**Noname manuscript No.**
(will be inserted by the editor)

# Inverse Design of Self-Oscillatory Gels through Deep Learning

**Doruk Aksoy* · Silas Alben · Robert D. Deegan · Alex A. Gorodetsky**

the date of receipt and acceptance should be inserted later

**Abstract** We develop a deep learning architecture for inverse design of a self-oscillating sheet propelled by an embedded chemical reaction. The dynamics of our problems are nonlinear and exhibit chaotic behavior, a challenging setting for existing deep-learning-based inverse design approaches. The aim is to explore data-driven design of soft robots using a novel locomotion mechanism. We train the architecture using a forward model of the locomotion mechanism developed recently by Alben et. al (2019). The architecture is shown to successfully map a snapshot of target motions of the gel into geometric and reaction parameters. The final architecture consists of a Multi-layer Perceptron (MLP) classifier for discrete parameters followed by a stacked MLP regressor (SMLPR) for continuous parameters.Our inverse design setting is unique in that it considers both discrete and continuous outputs, requiring an architecture capable of classification and regression. We are able to recover parameters within 2.87% accuracy. We also compare the simulated motion of the sheets at the recovered parameters. Because the motion has a chaotic quality, our demonstration is able to show quantitative agreement for a small time horizon and qualitative agreement over longer time horizons. We also demonstrate agreement of Lyapunov exponents up to 6.78% accuracy for suitable motions.

Doruk Aksoy* (Corresponding Author) · Alex A. Gorodetsky
Department of Aerospace Engineering, Universitiy of Michigan, Ann Arbor, MI, 48109 United States
* E-mail: doruk@umich.edu

Silas Alben
Department of Mathematics, University of Michigan, Ann Arbor, MI, 48109 United States

Robert D. Deegan
Department of Physics, University of Michigan, Ann Arbor, MI, 48109 United States

**Keywords** Inverse Design · Soft Robotics · Self-oscillating Gel · Simulation Based Design · Stacked Multi Layer Perceptron

## 1 Introduction

Conventional robots with rigid bodies can be designed and programmed for accomplishing specific tasks efficiently [1,2]. However, this efficiency comes with a price in adaptability. On the other hand, soft robots that are inspired by structures existing in nature offer flexibility to different tasks [3] and promise increased collaboration opportunities between man and machine [4]. As attention in this subfield of robotics increases, successful adoption of these approaches becomes more frequent across areas including surgical tools [5], marine exploration [6], wearables [7,8] and assistive devices [9].

One of the main challenges to deploying soft robots in real-life conditions is providing energy or signal to actuators to initiate or sustain locomotion. A variety of actuation possibilities have emerged in recent years: [10] uses pressurized air flow through special channels within the robot to actuate finger-like soft structures, [11] uses various cables to mimic the pushing-based locomotion and object grasping of octopi, [12] uses hydraulic pressure to create a helical gripper that mimics snakes and elephant trunks and [13] uses dielectric elastomer actuators to mimic the crawling motion of inchworms. However, all of those options share a common problem: they either need to be connected to external power/signal sources or are required to have onboard power supply. Furthermore, the control circuitry and actuators determine a lower limit on the size of the robots, which may pose a problem for operations in confined spaces.

A variety of propulsion methodologies have been proposed for smaller scale robots without those limitations: [14] proposes propulsion through altering the surrounding magnetic field to micromachines, [15] utilizes acoustic waves to actuate an untethered device and control its movement, and [16] uses structured light to trigger liquid crystal polymers to motion. In this paper, we consider an alternative approach based on the motion of chemical-reaction driven sheets of gels.

The idea of inducing motion of gels using oscillatory chemical reactions was introduced to soft robotics by Yoshida et al. [17]. Yoshida and collaborators have since shown various transduction schemes; a C-shaped gel that propels itself across a toothed surface as the raction in the active gel causes it to expand and contract [18]; artificial cilia [19,20]; surface waves that induce rolling and translation of a cylinder [21], and transport of a bubble within a tube [22]. The oscillatory motion resulting from the chemical reactions was investigated and modeled in [23]. On the other hand, instead of having the polymer gels react to their environment autonomously, [24] used simulations to investigate using light to guide the response of self oscillating gels. That non-invasive method of triggering the BZ reaction was promising, since it allowed the reaction (and therefore the resultant motion) within the soft material to be controlled in a purpose-driven way. Being able to control the motion of gels without a physical actuator will contribute greatly to completely untethered, self sufficient robots for the soft robotics field.

By extending previous applications on the luminously -triggered gels to sheet form, [25] demonstrates how useful light can be in this context by creating and modifying the pattern of the BZ reactions and therefore implying the controllability of this process. Furthermore, in the same work, a non-Euclidean elasticity approach was presented and verified with experiments and simulations. In [26], the authors used a semi implicit method to simulate self-oscillating gel, and the results of that paper form the basis of the present work.

Though the behaviour of these self oscillating gels can be accurately and efficiently simulated for a fixed set of parameters, a robust approach to find parameters that induces a target motion does not exist. Since these parameters are typically continuous, the motion of the sheet is sensitive to the choice of parameters, and the motion of the sheet is nonlinear and exhibits chaos; brute force tuning of parameters may not yield precise results.

This paper provides a systematic case study for the development of an inverse design tool that maps sheet motion to sheet parameters. Inverse design is an *in silico* technique adopted by different engineering disciplines that aims to accelerate the search process in multidimensional continuous spaces. [27] uses an MLP based inverse design architecture in design of microstrip antennas, [28] uses inverse design with generative deep neural networks to discover and design novel materials, [29] uses convolutional neural networks to design airfoils, and [30] uses inverse design to determine the sample structure information from measured spectrum.

Machine learning-based inverse design is also used in the soft robotics world; [31] uses an FEM-based machine learning architecture to obtain the inverse sensor function. We adopt a similar method to [32] for constructing the training and validation data in this work: instead of collecting real-life data of oscillating gels, we use the simulation of system proposed in [26] to create our data set, which makes our inverse design architecture a *simulation based inverse design.*

The contribution of this paper is a deep learning architecture that is able to effectively perform inverse design for determining geometric parameters and parameters of the chemical reaction to induce a target motion of the self-oscillating sheet. We step through a comprehensive set of hyper-parameter tuning activities and we finally demonstrate verification simulations. For at least 50% of the feasible domain of parameters, we are able to predict parameter settings for motions that are not in the test set within 2% with this architecture.

### 1.1 Comparison with Existing Approaches

In this section we compare and contrast other inverse design approaches with our proposed architecture and application. Inverse design is an emerging tool in the soft robotics world with very few examples and to our knowledge, this work is the first work bridging deep learning powered inverse design methods with autonomously actuated soft robotics. Inverse design has predominately been used in materials and crystals research, which exhibit different problem structures and can exhibit different types of complexity. Below we review several inverse design approaches, where we see a common theme that our application and approach is distinguished by the fact that (1) we consider a nonlinear, dynamical rather than a static, system; (2) we consider an architecture that does both regression and classification; and (3) we provide an extensive hyperparameter tuning procedure that also gives an insight into architecture robustness.

In [33], the authors investigated inverse design in context of nanoparticle light scattering spectra. Similar to our approach, they create training set through exhaustive numerical simulations on a sparse subset of the simulation domain. Then, they train an MLP NN

to replace a forward simulation approximating Maxwell interactions. By reversing the MLP network and training it in the "reverse" direction to fine-tune the inverse mapping, they obtain a framework calculating the necessary layer thicknesses for the nanoparticle that would give the prescribed scattering profile. In contrast to our proposed framework that provides a single architecture including both classification and regression, the authors assume the discrete parameter (number of layers of the nanoparticle) is known and train separate networks for each one.

In [34], the authors develop crystal graph CNNs framework to learn material properties from the connection of atoms in the crystal. Instead of simulated data or images, they use crystal graphs that encode atomic information and atomic configuration in the crystals. By passing that feature vectors through the CNN framework, they obtain both discrete and continuous properties of a given crystal. In contrast to this example, we were able to achieve our target accuracy without resorting to complicated architectures such as CNNs. However, in order to achieve our stringent target accuracy we needed to include a stacking network on top of the first MLPR. Another major difference between the two approaches is the fact that we are considering the dynamics of our system of interest, whereas [34] does not.

Inverse design has also been used for predicting airfoil/wing shapes and parameters from aerodynamical parameters. In [35], the authors train various types of NNs (including MLP) using a small dataset comprised of different airfoil types to obtain geometrical parameters of the requested airfoil/wing to accelerate aerodynamic design processes tailored to given criteria. In contrast to our application, the authors use self-organizing maps as multi-class classification to help their framework decide which airfoil type to choose and then determine the geometrical parameters of the selected airfoil type. By adopting this method, they alleviate the adverse effects of having a small training set and high-dimensional target. In [29], the authors considered a similar application, but instead they use 2D $C_p$ distribution images from simulations and various CNN architectures to estimate the airfoil shape responsible for the given $C_p$ distribution. They have conducted a more concise version of network hyperparameter tuning procedure that resembles to the one we have conducted in this work. However, they are using "static" images of the $C_p$ distribution whereas we are considering a dynamical system of the oscillation of the gel sheet in our framework.

In [27] the author uses an MLP based inverse design framework to obtain structural parameters of microstrip circular antennas that would give a prescribed frequency response. Similar to our approach, the author uses a forward simulation including PDEs to generate the training data. However, the input to their inverse design framework is in frequency domain, whereas our architecture works with inputs in time domain. Additionaly, the methodology in [27] lacks the classification stage that is essential to our method.

In [36], the authors use a supervised autoencoder (SAE) based inverse design architecture, where they feed in vectorized images of graphene kirigami structures and reduce to a 10-dimensional latent space including the ultimate stress and strain as supervised parameters. The authors embed inverse design into an architecture mainly used for dimensionality reduction and in addition, they force two elements of the latent space to be physical properties such as stress and strain. Another noteworthy achievement in [36] is the ability of the framework's ability of interpolation between two distinct cut orientations (horizontal, vertical) to get shapes that have diagonal slits. Since we do not investigate superposition of multiple wave types in our analysis, we refrain from using interpolation methods between classes in our framework. Moreover, we separate the inverse design stages for the distinct classes using a MLP classifier to make the regression afterwards easier.

In [37], the authors use an encoder-decoder architecture comprised of 1D convolutional layers to obtain geometrical parameters of an acoustic sink using given a sound absorption coefficient profile. Using the simulated parameter-absorption profile pairs, the authors train the encoder and decoder portions of the proposed framework separately. For some selected profiles, they manufacture the acoustic sinks using additive manufacturing and test the sound absorption profile using an experimental setup. Finally, they compare the sound absorption coefficient profiles of the prescribed, the estimated (using the decoder) and the built acoustic sinks. We do not seek to replace the forward simulation developed by [26], therefore we do not need anything similar to the decoder stage in [37]. Instead, we introduce stacking to the regression stage to account for the errors in the predicted parameters due to their position in the 3 dimensional coordinate space ($\mathbb{P} \subset \mathbb{R}^3$). Finally, [37] also lacks the classification stage that plays a crucial role in our framework.

To summarize, the existing approaches consider methods ranging from autoencoders to different CNN setups. Even though there exists similar MLP-based inverse design approaches in the literature, none of the aforementioned works (except [35] which implicitly does multi-class classification through self-organizing maps) pro-

vide classification and regression within a single architecture. Additionally, none of the listed works do inverse design on a dynamical system. Last but not least, despite the differences in applications and methodologies followed, only [29] follows a hyperparameter tuning procedure such as we do in the sections that follow. The rigorous procedure we follow in our work helps us justify the robustness of our methodology and provides explanation for the particular hyperparameter setting that we decide at the end.

The rest of this paper is organized as follows: Section 2 provides detail about the forward simulation by introducing the underlying equations and discretization, Section 3 describes the inverse design problem and introduces the proposed two-staged architecture, Section 4 explains the experimental procedure along with the obtained results for both classification and regression as well as with the verification simulations, and Section 5 summarizes our findings.

## 2 Forward simulation

In this section, the simulation model used as the forward simulation for generating data is described. This simulation model is based on a semi-implicit solver developed by Alben and co-authors [26].

### 2.1 Equations for the dynamics of flexible sheets

This section describes the equation used to simulate a hexagonal self-oscillating gel sheet defined via a triangular lattice (for example, left panel of Fig. 1). The lattice has stretching springs between neighboring vertices, and bending springs with energy proportional to the square of the (dihedral) angle between neighboring triangular faces. As the lattice spacing tends to zero, the energy closely approximates that of a continuum isotropic elastic sheet [38,39,26]. The motion of the lattice is driven by time- and space-varying distributions of the rest lengths of the stretching springs. In the motivating experiments on thin gel sheets [25], there are chemical waves, radial or spiral in form, that induce local swelling of the sheets. As a simple model, we study radial or planar (unidirectional) traveling waves in the corresponding simulations:

$$\textit{Radial wave}: \ \eta(x,y,t;A,k) = 1 + A\sin\Big(2\pi(k\sqrt{x^2+y^2}-t)\Big) \quad (1)$$

$$\textit{Planar wave}: \ \eta(x,t;A,k) = 1 + A\sin(2\pi(kx-t)) \quad (2)$$

Here $\eta$ is the relative change in rest length of the stretching springs, and is a sinusoidal traveling wave function of the $(x,y)$ location in the undeformed sheet and time $t$, with wave amplitude $A$, and wavenumber $k$. The sheet moves by overdamped dynamics

$$\mu\frac{\partial \mathbf{r}}{\partial t} = \mathbf{f}_S(\mathbf{r};K_s,\eta) + \mathbf{f}_B(\mathbf{r}). \quad (3)$$

with stretching force $\mathbf{f}_S$, bending force $\mathbf{f}_B$, and damping constant $\mu$. We nondimensionalize quantities with dimensions of length by the radius of the hexagonal sheets, and time by the period of $\eta$ (assumed to be time-periodic). $\mathbf{f}_S$ is proportional to the stretching stiffness $K_s$ (nondimensionalized by bending stiffness, and thus this constant is proportional to sheet thickness to the $-2$ power [26]). The rest strain function $\eta$ enters $\mathbf{f}_S$ explicitly, but not $\mathbf{f}_B$. If the sheet is initially nearly planar, with a small out-of-plane perturbation, and $\eta$ is spatially nonuniform, $\mathbf{f}_S$ can cause the perturbation to grow, i.e. buckling. Buckling occurs above a critical threshold of the rest strain amplitude $A$. The threshold decreases as $K_s$ increases, i.e. when the stretching force becomes more dominant over the bending force, which resists buckling.

To solve Eq.(3), we apply a semi-implicit method with a 2nd-order-in-time backward differentiation discretization, developed in [26]:

$$\mu A_p\frac{3\mathbf{r}^{n+1}-4\mathbf{r}^n+\mathbf{r}^{n-1}}{2\Delta t} = K_s\mathbf{L}\mathbf{r}^{n+1} + 2\mathbf{f}_{SE}(\mathbf{r}^n) - \mathbf{f}_{SE}(\mathbf{r}^{n-1}) + \mathbf{B}\mathbf{r}^{n+1} - 2\mathbf{B}\mathbf{r}^n + \mathbf{B}\mathbf{r}^{n-1} + 2\mathbf{f}_B(\mathbf{r}^n) - \mathbf{f}_B(\mathbf{r}^{n-1}) \quad (4)$$

The superscripts denote time steps. On the right hand side of Eq. (4), the first three terms represent the stretching force. Of these, the first, depending on $\mathbf{r}^{n+1}$, is implicit and linear (with $\mathbf{L}$ a discrete Laplacian matrix), and the second and third (denoted $\mathbf{f}_{SE}$) are explicit and nonlinear, but bounded in norm. The remaining terms are the bending force, again with a linear implicit term (involving $\mathbf{B}$, a discrete biharmonic matrix) that approximates the bending force at time step $n+1$, while the remaining terms approximate the remainder of the bending force with second-order temporal accuracy.

### 2.2 Examples of flexible sheet dynamics

We spatially discretize an initially flat hexagon of radius 1 with an equilateral triangular lattice mesh, with initially uniform mesh spacing 1/33, giving 3367 mesh

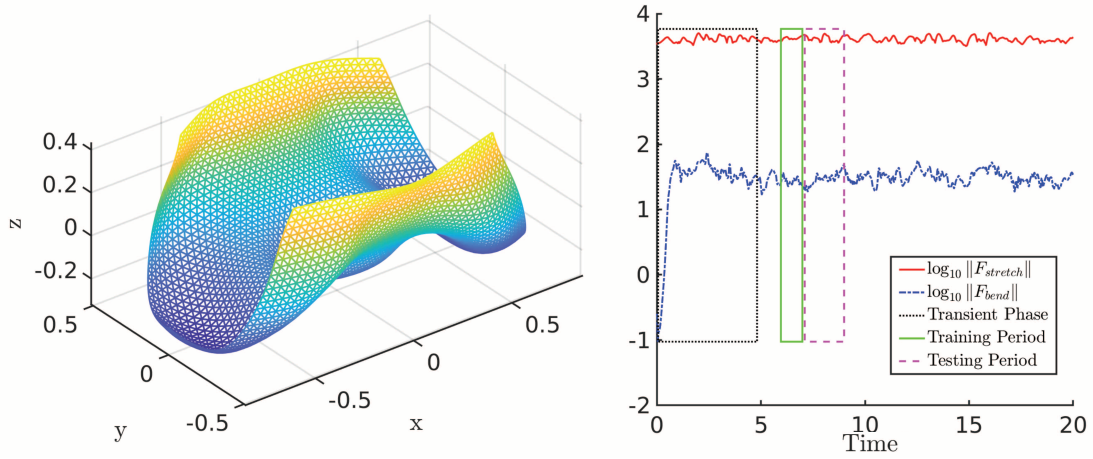



Fig. 1: A snapshot of the flexible sheet (left) and stretching and bending force norms (right) at $t = 20$. Sheet parameters: $A = 0.1$, $K_s = 7500$, $k = 2$, $T$ =Radial.

points. We apply a small out-of-plane perturbation, and evolve the sheet forward in time with Eq. (4). For large enough wave amplitude $A$, the sheet rapidly buckles into shapes with time-varying distributions of curvature, large in magnitude. Each simulation is run for $t \in [0, 20]$ to allow a sufficient amount of time for the sheet to evolve beyond the transient initial buckling motion. Fig. 1 shows a snapshot of the sheet (left panel) during an example of the dynamics. The right panel shows the corresponding time traces of the norms of the stretching force (red) and bending force vectors (blue), denoted $F_{stretch}$ and , $F_{bend}$.

A set of snapshots of the motion over time for varying $A$ is shown in Fig. 2. Note that with increasing wave amplitude the sheet experiences different patterns of curvature resulting from outward-moving circumferential bands of dilation and contraction. The range of the simulation parameters for which this occurs is discussed in Section 4.1. We take the sheet position $\mathbf{r}$—$x, y$ and $z$ coordinates at each of the 3367 lattice points, and for each time step—as the training features. We also use the stretching and bending force vectors, which have the same dimensions as the position. The stretching and bending forces involve second- and fourth-order derivatives, respectively, of the lattice positions. These forces can be written in terms of the sheet curvature and its spatial derivatives, and hence give measures of the sheet shape.

In Fig. 1 right panel, the force vector norms evolve nonperiodically in time, reflecting nonperiodic dynamics of the sheet position. Nonetheless, the sheet shape is often smooth, with certain approximate spatial symmetries, such as a bilateral symmetry for the snapshot in the left panel.

## 3 Inverse design for the flexible sheet analysis

In this section, we describe the inverse design problem — the task of mapping motion to parameters. Motion is explicitly defined by a discrete sequence of snapshots of the shape of a three dimensional lattice and associated derived quantities such as bending and stretching forces.

The inverse design problem is solved using data created by the simulation described in Section 2. The parameters, inputs, and simulation outputs used in this design problem are provided in Fig. 3. The sheet parameters are amplitude $A$, stretching stiffness constant $K_s$, wavenumber $k$, and wave type $T$. The amplitude $A$, sheet stiffness $K_s$ and wavenumber $k$ are all continuous parameters. The type $T$ is a binary variable specifying wave form type — 0 for radial travelling wave and 1 for planar travelling wave. The ranges of the continuous parameters are chosen to avoid sheet deformations which are not physically possible, as described in Section 4.1.

Our contribution is a learning architecture that converts target motion into sheet/reaction parameters. This architecture has two stages: the first stage is a classifier that determines wave types from lattice snapshots, and the second stage consists of two regressors — one for each wave-type — to identify the continuous parameters. Principal Component Analysis (PCA) is used as the main preprocessing step for the inputs in both stages.

Standardization of data before and/or after applying PCA is done for multitude of reasons: 1) to neutralize the effects of outlier in the data [40], 2) to make the units of the variables comparable [41] and 3) to neutralize the difference in the orders of magnitude between

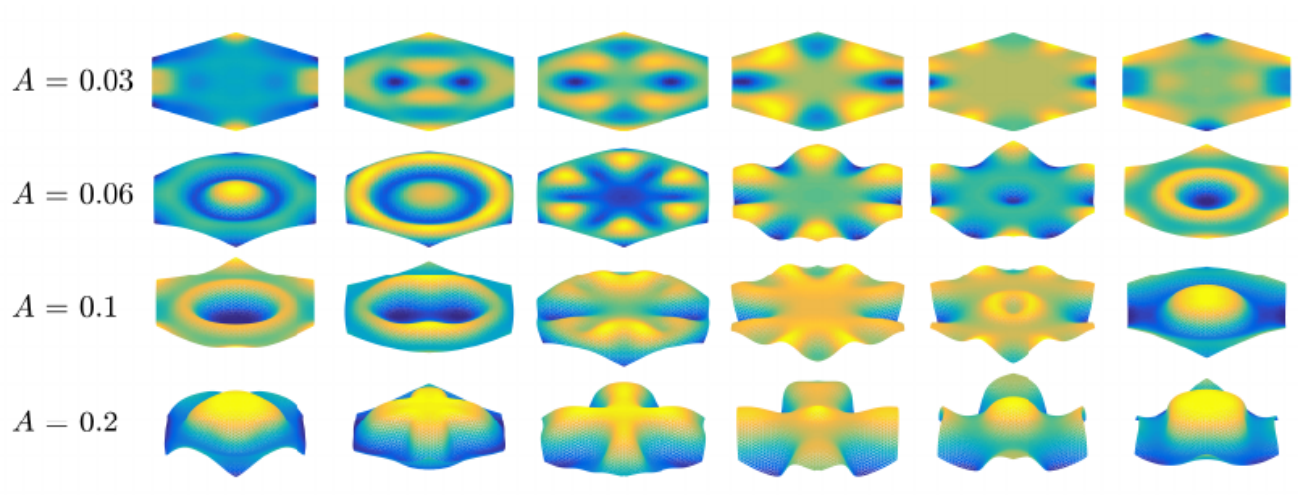



Fig. 2: Snapshots of the sheet with various values for $A$, listed at left, between $t = 19$ and $t = 20$ (0.2 time units between each image). Here we have a radial wave for $\eta$ in (1) with $k = 1$; and we take $K_s = 1.15 \cdot 10^4$. [26]

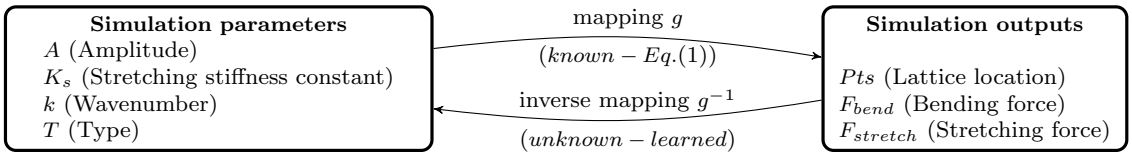



Fig. 3: Inverse engineering diagram for the flexible sheet analysis.

data [42]. A schematic of this architecture is shown in Fig. 4. In the following sections, the components of this architecture are discussed in detail.

### 3.1 Stage 1: MLP Classifier with PCA preprocessing

The first stage is a Multi-Layer Perceptron (MLP) classifier [43] that distinguishes between reaction wave types. Since there are different types of symmetries on the sheet for different types of motions (radial symmetry for radial waves and plane symmetry for planar waves) we found that using only the $x$ coordinates of the lattice points are sufficient to determine the motion type.

We also find that preprocessing with principal component analysis (PCA) both accelerates the learning algorithm as well as increases accuracy by up to 50% (for single snapshot training) and 3% (for 10 Snapshot training), as will be discussed in Section 4. For each training snapshot, the raw features used as inputs to PCA are the $x$ coordinates of 3367 lattice points obtained from the simulations. We found it is not necessary to use the time dependencies of the snapshots to obtain accurate predictions of reaction wave type.

PCA preprocessing extracts linearly uncorrelated features from the lattice positions and reparameterizes the data along new orthogonal axes with decreasing variance [44,45]. It can be computed using the singular value decomposition (SVD) [46] as follows. Let us consider $n$ data points, each with $m$ features. SVD decomposes the data set organized in an $m \times n$ matrix, $X \in R^{m \times n}$, into three factors

$$X = U\ \Sigma\ V^T \text{ where } U \in \mathbb{R}^{m \times m}, \quad \Sigma \in \mathbb{R}^{m \times n},\ V \in \mathbb{R}^{n \times n}. \quad (5)$$

The columns of $\tilde{U} = U\Sigma$ represent principal directions onto which $X$ is projected. The projected values are called principal components. The projection of $X$ onto the first $p$ principal directions ($\tilde{u}_1$ to $\tilde{u}_p$ with $\tilde{u}_i$ sorted in decreasing singular value order) creates the new data matrix $Y$, whose coordinates are defined by the columns of $U$

$$Y = [\tilde{u}_1, \cdots, \tilde{u}_p]^T\ X, \text{ where } Y \in \mathbb{R}^{p \times n},\ p \leq m \quad (6)$$

Using this formulation, the size of the feature space is reduced to $p$.

PCA preprocessing enables a reduction in the number inputs into a neural network, and thereby enables a more compact network architecture [47]. Furthermore, evidence suggests that the resulting PCA-NN architecture can achieve better performance in some cases [48], with multiple successes found in the literature [49–51].

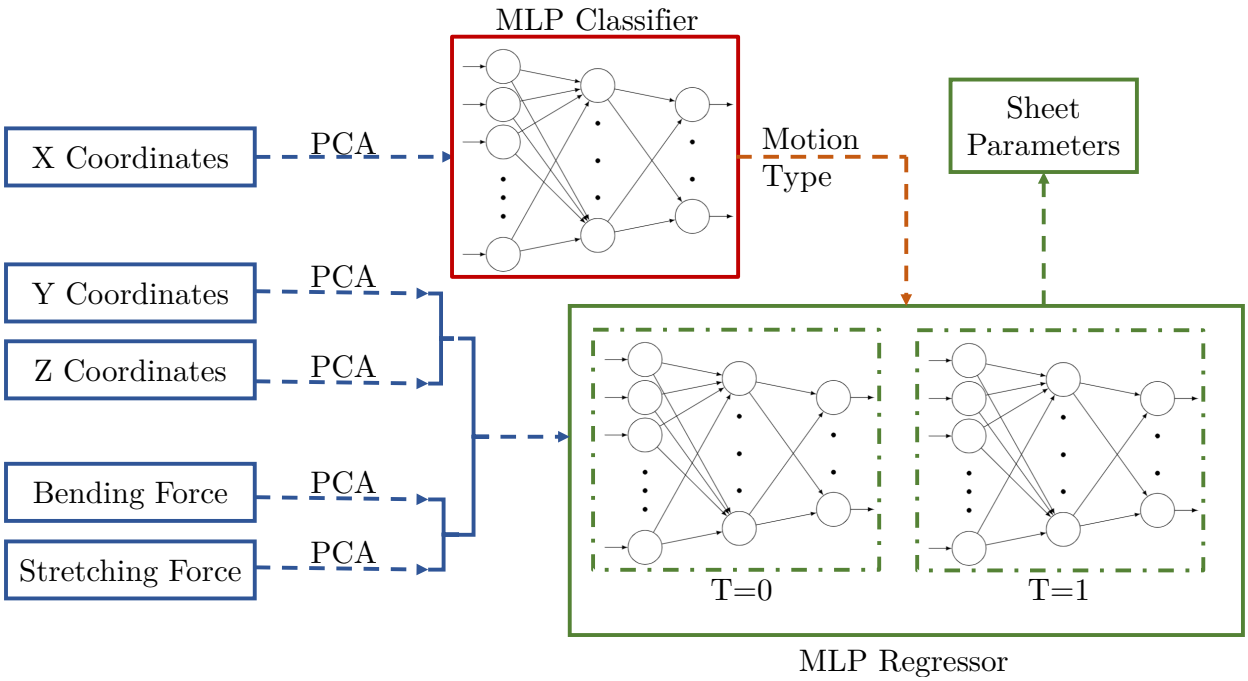



Fig. 4: Structure of the inverse mapping architecture. MLP classifier (red) is used to determine the motion type on the gel. According to the motion type, the architecture selects one of the specialized MLP regressors (green) and predicts the continuous sheet parameters.

Though PCA offers reduction of dimension for data sets, the size of the data sets used in learning can still hinder learning algorithms due to large memory requirements for standard PCA. To prevent this problem we used a memory efficient version of PCA, called Incremental PCA (iPCA) [52,53]. Incremental PCA determines principle components by using a sequence of mini-batches of the initial data set.

Following iPCA and renormalization, the new features passed through a classifier that is comprised of fully connected layers, of which at least one is a hidden layer. ReLU activation functions are used in the hidden layers. The output layer uses a sigmoid function to obtain a value between $[0, 1]$. The output is then rounded to the nearest integer to indicate one of the labels for the motion of the gel.

### 3.2 Stage 2: MLP Regressor with PCA Preprocessing

The second stage of the architecture is a stacked set of regressors, one stacked regressor for each wave type. These regressors aim to identify the three continuous sheet parameters: amplitude, stiffness, and wavenumber ($A, K_s, k$ respectively).

The features used for this portion of the architecture are the $y$ and $z$ coordinates of the lattice points and the resultant bending and stretching forces on the gel sheet ($f_b$ and $f_s$, respectively). The force components are nonlinear combinations of higher order derivatives of the lattice positions. We found that using this physics-inspired quantity avoids the need to incorporate convolutional layers that also essentially extract derivatives.

Since the regressor networks are specialized to single reaction wave type (either radial or planar), only the simulations of one wave type is used as training set. For each snapshot, the raw features used as inputs to another PCA preprocessing procedure (see Fig.5) are the $y$ and $z$ coordinates of 3367 lattice points as well as 10 bending, and 10 stretching force norms that are calculated between each snapshot (a sum of $6754 = 2 \times 3367 + 10 + 10$ total features). Similar to the classification stage, the time dependencies of the training snapshots are not taken into consideration.

To keep the analysis concise, we are using a fixed dimension reduction via PCA transformation from 10 to 7 for both bending and stretching force data. We want to keep the number of the principal components for force components as high as possible so that the force components are not outnumbered by the coordinate components at the input layer of the networks. The performance may increase by changing this coor-

dinate transformation but we keep it out of our analysis because we were able to achieve excellent results even by fixing it at 7.

This regressor structure outputs the three continuous sheet parameters. Unlike the classification stage, this stage consists of two stacked networks as shown in Fig. 5.

This parameter estimation problem can be visualized as following: a given simulation's data coresponds to a unique coordinate in the feasible parameter space ($\mathbb{P} \subset \mathbb{R}^3$) and our aim is to find that coordinate in $\mathbb{P}$ using the proposed SMLPR structure. The first regressor network stage makes an initial prediction about the coordinate of a given simulation data in $\mathbb{P}$. However, this estimation has low robustness (high percentile values over the test set). The second regressor network uses the outputs of the first network (preliminary coordinate predictions of the three continuous sheet parameters in $\mathbb{P}$) on top of the inputs used at the first network (preprocessed $x, y$ coordinates and $f_b, f_s$) as input and outputs the final predictions for the three continuous sheet parameters. Adding the preliminary coordinate estimates from the first network to the coordinate and force components reduces the distance between the actual and estimated parameters in the parameter space $\mathbb{P}$.

Furthermore by including the initial predictions of the input data in the second network's inputs, we aim to eliminate the underlying effects of the location of the parameters in the 3D parameter space on the estimation accuracy.

Similar to the classification portion, ReLU activation function is used for all fully connected layers with the exception of output layers of each network due to regression purposes.

## 4 Experiments and Results

In this section we describe the data generation and training procedures; present our approach to architecture determination; and demonstrate inverse design performance.

Scikit-Learn [54] is used for all preprocessing and PyTorch [55] is used for all neural network training. Computers used for training the networks are equipped either with an NVIDIA Quadro P1000 GPU with 4GB memory[1] or with an NVIDIA GeForce GTX 1050 Ti with Max-Q Design with 4GB memory[2]. All of the graphics card drivers are up to date at the time of the trainings[3]. The classifiers are trained using binary cross-entropy loss. The regressors are trained using squared loss. The optimization procedure use the Adaptive Moment Estimation (ADAM) stochastic gradient descent [56]. Hyperparameters for
ADAM are set as in [56]: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. In addition to the hyperparameters, the sizes of the batches of each training method are adjusted so that there are 26 batches for all classification case studies to enable reasonable memory usage on our hardware and provide a fair ground to evaluate the classification accuracy by fixing the number of updates on the weights.

We use a fixed number of 20 training epochs for classification case studies to keep the training time short and avoid overfitting of the network. For regression we use 200 epochs as baseline epoch count. The number of batches for regression studies are limited to 256 to limit the memory usage on our hardware. We did not find significant sensitivity to the number of epochs in the classification case. The effect of the number of epochs for the regressors is investigated in Section 4.3.

### 4.1 Data generation

In this section, we identify the parameter domain, training data, and validation data.

The parameter bounds are determined through random and targeted sampling across the simulation space to identify a rectangular region containing feasible motions. This procedure yields $A \in [0.01, 0.22]$ for the amplitude; $K_s \in [10^3, 10^4]$ for the stretching stiffness; and $k \in [0.1, 10]$ for the wave number.

The parameter domain for these continuous variables is then obtained as a tensor product space $\mathcal{P}$.

To generate the training data we discretize $\mathcal{P}$ into a $16 \times 20 \times 20$ grid of equidistant points. For each parameter combination $p^{(i)} \in \mathcal{P}$ we generate a simulation for both wave types — yielding $6400 = 16 \times 20 \times 20$ simulations per wave type.

Each simulation results in a solution $r^{(i)}(t)$ for $t \in [0, 20]$. Our networks are trained at certain snapshots of this solution, and these snapshots are selected from $t \in [6.1, 7.0]$. Fig. 1 shows that this time is after the initial transient phase and still provides us with times in the future with which we can test extrapolation.

[1] source: www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/301968-DS-NV-Quadro-Pascal-P1000-US-03Feb17-NV-fnl-WEB.pdf

[2] source: www.geforce.com/hardware/desktop-gpus/geforce-gtx-1050-ti/specifications

[3] February 20, 2020

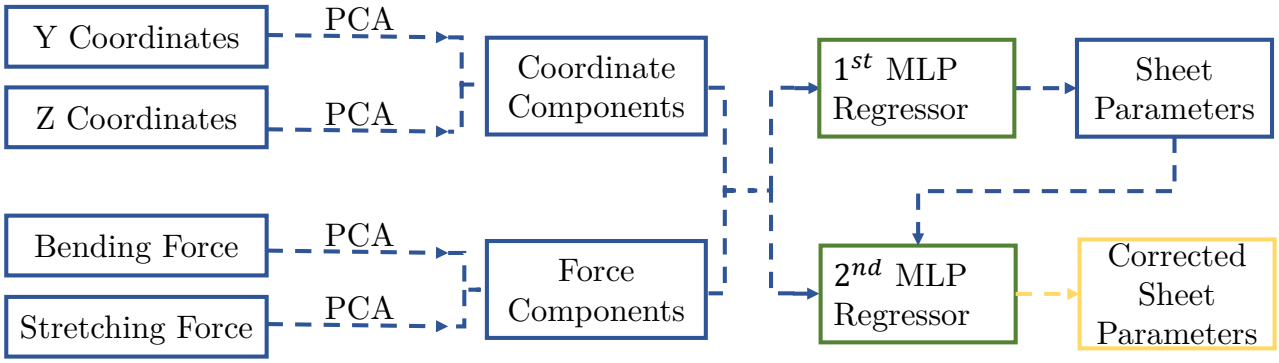



Fig. 5: Stacked MLP Regressor (SMLPR) architecture consists of two components (green boxes). First layer MLPR outputs the sheet parameters and those sheet parameters are fed into the second MLPR stage along with force and coordinate components. The second stage outputs the sheet parameters more accurately (yellow box).

We generate two different types of training data sets from these trajectories. In the first version, which we call *single-snapshot-per-simulation*, we choose only a single time instance for the training data. In this case we will use ten different training data sets to train ten different networks. Specifically, each of the data sets corresponds to snapshots from $t = 6.1, 6.2, \ldots, 7.0$. In the second version, *multi-snapshot-per-simulation*, we use a single larger training set with all ten-snapshots-per-simulation in the training data.

Both networks will be tested on a validation set $\mathcal{V}$. This validation set is obtained by generating 15000 random uniformly distributed parameters across the input domain $\mathcal{P}$ and then obtaining simulation results for each wave-type. To test extrapolation performance, a total of 20 validation snapshots are taken from each of these simulation from the time span $t \in [7.1, 9.0]$.

## 4.2 Classification

In this section we perform a parametric study over network architectures to determine a classifier. This study is a single feed-forward process by which we first optimize the number of hidden units, then the number of hidden layers, and then the number of principle components. We show that this process leads to sufficient accuracy for the classification network. The baseline architecture from which we begin has 1 hidden layer and 10 hidden units. This process is performed for both the single- and multi-snapshot training sets outlined above. The single-snapshot results are provided in Section 4.2.1, and the multi-snapshot results are provided in Section 4.2.2.

***Metrics*** These studies consider three performance metrics: mean classification accuracy (MCA), worst case classification accuracy (WCA), and training time.

For one snapshot training, $MCA_1$ is the average classification accuracy that networks trained using a single $r^{(i)}(t)$ from $t \in [6.1, 7.0]$ achieve over $\mathcal{V}$ (Eq. 7).

$$MCA_1 = \frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\frac{1}{20}\sum_{k=1}^{20}\frac{1}{10}\sum_{j=1}^{10}\delta(T^{(i)}, \hat{T}^{(i)}(t_k; t_{train,j})) \quad (7)$$

where $T^{(i)}$ is the true type of the $i$-th simulation in the validation set $\mathcal{V}$; $\hat{T}^{(i)}$ is the predicted type for the $i$-th simulation in the validation set where the prediction is based on a snapshot in $\mathcal{V}$ from time $t_k \in \{7.1, 7.2, \ldots, 9.0\}$ (20 total); $t_{train,j}$ is the time at which snapshots for the training data were taken; and finally $\delta$ is the delta function (1 if arguments are equal, and 0 otherwise). The sum over the time of the training snapshot is used to obtain to the average performance of a single-snapshot-trained network.

$WCA_1$ is the minimum average classification accuracy that networks trained using a single $r^{(i)}(t)$ from $t \in [6.1, 7.0]$ achieve over $\mathcal{V}$ (Eq. 8).

$$WCA_1 = \min_{j\in[1,10],k\in[1,20]}\frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\delta(T^{(i)}, \hat{T}^{(i)}(t_k; t_{train,j})) \quad (8)$$

For the ten-snapshot training, we do not average over different training data sets because the training is performed using all ten snapshots of $t \in [6.1, 7.0]$. As a result, the $MCA_{10}$ becomes:

$$MCA_{10} = \frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\frac{1}{20}\sum_{k=1}^{20}\delta(T^{(i)}, \hat{T}^{(i)}(t_k)), \quad (9)$$

and the $WCA_{10}$ becomes:

$$WCA_{10} = \min_{k\in[1,20]} \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \delta(T^{(i)}, \hat{T}^{(i)}(t_k)). \quad (10)$$

MCA [57,58] and WCA [59,60] are used widely in the learning literature to provide a more holistic view by accounting for both the overall performance and the reliability of the proposed networks.
Training time is used to differentiate between networks with similar MCA and WCA values. However, in other applications, this timing can have a higher impact factor [33,30]. If all three performance metrics yield similar results, the network with the minimum dimensions (number of hidden layers, number of units or number of principal components) is preferred. Our aim is to have both MCA and WCA $\geq 99\%$ for classification.

***Procedure*** The analysis proceeds by first choosing the number of hidden units, then the number of hidden layers, and finally refining the number of principal components. In the first two steps we also search over a coarse grid of principal components, ranging between 10 to 2000, so that the hyperparameter search is not entirely a greedy search. Results for each network combination are provided in Figs. 7-11 and Figs. S1-S3.

Figs. 7-9 and Figs. S1-S3 (provided among supplementary materials for more detailed analysis) correspond to single snapshot-per-simulation training. Each column represents training with a different time $t_k$, where $t_k$ is denoted at the top of the figures. Each row of these figures represent testing on different time-intervals of the validation set: the first row uses $t$ between $7.1 - 8.0$ and the second row uses $t$ between $8.1 - 9.0$. Both rows demonstrate extrapolation in time.

Fig. 11 corresponds to the ten snapshot training studies, recall that this approach trains a single network on multiple snapshots (compared to ten networks at single snapshot training). Each column of these figures represent testing on different snapshots in the validation set $v^{(i)}(t) \in \mathcal{V}$. The first and third column of Fig.11a uses $t$ between $7.1 - 8.0$ and the second and fourth column of Fig. 11a uses $t$ between $8.1 - 9.0$. For Fig. 11b, first column uses $t$ between $7.1 - 8.0$ and second column uses $t$ between $8.1 - 9.0$.

We now describe hyper-parameter tuning results in detail.

#### *4.2.1 Single snapshot-per-simulation training*

We first investigate whether training on a single snapshot-per-simulation is sufficient. Fig. 6 summarizes the results of the parametric studies done using single snapshot along with the selected network parameters. Upon completion of the pass we achieve a network with $MCA_1$ of 95.3% and $WCA_1$ of 84.6%.

Figs. S1-S3 are provided among supplementary materials for more detailed results.

***Number of Hidden Units*** In this subsection we study the effect of the number of hidden units on classification accuracy. The worst performance is observed with 10 hidden units (87.2% $MCA_1$, 72.3% $WCA_1$) and the best performances are observed with 300 and 100 hidden units (95.3% $MCA_1$, 84.9% $WCA_1$ respectively).

Pointwise classification errors — per training validation simulation and time snapshot — are shown in Figs S1 for 10 units, S2 for one hundred units, and S3 for three hundred units.

From these figures, we calculate that $WCA_1$ and $MCA_1$ becomes better as the number of hidden units increases from 10 to 100 (Fig. S2 94.3% $MCA_1$ at 30 PCs, 84.9% $WCA_1$ at 30 PCs). However, beyond 100 hidden units, the performance of the MLP classifiers start to saturate. Specifically, as the number of hidden units increases from 100 to 300, $MCA_1$ increases up to 95.3% for 30 PCs, and we obtain a $WCA_1$ of 84.6% at 30 PCs (Fig. S3). Though the $WCA_1$ seems to deteriorate slightly, we choose 300 hidden units as the result of our analysis.

***Number of Hidden Layers*** Next we increase the number of hidden layers. Following this process we find an improved three-layer architecture with 97.0% $MCA_1$ and 84.2% $WCA_1$.

Pointwise classification errors — per training validation simulation and time snapshot — are shown in Figs 7 (for 2 layers) and 8(for 3 layers).

From these figures we calculate that $WCA_1$ and $MCA_1$ becomes better as the number of hidden layers increases to 2 (Fig. 7 96.3% $MCA_1$ at 30 PCs, 84.4% $WCA_1$ at 30 PCs). However, beyond 2 hidden layers, the performance of the MLP classifiers start to saturate. Specifically, as the number of hidden layers increases from 2 to 3, $MCA_1$ increases up to 97.0% for 10 PCs, and we obtain a $WCA_1$ of 84.2% (Fig. 8). Though the $WCA_1$ seems to deteriorate slightly, we choose 3 hidden layers as the result of our analysis. For further examples of networks that are not included in this work, the bitbucket repository[4] can be visited.

***Number of Principal Components*** Next we consider the number of principal components. The worst performance is observed under 5 PCs (91.2% $MCA_1$, 70.6% $WCA_1$) and the best performance is observed under 15 PCs (97.3% $MCA_1$, 87.5% $WCA_1$).

[4] https://bitbucket.org/dorukaks/workspace/projects/CGL

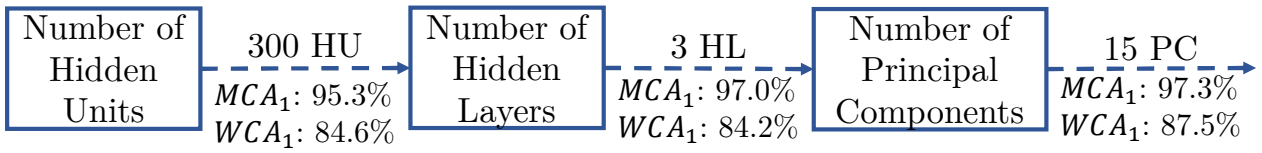



Fig. 6: Flow diagram for classification architecture determination using single snapshot-per-simulation training. Each box represents a sequential step in the parametric study of this section. The architectural choice is written above each arrow and the performance is written below.

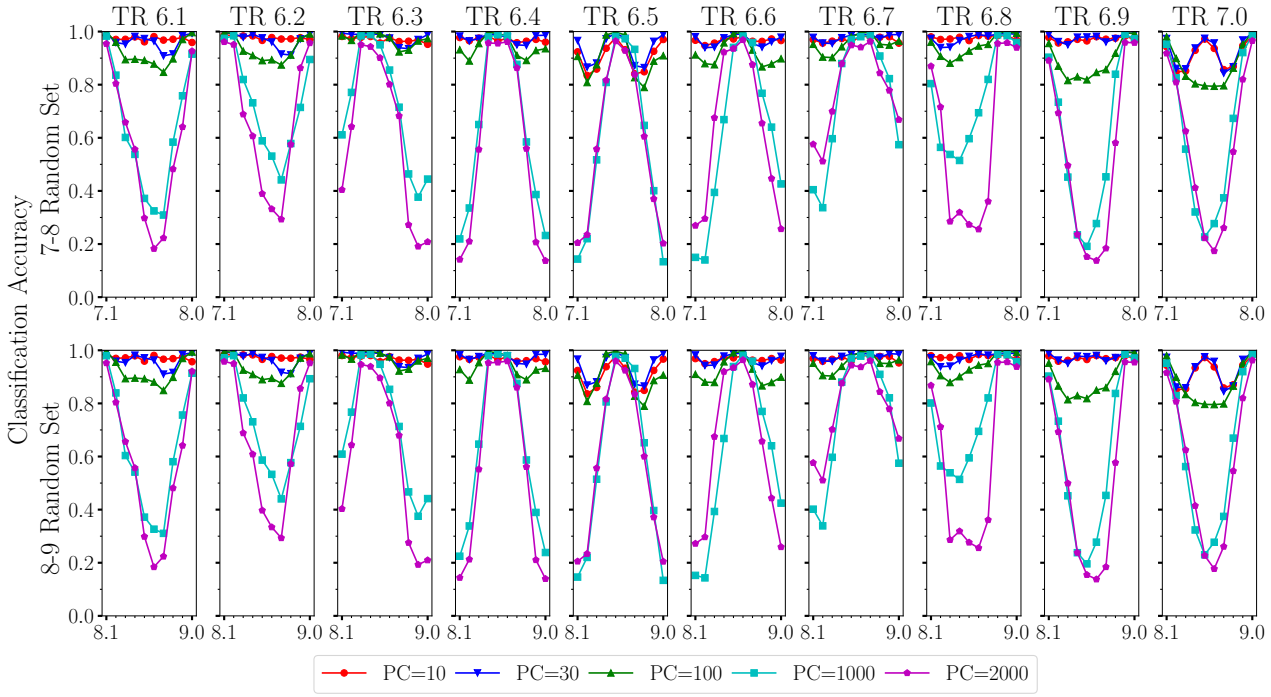



Fig. 7: Classification accuracies of 2-layer neural networks with $N_{units} = 300$ on snapshots from the validation set (top row: $t \in [7.1, 8]$, bottom row: $t \in [8.1, 9]$.). Columns correspond to networks trained with single-snapshots obtained at the indicated training time (TR). Summary: highest $MCA_1$=96.3% obtained with PC=30; highest $WCA_1$=84.4% obtained with PC=30.

Pointwise classification errors — per training validation simulation and time snapshot — are shown in Fig 9 for all of the analyses mentioned in this section.

Fig. 8 demonstrates peak performance when the number of principal components is $\leq 30$. Therefore we refine our analysis in this region, and the results are shown in Fig. 9. From these figures, we calculate that the $MCA_1$ are $> 96.7\%$ for $10, 15, 20$ and $25$ principal components. We also find that their training times are similar $(1.95 - 2.25s)$. As a result, we choose 15 PCs for the final architecture.

***Single-snapshot summary*** Our final architecture is an MLP classifier structure with 3 hidden layers, 300 hidden units and 15 PCs. A comparison of the pointwise classification error of this architecture with a neural network lacking PCA processing is shown in Fig. 9. We observe that without PCA preprocessing, all of the performance metrics become significantly worse. The $MCA_1$ drops down to 73.7% and training time (which is around $2s$ for the network with 3 hidden layers and 300 hidden units) almost doubles itself and jumps to $3.8s$.

Fig. S1 shows that the accuracy peaks at the validation snapshots from the same temporal phase as the training data for a given network (e.g. network trained with snapshots from $t = 6.7$ shows its peak performance at validation snapshots from $t = 8.7$, $9.7$ etc.). We see that all the single-snapshot analyses indicate periodic patterns in the classification accuracy — see Figs. 9 and S3 for the more obvious cases. This sinusoidal trend in the classification accuracy hints that a single time snap-

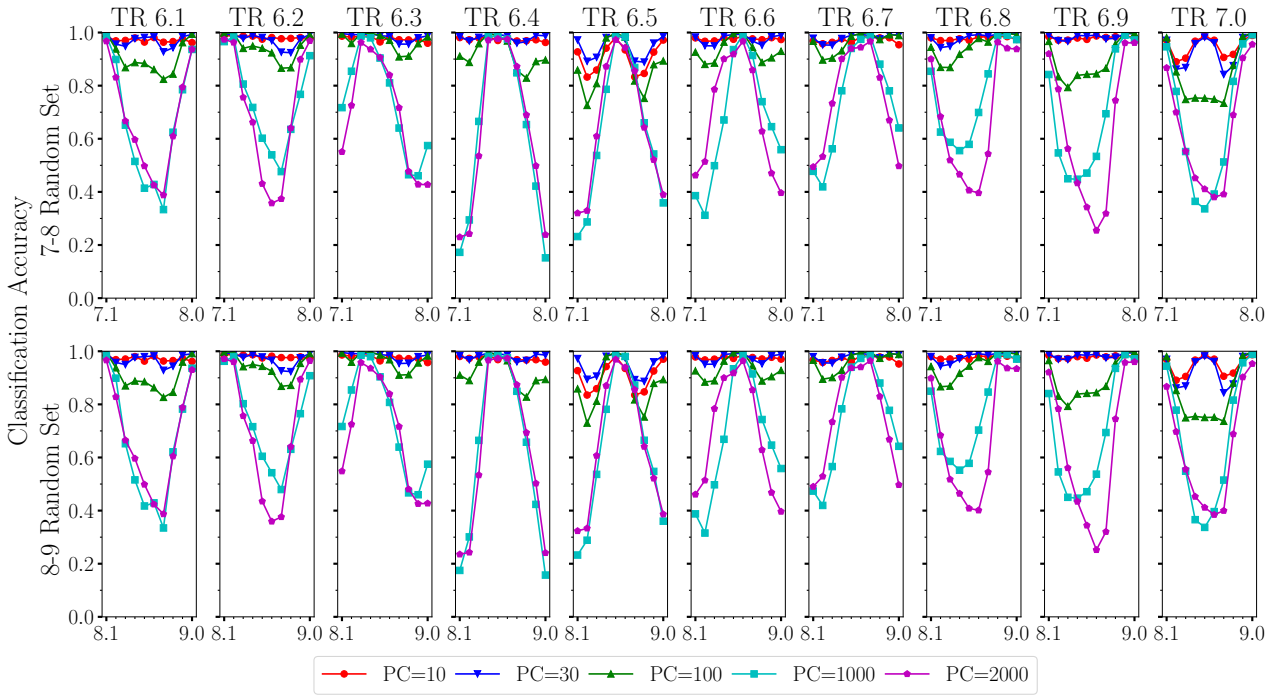



Fig. 8: Classification accuracies of 3-layer neural networks with $N_{units} = 300$ on snapshots from the validation set (top row: $t \in [7.1, 8]$, bottom row: $t \in [8.1, 9]$.). Columns correspond to networks trained with single-snapshots obtained at the indicated training time (TR). Summary: highest $MCA_1$=97.0% obtained with PC=30; highest $WCA_1$=84.2% obtained with PC=30.

shot cannot capture the full range of motion. The recurring high accuracy is caused by the cyclic nature of the underlying physical motion and therefore we conclude that multi-snapshot training is required to achieve better accuracy.

### *4.2.2 Ten snapshots-per-simulation training*

We now repeat the above experiments for the case where ten snapshots are gathered from each training simulation (i.e entire $r^{(i)}(t)$). Figure 10 indicates a significant improvement, eventually obtaining greater than 99% $MCA_{10}$ and $WCA_{10}$ accuracy. Since we achieve the desired classification performance with a single layer MLPC network, we skip tuning the number of layers as shown in Fig. 10.

This improvement comes with significantly increased training data size. The ten snapshot training data occupies 1.6GB of memory compared to 160MB for the single-snapshot training. Nevertheless, we find the training times of both networks have the same order of magnitude.

***Number of Hidden Units*** In this subsection we study the effect of the number of hidden units on classification accuracy. The worst performance is observed under 10 hidden units (99.1% $MCA_{10}$, 98.5% $WCA_{10}$) and the best performance is observed under 30 hidden units (99.7% $MCA_{10}$, 99.5% $WCA_{10}$).

Pointwise classification errors — per training validation simulation and time snapshot — are shown in Fig 11a for 10 units and 30 units.

From Fig. 11a we calculate that the $WCA_{10}$ and the $MCA_{10}$ becomes slightly better as the number of hidden units increases from 10 (99.1% $MCA_{10}$ and 98.5% $WCA_{10}$ at 1000 PCs) to 30 (99.7% $MCA_{10}$ and 99.5% $WCA_{10}$ at 300 PCs). As the number of units increase to 30, both of the criteria are met.

Unlike in the single snapshot training cases, the networks using many principal components are performing close to the desired level of accuracy — $\geq 99\%$ for all simulations in the validation set $\mathcal{V}$ among the networks with 30 hidden units ($PC \geq 100$). Due to memory efficient computing concerns, the focus is to achieve the highest possible accuracy with the simplest network possible and therefore we select the network with 30 hidden units as the result of this section.

***Number of Principal Components*** Next we refine the number of principal components. The worst performance is observed with 50 principal components (97.2% $MCA_{10}$, 96.0% $WCA_{10}$) and the best performance is

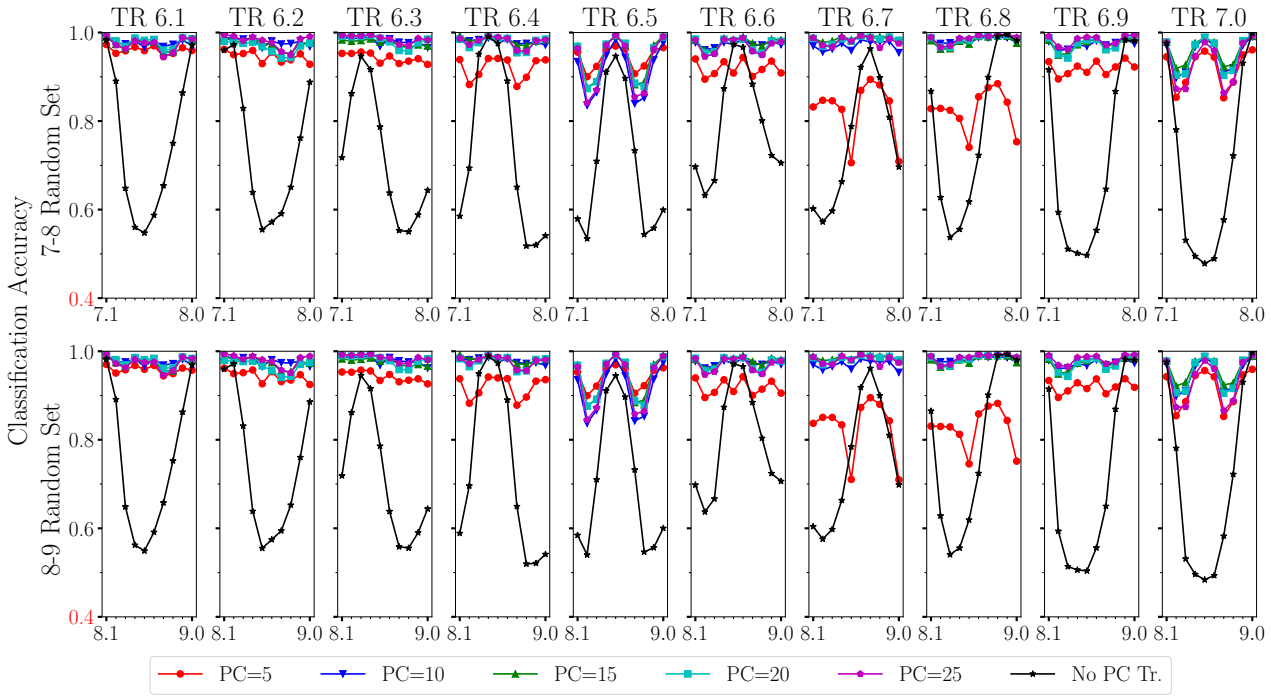



Fig. 9: Classification accuracies of 3-layer neural networks with low to no PCs and $N_{units} = 300$ on snapshots from the validation set (top row: $t \in [7.1, 8]$, bottom row: $t \in [8.1, 9]$.). Columns correspond to networks trained with single-snapshots obtained at the indicated training time (TR). Summary: highest $MCA_1$=97.3% obtained with PC=15; highest $WCA_1$=87.5% obtained with PC=15. The figures are zoomed in to provide more detailed overview and the shifted lower bounds of the figure axes are denoted in red

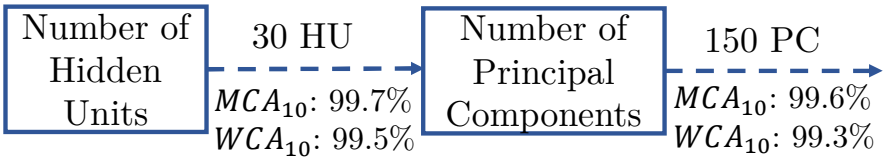



Fig. 10: Flow diagram for classification architecture determination using ten snapshots-per-simulation training. Each box represent a sequential step in the parametric study of this section. The architectural choice is written above each arrow and the performance is written below.

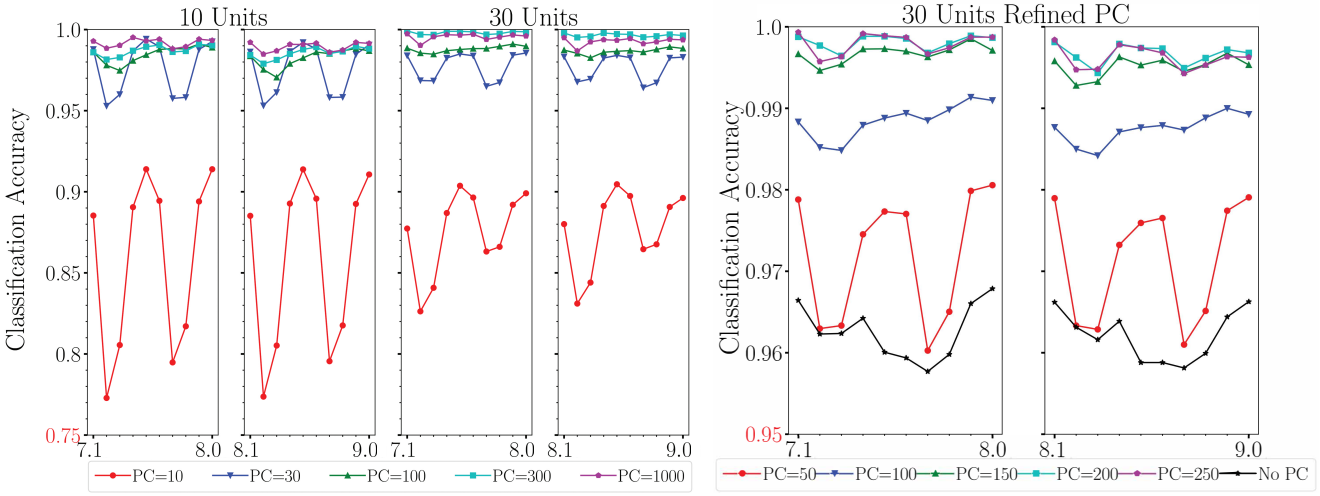
observed under 200 principal components (99.7% $MCA_{10}$, 99.4% $WCA_{10}$). Furthermore, a comparison between the best performing PCA hybridized MLP classifier and non-PCA version of the same network shows that the PCA-based classifier is better by 3.7% $MCA_{10}$ and 3.6% $WCA_{10}$

Pointwise classification errors — per training validation simulation and time snapshot — are shown in Fig 11b for all of the analyses mentioned in this section.

Fig. 11a demonstrates peak performance when the number of principal components is between 100 and 300 PCs. Therefore we refine our analysis in this region and focus on the region between $50 - 250$ PCs. The results of this analysis are shown in Fig. 11b.

As can be seen in Fig.11b all of the networks except the ones with 50 and 100 principal components achieve comparable and satisfactory $MCA_{10}$ that is above 99.5%. Since the training times and the $WCA_{10}$ accuracy performances of the networks are not different from each other, it can be concluded that the MLPC with 1 layer and 30 units having 150 principal components is the best performing combination for this architecture.

***Ten-snapshots summary*** Our final architecture is an MLP classifier structure with 1 layer and 30 units and 150 PCs. A comparison of the pointwise classification error of this architecture with a neural network lacking PCA processing is shown in Fig. 11b. We ob-




(a) Classification accuracies of 1-layer neural networks with $N_{units} = 10$ (left two columns - highest $MCA_{10}$=99.1% obtained with PC=1000; highest $WCA_{10}$=98.5% obtained with PC=1000) and $N_{units} = 30$ (right two columns - highest $MCA_{10}$=99.7% obtained with PC=300; highest $WCA_{10}$=99.5% obtained with PC=300) on snapshots from the validation set.

(b) Classification accuracies of 1-layer neural networks with low to no PCs and $N_{units} = 30$. Summary: highest $MCA_{10}$=99.7% obtained with PC=200; highest $WCA_{10}$=99.4% obtained with PC=200.

Fig. 11: Ten snapshot classification results with coarse principal component search (Fig. 11a). Ten snapshot classification results with fine principal component search and no-PCA comparison (Fig. 11b). Networks are trained using entire $r^{(i)}(t)$ with $t \in [6.1, 7.0]$. The figures are zoomed in to provide more detailed overview and the shifted lower bounds of the figure axes are denoted in red. Time of the validation snapshots are denoted along the x-axis.

serve that without PCA preprocessing, networks perform comparably but still worse. The $MCA_{10}$ drops down to 96.2% and the $WCA_{10}$ drops down to 95.8%. These results give the impression that the desired specifications may be achieved using a more complex network architecture. However, the training time grows substantially with the absence of PCA preprocessing. The MLP classifier takes around $1.3s$ on average to train with PCA preprocessing whereas without the PCA preprocessing, the training takes around $13s$.

***Classification summary*** Though $MCA_1$ of the networks trained with single-snapshot are close to the desired level, networks trained with ten-snapshots outperform their single-snapshot counterparts by a large margin. Furthermore, that increase in both $MCA$ and $WCA$ are achieved with a much more compact MLP classifier. Therefore, our inverse design architecture is equipped with a MLP classifier with PCA preprocessing trained using ten-snapshots per simulation.

### 4.3 Regression

In this section we perform a parametric study over the network architecture of the regressor stage described in Section 3.2. We focus only on training with ten-snapshots-per-simulation.

This study is a single forward process by which we first optimize the number of hidden units, then the number of hidden layers, then narrow in on the number of principle components, and finally the number of epochs as shown in Fig. 12. As with the classification procedure, the initial phases also consider parametric dependence on the number of principal components.

The baseline architecture has 1 hidden layer and 30 hidden units.

***Metrics*** Two types of performance metrics are considered: (1) mean $\alpha-$ quantiles ($MQ_\alpha$) of percent error, and (2) mean maximum ($MMax$) percent error. The $MQ_\alpha$ percent error is defined as

$$MQ_\alpha = \frac{1}{20}\sum_{k=1}^{20} \texttt{quant}(\alpha, PE_k),$$

$$PE_k = 100\left(\frac{p^{(i)} - \hat{p}^{(i)}(t_k)}{p^{(i)}}\right). \quad (11)$$

In particular, this is the average (over 20 snapshots in the testing regime $t_k \in [7.1, 9.0]$) median percentage errors achieved by the stacked regressor networks trained using a ten-snapshot sequence $r^{(i)}(t)$. The function $\texttt{quant}(\alpha, PE_k)$ returns the $\alpha$-th quantile values

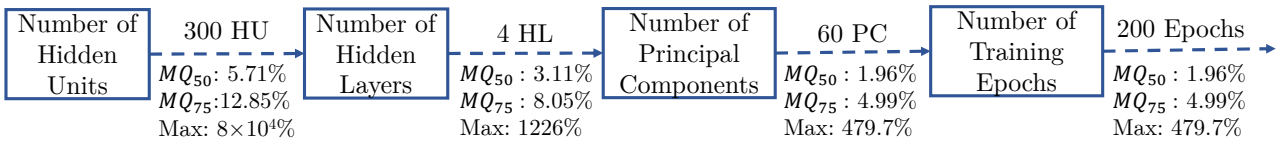



Fig. 12: Flow diagram for regression architecture determination using ten snapshots-per-simulation training. Each box represents a sequential stage in the parametric study of this section. The best architectural choice following a particular stage is written above each arrow and the performance is written below.

for a set of data points $PE_k$. Here $PE_k$ is a vector of percentage errors in recovering some parameter $p^{(i)}$ in the testing set of parameters using snapshot $r(t_k)$ for $t_k \in [7.1, 9.0]$ (the testing time-regime). Tables A1-A5 show the 25th, 50th, and 75th quantiles.

Similarly, the average maximum error is defined as

$$MMax = \frac{1}{20}\sum_{k=1}^{20} \max(PE_k) \tag{12}$$

However, since the extreme outliers can shift the mean substantially, we prefer median as an unbiased metric for the regression studies.

The aim of the case studies is to obtain a neural network that has small median PE. We also want the final network to be robust — having $MQ_{75} \leq 5\%$. Unless there is a substantial difference between two $MMax$ values, it will not play a decisive role in the analysis.

***Procedure*** The analysis proceeds by first choosing the number of hidden units, then the number of hidden layers, then refining the number of principal components and finally the number of epochs on the training set. In the first two steps we also search over a coarse grid of principal components, ranging between 20 to 640, so that the hyperparameter search is not entirely a greedy search.

For the simplicity of the analysis, both MLPs within the stacked MLP regressors (SMLPR), defined in Section 3.2, have the same network shape i.e. if a regressor network is described as having 5 layers and 100 units, then the neural network consists of two MLP regressors having 5 hidden layers with 100 neurons each. Results for each network combination are provided in Tables A1-A5, Figs. 13-15 and Figs. S4-S8 (Figs. S4-S8 are provided in the supplementary material for the interested reader).

Tables A1-A5 provide detailed results for the error metrics of each network configuration in the analyses. The columns are divided into three groups, one for each sheet parameters. The rows are divided into subgroups according to the number of different parameter options investigated under each study and for Tables A1-A4, regression metrics of the networks with different numbers of principal components are listed. Minimum values of each column are written in bold and the selected value for that study is highlighted.

In Figs. 13-15 and Figs. S4-S8, each column represents the regression metrics for a different continuous variable (amplitude, stiffness, wavenumber, respectively) and each row gives the performance of the trained network in terms of one of the three regression metrics (25th quantile, median, and 75th quantile, respectively).

***Number of Hidden Units*** In this subsection, we study the effect of the number of hidden units on regression performance. Following this analysis step, the highest $MQ_{50}$ (8.97%) is achieved with 30 hidden units (for wavenumber, 20 PCs) and the lowest $MQ_{50}$ under this analysis (1.23%) is obtained with 1000 hidden units (for stiffness, 40 PCs) (see Table A1).

Pointwise regression results — per training validation simulation and time snapshot — are shown in Figs S4 for 30 units, 13) for three hundred units, and S5) for one thousand units. Then Table A1, the derived regression metrics from Figs. S4-S5 are presented.

Our baseline architecture (1 hidden layer 30 hidden units - Fig. S4) achieves highest and lowest $MQ_{50}$ recorded 8.97% (for wavenumber, 20 PCs) and 1.96% (for stiffness, 80 PCs), respectively. Increasing the number of hidden units to 300 decreases the $MQ_{50}$ for all three parameters. The highest and the lowest $MQ_{50}$ recorded among networks with 300 hidden units are 5.71% (for wavenumber, 20 PCs) and 1.38% (for stiffness, 80 PCs) respectively (Fig. 13).

Further increasing the number of hidden units to 1000 does not result in significant decrease of $MQ_{50}$. The maximum $MQ_{50}$ observed among the networks with 1000 hidden units are 5.97% (640 PCs, for wavenumber) and 1.23% (40 PCs, for stiffness) respectively (Fig. S5).

In contrast to $MQ_{50}$, increasing the number of units does not result in a consistent reduction in MMax as shown in Table A1. This is caused by the random occur-

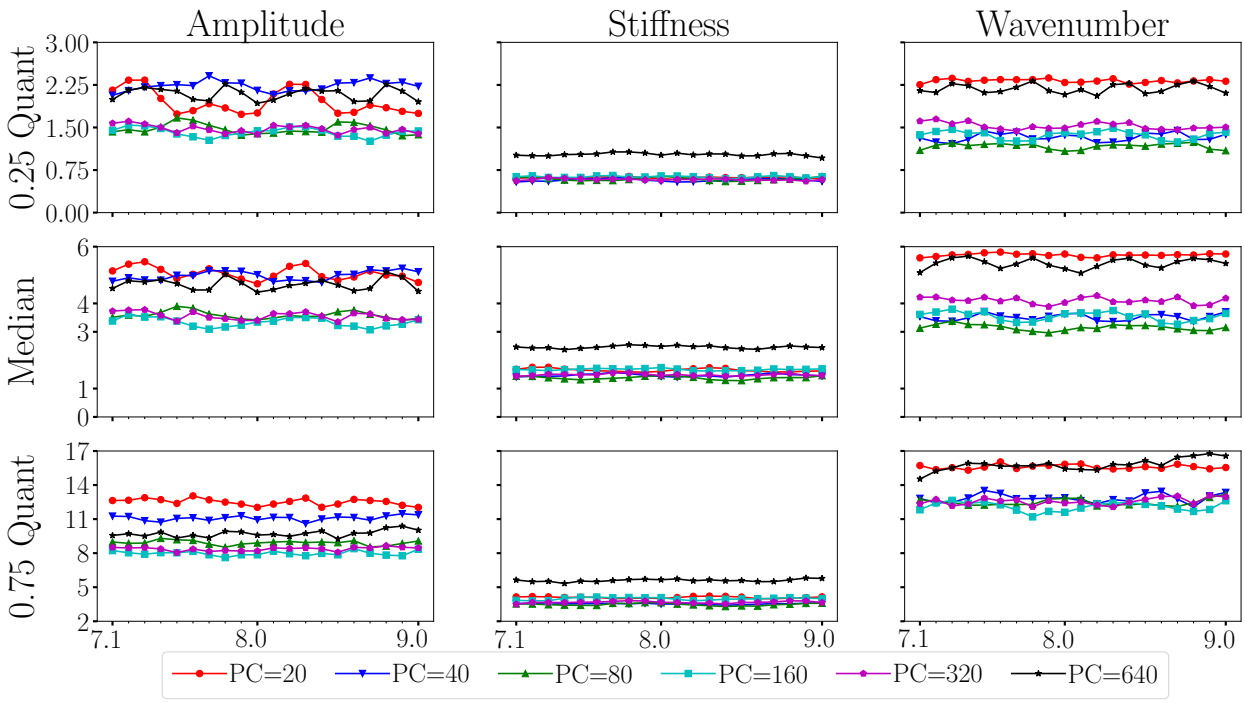



Fig. 13: Regression metrics of 1-layer SMLPR network with $N_{units} = 300$ (trained using entire $r^{(i)}(t) \in [6.1, 7.0]$) on 20-snapshots from the validation set ($t \in [7.1, 9]$). Columns correspond to different continuous sheet parameters, rows correspond to different metrics. Summary: lowest $MQ_{50}$=1.38% obtained with PC=80 for stiffness; lowest $MQ_{75}$=3.48% obtained with PC=80 for stiffness.

ring extreme outliers shifting the mean. It is also noteworthy that this phenomenon becomes apparent for the networks trained with higher numbers of principal components.

Though most of the minimum values of each metric appear among networks with $N_{units} = 1000$, most of the observed metrics under the networks with $N_{units} = 300$ have less difference than 0.5% from the minimum metrics observed in Table A1 Since both networks can achieve comparable MMax values we proceed with $N_{units} = 300$ because of smaller network size (as highlighted in Table A1).

***Number of Hidden Layers*** In this subsection we seek to improve accuracy by increasing the depth of the networks with layers of 300 hidden units. The highest $MQ_{50}$ (5.71%) is obtained with 1 hidden layer and 20 PC) and the lowest $MQ_{50}$ within this analysis (1.0%) is obtained with 3 hidden layers and 80 PCs, according to Table A2.

Pointwise regression results — per training validation simulation and time snapshot — are shown in Figs S6 for 2 hidden layers, S7) for three hidden layers, 14) for four hidden layers, and S8 for five hidden layers. Then Table A2, the derived regression metrics from Figs. S6-S8 are presented.

We observe that adding the second hidden layer results in a median PE drop up to 50% for all of the parameters as it can be seen in Table A2. Additionally, the MaxPE also drops significantly for amplitude and wavenumber as the number of hidden layers increase to 2. The highest and lowest $MQ_{50}$ recorded with this network combination are 2.93% (for amplitude, 640 PCs) and 1.07% (for stiffness, 80 PCs) respectively.

Introduction of the third layer doesn't have a substantial effect on the regression performance other than reducing MMax for all of the parameters.

As the number of layers increases from 2 to 3, the metrics for $N_{layers} = 3$ (in Fig.S7) have slightly better values than $N_{units} = 2$ (in Fig.S6). The highest $MQ_{50}$ recorded among networks with 3 hidden layers is 2.43% (for amplitude, 80 PCs) whereas the lowest $MQ_{50}$ is 1.0% (for stiffness, 80 PCs).

As the number of hidden layers increases to 4, the performance metrics get close to the desired values. Fig. 14 demonstrates that the metrics for 80 principal components exhibit results that comply with the goal metrics almost for all test snapshots. The highest and lowest $MQ_{50}$ recorded for this network combination are 3.11% (for amplitude, 20 PCs) and 1.11% (for stiffness, 160 PCs) respectively.

At this point of the analysis, each additional layer to the SMLPR architecture with 300 hidden units adds between $30 - 50s$ of total training time depending on the number of principal components used and number of epochs selected to train the network.

Adding the fifth layer brings almost no improvement to the regression metrics. In Fig. S8, the regression metrics of the network with 5 hidden layers with 300 units can be seen. The maximum and minimum $MQ_{50}$ recorded among networks with this combination are 3.1% (for amplitude, 640 PCs) and 1.23% (for stiffness, 80 PCs) respectively. This range is similar to the results of the SMLPR with 4 hidden layers. Therefore, SMLPR networks with 4 hidden are selected as the result of this study.

***Number of Principal Components*** Next we tune the number of principal components. The highest $MQ_{50}$ is achieved with 70 principal components (for amplitude,
2.11%) and the lowest $MQ_{50}$ is achieved with 70 principal components (for stiffness, 1.06%). Table A2 indicates that the best PCA values are between 40 and 80, so we zoom into this region in this study.

Pointwise regression results — per training validation simulation and time snapshot — are shown in Fig 15 for all the cases. Then Table A3, the derived regression metrics from Fig. 15 are presented.

Table A3 summarizes the results of our analysis. All of the trained networks except the one with 70 principal components satisfy our criterion for $MQ_{50} \leq 2\%$. However, the $0.75^{th}$ quantile $\leq 5\%$ criterion is just satisfied at the networks with 60 and 80 principal components. Since there is not a significant difference between the MMax values or the mean of $0.25^{th}$ quantiles of PE of two networks, the network with 60 principal components is selected due to its smaller network size.

***Number of Epochs*** In this study, the effect of training epochs on the regression performance is investigated. Furthermore, a comparison between the PCA hybridized and non-PCA version of the same network architecture is done to investigate the benefits of PCA hybridization. The worst performance within this section is observed under 100 epochs (3.44% $MQ_{50}$ for amplitude) and the best performance is observed under 300 epochs (0.98% $MQ_{50}$ for stiffness).

Pointwise regression results — per training validation simulation and time snapshot — are only shown for 200 epochs in Fig 15. Results of 100 and 300 epochs are only included in Table A4 along with the derived regression metrics from Fig. 15 for epoch case studies. The regression metrics for the Non-PCA comparison are listed in Table A5.

The final network parameter of interest is the number of epochs completed to train the SMLPR networks. Please note that the numbers given here represent the number of epochs completed by each individual stage of the SMLPR networks. Though we select a specific value for the dimension of the coordinate data after principle component transformation in the previous section, we investigate the $[30, 80]$ principle component range in this case study again to check if changing the number of epochs have any effect on the best performing PC choice in Table A4

To investigate the possibility of overfitting, we modify the number of epochs used for training. In particular, we test using hundred more and hundred fewer epochs. Table A4 shows that changing the number of epochs increases the $MQ_{75}$ for amplitude by at least 0.5 and pushes the error in amplitude prediction beyond our admissible limits. For stiffness and wavenumber, the performance becomes slightly better by increasing the number of epochs. However, the best SMLPR network combination that satisfies our criteria remains as the last section (4 layers 300 units with 60 PCs and 200 epochs).

***Regression Summary*** Finally, we again compare performance of our network to that which lacks PCA preprocessing. Table A4 and Fig.15 show that removing PCA hybridization from the coordinate components increases the error in stiffness estimation to limit of our predefined admissible values. In our analyses, the SMLPR network with 60 principal components completes the training in $514s$ whereas the Non-PCA version of the same network completes the training in $638s$. As PCA preprocessing proves itself beneficial, we equip our inverse design architecture with a SMLP regressor with 4 layers 300 units 60 PCs that is trained using the ten-snapshot-per-simulation data over 200 epochs and conclude our investigation.

### 4.4 Simulated Performance of Recovered Parameters

In this section, we provide examples of the motion recovered by the resulting inverse mapping architecture in comparison to known actual parameters.

Figs. 16-19 show the performance for four cases with parameters chosen to display notable vertical displacements (two with planar and two with radial motion). In the top subfigures we provide the motion with the true

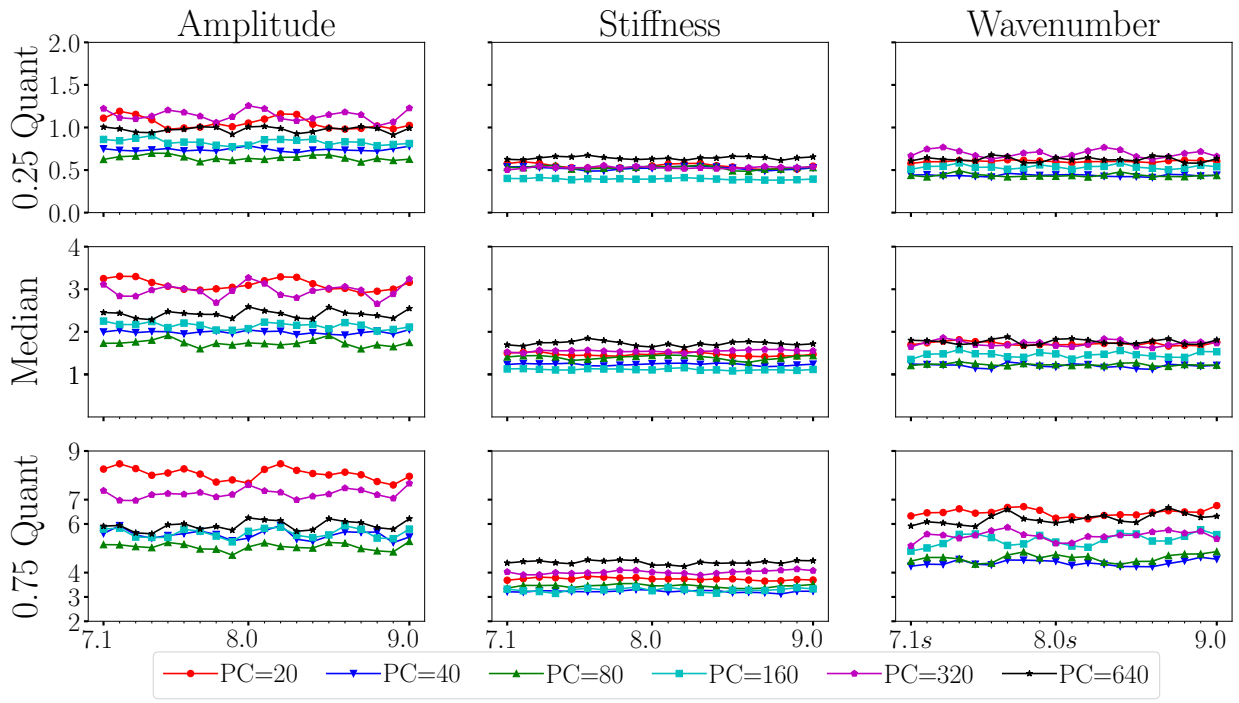



Fig. 14: Regression metrics of 4-layer SMLPR network with $N_{units} = 300$ (trained using entire $r^{(i)}(t) \in [6.1, 7.0]$) on 20-snapshots from the validation set ($t \in [7.1, 9]$). Columns correspond to different continuous sheet parameters, rows correspond to different metrics. Summary: lowest $MQ_{50}$=1.11% obtained with PC=160 for stiffness; lowest $MQ_{75}$=3.22% obtained with PC=40 for stiffness.

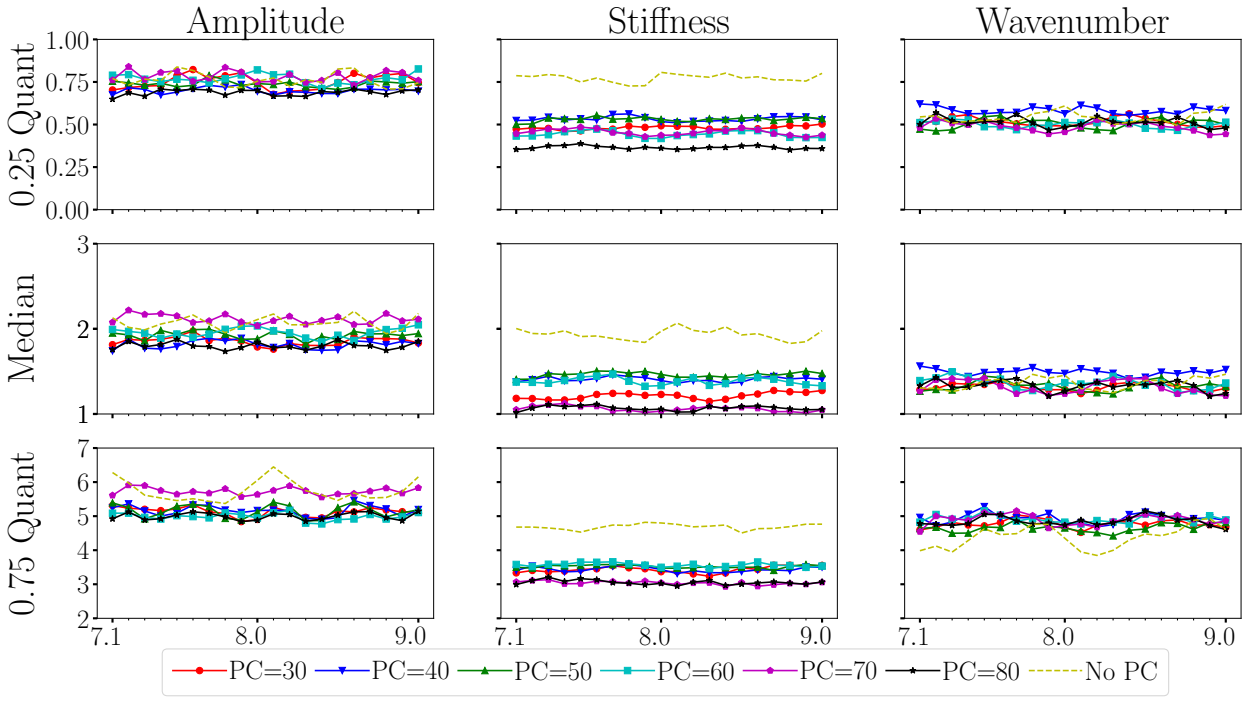



Fig. 15: Regression metrics of 4-layer SMLPR network with $N_{units} = 300$ (trained using entire $r^{(i)}(t) \in [6.1, 7.0]$) on 20-snapshots from the validation set ($t \in [7.1, 9]$) for fine principal component search and no-PCA comparison. Columns correspond to different continuous sheet parameters, rows correspond to different metrics. Summary: lowest $MQ_{50}$=1.06% obtained with PC=70 for stiffness; lowest $MQ_{75}$=3.04% obtained with PC=70 for stiffness.

parameters, and the bottom subfigures provide the motion with the inverted parameters. The selected samples and the respective outputs of the inverse design architecture are listed in Table A6. Inverse design radial motion continuous parameters were recovered with $\leq 2.87\%$ error, whereas the continuous parameters in the planar motion case were recovered with $\leq 2.47\%$ error. For all cases, the correct wave type was predicted. Videos for the motions can be found in supplementary material[5].

Figs. 16-17 provide both the shape of the gel at a time instance and a time series of the vertical displacement and forces for the planar waves. We note the general agreement of the time series data.

The simulations with radial waves (Figs. 18 and 19) indicate further discrepancy between shapes at a fixed timestep. However, we see that these motions are fairly chaotic, and such qualitative mismatch at a fixed time does not necessarily indicate a significant mismatch in motion type. Indeed the time series of the vertical displacement and forces still indicate qualitative similarity. To obtain a more quantitative comparison, we compare Lyapunov exponents using a long-time simulation over 300 periods using the approach provided in [61]. Here, we find at most 6.78% error in their dominant Lyapunov exponents.

Note, we do not perform a similar comparison for the planar cases because (1) they already indicate strong agreement and (2) since they have one high frequency and one low frequency oscillations, they are unsuitable for Lyapunov exponent estimation using our chosen tool[6] prepared according to [61].

## 5 Conclusion

We have successfully created an inverse design mapping to learn sheet parameters of a self-oscillating gel to obtain a target motion. Our final architecture consists of several integrated neural networks — a single classifier that determines a discrete parameter and two regressors for determining continuous parameters. The optimal classifier uses 150 principal components for PCA preprocessing of the input into an MLP with 1 hidden layer and 30 units with 150 principal components. The regressor is a stacked two-network architecture, each having 4 hidden layers and 300 units with 60 principal components. For both classification and regression, the networks are trained using snapshots from 10 successive steps of a forward simulation across a wide range of parameter settings.

We also demonstrated the efficacy of the developed architectures by comparing the simulations of the sheets with the recovered parameters and the true parameters both qualitatively through inspection of the force and vertical displacement trajectories and quantitatively through comparing the Lyapunov exponents of the resulting chaotic systems.

Future work will seek to verify our approach using experimental data on a gel.

**Acknowledgements** We would like to thank Donghak Kim for creating the simulation data that we used to train the inverse design architecture.

*Funding:* This work was supported by the MICDE Catalyst Grant program at the University of Michigan, the DARPA AIRA program under Agreement No. HR0011199002, "Artificial Intelligence guided multi-scale multi-physics framework for discovering complex emergent materials phenomena", and the DOE Office of Science, ASCR.

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

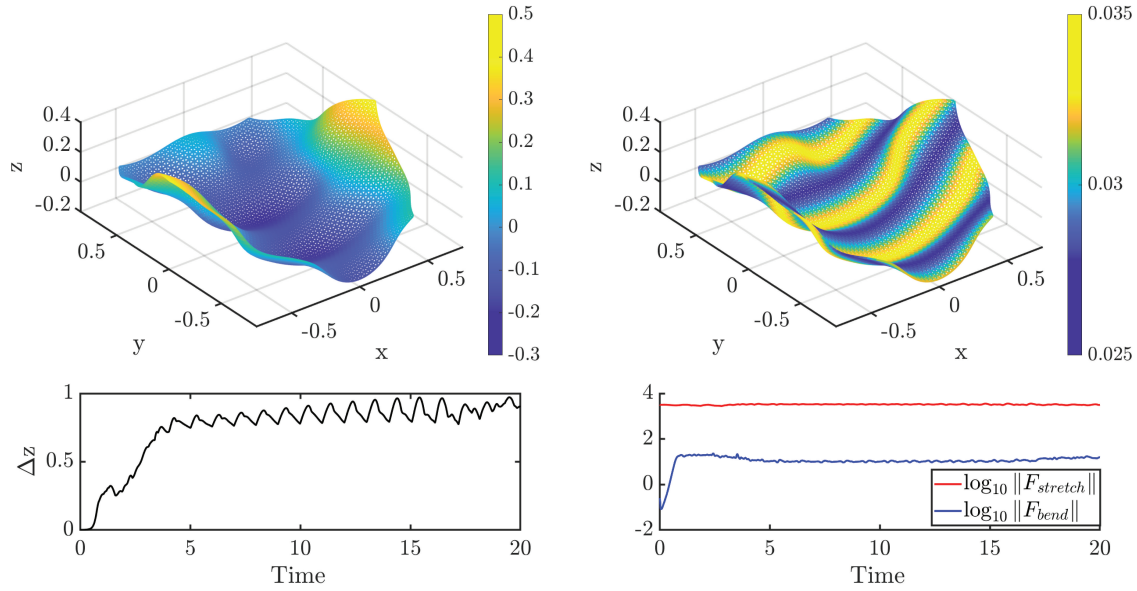
*Code Availability:* All the trained networks in this paper along with the python and MATLAB scripts responsible for creating them can be found in the Bitbucket repository bitbucket.org/dorukaks/workspace/projects/CGL.

## References

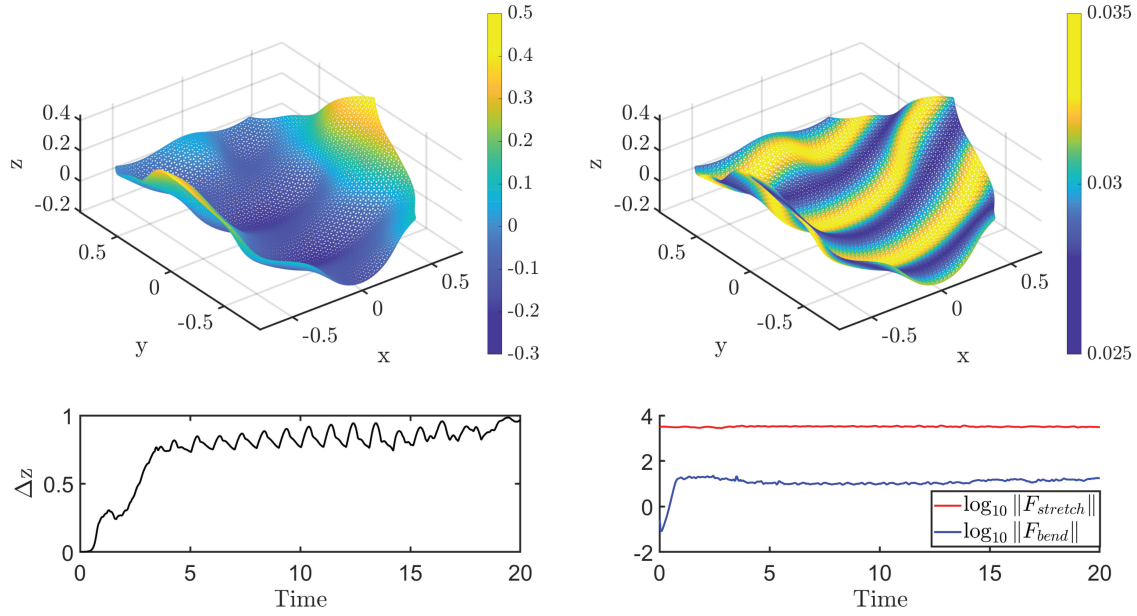
1. Y. Takeuchi, N. Asakawa, and D. Ge, "Automation of Polishing Work by an Industrial Robot : System of Polishing Robot," *JSME international journal. Ser. C, Dynamics, control, robotics, design and manufacturing*, vol. 36, no. 4, pp. 556–561, 1993.
2. C. Lenz, S. Nair, M. Rickert, A. Knoll, W. Rosel, J. Gast, A. Bannat, and F. Wallhoff, "Joint-action for humans and industrial robots for assembly tasks," in *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 130–135, IEEE, aug 2008.
3. J. Zhou, S. Chen, and Z. Wang, "A Soft-Robotic Gripper With Enhanced Object Adaptation and Grasping Reliability," *IEEE Robotics and Automation Letters*, vol. 2, pp. 2287–2293, oct 2017.
4. S. Haddadin and E. Croft, *Physical Human–Robot Interaction*, pp. 1835–1874. Cham: Springer International Publishing, 2016.
5. M. Cianchetti, T. Ranzani, G. Gerboni, T. Nanayakkara, K. Althoefer, P. Dasgupta, and A. Menciassi, "Soft Robotics Technologies to Address Shortcomings in Today's Minimally Invasive Surgery: The STIFF-FLOP Approach," *Soft Robotics*, vol. 1, no. 2, pp. 122–131, 2014.

[5] The files are named as `set[setnumber]_[actual/predicted] _[300/20]s.gif`

[6] Alan Wolf (2021). Wolf Lyapunov exponent estimation from a time series. (https://www.mathworks.com/matlabcentral/fileexchange/48084-wolf-lyapunov-exponent-estimation-from-a-time-series), MATLAB Central File Exchange. Retrieved December 22, 2020.

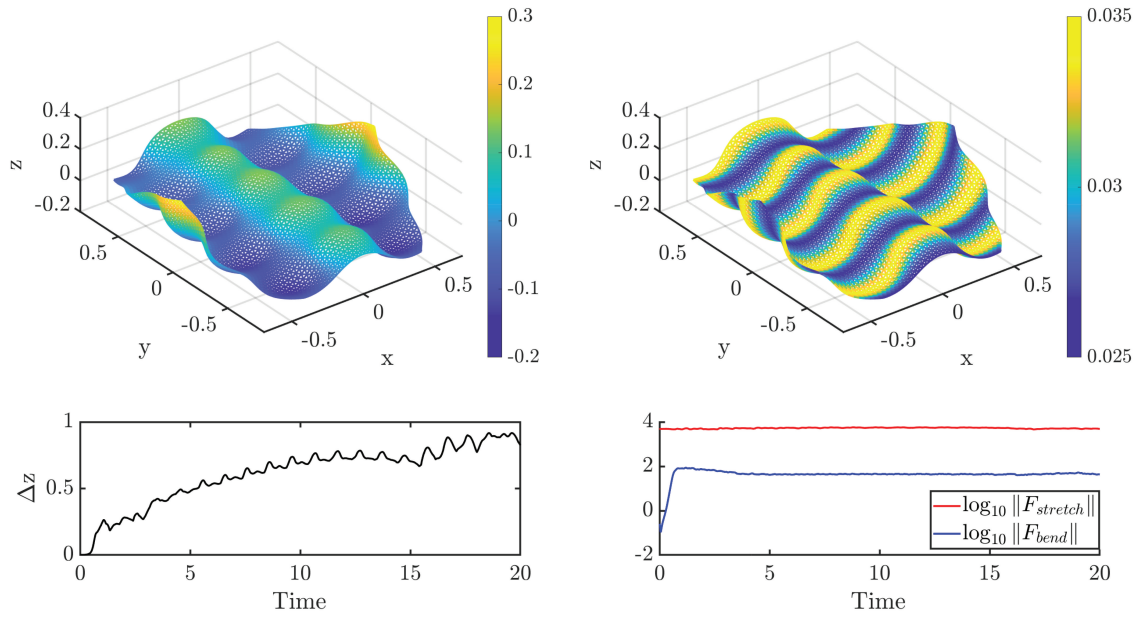



(a) MATLAB simulations of the actual parameters ($A = 0.0824$ $K_s = 5875.3$ $k = 1.9100$) for parameter set 1.

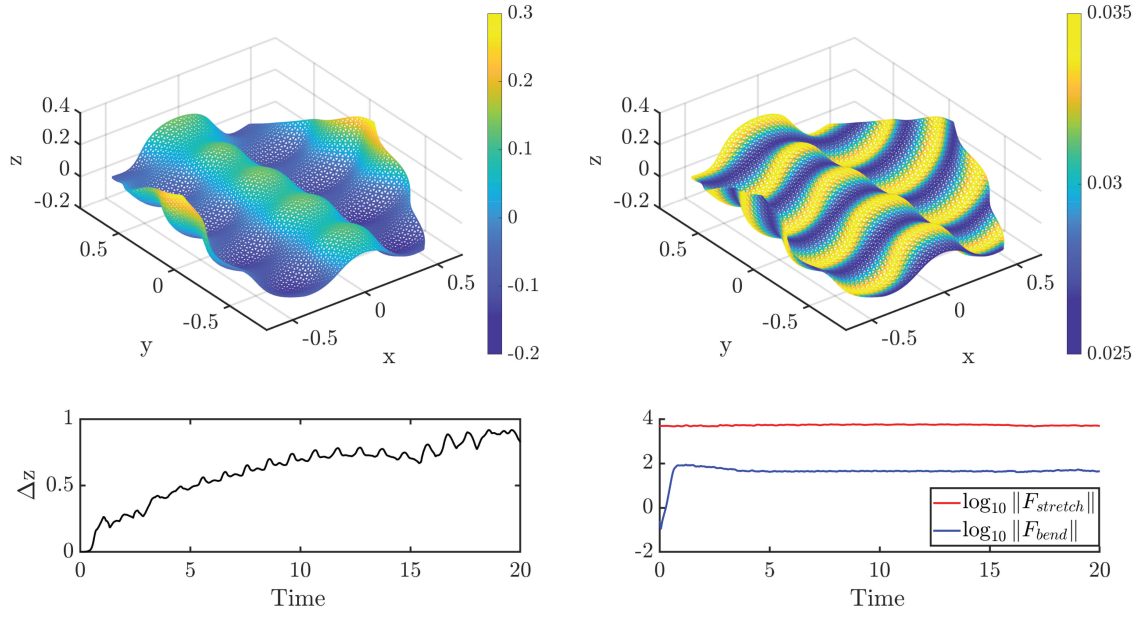



(b) MATLAB simulations of the inverted parameters ($A = 0.0810$ $K_s = 5896.4$ $k = 1.8782$) for parameter set 1.

Fig. 16: Comparison of true and estimated motions. Each subfigure shows the vertical displacement (top left); the lattice springs' rest lengths at $t = 3$ (top right); time series plot of vertical displacement (bottom left); and time series of the force terms (bottom right) for $t = [0, 20]$.

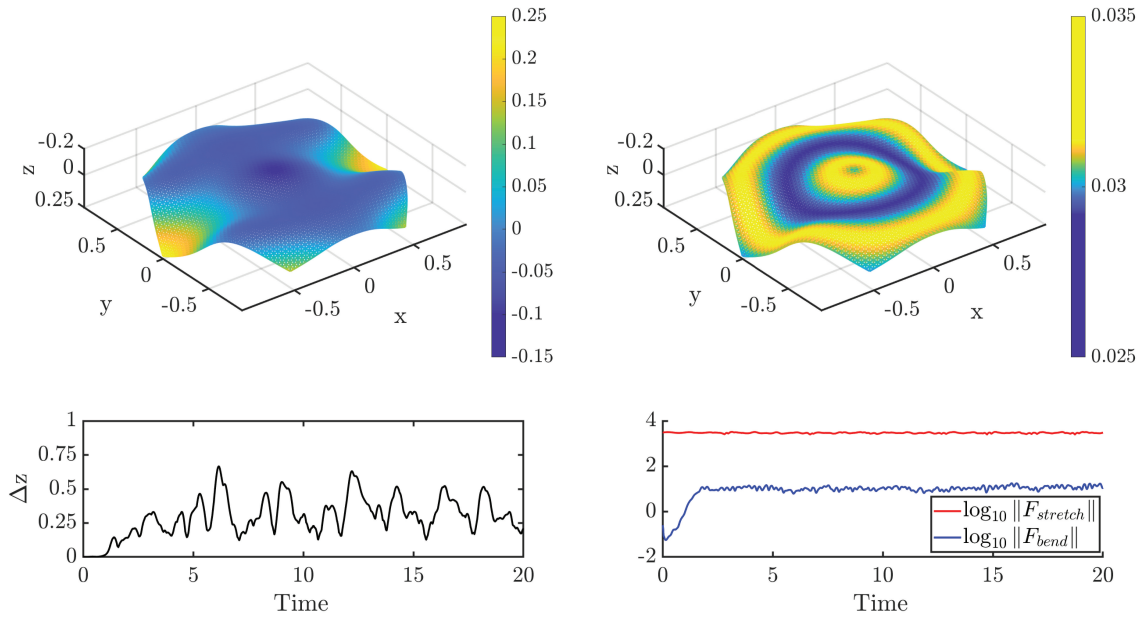



(a) MATLAB simulations of the actual parameters ($A = 0.1182$ $K_s = 8829.5$ $k = 2.7213$) for parameter set 2.

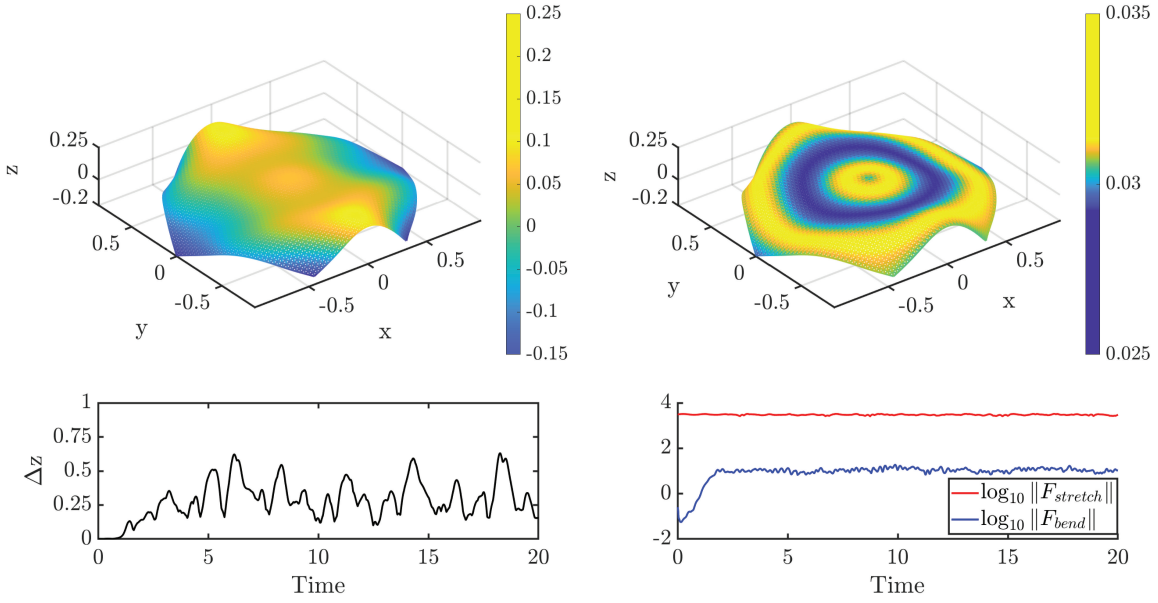



(b) MATLAB simulations of the inverted parameters ($A = 0.1211$ $K_s = 8806.1$ $k = 2.7090$) for parameter set 2.

Fig. 17: Comparison of true and estimated motions. Each subfigure shows the vertical displacement (top left); the lattice springs' rest lengths at $t = 4$ (top right); time series plot of vertical displacement (bottom left); and time series of the force terms (bottom right) for $t = [0, 20]$.

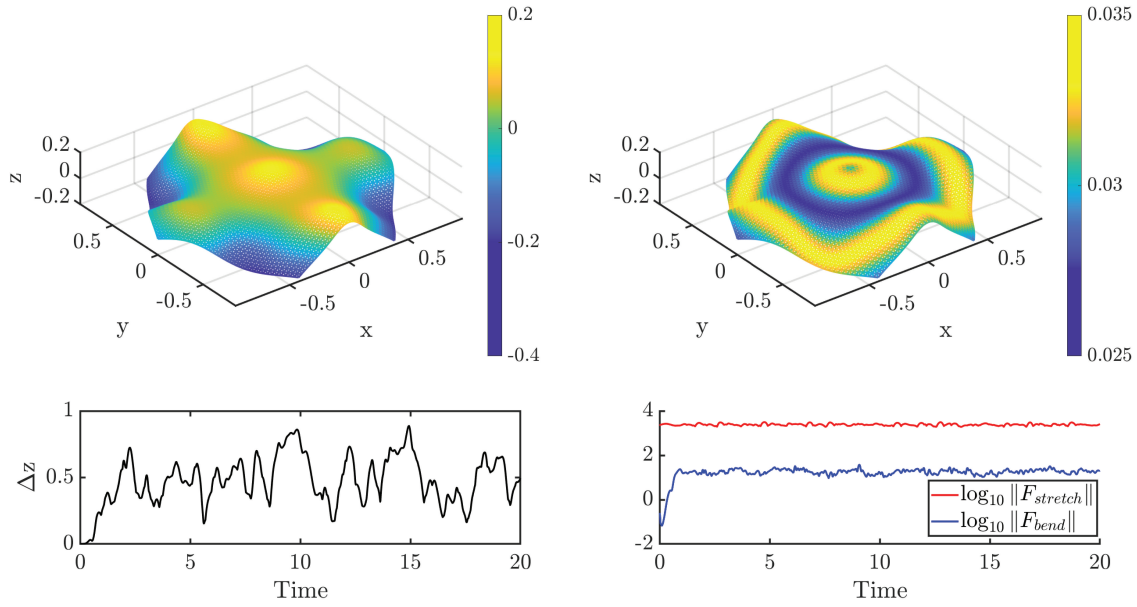



(a) MATLAB simulations of the actual parameters ($A = 0.0366$ $K_s = 5830.4$ $k = 1.6545$) for parameter set 3.

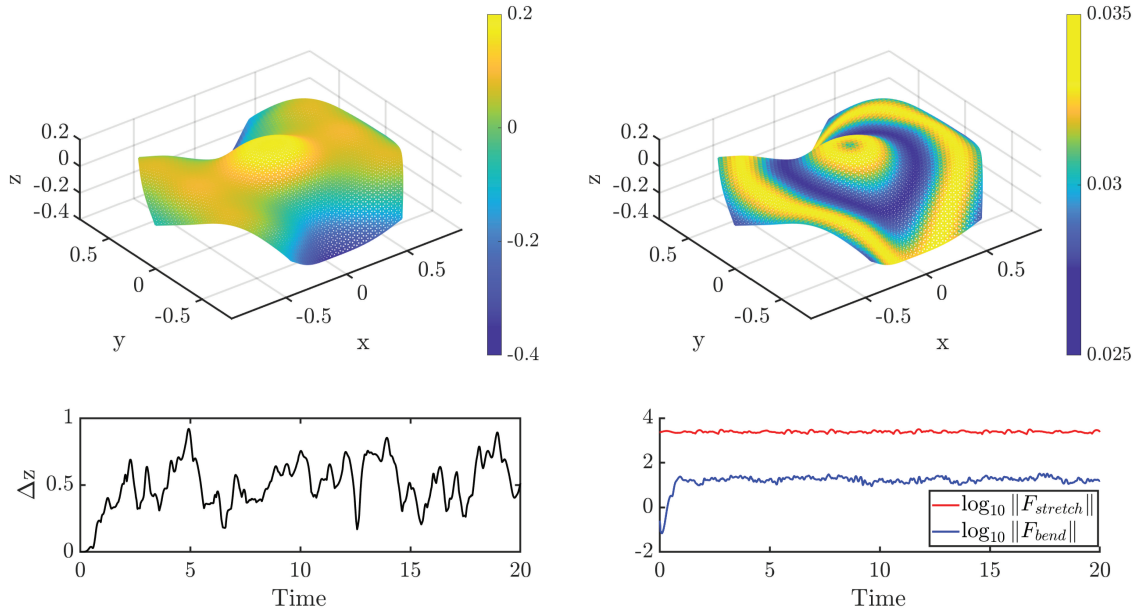



(b) MATLAB simulations of the inverted parameters ($A = 0.0356$ $K_s = 5913.4$ $k = 1.6688$) for parameter set 3. Z axes of the top plots are flipped to match the shapes of the sheets to the actual simulations.

Fig. 18: Comparison of true and estimated motions. Each subfigure shows the vertical displacement (top left); the lattice springs' rest lengths at $t = 15$ (top right); time series plot of vertical displacement (bottom left); and time series of the force terms (bottom right) for $t = [0, 20]$.




(a) MATLAB simulations of the actual parameters ($A = 0.0903$ $K_s = 4552./ = 8$ $k = 1.7106$) for parameter set 4.




(b) MATLAB simulations of the predicted parameters ($A = 0.0911$ $K_s = 4561.8$ $k = 1.7446$) for parameter set 4.

Fig. 19: Comparison of true and estimated motions. Each subfigure shows the vertical displacement (top left); the lattice springs' rest lengths at $t = 11$ (top right); time series plot of vertical displacement (bottom left); and time series of the force terms (bottom right) for $t = [0, 20]$.

6. R. K. Katzschmann, J. DelPreto, R. MacCurdy, and D. Rus, "Exploration of underwater life with an acoustically controlled soft robotic fish," *Science Robotics*, vol. 3, no. 16, p. eaar3449, 2018.
7. M. A. Delph, S. A. Fischer, P. W. Gauthier, C. H. Luna, E. A. Clancy, and G. S. Fischer, "A soft robotic exomusculature glove with integrated sEMG sensing for hand rehabilitation," *IEEE International Conference on Rehabilitation Robotics*, pp. 1–7, 2013.
8. D. Copaci, E. Cano, L. Moreno, and D. Blanco, "New Design of a Soft Robotics Wearable Elbow Exoskeleton Based on Shape Memory Alloy Wire Actuators," *Applied Bionics and Biomechanics*, vol. 2017, pp. 1–11, 2017.
9. Y. Ansari, M. Manti, E. Falotico, Y. Mollard, M. Cianchetti, and C. Laschi, "Towards the development of a soft manipulator as an assistive robot for personal care of elderly people," *International Journal of Advanced Robotic Systems*, vol. 14, no. 2, pp. 1–17, 2017.
10. B. Mosadegh, P. Polygerinos, C. Keplinger, S. Wennstedt, R. F. Shepherd, U. Gupta, J. Shim, K. Bertoldi, C. J. Walsh, and G. M. Whitesides, "Pneumatic networks for soft robotics that actuate rapidly," *Advanced Functional Materials*, vol. 24, no. 15, pp. 2163–2170, 2014.
11. M. Calisti, M. Giorelli, G. Levy, B. Mazzolai, B. Hochner, C. Laschi, and P. Dario, "An octopus-bioinspired solution to movement and manipulation for soft robots," *Bioinspiration and Biomimetics*, vol. 6, no. 3, 2011.
12. T. T. Hoang, P. T. Phan, M. T. Thai, N. H. Lovell, and T. N. Do, "Bio-Inspired Conformable and Helical Soft Fabric Gripper with Variable Stiffness and Touch Sensing," *Advanced Materials Technologies*, vol. 2000724, pp. 1–14, 2020.
13. S. Pfeil, M. Henke, K. Katzer, M. Zimmermann, and G. Gerlach, "A Worm-Like Biomimetic Crawling Robot Based on Cylindrical Dielectric Elastomer Actuators," *Frontiers in Robotics and AI*, vol. 7, no. February, pp. 1–11, 2020.
14. S. Tottori, L. Zhang, F. Qiu, K. K. Krawczyk, A. Franco-Obregõn, and B. J. Nelson, "Magnetic helical micromachines: Fabrication, controlled swimming, and cargo transport," *Advanced Materials*, vol. 24, no. 6, pp. 811–816, 2012.
15. T. Qiu, S. Palagi, A. G. Mark, K. Melde, F. Adams, and P. Fischer, "Wireless actuation with functional acoustic surfaces," *Applied Physics Letters*, vol. 109, no. 19, pp. 1–5, 2016.
16. H. Zeng, P. Wasylczyk, D. S. Wiersma, and A. Priimagi, "Light Robots: Bridging the Gap between Microrobotics and Photomechanics in Soft Materials," *Advanced Materials*, vol. 30, no. 24, pp. 1–9, 2018.
17. R. Yoshida, T. Takahashi, T. Yamaguchi, and H. Ichijo, "Self-oscillating gel," *Journal of the American Chemical Society*, vol. 118, no. 21, pp. 5134–5135, 1996.
18. S. Maeda, Y. Hara, T. Sakai, R. Yoshida, and S. Hashimoto, "Self-walking gel," *Advanced Materials*, vol. 19, no. 21, p. 3480, 2007.
19. O. Tabata, H. Hirasawa, S. Aoki, R. Yoshida, and E. Kokufuta, "Ciliary motion actuator using self-oscillating gel," *Sensors and Actuators A-Physical*, vol. 95, no. 2-3, pp. 234–238, 2002.
20. O. Tabata, H. Kojima, T. Kasatani, Y. Isono, and R. Yoshida, "Chemo-mechanical actuator using self-oscillating gel for artificial cilia," in *MEMS-03: IEEE The Sixteenth Annual International Conference on Micro Electro Mechanical Systems*, Proceedings: IEEE Micro Electro Mechanical Systems, pp. 12–15, 2003.
21. S. Maeda, Y. Hara, R. Yoshida, and S. Hashimoto, "Peristaltic motion of polymer gels," *Angewandte Chemie-International Edition*, vol. 47, no. 35, pp. 6690–6693, 2008.
22. Y. Shiraki and R. Yoshida, "Autonomous Intestine-Like Motion of Tubular Self-Oscillating Gel," *Angewandte Chemie-International Edition*, vol. 51, no. 25, pp. 6112–6116, 2012.
23. V. V. Yashin and A. C. Balazs, "Modeling polymer gels exhibiting self-oscillations due to the Belousov - Zhabotinsky reaction," *Macromolecules*, vol. 39, no. 6, pp. 2024–2026, 2006.
24. P. Dayal, O. Kuksenok, and A. C. Balazs, "Directing the behavior of active, self-oscillating gels with light," *Macromolecules*, vol. 47, no. 10, pp. 3231–3242, 2014.
25. I. Levin, R. Deegan, and E. Sharon, "Self-oscillating membranes: Chemomechanical sheets show autonomous periodic shape transformation," *Phys. Rev. Lett.*, vol. 125, p. 178001, Oct 2020.
26. S. Alben, A. A. Gorodetsky, D. Kim, and R. D. Deegan, "Semi-implicit methods for the dynamics of elastic sheets," *Journal of Computational Physics*, vol. 399, p. 108952, 2019.
27. K. Siakavara, "Artificial neural network based design of a three-layered microstrip circular ring antenna with specified multi-frequency operation," *Neural Computing and Applications*, vol. 18, pp. 57–64, jan 2009.
28. C. Chen and G. X. Gu, "Generative Deep Neural Networks for Inverse Materials Design Using Backpropagation and Active Learning," *Advanced Science*, vol. 7, p. 1902607, mar 2020.
29. V. Sekar, M. Zhang, C. Shu, and B. C. Khoo, "Inverse Design of Airfoil Using a Deep Convolutional Neural Network," *AIAA Journal*, vol. 57, pp. 993–1003, mar 2019.
30. M. M. Li, B. Verma, X. Fan, and K. Tickle, "RBF neural networks for solving the inverse problem of backscattering spectra," *Neural Computing and Applications*, vol. 17, pp. 391–397, aug 2008.
31. L. Massari, E. Schena, C. Massaroni, P. Saccomandi, A. Menciassi, E. Sinibaldi, and C. M. Oddo, "A Machine-Learning-Based Approach to Solve Both Contact Location and Force in Soft Material Tactile Sensors," *Soft Robotics*, vol. 7, no. 4, pp. 409–420, 2020.
32. X. Liu and A. Fotouhi, "Formula-E race strategy development using artificial neural networks and Monte Carlo tree search," *Neural Computing and Applications*, vol. 32, no. 18, pp. 15191–15207, 2020.
33. J. Peurifoy, Y. Shen, L. Jing, Y. Yang, F. Cano-Renteria, B. G. DeLacy, J. D. Joannopoulos, M. Tegmark, and M. Soljačić, "Nanophotonic particle simulation and inverse design using artificial neural networks," *Science Advances*, vol. 4, no. 6, pp. 1–8, 2018.
34. T. Xie and J. C. Grossman, "Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties," *Physical Review Letters*, vol. 120, no. 14, p. 145301, 2018.
35. G. Sun, Y. Sun, and S. Wang, "Artificial neural network based inverse design: Airfoils and wings," *Aerospace Science and Technology*, vol. 42, pp. 415–428, 2015.
36. P. Z. Hanakata, E. D. Cubuk, D. K. Campbell, and H. S. Park, "Forward and inverse design of kirigami via supervised autoencoder," *Physical Review Research*, vol. 2, no. 4, pp. 1–6, 2020.
37. N. Gao, M. Wang, B. Cheng, and H. Hou, "Inverse design and experimental verification of an acoustic sink based on machine learning," *Applied Acoustics*, vol. 180, p. 108153, 2021.

38. H. S. Seung and D. R. Nelson, "Defects in flexible membranes with crystalline order," *Physical Review A*, vol. 38, pp. 1005–1018, jul 1988.
39. B. Schmidt and F. Fraternali, "Universal formulae for the limiting elastic energy of membrane networks," *Journal of the Mechanics and Physics of Solids*, vol. 60, pp. 172–180, jan 2012.
40. A. AKILLI and H. ATIL, "Evaluation of Normalization Techniques on Neural Networks for the Prediction of 305-Day Milk Yield," *Turkish Journal of Agricultural Engineering Research*, vol. 1, pp. 354–367, 2020.
41. M. S. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, 1996.
42. S. U. Yongkui, C. A. Yuan, X. I. Guo, and W. E. Tao, "Condition monitoring for railway point machines based on sound analysis and support vector machine," *Chinese Journal of Electronics*, vol. 29, no. 4, pp. 786–792, 2020.
43. F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, no. 5-6, pp. 183–197, 1991.
44. A. Malhi and R. X. Gao, "Pca-based feature selection scheme for machine defect classification," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 6, pp. 1517–1525, 2004.
45. W. Stacklies, H. Redestig, M. Scholz, D. Walther, and J. Selbig, "pcaMethods - A bioconductor package providing PCA methods for incomplete data," *Bioinformatics*, vol. 23, no. 9, pp. 1164–1167, 2007.
46. R. E. Madsen, L. K. Hansen, and O. Winther, "Singular Value Decomposition and Principle Component Analysis," *Neural Networks*, vol. 1, no. February, pp. 1–5, 2004.
47. S. Sousa, F. G. Martins, M. Alvim-Ferraz, and M. C. Pereira, "Multiple linear regression and artificial neural networks based on principal components to predict ozone concentrations," *Environmental Modelling & Software*, vol. 22, no. 1, pp. 97–103, 2007.
48. S. Ghosh-Dastidar, H. Adeli, and N. Dadmehr, "Principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection," *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 2, pp. 512–518, 2008.
49. J. Hesthaven and S. Ubbiali, "Non-intrusive reduced order modeling of nonlinear problems using neural networks," *Proceedings of the 20th USENIX Security Symposium*, no. 363, pp. 55–78, 2018.
50. V. Ravi and C. Pramodh, "Threshold accepting trained principal component neural network and feature subset selection: Application to bankruptcy prediction in banks," *Applied Soft Computing Journal*, vol. 8, no. 4, pp. 1539–1548, 2008.
51. H. Abdi, D. Valentin, B. Edelman, and A. J. O'Toole, "More about the difference between men and women: evidence from linear neural networks and the principal-component approach.," *Perception*, vol. 24, no. 5, pp. 539–562, 1995.
52. D. A. Ross, J. Lim, R. S. Lin, and M. H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 125–141, 2008.
53. V. Lippi and G. Ceccarelli, "Incremental principal component analysis: Exact implementation and continuity corrections," *ICINCO 2019 - Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics*, vol. 1, pp. 473–480, 2019.
54. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
55. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8026–8037, Curran Associates, Inc., 2019.
56. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
57. Z. Zhou, J. B. Wang, Y. F. Zang, and G. Pan, "PAIR comparison between two within-group conditions of resting-state fMRI improves classification accuracy," *Frontiers in Neuroscience*, vol. 11, no. JAN, pp. 1–13, 2018.
58. D. Q. Zeebaree, H. Haron, and A. M. Abdulazeez, "Gene Selection and Classification of Microarray Data Using Convolutional Neural Network," *ICOASE 2018 - International Conference on Advanced Science and Engineering*, pp. 145–150, 2018.
59. D. S. Guru, P. B. Mallikarjuna, S. Manjunath, and M. M. Shenoi, "Machine Vision Based Classification Of Tobacco Leaves For Automatic Harvesting," *Intelligent Automation and Soft Computing*, vol. 18, no. 5, pp. 581–590, 2012.
60. I. Veisi, N. Pariz, and A. Karimpour, "Fast and robust detection of epilepsy in noisy BEG signals using permutation entropy," *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering, BIBE*, pp. 200–203, 2007.
61. A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining Lyapunov exponents from a time series," *Physica D: Nonlinear Phenomena*, vol. 16, pp. 285–317, jul 1985.

## Appendix

## A Tables

Table A1: Regression metrics of the hidden unit case studies for regression ($MQ_{25}$: mean 25-th quantile, $MQ_{50}$: mean 50-th quantile, $MQ_{75}$: mean 75-th quantile, MMax: mean maximum as defined in Section 3.2)

| **Number of Units** | **PCs** | **Amplitude** | | | | **Stiffness** | | | | **Wavenumber** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ |
| **30** | 20 | 2.91 | 7.33 | 17.68 | 323.9 | 1.14 | 2.63 | 5.36 | 58.1 | 3.8 | 8.97 | 21.06 | 3200.9 |
| | 40 | 2.06 | 4.88 | 10.68 | 316.7 | 1.01 | 2.33 | 4.78 | 299.5 | 2.7 | 6.58 | 16.39 | 1813.2 |
| | 80 | 1.72 | 4.08 | 9.24 | 1233.6 | 0.8 | 1.96 | 4.29 | 894.9 | 2.04 | 5.07 | 15.25 | 17721.1 |
| | 160 | 2.05 | 4.76 | 9.24 | 1565.8 | 0.88 | 2.03 | 4.37 | 928.0 | 2.42 | 5.85 | 17.21 | 17403.0 |
| | 320 | 2.12 | 5.04 | 11.11 | 3897.4 | 0.93 | 2.26 | 4.7 | 3329.1 | 2.43 | 6.34 | 17.47 | 140166.9 |
| | 640 | 2.18 | 5.21 | 11.62 | 3706.2 | 0.9 | 2.28 | 5.2 | 4121.3 | 2.41 | 6.04 | 17.66 | 133543.6 |
| **100** | 20 | 2.39 | 5.98 | 13.81 | 247.5 | 0.94 | 2.17 | 4.47 | 47.7 | 2.9 | 7.05 | 16.87 | 2825.1 |
| | 40 | 1.74 | 4.09 | 9.28 | 521.2 | 0.66 | 1.63 | 3.8 | 403.2 | 1.77 | 4.54 | 13.11 | 1033.3 |
| | 80 | 1.48 | 3.55 | 8.51 | 1953.0 | 0.56 | 1.44 | 3.65 | 1049.1 | 1.73 | 4.42 | 13.53 | 6123.3 |
| | 160 | 1.8 | 4.25 | 9.86 | 2427.1 | 0.53 | 1.43 | 3.73 | 1340.2 | 1.74 | 4.44 | 14.41 | 15501.2 |
| | 320 | 1.73 | 4.27 | 10.29 | 3215.0 | 0.92 | 2.34 | 5.18 | 2556.7 | 1.95 | 4.98 | 15.87 | 32166.0 |
| | 640 | 1.81 | 4.33 | 9.89 | 4891.3 | 0.74 | 1.73 | 4.01 | 5986.5 | 2.16 | 5.45 | 15.9 | 41438.8 |
| **300** | 20 | 1.95 | 5.06 | 12.51 | 240.1 | 0.61 | 1.65 | 4.1 | **44.46** | 2.32 | 5.71 | 15.6 | 2248.7 |
| | 40 | 2.23 | 4.98 | 11.09 | 558.0 | 0.57 | 1.47 | 3.54 | 809.23 | 1.33 | 3.52 | 12.85 | 3387.5 |
| | 80 | 1.47 | 3.59 | 8.92 | 2424.2 | 0.59 | 1.38 | 3.48 | 1510.0 | 1.17 | 3.15 | 12.47 | 5193.8 |
| | 160 | 1.41 | 3.34 | **7.98** | 1888.3 | 0.63 | 1.67 | 3.98 | 1862.2 | 1.37 | 3.55 | 12.1 | 17117.9 |
| | 320 | 1.47 | 3.50 | 8.37 | 4960.4 | 0.58 | 1.48 | 3.68 | 3665.0 | 1.53 | 4.1 | 12.56 | 70353.4 |
| | 640 | 2.09 | 4.67 | 9.73 | 8063.9 | 1.02 | 2.46 | 5.6 | 5053.7 | 2.18 | 5.41 | 15.8 | 77499.7 |
| **1000** | 20 | 1.39 | 3.82 | 10.18 | **212.4** | 0.56 | 1.44 | 3.65 | 49.9 | 1.71 | 4.5 | 13.65 | **1872.0** |
| | 40 | **1.34** | **3.32** | 8.38 | 387.4 | **0.45** | **1.23** | **3.23** | 566.3 | 1.24 | 3.14 | **8.99** | 4427.0 |
| | 80 | 1.47 | 3.66 | 8.83 | 879.2 | 0.76 | 1.72 | 3.97 | 1915.1 | **1.05** | **2.73** | 9.7 | 8684.3 |
| | 160 | 1.49 | 3.59 | 8.57 | 2371.4 | 0.9 | 1.77 | 3.91 | 1373.9 | 1.54 | 3.86 | 12.47 | 19363.8 |
| | 320 | 2.13 | 5.2 | 12.49 | 6334.6 | 1.19 | 2.77 | 6.04 | 6825.6 | 1.67 | 4.37 | 13.63 | 59121.9 |
| | 640 | 2.27 | 5.23 | 11.42 | 10535.3 | 1.19 | 3.2 | 7.36 | 16774.8 | 2.54 | 5.97 | 17.3 | 133128.3 |

Table A2: Percent error (PE) metrics of the layer case studies for regression ($MQ_{25}$: mean 25-th quantile, $MQ_{50}$: mean 50-th quantile, $MQ_{75}$: mean 75-th quantile, MMax: mean maximum as defined in Section 3.2)

| **Layers** | **PCs** | **Amplitude** | | | | **Stiffness** | | | | **Wavenumber** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ |
| **1** | 20 | 1.95 | 5.06 | 12.51 | 240.1 | 0.61 | 1.65 | 4.1 | 44.5 | 2.32 | 5.71 | 15.6 | 2248.7 |
| | 40 | 2.23 | 4.98 | 11.09 | 558.0 | 0.57 | 1.41 | 3.54 | 809.2 | 1.33 | 3.52 | 12.85 | 3387.5 |
| | 80 | 1.47 | 3.59 | 8.92 | 2424.2 | 0.59 | 1.38 | 3.48 | 1510.1 | 1.17 | 3.15 | 12.47 | 5193.8 |
| | 160 | 1.41 | 3.34 | 7.98 | 1888.3 | 0.63 | 1.67 | 3.98 | 1862.2 | 1.37 | 3.55 | 12.1 | 17117.9 |
| | 320 | 1.47 | 3.56 | 8.37 | 4960.4 | 0.58 | 1.48 | 3.68 | 3665.0 | 1.53 | 4.1 | 12.56 | 70353.4 |
| | 640 | 2.09 | 4.67 | 9.73 | 8063.9 | 1.02 | 2.46 | 5.6 | 5053.7 | 2.18 | 5.41 | 15.8 | 77499.7 |
| **2** | 20 | 0.9 | 2.53 | 7.59 | 187.4 | 0.35 | 1.04 | 3.19 | **42.7** | 0.77 | 2.23 | 8.63 | 1036.9 |
| | 40 | 0.81 | 2.23 | 6.28 | 451.3 | 0.69 | 1.61 | 3.71 | 290.0 | 0.62 | 1.67 | 6.46 | 522.0 |
| | 80 | 0.75 | 2.07 | 5.82 | 801.5 | 0.35 | 1.07 | 3.08 | 528.0 | 0.63 | 1.74 | 6.5 | 1368.3 |
| | 160 | 1.2 | 2.73 | 6.58 | 1106.3 | 0.54 | 1.35 | 3.39 | 370.4 | 0.85 | 2.02 | 6.6 | 1452.4 |
| | 320 | 0.89 | 2.23 | 5.98 | 1193.1 | 0.66 | 1.87 | 4.54 | 593.7 | 0.75 | 2.03 | 6.88 | 1537.1 |
| | 640 | 1.21 | 2.93 | 7.1 | 1256.4 | 0.55 | 1.49 | 3.91 | 1008.6 | 1.02 | 2.72 | 8.94 | 3822.3 |
| **3** | 20 | 0.86 | 2.27 | 6.76 | 142.3 | 0.34 | 1.06 | 3.14 | 45.7 | 0.61 | 1.84 | 7.06 | 1041.0 |
| | 40 | 0.74 | 2.05 | 5.48 | 216.9 | 0.4 | 1.14 | 3.08 | 173.6 | 0.5 | 1.35 | 5.19 | 383.9 |
| | 80 | 1.03 | 2.43 | 6.26 | 633.0 | **0.33** | **1.0** | **2.95** | 193.0 | 0.73 | 1.79 | 5.7 | 659.2 |
| | 160 | 0.89 | 2.19 | 5.61 | 905.1 | 0.47 | 1.53 | 3.88 | 368.5 | 0.58 | 1.56 | 6.02 | 1601.2 |
| | 320 | 0.9 | 2.2 | 5.73 | 827.7 | 0.39 | 1.11 | 3.34 | 471.3 | 0.61 | 1.68 | 6.1 | 1562.9 |
| | 640 | 0.99 | 2.42 | 5.89 | 952.4 | 0.48 | 1.43 | 3.91 | 959.8 | 0.74 | 1.96 | 6.68 | 2614.7 |
| **4** | 20 | 1.05 | 3.11 | 8.05 | 175.9 | 0.54 | 1.47 | 3.74 | 47.1 | 0.6 | 1.72 | 6.47 | 1226.7 |
| | 40 | 0.74 | 1.99 | 5.56 | 167.0 | 0.52 | 1.23 | 3.22 | 285.9 | **0.44** | **1.2** | **4.4** | 395.4 |
| | 80 | **0.64** | **1.74** | 5.07 | 329.3 | 0.53 | 1.4 | 3.45 | 236.4 | **0.44** | 1.23 | 4.6 | 531.1 |
| | 160 | 0.83 | 2.14 | 5.62 | 157.5 | 0.4 | 1.11 | 3.29 | 231.5 | 0.54 | 1.46 | 5.35 | 991.9 |
| | 320 | 1.14 | 2.97 | 7.25 | 224.0 | 0.53 | 1.55 | 4.02 | 476.0 | 0.69 | 1.72 | 5.52 | 1064.2 |
| | 640 | 0.98 | 2.42 | 5.94 | 216.3 | 0.64 | 1.72 | 4.42 | 474.0 | 0.62 | 1.77 | 6.21 | 1105.0 |
| **5** | 20 | 0.82 | 2.3 | 6.54 | 161.2 | 0.57 | 1.51 | 3.71 | 44.5 | 0.61 | 1.68 | 6.0 | 808.9 |
| | 40 | 0.77 | 1.84 | **5.0** | **133.7** | 0.48 | 1.28 | 3.23 | 125.8 | 0.54 | 1.32 | 4.51 | **362.5** |
| | 80 | 0.7 | 1.85 | 5.16 | 145.6 | 0.43 | 1.23 | 3.48 | 210.0 | 0.48 | 1.33 | 4.8 | 515.0 |
| | 160 | 0.95 | 2.46 | 6.23 | 152.2 | 0.91 | 1.88 | 4.02 | 166.0 | 0.57 | 1.43 | 4.82 | 599.3 |
| | 320 | 0.93 | 2.31 | 5.64 | 186.7 | 0.51 | 1.47 | 3.99 | 200.4 | 0.5 | 1.38 | 5.27 | 819.0 |
| | 640 | 1.27 | 3.1 | 7.36 | 191.3 | 0.89 | 2.16 | 5.17 | 241.8 | 0.63 | 1.69 | 6.19 | 859.8 |

Table A3: Percent error (PE) metrics of the principal component case studies for regression ($MQ_{25}$: mean 25-th quantile, $MQ_{50}$: mean 50-th quantile, $MQ_{75}$: mean 75-th quantile, MMax: mean maximum as defined in Section 3.2)

| **Layers** | **PCs** | **Amplitude** | | | | **Stiffness** | | | | **Wavenumber** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ |
| **4** | 30 | 0.75 | 1.86 | 5.13 | **144.4** | 0.48 | 1.21 | 3.43 | **96.7** | 0.51 | 1.32 | 4.77 | 503.9 |
| | 40 | 0.7 | 1.82 | 5.16 | 167.5 | 0.53 | 1.41 | 3.43 | 101.4 | 0.58 | 1.49 | 4.93 | **424.4** |
| | 50 | 0.74 | 1.93 | 5.15 | 270.8 | 0.53 | 1.46 | 3.52 | 308.9 | 0.51 | 1.33 | **4.65** | 450.1 |
| | 60 | 0.77 | 1.96 | **4.99** | 231.2 | 0.44 | 1.38 | 3.57 | 202.5 | 0.5 | 1.36 | 4.91 | 479.7 |
| | 70 | 0.78 | 2.11 | 5.73 | 375.5 | 0.45 | **1.06** | **3.04** | 158.1 | **0.49** | **1.32** | 4.89 | 493.7 |
| | 80 | **0.69** | **1.8** | 5.0 | 194.9 | **0.37** | 1.07 | 3.05 | 229.1 | 0.51 | 1.33 | 4.85 | 567.9 |

Table A4: Percent error (PE) metrics of the epoch case studies for regression ($MQ_{25}$: mean 25-th quantile, $MQ_{50}$: mean 50-th quantile, $MQ_{75}$: mean 75-th quantile, MMax: mean maximum as defined in Section 3.2). *:200 epochs is the standard number of epochs used throughout the previous sections

| **Epochs** | **PCs** | **Amplitude** | | | | **Stiffness** | | | | **Wavenumber** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ |
| **100** | 30 | 0.89 | 2.22 | 5.89 | **129.9** | 0.88 | 2.08 | 4.65 | **45.7** | 0.47 | 1.3 | 5.28 | **345.0** |
| | 40 | 1.57 | 3.2 | 6.93 | 215.5 | 0.59 | 1.44 | 3.6 | 125.1 | 0.84 | 1.99 | 6.19 | 425.2 |
| | 50 | 1.74 | 3.44 | 7.26 | 261.8 | 0.7 | 1.47 | 3.31 | 224.0 | 0.61 | 1.5 | 4.92 | 408.9 |
| | 60 | 0.83 | 2.22 | 6.0 | 323.1 | 0.48 | 1.22 | 3.26 | 180.4 | 0.54 | 1.37 | 4.96 | 391.1 |
| | 70 | 0.82 | 2.07 | 5.6 | 521.3 | 0.57 | 1.21 | 3.15 | 186.8 | 0.49 | 1.3 | 4.99 | 746.2 |
| | 80 | 0.98 | 2.41 | 6.16 | 392.7 | 0.51 | 1.41 | 3.58 | 145.9 | 0.58 | 1.49 | 5.12 | 549.1 |
| **200*** | 30 | 0.75 | 1.86 | 5.13 | 144.4 | 0.48 | 1.21 | 3.43 | 96.7 | 0.51 | 1.32 | 4.77 | 503.9 |
| | 40 | 0.7 | 1.82 | 5.16 | 167.5 | 0.53 | 1.41 | 3.43 | 101.4 | 0.58 | 1.49 | 4.93 | 424.4 |
| | 50 | 0.74 | 1.93 | 5.15 | 270.8 | 0.53 | 1.46 | 3.52 | 308.9 | 0.51 | 1.33 | **4.65** | 450.1 |
| | 60 | 0.77 | 1.96 | **4.99** | 231.2 | 0.44 | 1.38 | 3.57 | 202.5 | 0.5 | 1.36 | 4.91 | 479.7 |
| | 70 | 0.78 | 2.11 | 5.73 | 375.5 | 0.45 | 1.06 | 3.04 | 158.1 | 0.49 | 1.32 | 4.89 | 493.7 |
| | 80 | **0.69** | **1.8** | 5.0 | 194.9 | 0.37 | 1.07 | 3.05 | 229.1 | 0.51 | 1.33 | 4.85 | 567.9 |
| **300** | 30 | 0.71 | 1.97 | 5.65 | 430.7 | **0.36** | **0.98** | 2.99 | 157.6 | **0.41** | **1.2** | 4.87 | 559.3 |
| | 40 | 0.77 | 2.05 | 5.68 | 254.6 | 0.4 | 1.19 | 3.27 | 308.1 | 0.53 | 1.37 | 4.76 | 394.6 |
| | 50 | 0.92 | 2.31 | 5.92 | 382.9 | 0.45 | 1.18 | 3.03 | 203.8 | 0.53 | 1.34 | 4.78 | 517.5 |
| | 60 | 0.73 | 1.97 | 5.41 | 355.6 | **0.36** | 1.01 | **2.85** | 162.9 | 0.51 | 1.34 | 4.62 | 624.9 |
| | 70 | 0.97 | 2.37 | 5.83 | 408.4 | 0.48 | 1.51 | 3.79 | 213.8 | 0.5 | 1.31 | 4.81 | 500.3 |
| | 80 | 0.87 | 2.05 | 5.31 | 605.0 | 0.44 | 1.15 | 3.14 | 239.2 | 0.48 | 1.34 | 4.87 | 557.2 |

Table A5: Percent error (PE) metrics of the Non-PCA comparison case study for regression ($MQ_{25}$: mean 25-th quantile, $MQ_{50}$: mean 50-th quantile, $MQ_{75}$: mean 75-th quantile, MMax: mean maximum as defined in Section 3.2).

| **Epochs** | **PCs** | **Amplitude** | | | | **Stiffness** | | | | **Wavenumber** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ | $MQ_{25}$ | $MQ_{50}$ | $MQ_{75}$ | $MMax$ |
| **200** | 60 | **0.77** | **1.96** | **4.99** | **231.2** | **0.44** | **1.38** | **3.57** | **202.5** | **0.5** | **1.36** | 4.91 | **479.7** |
| | N/A | 0.76 | 2.07 | 5.75 | 241.1 | 0.77 | 1.93 | 4.69 | 214.7 | 0.54 | **1.36** | **4.39** | 643.8 |

Table A6: Errors in our inverse design estimation for four representative parameter settings P: planar motion, R: radial motion. In addition to three continuous sheet parameters, Lyapunov exponents of radial motions are provided for the radial cases. Video column denotes the name of the motion video of given parameter combination in supplementary material.

| | | **Video** | **Amplitude** | **Stiffness** | **Wavenumber** | **Lyap. Exp.** |
|---|---|---|---|---|---|---|
| **Set 1** (P) | **Actual** (Fig. 16a) | set1_actual_300s.gif | 0.0824 | 5875.3 | 1.9100 | N/A |
| | **Predicted** (Fig. 16b) | set1_predicted_300s.gif | 0.0810 | 5896.4 | 1.8782 | N/A |
| | **Error** (%) | — | 1.73 | 0.36 | 1.66 | N/A |
| **Set 2** (P) | **Actual** (Fig. 17a) | set2_actual_300s.gif | 0.1211 | 8806.1 | 2.7090 | N/A |
| | **Predicted** (Fig. 17b) | set2_predicted_300s.gif | 0.1182 | 8829.5 | 2.7213 | N/A |
| | **Error** (%) | — | 2.47 | 0.26 | 0.45 | N/A |
| **Set 3** (R) | **Actual** (Fig. 18a) | set3_actual_300s.gif | 0.0366 | 5830.4 | 1.6545 | 1.9956 |
| | **Predicted** (Fig. 18b) | set3_predicted_300s.gif | 0.0356 | 5913.5 | 1.6688 | 2.1309 |
| | **Error** (%) | — | 2.87 | 1.43 | 0.86 | 6.78 |
| **Set 4** (R) | **Actual** (Fig. 19a) | set4_actual_300s.gif | 0.0911 | 4561.8 | 1.7446 | 1.3115 |
| | **Predicted** (Fig. 19b) | set4_predicted_300s.gif | 0.0903 | 4552.8 | 1.7106 | 1.3919 |
| | **Error** (%) | — | 0.93 | 0.20 | 1.99 | 6.13 |