UNIVERSITY OF CALIFORNIA

Los Angeles

Mechanical Neural-Networks: Materials That Learn Their Properties and Behaviors

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Mechanical Engineering

by

Ryan Hansen Lee

2023

ABSTRACT OF THE DISSERTATION

Mechanical Neural-Networks: Materials That Learn Their Properties and Behaviors

by

Ryan Hansen Lee

Doctor of Philosophy in Mechanical Engineering

University of California, Los Angeles, 2023

Professor Jonathan B. Hopkins, Chair

This dissertation builds the foundational knowledge required for creating a general material capable of adapting and learning to meet changing requirements. The novel material demonstrates its ability to learn mechanical behaviors through changes in its structure in an experimental setting which represents a broad leap in the capabilities of architected materials. Architected materials are systems which derive their apparent bulk properties from their structure rather than their chemical composition. In this dissertation, we propose that a mechanical structure with many similarities to the compositional framework used in artificial intelligence (AI) that can enable a material to learn, adapt, and relearn behaviors. The presented architected material is inspired by the artificial neural network (ANN) and uses mechanically analogous elements called a Mechanical Neural Network (MNN). To demonstrate the MNN's potential, we

constructed a simulation tool as well as a physical apparatus to show an MNN's ability to learn and then uses these tools to explore several factors for increasing an MNNs utility. Specifically, by improving the accuracy of the learned behaviors, increasing an MNN's potential to learn more properties simultaneously, and decreasing the amount of time required to obtain new behaviors. The findings from these studies are then used to explore avenues for future microscale MNNs by creating designs that can be scaled arbitrarily. The work shown in this dissertation has the potential to have dramatic effects on the material selection process and the life cycle of future products by granting flexibility in design requirements and giving the freedom to reteach behaviors to compensate for aging and damage.

The dissertation of Ryan Hansen Lee is approved.

Lihua Jin

Jason L. Speyer

Tsu-Chin Tsao

Jonathan Hopkins, Committee Chair

University of California, Los Angeles

2023

To my family for their endless love and support.

Table of Contents

List of Figures

Supplementary Materials

Movie S1: Demonstrating the tunable beam's active control. An Instron testing machine cycling the tunable beam of Fig. 2.2A as it is being actively controlled with different proportional gain, K_p, values with $f(e[k])$ set equal to $e[k]$ (Fig. 2.13A) so that the beam exhibits different axial stiffness values with linear force-displacement responses.

Movie S2: Mechanical neural network (MNN) features. The 21-beam MNN used to conduct the experimental learning study of this work.

Movie S3: Mechanical neural network (MNN) learning demonstration. The fabricated MNN simultaneously learning the two shape-morphing behaviors of Fig. 2.4A. The MNN vibrates each time its input nodes are loaded by its behavior's step forces after a new combination of axial stiffness values has been assigned to the MNN's 21 tunable beams.

Movie S4: Learning comparison between different optimization algorithms. The two output nodes of the mechanical neural network (MNN) (i.e., Nodes 1 and 2) move from one displacement to the next as improved combinations of axial stiffness values are assigned to its beams, according to the genetic algorithm (GA) and the partial pattern search (PPS) algorithm, in such a way that produces decreasing mean squared error (MSE) values toward achieving the desired behaviors of Fig. 2.4A.

Acknowledgments

1.1 Personal

First, I would like to thank Jonathan Hopkins, Ph.D. who has helped to guide and shape my doctoral journey. His work ethic and drive has shown me what I can achieve when I fully devote myself to a project. Next, I would like to thank my friends and my lab mates Samira, Rudy, Miles, Amin, Armin, Sam, Talmage, Nigel, Pietro, Zhidi, Ivan, and Melika. Thank you all so much for the wonderful memories that we made together. Graduate school was not always easy, but it was better going through it with you all. Finally, I would like to thank my mother, father, and sister that were here to provide support, guidance, and endless love.

1.2 Funding

I would like to thank the Air Force Office of Scientific Research (AFOSR) and Byung "Les" Lee who has supported this work through (Air Force Office of Scientific Research FA9550-18-1-0459 (JBH) and Air Force Office of Scientific Research FA9550-22-1-0008 (JBH)), which has made my Ph.D. possible. I would also like to thank the UCLA Department of Mechanical and Aerospace for their support by appointing me as a departmental fellowship.

1.3 Journals

Chapter 2 contains a version of: Lee, R. H., Mulder, E. A. B., and Hopkins, J. B., 2022, "Mechanical Neural Networks: Architected Materials That Learn Behaviors," Science Robotics, 7(71), p. eabq7278. The final manuscript for this paper was drafted by JH based on material by RL, inspired by samples from EM.

Chapter 3 is a version of: Lee, R. H., Sainaghi, P., Hopkins, J. B., "Comparing Mechanical Neural-Network Learning Algorithms," submitted to ASME Journal of Mechanical

Design, 2023. The final manuscript for this paper was drafted by RL revised by JH with assistance from PS.

Chapter 4 is a version of a journal submission currently under review: Hopkins, J.B, Lee, R. H., Sainaghi P., “Mechanical Neural-network Metamaterials that Learn via Binary-stiffness Beams,” Smart Materials and Structures. The final manuscript for this paper was written by JH based revised by RL and PS.

Vita

Education

2014-2018	Bachelor of Science in Mechanical Engineering
	University of California, Santa Barbara

Employment

Summer 2016	Mechanical Engineering Intern. Northrop Grumman
Summer 2017	Mechanical Engineering Intern, Northrop Grumman
Summer 2018	Mechanical Engineering Intern, Northrop Grumman
2018-2023	University of California, Los Angeles (UCLA)
	Mechanical and Aerospace Engineering (MAE) Department
	Graduate Fellows, Research, and Teaching Assistant

Conference Presentations

2019	International Design Engineering Technical Conference (IDETC) presentation

Awards, Grants, and Fellowships

2018-2019	Departmental Fellowship

Teaching

2020-2022	Signatory for MAE Graduate Interests Council (MAEGIC) at UCLA

Journal Publications

Lee, R. H., Mulder, E. A. B., and Hopkins, J. B., 2022, "Mechanical Neural Networks: Architected Materials That Learn Behaviors," Science Robotics, **7**(71), p. eabq7278.

Hopkins, J.B **, Lee, R. H.**, Sainaghi P., "Mechanical Neural-network Metamaterials that Learn via Binary-stiffness Beams," Smart Materials and Structures, UNDER REVIEW, 2022

Lee, R. H., Sainaghi, P., Hopkins, J. B., "Comparing Mechanical Neural-Network Learning Algorithms," UNDER REVIEW, 2022

Yang, Z., **Lee, R. H.**, and Hopkins, J. B., 2022, "Hexblade Positioner: A Fast Large-Range Six-Axis Motion Stage," Precision Engineering, **76**, pp. 199–207.

Shimohara, S., **Lee, R. H.**, and Hopkins, J. B., 2022, "Compliant Mechanisms That Achieve Binary Stiffness along Multiple Degrees of Freedom," Journal of Composite Materials, p. 00219983221146262.

CHAPTER 1. INTRODUCTION

1.1 Background and Motivation

The unmatched ability of the human brain to quickly and efficiently learn has inspired many innovations which attempt to mimic its capabilities using interconnected neuron like structures. Recent attempts have been successful in creating artificial intelligence (AI) systems to solve complex problems like self-driving vehicles [1], text prediction [2], and image recognition [3] by using computational networks called Artificial Neural Networks (ANN)s. This approach is mainly focused on a creating a software solution to learning. However, an engineering material with the ability to alter its properties could alter the way that systems are designed. Rather than requiring a designer to select a material based on design goals, a learning material can be trained to adapt the optimal material properties for specified design requirements. The architected material presented in this dissertation is called a Mechanical Neural Network (MNN) and is able to fill this niche by altering its stiffness to acquire desired deformations. As an architected material, the effective material properties of the MNN are derived from the structure and the structural adaptability of the network rather than the type and arrangement of its chemical components.

For this work, the structure of the MNN is inspired by the computational structure of ANNs which have layers of neurons connected by a lattice of interconnected weight coefficients [4]. When input signals are provided to the input neurons in an ANN, the size of each weight determines how much of the input signals are passed to neurons in the next layer of the network. This new layer of neurons then applies a modifying nonlinear function to the weighted sum of outputs from the previous layer. This process is then repeated, and the output of the non-linear

functions then have a weighted sum applied as they transmit values to successive layers of neurons in the ANN. The weighting, modification, and transmission process propagates through the entire network to create a final set of outputs. The relationship between input and output is a complicated function of the network's parameters. ANNs are very powerful for modeling systems from data alone; in fact, a sufficiently deep neural network can model any arbitrary nonlinear functions [5]. To convert an untrained ANN (i.e., one with randomized weights) into an accurate model, the ANN's weights are adjusted to map known input values to corresponding output values. Given sufficient data, a trained ANN lattice can accurately represent any trends and features in the dataset.

The MNN's structure mimics the parameterization of the ANN with mechanical elements, this is done to create desired displacements for a specified loading condition. MNNs consist of several layers of mechanical nodes connected by a lattice of beams with adjustable axial-stiffnesses. These beams are analogous elements to the weight parameters in the ANN since increasing the stiffness of a beam also increases the amount of force transmitted between nodes. In this structure, the displacement of each node in the network depends on the stiffnesses of all the beams connecting to it. During training of an MNN, the stiffnesses of the beams are adjusted until a configuration is determined that produces the desired motions under specified loads [6]. After training, the desired motions achieved by the MNN are a property of the lattice, allowing an MNN to gain or remove behaviors simply by altering the same beam stiffnesses.

Though the framework of the MNN as a learning material is novel, other physical systems have used analogous physical properties to perform learning. In fact, the field of neuromorphic computing relies on using interconnected physical elements to perform neural computation. These neuromorphic computers use a range of physical phenomena such as

optics[7, 8], electromagnetics[9, 10], and acoustics [11, 12] to learn a variety of properties. In many ways, the structure of acoustic neuromorphic computers is similar to MNNs. These devices perform computation by creating mass distributions and geometries that respond to input vibrations with desired AI properties, like vowel recognition [12]. Many of these neuromorphic computers are trained in simulation by manipulating the shape and density of the model until the desired properties are reached, after which the optimized system can be fabricated for use. One issue with designing mechanical AI devices in this manner, opposed to the proposed MNN, is that altering the device's behavior can be challenging if new properties are desired as removing material is often easier than adding material.

One common learning strategy that avoids this problem is Reservoir Computing (RC) because the RC framework learns without modifying the physical elements of the neuromorphic computer. The RC process moves the learning process away from the material components and into a simple computer. In these RC computers, a single layer of weights is trained to create the desired responses from direct measurements of the system rather than attempting to alter the physical system [13]. This process creates an AI device that is easy to train using a single linear operation [14]. This is advantageous when computational power is limited, as preexisting physical systems can offload computation other computer systems [15]. One limitation of RC computers is the physicality of the learned behaviors. Rather than creating direct properties, these systems require feedback actuators for the learned behaviors to be exhibited in the physical world [16].

Whereas RC systems opt to create no changes in the physical system and learn using the natural dynamics of the hardware; another approach to neuromorphic computing, Physical Neural Networks (PNNs), uses disturbing forces and the forced dynamics of the physical

elements for neural computation [17]. Because PNNs rely on the forced dynamics, this framework is easily transferable to controllable systems and has been demonstrated using neuromorphic computers with electrical, optical, or mechanical elements. Further, a single layer PNN can easily be extended to deeper MNNs, by feeding the outputs from a single physical system to other similar layers. The measurements from these deep PNNs have been used to perform AI tasks like image and vowel classification [17]. While using forcing signals to shape system dynamics is an effective technique for physical learning, the resulting properties are closer to temporary signals, with utility for computation, rather than permanent physical changes as produced by a material.

The neuromorphic computation scheme that is most similar the MNNs are Physics Driven Learning (PDL) systems which can learn to produce permanent changes using lattices of interconnected physical elements such as circuits, pipes, or springs [18]. What makes PDL special is the training technique teach the system the desired outputs. PDL adjust the connection strength in through a two-stage process. During the first stage, the system's response to the specified loads is measured. In the second stage, the output nodes in the network are perturbed toward the desired output values. The difference between the natural and perturbed response is used to update the lattice connections [19]. The PDL learning process is very fast, for small steps, it approaches the speed of gradient descent [18]. It is a capable tool and it was able to teach a simulated piece of origami to classify images using forces proportional to measured image properties [20]. Despite these advantages, the framework is limited by its requirement for control over both the networks input and output nodes.

The MNN learning device and framework presented in this dissertation are differentiated from earlier work in neuromorphic computers for the following reasons. MNNs were

experimentally demonstrated to have retrainable physical material capable of learning and relearning permanent structural properties without external clamping. This framework gives a single MNN lattice the potential to learn and relearn properties in-situ after experiencing damage or to adapt to changes in requirements by simply retraining the architected material. Since retraining only requires adjusting the stiffness of the lattice elements and does not require altering the lattice structure or order, a single MNN can take on many properties throughout its useful life. The generalization of learning as a material and its demonstration in an experimental setting presents a broad leap in the capabilities of architected materials.

1.2 Dissertation Overview

Chapter 2 explorers the basic geometrical layouts required to construct a mechanical neural network through simulation. It proposes one method for creating adjustable stiffness beams and uses this understanding to fabricate an MNN device containing 21 beams for the simultaneous learning of behaviors. The MNN device is used to experimentally demonstrate the architected materials potential for learning and examines the impact of stiffness linearity on mechanical learning. During the experiments, two different algorithms are proposed and implemented to teach behaviors to MNNs.

In Chapter 3 the effects of various learning algorithms on the speed and accuracy of MNNs are investigated. A simulation capable of modeling realistic MNN behaviors including steady state noise is created for the 21-beam lattice from Chapter 2. This simulation is used to compare the speed and accuracy for six different learning algorithms when training MNNs under the effects that mechanical disturbances and noise. The fabricated MNN from Chapter 2 is then used to validate the findings from the earlier simulations. In this comparison, the relative performance for each algorithm was consistent between experimental learning and simulated

learning. In general, Nelder-Mead was identified as an exceptional learning algorithm due to its high training speed and accuracy as well as its consistent performance in the presence of sensor noise.

Chapter 4 explores the potential use of a simplified beam, capable of only achieving two different stiffness values, as a beam element in MNNs. Due to the simple, flexure-based, scalable mechanism used in this beam, microscale fabrication of these binary mechanical neural networks would be possible. This chapter describes the process of simulating these binary MNNs and investigates the effects of changing the relative stiffnesses of the stiff and compliant states as well as lattice size on mechanical neural network learning for a wide range of simultaneously learned behaviors.

Chapter 5 concludes this PhD dissertation with a brief summary and overview of the findings and how they can be applied to future mechanical neural networks and neural network research.

1.3 References

[1] Rao, Q., and Frtunikj, J., 2018, "Deep Learning for Self-Driving Cars: Chances and Challenges," *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, Association for Computing Machinery, New York, NY, USA, pp. 35–38.

[2] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K., 2019, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding."

[3] He, K., Zhang, X., Ren, S., and Sun, J., 2016, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

[4] LeCun, Y., Bengio, Y., and Hinton, G., 2015, "Deep Learning," Nature, **521**(7553), pp. 436–444.

[5] Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks Are Universal Approximators," Neural Networks, **2**(5), pp. 359–366.

[6] Lee, R. H., Mulder, E. A. B., and Hopkins, J. B., 2022, "Mechanical Neural Networks: Architected Materials That Learn Behaviors," Science Robotics, **7**(71), p. eabq7278.

[7] Lin, X., Rivenson, Y., Yardimci, N. T., Veli, M., Luo, Y., Jarrahi, M., and Ozcan, A., 2018, "All-Optical Machine Learning Using Diffractive Deep Neural Networks," Science, **361**(6406), pp. 1004–1008.

[8] Steck, J. E., Skinner, S. R., Cruz-Cabrera, A. A., Yang, M., and Behrman, E. C., 1999, "Backpropagation Training of an Optical Neural Network," *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, IEEE Computer Society, USA, p. 346.

[9] Han, R., Huang, P., Xiang, Y., Liu, C., Dong, Z., Su, Z., Liu, Y., Liu, L., Liu, X., and Kang, J., 2019, "A Novel Convolution Computing Paradigm Based on NOR Flash Array With High Computing Speed and Energy Efficiency," IEEE Transactions on Circuits and Systems I: Regular Papers, **66**(5), pp. 1692–1703.

[10] Adhikari, S. P., Yang, C., Kim, H., and Chua, L. O., 2012, "Memristor Bridge Synapse-Based Neural Network and Its Learning," IEEE Transactions on Neural Networks and Learning Systems, **23**(9), pp. 1426–1435.

[11] Zuo, S., Wei, Q., Tian, Y., Cheng, Y., and Liu, X., 2018, "Acoustic Analog Computing System Based on Labyrinthine Metasurfaces," Sci Rep, **8**(1), p. 10103.

[12] Hughes, T. W., Williamson, I. A. D., Minkov, M., and Fan, S., 2019, "Wave Physics as an Analog Recurrent Neural Network," Science Advances, **5**(12), p. eaay6946.

[13] Coulombe, J. C., York, M. C. A., and Sylvestre, J., 2017, "Computing with Networks of Nonlinear Mechanical Oscillators," PLOS ONE, **12**(6), p. e0178663.

[14] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A., 2019, "Recent Advances in Physical Reservoir Computing: A Review," Neural Networks, **115**, pp. 100–123.

[15] Urbain, G., Degrave, J., Carette, B., Dambre, J., and Wyffels, F., 2017, "Morphological Properties of Mass–Spring Networks for Optimal Locomotion Learning," Frontiers in Neurorobotics, **11**.

[16] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W., 2012, "The Role of Feedback in Morphological Computation with Compliant Bodies," Biol Cybern, **106**(10), pp. 595–613.

[17] Furuhata, G., Niiyama, T., and Sunada, S., 2021, "Physical Deep Learning Based on Optimal Control of Dynamical Systems," Phys. Rev. Applied, **15**(3), p. 034092.

[18] Dillavou, S., Stern, M., Liu, A. J., and Durian, D. J., 2022, "Demonstration of Decentralized, Physics-Driven Learning."

[19] Stern, M., Arinze, C., Perez, L., Palmer, S. E., and Murugan, A., 2020, "Supervised Learning through Physical Changes in a Mechanical System," Proceedings of the National Academy of Sciences, **117**(26), pp. 14843–14850.

[20] Stern, M., Hexner, D., Rocks, J. W., and Liu, A. J., 2021, "Supervised Learning in Physical Networks: From Machine Learning to Learning Machines," Phys. Rev. X, **11**(2), p. 021045.

CHAPTER 2. FOUNDATIONAL MECHANICAL NEURAL-NETWORK MODELS AND FABRICATED SYSTEM EXPERIMENTS

2.1 Abstract

Aside from some living tissue, few materials can autonomously learn to exhibit desired behaviors as a consequence of prolonged exposure to unanticipated ambient loading scenarios. Still fewer materials can continue to exhibit previously learned behaviors in the midst of changing conditions (e.g., rising levels of internal damage, varying fixturing scenarios, and fluctuating external loads) while also acquiring new behaviors best suited for the situation at hand. In this work, we describe a class of architected materials, called mechanical neural networks (MNNs), that achieve such learning capabilities by tuning the stiffness of their constituent beams similar to how artificial neural networks (ANNs) tune their weights. An example lattice was fabricated to demonstrate its ability to learn multiple mechanical behaviors simultaneously, and a study was conducted to determine the effect of lattice size, packing configuration, algorithm type, behavior number, and linear-versus-nonlinear stiffness tunability on MNN learning as proposed. Thus, this work lays the foundation for artificial-intelligent (AI) materials that can learn behaviors and properties.

2.2 Introduction

Scientists have been inspired by the interconnected network of neurons that constitute biological brains and enable complex learning with unmatched speed and energy efficiency. Consequently, many have sought to leverage a variety of interconnected networks to mimic natural learning for numerous artificial-intelligent (AI) applications [1-3].

Some of the first networks developed for AI purposes were purely mathematical in form. The concepts underlying these mathematical networks, called artificial neural networks (ANNs) (Fig. 2.1A), were first introduced by McCulloch and Pitts [4] but were later matured by Rosenblatt [5]. The mathematical formulation underlying ANNs can be diagramed using interconnected lines, shown blue in Fig. 2.1A, that represent scalar values, called weights [6], which are multiplied by input numbers that are fed into multiple layers of activation functions [6], called neurons, which ultimately produce output values. If the ANN is provided with a set of known input and output values, the network can be trained by tuning its weights so that it accurately predicts previously unknown output values that result for any desired input values. Hornik et al. [7] proved the true AI potential of ANNs by demonstrating that with sufficiently large numbers of neurons and layers, ANNs could learn to model almost anything by accurately mapping any number of inputs to any number of outputs. Tuning the weights of sizeable ANNs, however, proved to consume large amounts of computational time and energy using traditional digital computers.

Thus, further inspired by the physical nature of biological brains, scientists began developing physical networks to more rapidly tune weights (i.e., learn) with higher efficiencies due to their analogue nature. Most of these physical networks can be classified as electrical [8-12] or optical [13-17] networks. Although some physical neural networks utilize the vibrations of mechanical structures to improve the speed and efficiency of learning, none yet exist that are purely mechanical. Roboticists have learned to leverage the dynamics of mechanical bodies as a computational resource for enabling mathematical ANNs to be more efficiently trained by restricting only the final layer's weights to be tuned. This approach, called morphological computation [18], is a mechanical version of the concept of reservoir computing [19, 20], where

the reservoir used to simplify the mathematical computation is the structure of the robot itself. Networks of springs and point masses [21,22], tensioned cables and rigid bodies [23, 24], as well as soft bodies [25,26] have been employed to demonstrate this approach. The most mechanical instantiation of a neural network to date was proposed by Hermans et al. [27]. This network consists of a vibrating plate that is excited by acoustic waves as inputs and outputs. Instead of tuning the mechanical properties of the plate itself (i.e., its stiffness, damping, or mass properties) to tune the network's weights, masking signals of interfering acoustic waves were electrically generated to train the network. This concept was recently extended by Wright et al. [28] using multiple layers of vibrating plates to achieve a deep physical neural network.

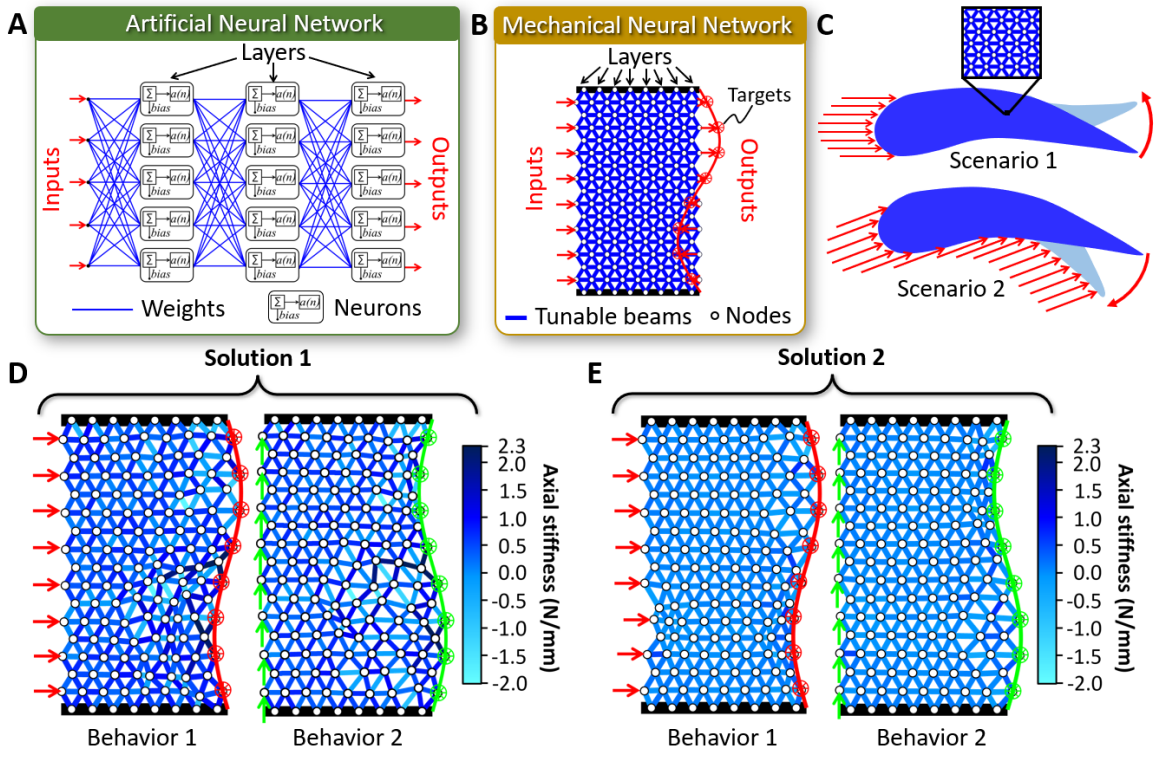

Figure 2.1 Introduction to mechanical neural networks (MNNs) and how they learn mechanical behaviors. (**A**) Artificial neural networks (ANNs) mathematically map numerical inputs to outputs by tuning scalar weights within layers of neurons consisting of activation functions. (**B**) MNNs are mechanical analogues to ANNs in that they map force and displacement inputs and outputs using tunable beams, which are analogous to weights, and physical nodes, which are analogous to neurons. (**C**) Example shape-morphing behaviors that could be learned by MNN aircraft wings in two different scenarios. (**D** and **E**) Two different combinations of beam stiffness values (i.e., solutions) that achieve the same two shape-morphing behaviors.

In this work, a different physical network, called a mechanical neural network (MNN), is introduced. MNNs are lattices of interconnected tunable beams, shown blue in Fig. 2.1B, that join at nodes, which are driven by force or displacement inputs and outputs. The stiffness values of the interconnected beams are tuned as network weights to train the lattice such that it can learn desired mechanical behaviors (e.g., shape morphing, acoustic wave propagation, and mechanical computation) and bulk properties (e.g., Poisson's ratio, shear and Young's modulus, and density). Thus, this work introduces a class of architected materials (a.k.a., mechanical metamaterials) [29] that learn as a consequence of prolonged exposure to unanticipated ambient loading conditions. Although others have proposed acoustic metamaterials that can perform specific mechanical computations [30,31], these materials are not neural networks and thus cannot learn. Hughes et al. [32] proposed an acoustic metamaterial that behaves as a trained neural network, but a fabricated version of the proposed design could not learn new behaviors since training is performed during the design process by adjusting the mass within a vibrating plate using simulation. While other mechanical concepts have also been proposed and demonstrated using simulation only [33, 34], the MNN concept introduced here is physically demonstrated experimentally. The concept can also be extended into complex three-dimensional lattices, which can occupy volumes of arbitrary shape and accommodate desired fixturing requirements for practical material applications. And, since MNNs inherently possess numerous layers of nodes, which are analogous to the neurons within ANNs, MNNs behave as deep neural networks that can learn many complex behaviors simultaneously. If a MNN is damaged, cut to occupy an alternate volume, or fixtured differently, it can relearn previously mastered behaviors and acquire new behaviors as needed with exposure to changing ambient conditions. An application could include MNN aircraft wings (Fig. 2.1C) that learn to morph their airfoil shape

as desired in response to certain wind-loading scenarios such that the aircraft achieves greater efficiency and maneuverability as it accrues flight experience. This work demonstrates the ability of a MNN to learn two different shape-morphing behaviors using two different algorithms. Experimental and simulated studies are performed to determine the effect of lattice size, packing configuration, algorithm type, behavior number, and linear-versus-nonlinear stiffness tunability on MNN learning.

2.3 Results

2.3.1 Learning process

MNNs mechanically learn behaviors analogously to how ANNs mathematically map numerical inputs to outputs. To understand the specifics of how MNNs learn, consider the 8 layer-deep 2D MNN lattice of tunable beams packed in a triangular configuration with 8 input and output nodes shown in Fig. 2.1B. The black bars shown at the top and bottom of the lattice represent fixed ground. Suppose when the input nodes are loaded by equal horizontal forces, shown as red arrows in Fig. 2.1B, it is desired that the output nodes respond by moving to target displacements along the contour of the red sinusoidal curve shown in Fig. 2.1B. To learn this behavior in the midst of unexpected and changing loading scenarios, each tunable beam in the lattice would be prescribed with a random stiffness value. Sensors (e.g., strain gauges on each beam) would then determine the displacement of each node in the lattice for each loading scenario. Since the beam stiffness values and the node displacements are known (i.e., prescribed and measured respectively), the MNN could determine when the lattice has been loaded with the desired behavior's loading scenario (i.e., the horizontal forces shown in Fig. 2.1B). Anytime the desired loading scenario occurs, the lattice sensors would measure the resulting displacements of the output nodes on the lattice's right side and the mean squared error (MSE) of these

displacements would be calculated by subtracting them from the target displacements and averaging the resulting differences squared. The tunable beams would then change their stiffness values according to an optimization algorithm such that when the process of loading, measuring, and calculating the MSE is repeated, the MSE is minimized until a working combination of beam stiffness values is identified. One possible combination of beam stiffness values that achieve the desired behavior of Fig. 2.1B is shown on the left side of Fig. 2.1D. Different shades of blue are used to denote different axial stiffness values.

Suppose it is desired that the MNN learn another behavior in addition to retaining the first behavior shown in Fig. 2.1B. Specifically, suppose it is desired that the lattice's output nodes displace to an inverted sinusoidal contour, shown as a green curve on the right side of Fig. 2.1D, in response to its input nodes being loaded by equal vertical input forces, shown as green arrows, instead. To learn the new behavior while maintaining the ability to simultaneously achieve the first behavior, the lattice of tunable beams would begin with the combination of stiffness values that were found to successfully achieve the first behavior. Then those stiffness values would be adjusted according to the same optimization algorithm to find a new combination of stiffness values that achieve both behaviors simultaneously. This optimization would be achieved by measuring the displacements of the output nodes in response to loading the material's input nodes with alternating horizontal and vertical forces. A single MSE would be calculated that simultaneously considers the results of both loading scenarios. That cumulative MSE would then be minimized so that a desired combination of beam stiffness values would be identified that successfully produced both the new and original behavior. Note that all the tunable beams are colored with the same shades of blue between the two corresponding lattice

images of Fig. 2.1D since a single combination of stiffness values was identified that could successfully enable the MNN to achieve both behaviors.

Since MNNs typically possess multiple layers, they can learn the same set of desired behaviors using many different combinations of beam stiffness values. Note that although Solution 2 of Fig. 2.1E exhibits the same desired behaviors as Solution 1 of Fig. 2.1D, it does so with a different combination of beam stiffness values. The fact that many different combinations of beam stiffness values can achieve the same behaviors enables MNNs to learn many new behaviors. Moreover, MNNs don't need to be configured, fixtured, or loaded as shown in the example of Fig. 2.1B to learn. Any combination of nodes within a MNN could be fixed, loaded as an input, and sensed as an output to learn almost any mechanical behaviors desired.

2.3.2 Tunable beams

There are many ways the stiffness of a beam could be tuned to enable MNN learning. Principles of jamming [35], phase changing [36], static balancing [37], and electrorheology [38] among other approaches [39] could be employed. Approaches that enable beams to continue exhibiting their prescribed stiffness without external influence (e.g., electrical power, magnetic fields, or temperature) are preferable for MNNs applications since such networks physically store a kind of mechanical 'muscle memory' in their architecture for manifesting the desired behaviors previously learned.

The beams used to demonstrate the concept of MNNs in this work (Fig. 2A), however, were designed to achieve tunable stiffness via closed-loop active control. This approach was chosen so that any desired linear or nonlinear force-displacement responses (including responses with negative stiffness) could be prescribed to each beam to enable a broader study of MNN

learning. These actively controlled beams use voice coils and strain gauges to actuate and sense the deformations of flexures, which guide the extensions and contractions of the beams along their axes. These beams can be seen in Fig. 2.2A and Fig. 2.2B and are described in the next section.

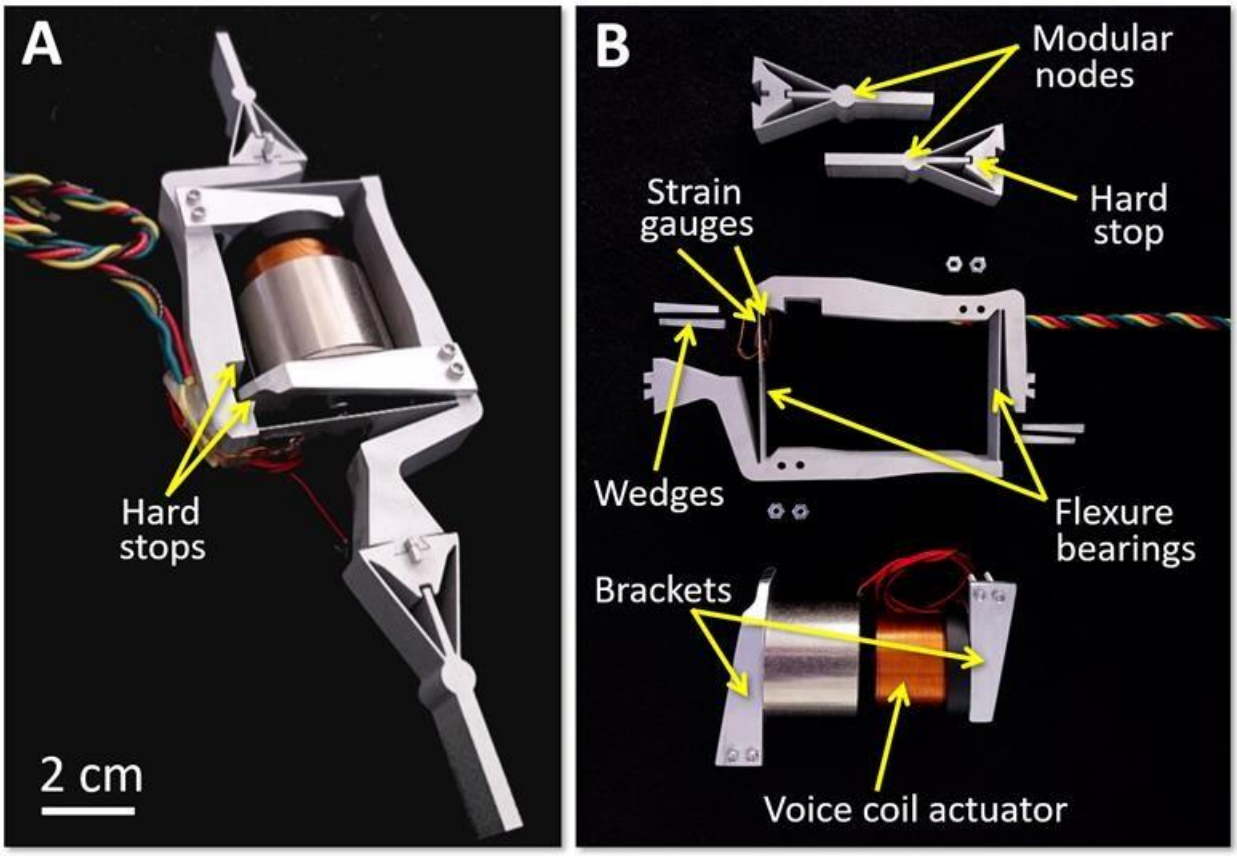

Figure 2.2 Tunable beam used within the experimental mechanical-neural-network (MNN) study of this work. The beam shown with its labeled parts (**A**) assembled and (**B**) disassembled.

A detailed discussion about their fabrication and function is provided in section 2.5.3. A diagram (Fig. 2.13A) detailing each beam's closed-loop controller is also provided in Section 2.5.2 along with examples of the four calibration plots (Fig. 2.13, B to E) that need to be generated by an Instron testing machine to control each beam's axial stiffness.

A discussion about the closed-loop controller and its calibration plots is provided in Section 2.5.2. The controller was designed so that when it was set to achieve a linear force-displacement response, the axial stiffness of the tunable beam (i.e., the slope of its response) would be the controller's proportional gain, K_p. Figure 2B provides the force-displacement responses of the tunable beam in Fig. 2A as it is controlled using the linear scenario with different K_p values to achieve various positive and negative stiffness values. A video showing the Instron cycling the beam as it is being actively controlled with different K_p values is provided (Movie S1). The maximum and minimum stiffness values that the beam can be controlled to achieve without becoming unstable or exceeding the actuator's force capabilities was measured to be 2.3 N/mm and -2 N/mm respectively as shown in Fig. 2C.

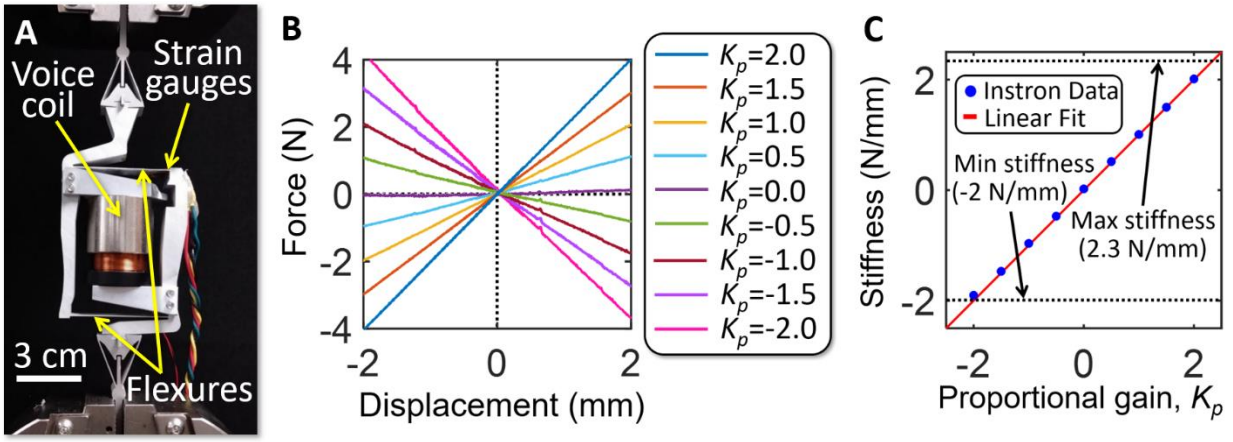

Figure 2.3 Tunable beams that use closed-loop control to achieve variable axial stiffnesses. (**A**) Voice coil and strain gauges are used as actuators and sensors to control the axial stiffness of the beam. (**B**) Data collected from an Instron as it stretches and compresses the tunable beam while it is actively controlled to achieve linear force-displacement responses using different proportional gain values, K_p. (**C**) Plot demonstrating how well the controller's prescribed proportional gain corresponds with the beam's resulting axial stiffness.

2.3.3 Experimental demonstration and study

Tunable beams of the kind shown in Fig. 2A were fabricated and assembled (Fig. 3) within a lattice to experimentally demonstrate and study MNN learning. Four additional actuators were used within decoupling flexures (Fig. 3A) to load the MNN's two input nodes

with desired in-plane forces, and cameras (Fig. 3B) were used to directly measure the displacements of the two output nodes. A discussion about the MNN's features, fabrication, and control electronics (Fig. 2.14, A and B) is provided in Section 2.5.3. Additional photos (Fig. 2.4, A and B), a video (Movie S2), and a discussion about how the MNN was calibrated after its beams were assembled in the lattice are provided in Section 2.5.4. Plots demonstrating the importance of calibrating MNNs are provided in Fig. 2.15.

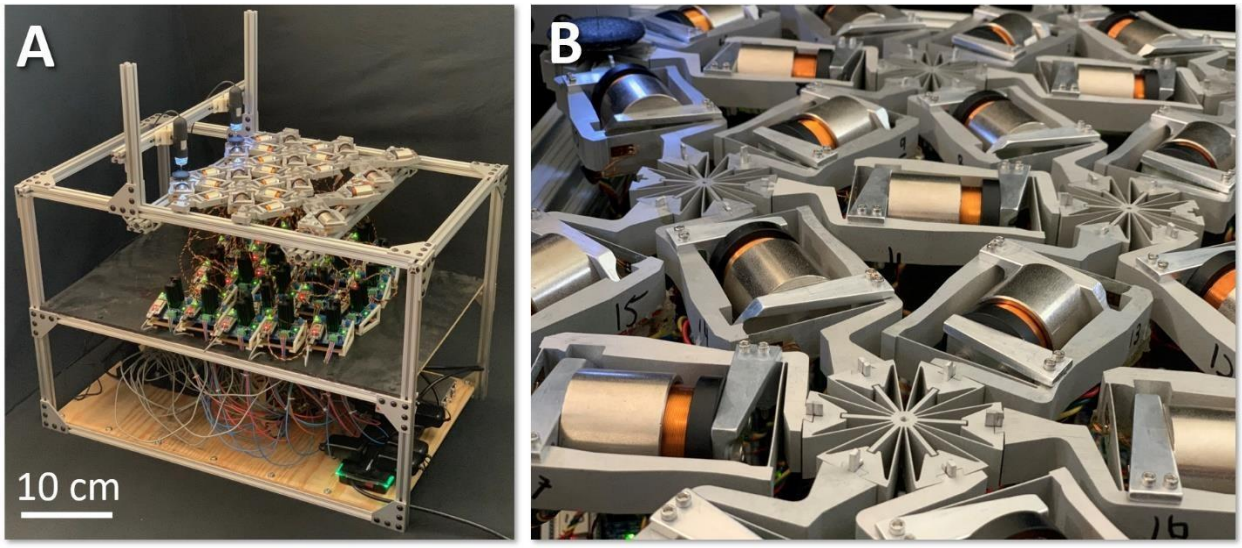

Figure 2.4 Mechanical neural network (MNN) fabricated for the experimental study of this work. (**A**) Full-view and (**B**) close-up photographs of the fabricated MNN lattice and its constituent tunable beams.

MNNs that require external sensors (e.g., cameras) to directly measure the displacements of their output nodes cannot learn without being placed in a testing rig, which is not practical for most applications that require in-field learning. Thus, it's important that the same sensors (e.g., strain gauges) that measure and help control the extension and contraction of their corresponding beams be used to also measure the output-node displacements indirectly to demonstrate practical MNN learning. Thus, the cameras mounted to the frame of the MNN in Fig. 3B were used to validate this indirect approach (i.e., the strain-gauge approach) for measuring output-node

displacements. The results of this validation (Fig. 2.16, A and B) as well as a discussion about how the validation was performed are provided in Section 2.5.5.

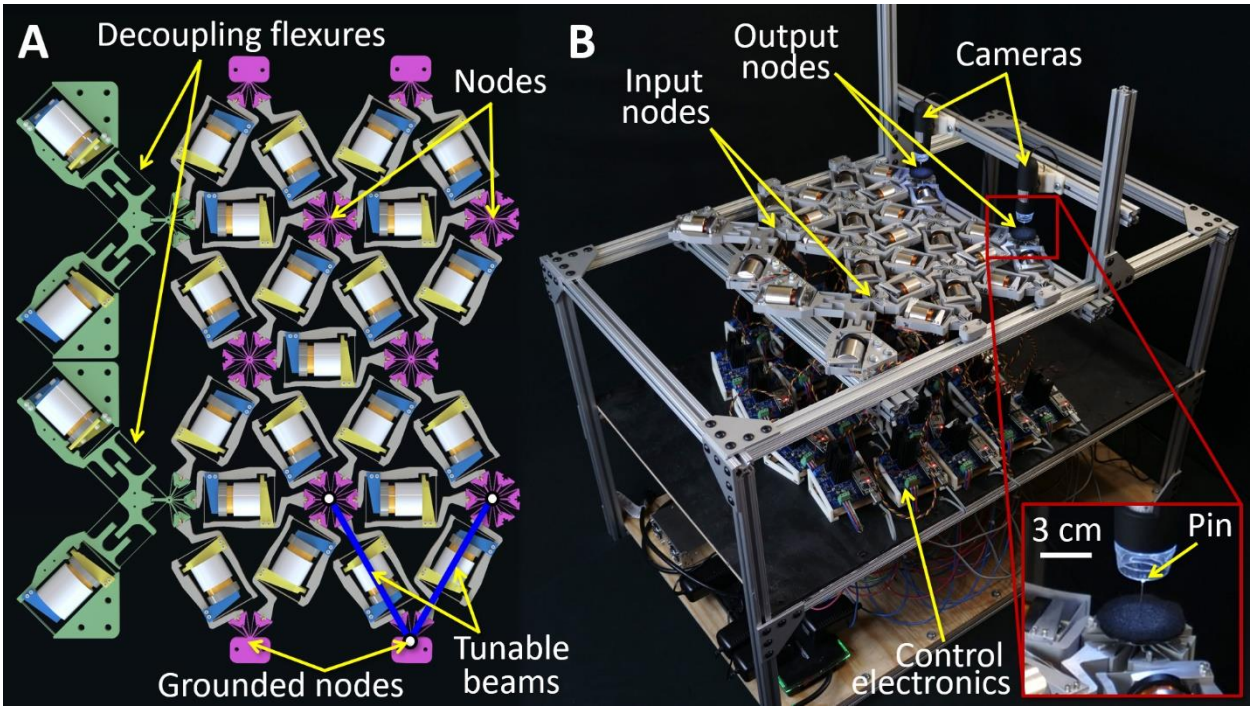

Figure 2.5 A mechanical neural network (MNN). (**A**) A computer-aided design (CAD) model and (**B**) a photo of the MNN used to conduct the experimental learning study of this work.

The MNN of Fig. 3B was used in conjunction with this strain-gauge approach to demonstrate that a triangular lattice of 21 tunable beams (shown as blue lines in Fig. 4A) could simultaneously learn two different sinusoidal shape-morphing behaviors (shown red and green in Fig. 4A for behaviors 1 and 2 respectively). Behavior 1 is manifest when the output nodes, labeled Node 1 and Node 2 in Fig. 4A, displace to the right and to the left by 0.5 mm respectively as the two input nodes are both pushed to the right (as shown by the red arrows) with equal magnitude. Behavior 2 is manifest when Node 1 and Node 2 displace in the opposite directions (i.e., to the left and to the right by 0.5 mm respectively) as the two input nodes are both sheared upwards (as shown by the green arrows) with equal magnitude. As the MNN

attempted to exhibit these two desired behaviors according to the learning process, the axial stiffness values of each beam were allowed to be tuned between the maximum and minimum values of 2.3 N/mm and -2 N/mm respectively according to the limits measured in Fig. 2C.

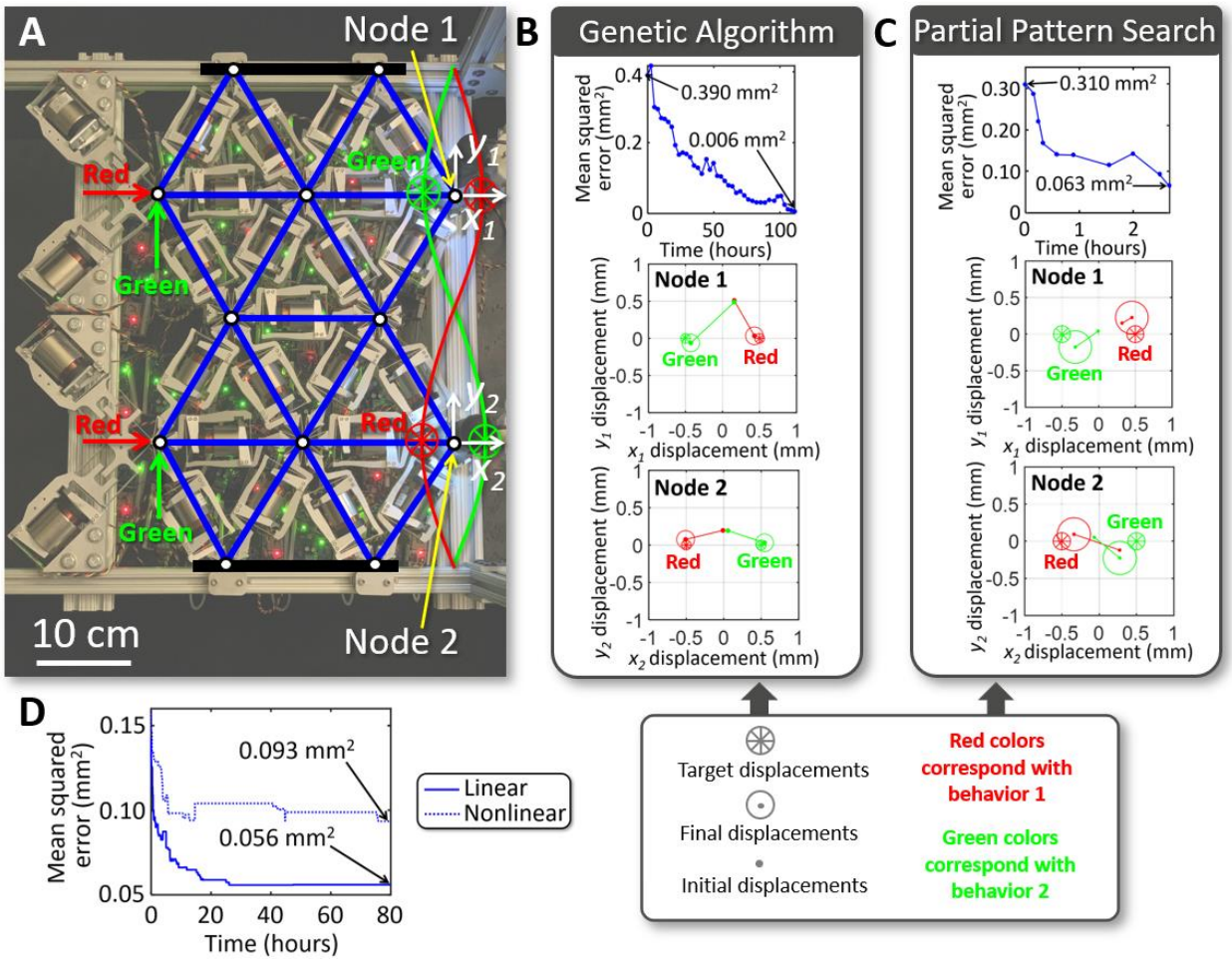

Figure 2.6 Experimental study results. (**A**) Two behaviors (shown red and green) that the mechanical neural network (MNN) attempted to learn using two different optimization algorithms. The results of (**B**) the genetic algorithm (GA) and (**C**) partial pattern search (PPS) showing mean squared error (MSE) over time and the initial and final displacements of the output nodes (i.e., Nodes 1 and 2) relative to their target displacements. (**D**) The MNN's MSE plotted over time as its beams are controlled to exhibit tunable linear and nonlinear force-displacement responses.

Two optimization algorithms—genetic algorithm (GA) [40] and partial pattern search (PPS) [41]—were used to learning the two behaviors to compare their performance. The details underlying each optimization algorithm are provided in Section 2.5.6, and a video showing the

MNN learning is provided in Supplementary Materials (Movie S3). The learning results of the GA and PPS are provided in Fig. 4, B and C respectively. The MSE of each algorithm is plotted over time as the MNN learns the two desired behaviors simultaneously, and the initial and final displacements of Nodes 1 and 2, relative to the desired target displacements, are also provided for each behavior. A video showing how the nodes move from one displacement to the next (corresponding to each blue dot in the MSE plots of Fig. 4, B and C) is provided in Supplementary Materials (Movie S4).

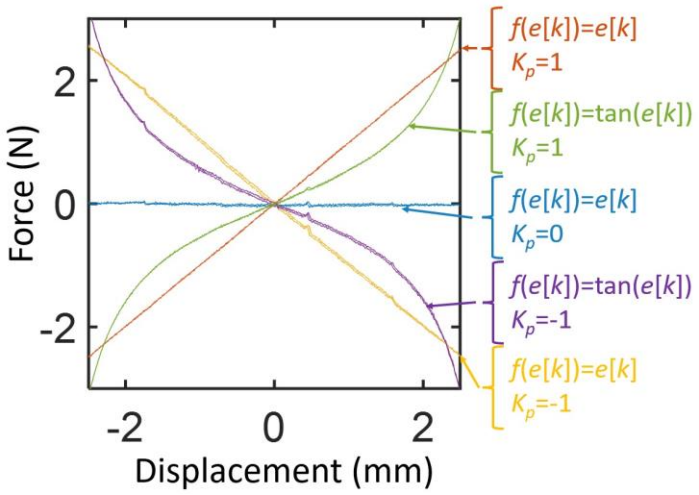

Figure 2.7 Effect of proportional gain, K_p, and $f(e[k])$ on tunable beam stiffness as labeled in Fig. 2.13A. Example force-displacement responses achieved via closed-loop control of the tunable beam in Fig. 2.3A as measured by an Instron testing machine along the beam's axis for different values of K_p (i.e., 1, 0, and -1) and for different functions, $f(e[k])$ (i.e., $e[k]$ and $\tan(e[k])$). These functions were used to compare the effect of linear versus nonlinear beam stiffness on the process of mechanical-neural-network (MNN) learning.

The MNN of Fig. 2.4A was also used to compare learning with tunable beams that exhibit linear (e.g., Fig. 2.2B) versus nonlinear force-displacement responses. Specifically, tangent functions (e.g., the responses shown in Fig. 2.7 for different K_p values) were used for the nonlinear scenario. The MNN's tunable beams were initially set to only exhibit linear force-displacement responses with stiffness values that could vary between 2.3 N/mm and -2 N/mm

according to the limits measured in Fig. 2C. Two random but different shape-morphing behaviors were generated for the MNN to learn. Each behavior was generated by selecting forces with randomly generated *x*- and *y*-axis components between ±2 N, which cause the MNN's output nodes to move selected displacements with randomly generated *x*- and *y*-axis components between ±0.5 mm when the selected forces load the input nodes. The MNN then used the PPS algorithm to learn the generated pair of random behaviors simultaneously. The MSE of this learning process over time was recorded similar to the example plots shown in Fig. 4, B and C. Five additional random but unique pairs of behaviors were then generated and learned independently by the MNN. The 6 total resulting MSE-versus-time plots were averaged to produce the single solid-line plot of Fig. 4D (i.e., the plot corresponding to the linear scenario). The same 6 pairs of generated behaviors were then learned by the same MNN but its tunable beams were set to only exhibit tangent force-displacement responses (i.e., a nonlinear response) with instantaneous stiffness values that could vary between 2.3 N/mm and -2 N/mm according to the limits measured in Fig. 2C. Note that although 2.3 N/mm was found to be the largest axial stiffness value achievable by the tunable beams of this study, that finding is conservative and is only true for instantaneous stiffness values (i.e., beam stiffness values prior to deformation). When the beam is deformed an appreciable amount, it can be stably controlled with larger stiffness values to accommodate the rising tangent function profile. The 6 resulting MSE-versus-time plots were averaged to produce the single dotted-line plot of Fig. 4D (i.e., the plot corresponding to the nonlinear scenario).

2.3.4 Previous MNN attempts and reasons for failure

Prior to the successful demonstration of the MNN of Fig. 4, other fabricated MNNs as shown in (Fig. 2.8, A and B) failed to learn mechanical behaviors prior to the successful design

of Fig. 4A. Identifying the reasons why they failed to learn is important for understanding how to design MNNs that successfully learn. The failed MNNs used the tunable beam design shown in Fig. 2.8A within their lattice.

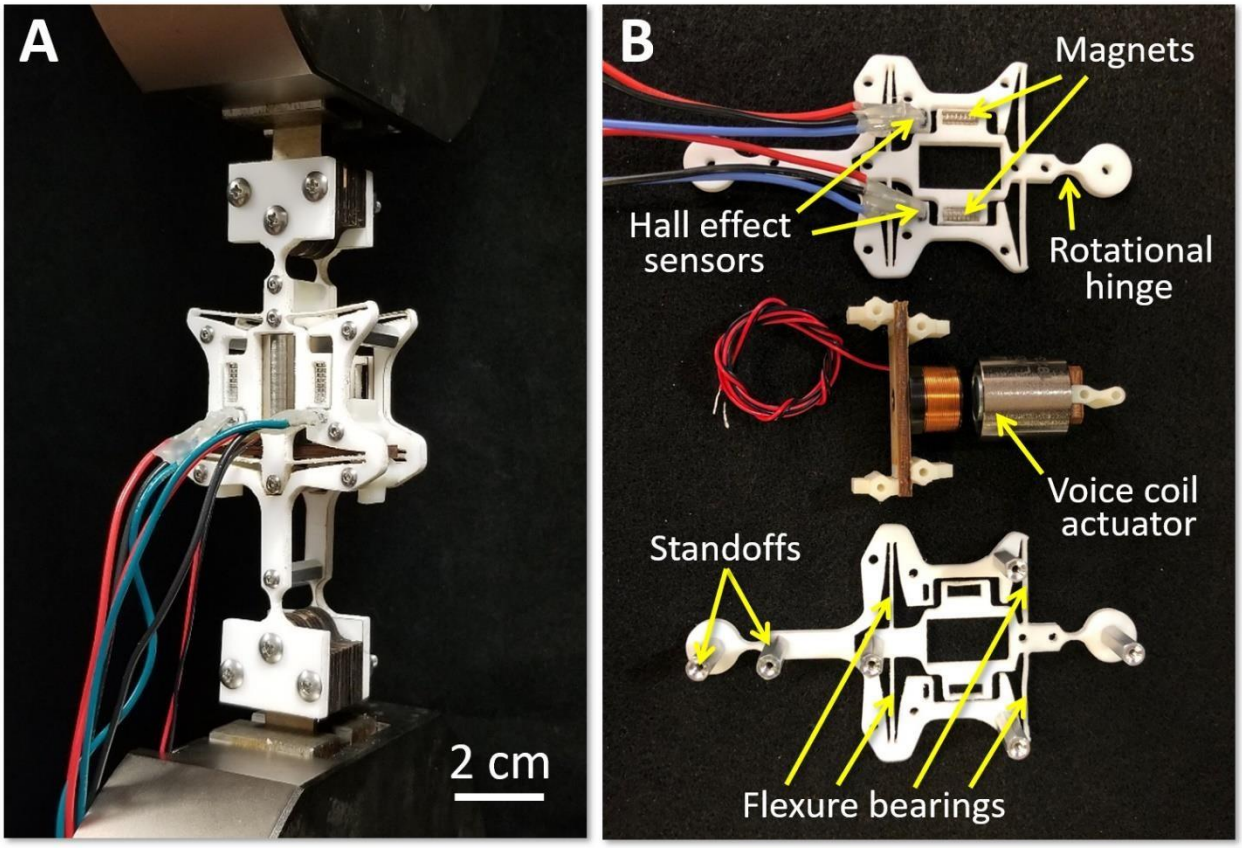

Figure 2.8 Tunable beam design used within prior mechanical-neural-network (MNN) attempts that failed to learn. The beam shown with its labeled parts (**A**) assembled and (**B**) disassembled.

The beam was assembled by bolts using two layers of laser-cut polytetrafluoroethylene (PTFE) sheets (Fig. 2.8B), which were separated by aluminum standoffs. Flexure bearings, which guided the axial displacements of the beam, were cut within these sheets. Two stacks of small cylindrical magnets were embedded in the upper sheet and were used so two Allegro MicroSystems Hall effect sensors (A1324LUA-T) could accurately measure the beam's axial displacements. A BEI Kimco linear voice coil (LA08-10-000A) was used to actuated the beam's

axial displacements and was mounted to a laser-cut wooden support, which was fixtured by 3D printed parts between the two PTFE sheets. The entire MNN lattice (Fig. 2.9A) was assembled using similar parts to those shown in Fig. 2.8B but required only two large laser-cut sheets of PTFE for all the beams combined. Decoupling flexures, attached to both the input and output nodes, were also included within the two PTFE sheets. The input nodes were driven by four additional voice-coil actuators and the output nodes were sensed by four additional Hall effect sensors (Fig. 2.9A). The MNN was mounted on a wooden box and was controlled from below by custom-designed control circuitry from within the box (Fig. 2.9B) using five Arduino Megas. When the MNN failed to learn, a wooden frame was built around it so that Nylon thread could be used to prevent the MNN from sagging the small amount that it previously did (Fig. 2.9C). A typical MSE-versus-generations plot produced by the MNN when the genetic algorithm (GA) was applied to the learning process is shown in Fig. 2.9D. Note from the plot that although learning appeared to be occurring for certain periods of time, the MSE would occasionally jump to higher values and would never settle to a value small enough for the MNN to successfully learn any shape-morphing behaviors. This occurred because the system's loading response would change with unacceptably poor repeatability so that the optimization algorithm's decisions were misinformed by outdated response data.

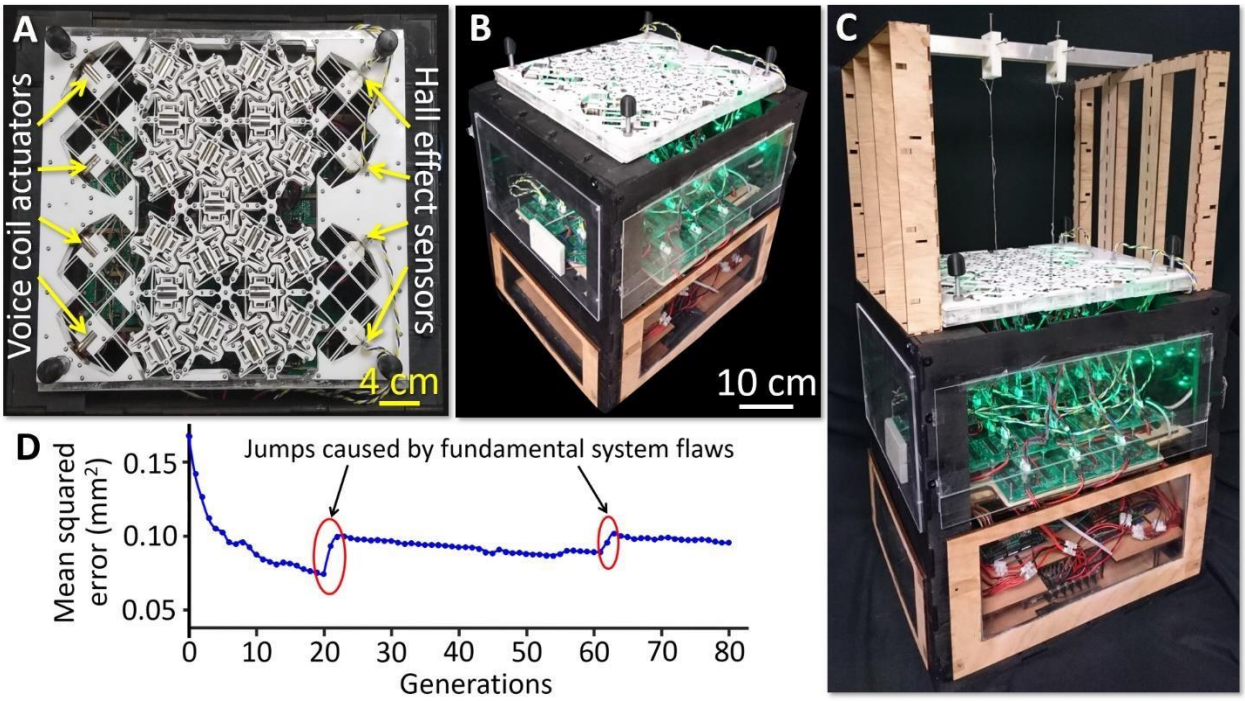

Figure 2.9 Prior mechanical-neural-network (MNN) attempts that failed to learn primarily due to a lack of precision (i.e., repeatability). Photograph of a fabricated MNN design taken (**A**) from above and (**B**) from an isometric view that shows its control electronics in a box below the MNN. (**C**) A different version of the design shown with Nylon threads attached to nodes for preventing the MNN from sagging under its weight. (**D**) A mean-squared-error-versusgenerations plot that was produced by the MNN of (C) using the genetic algorithm (GA). The plot demonstrates that its MNN failed to settle on a low enough error value sufficient for the MNN to learn its desired shape-morphing behaviors. Similar plots with equally unacceptable jumps in error were generated by all the learning attempts of the MNN designs of this figure.

There are multiple reasons for the unacceptably poor repeatability of the MNNs in Fig. S9, A to C. Due to the fact that they were made predominantly from PTFE, the MNNs experienced stress relaxation and creep as a consequence of the changing voice-coil actuator loads and gravity. Moreover, since their flexure bearings (Fig. S8B) were made of PTFE, each tunable beam within the MNNs exhibited unacceptable hysteresis as they deformed due to internal friction caused by polymer chains slipping within the PTFE flexures. The gaps between the voice-coil actuators' magnets and coils were also not sufficiently large and thus the magnets and coils would collide and rub past each other during learning, which would generate friction and noise that also contributed to the MNNs' poor repeatability. It is also possible that some of their many bolts would loosen over time due to the vibrations induced by the step forces

imparted by the input actuators during learning, which would produce additional friction. Moreover, the Hall effect sensors produced significantly more signal noise than the strain gauge sensors of the MNN in Fig. 4A. And, despite there being two Hall effect sensors per beam, they experienced unwanted cross-talk from the magnetic fields induced by the voice-coil actuators instead of exclusively experiencing the intended changing magnetic fields of their cylindrical magnet stacks. Furthermore, the symmetric flexure bearings and rotational hinges of the beam design of Fig. 2.8B in conjunction with the decoupling flexures of the input-node actuators and output-node sensors produced a two-layer flexure system that is highly over-constrained. Over-constrained systems can exhibit nonlinearity, bifurcation, and stress buildup, which negatively impacts precision (i.e., repeatability).

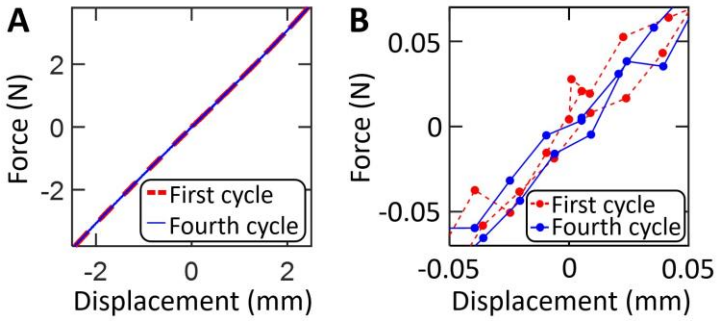

Figure 2.10 Minimal hysteresis measured in the flexure bearings of the tunable beam design of Fig. 2.3A. First and fourth loading cycle measured by an Instron testing machine of the beam in Fig. 2.3A shown (**A**) zoomed out over a larger displacement range and (**B**) zoomed in over a smaller displacement range.

The MNN of Fig. 4A was designed to overcome the failings of the MNN designs of Fig. 2.9, A to C so that MNN learning could be successfully demonstrated. Since its flexures were made of aluminum, the design of Fig. 4A did not experience appreciable stress relaxation, creep, sagging, or hysteresis. The plot of Fig. 2.10A shows the first and fourth cycle axial force-displacement response of the tunable beam of Fig. 2.3A. Note that hysteresis is barely apparent even when the plot is shown zoomed in over a much smaller range of deformation (Fig. 2.3B).

The force displacement responses plotted in Fig. 2B included two cycles of loading for each K_p value to demonstrate that hysteresis remains minimal in the beam design of Fig. 2A even when closedloop control is applied. The gaps between the voice-coil actuators' magnets and coils in the MNN of Fig. 4A were selected to provide sufficient clearance so no bodies would collide or rub within the range set by the tunable beam's hard stops (Fig. 2.3A). Wedges (Fig. 2.3B) were used to join each beam together within the MNN of Fig. 4A via press-fit principles to prevent slip while allowing the beams to be removed and reattached on demand. This modularity was necessary so that each beam could be individually calibrated, which was not possible to achieve for the tunable beams permanently joined within the MNNs of Fig. 2.9, A to C. The flexure bearings and strain gauge sensors within the MNN of Fig. 4A were also chosen since they are significantly more linear than the flexure bearings and Hall effect sensors within the MNNs of Fig. 2.9, A to C. This improved linearity made controlling the axial stiffness of the beams within the MNN of Fig. 4A significantly easier. The output-node decoupling flexures of the MNNs in S9, A to C were removed from the MNN design of Fig. 4A entirely and the input-node decoupling flexures (Fig. 3A) were adapted to include rotational blade flexures so that their input nodes would not be constrained to prevent rotation. Although still technically over-constrained, the flexures within the design of Fig. 4A constrain their MNN with significantly less over-constraint than the flexures within the designs of Fig. 2.9, A to C. Moreover, note that the electronics (Fig. 2.14) used to control the MNN of Fig. 4A was upgraded from the electronics used to control the MNNs of Fig. 2.9, A to C to being current controlled. This upgrade allows the MNN of Fig. 4A to achieve greater force stability when learning dynamic behaviors while increasing the range of axial stiffness over which its constituent beams can be controlled.

2.3.5 Simulation study

A computational tool, informed by the measured and modeled (Fig. 2.31) characteristics of the tunable beam of Fig. 2A, was created and used to simulate MNN learning scenarios, which the physical MNN of Fig. 3B was not designed to attempt. The tool's assumptions are detailed in Section 2.5.7, and a discussion about how the tool was verified using finite element analysis (FEA) (Fig. 2.18, A to E) is provided in Section 2.5.8. The computational tool was used to generate the example of Fig. 2.1, D and E according to the details also provided in Materials and Methods.

Three simulation studies were conducted using the tool. The MNNs of the first simulation study were all configured as triangular lattices (e.g., Fig. 2.1B) with 8 input and 8 output nodes. Their tunable beams were assigned axial stiffness values between 4 N/mm and -2 N/mm. Learning was simulated using different numbers of layers and different numbers of random behaviors. Random behaviors were generated by selecting input-node forces and output-node displacements with randomly generated x- and y-axis components between ±1 N and ±0.5 mm respectively. To ensure that each new behavior generated was sufficiently different from all previously generated behaviors, a MSE was calculated for each previous behavior by averaging the difference between the previous and new behavior's input forces squared. As long as the MSEs that were calculated from each of the previously generated behaviors all exceeded 0.3 N^2, the new behavior was deemed sufficiently different. Once sufficiently different behaviors were generated, three additional unique sets of different behaviors were generated for each scenario. The simulated MNN then attempted to simultaneously learn each unique set of behaviors four times and the final MSE (i.e., the last MSE that the optimization algorithm achieved by comparing the output-node displacements with the target displacements as described previously)

of the attempt that yielded the lowest value was averaged with the lowest final MSEs generated by learning the other unique sets of behaviors. The resulting MSE average was plotted for different numbers of layers and behaviors in Fig. 5A. Ten example behaviors that were randomly generated for this study are shown in Fig. 2.11.

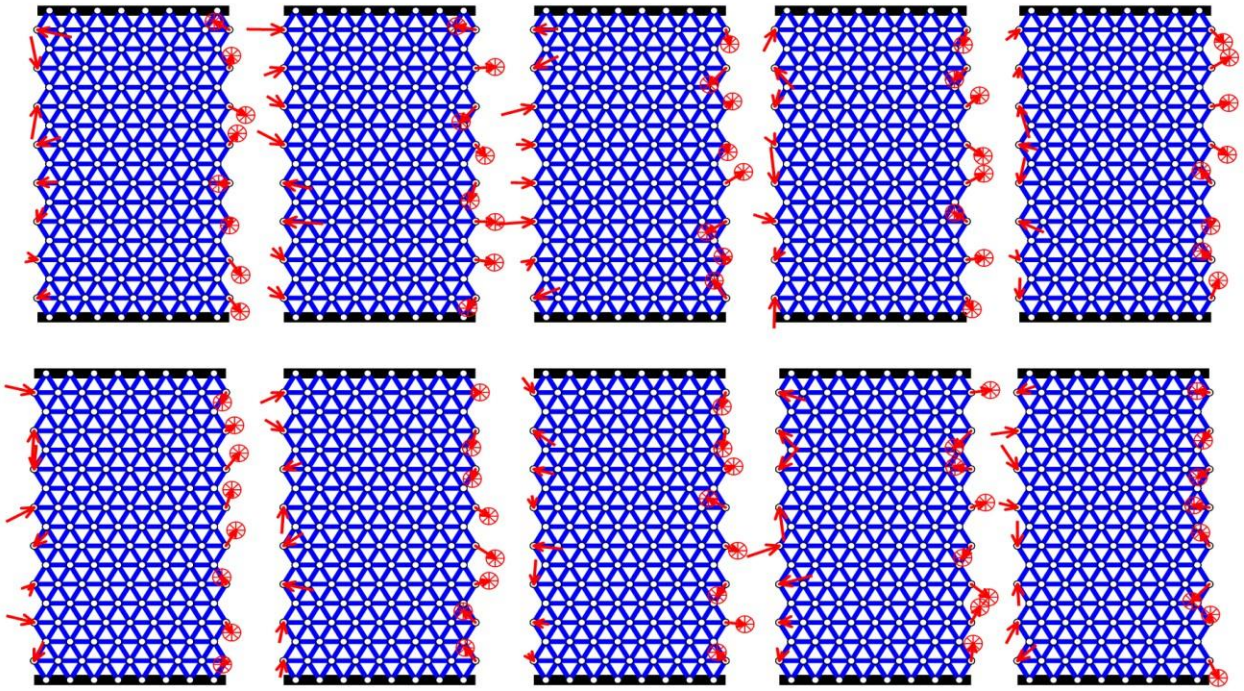

Figure 2.11 Ten example behaviors randomly generated for 8-layer deep mechanical neural networks (MNNs) to learn within the first simulation study of Fig. 5A. Arrows applied to the input nodes on the left side of each lattice are desired forces and arrows pointing from the output nodes to target locations on the right side of each lattice are the corresponding desired displacements.

The MNNs of the second simulation study were all configured as triangular lattices (e.g., Fig. 2.1B) with tunable beams that were assigned axial stiffness values between 4 N/mm and -2 N/mm. Learning was simulated using different numbers of layers and different numbers of output nodes (note that the number of output nodes is equal to the number input nodes). Regardless of the scenario, each MNN attempted to learned the same two behaviors shown in Fig. 2.1, D and E, except that the amplitudes of both behaviors' sinusoidal contours on which their target displacements lie were set to 2.5 mm and the shearing input-node forces of the

second behavior pushed downward instead of upward with a magnitude of 1 N. Each scenario was attempted 15 times and the MSE of the simulation that produced the lowest final value was plotted in Fig. 5B.

The MNNs of the third simulation study had 8 input and 8 output nodes. Their tunable beams were assigned axial stiffness values between 4 N/mm and -2 N/mm. Learning was simulated using both triangular and square lattices (Fig. 5C) that learn different numbers of random behaviors with two, four, and eight layers. Random behaviors were generated the same way they were in the first simulated study. Once sufficiently different behaviors were generated, three additional unique sets of different behaviors were generated for each scenario. The simulated MNN then attempted to simultaneously learn each unique set of behaviors four times and the final MSE of the attempt that yielded the lowest value was averaged with the lowest final MSEs generated by learning the other unique sets of behaviors. The resulting MSE average was plotted in Fig. 5C for the different triangular and square lattice scenarios (shown as green and red lines respectively in Fig. 5C). Dotted lines correspond to two layers, dashed lines correspond to four layers, and solid lines correspond to eight layers.

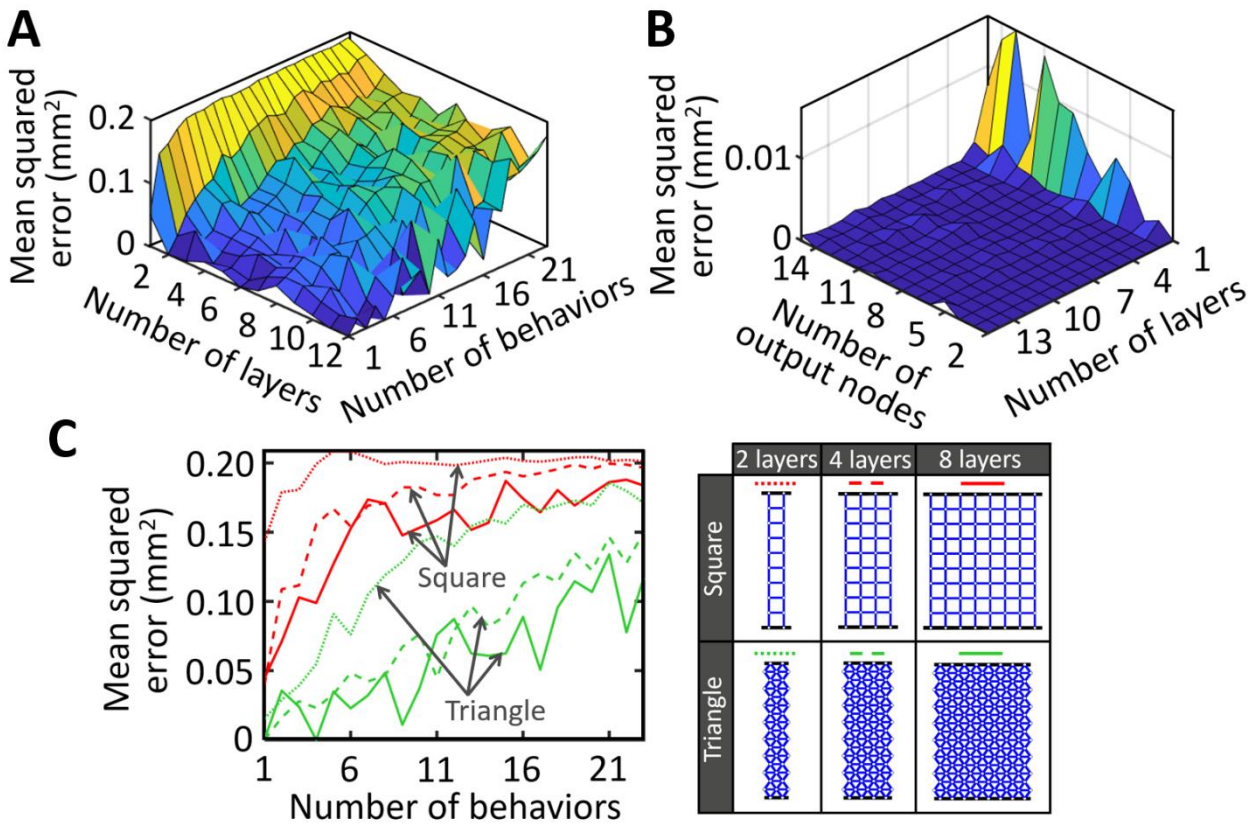

Figure 2.12 Simulation study results. The lowest mean squared error (MSE) achieved when (**A**) mechanical neural networks (MNNs) with different numbers of layers learn different numbers of random behaviors, (**B**) MNNs of different numbers of layers and output nodes learn the same two behaviors, and (**C**) MNNs of different configurations (i.e., triangular and square) learn different numbers of random behaviors.

2.4 Discussion

The concept of MNNs as architected materials that learn behaviors was introduced and experimentally demonstrated using two optimization algorithms—GA and PPS (Fig. 4, B and C). Although the GA proved to be more than 41 times slower than PPS (i.e., GA required 111.13 hours whereas PPS required 2.68 hours), the MNN of Fig. 3 learned its behaviors 10.5 times more accurately using the GA (i.e., GA achieved a MSE of 0.006 mm^2 whereas PPS achieved 0.063 mm^2). An experimental study was also conducted to compare the learning capabilities of MNNs consisting of tunable beams that are controlled to exhibit linear versus nonlinear stiffness. Figure 4D indicates that MNNs with linear-stiffness beams learn with greater accuracy than

MNNs with nonlinear-stiffness beams (i.e., the lowest linear and nonlinear MSE was 0.056 mm^2 and 0.093 mm^2 respectively). A computational tool was also created to simulate the effect of lattice size, behavior number, and packing configuration on MNN learning. The plots of Fig. 5, A and C, demonstrate that the more layers a MNN possesses and the fewer random behaviors it's tasked to simultaneously learn, the more accurately it can learn (i.e., the lower its final MSE can become). The plot of Fig. 5B demonstrates that as long as a MNN is 3 or more layers deep, it possesses enough tunable beams to accurately learn two shape-morphing behaviors regardless of the number of layers and output nodes. Note that although a MNN with fewer output nodes has fewer tunable beams with which to learn, it also has fewer force-input and displacement-output requirements for the beams to satisfy during learning. Thus, the number of output nodes is largely irrelevant. The plot of Fig. 5C demonstrates that triangular lattices can learn more accurately than square lattices because more tunable beams constitute triangular lattices than square lattices given the same number of layers and output nodes. Moreover, the beams of triangular lattices can more effectively propagate displacements in all directions rather than predominantly along orthogonal directions like square lattices.

2.5 Materials and Methods

2.5.1 Tunable beam fabrication and function

A photo of a tunable beam used within the MNN of this work is provided in Fig. 2.3A. Its parts are shown disassembled in Fig. 2.3B. It consists of a Moticont linear voice coil motor (LCVM-032-025-02) actuator and two HBM strain gauge (1-LM13-1.5/350GE) sensors, which are mounted on opposing sides near the base of one of the flexure bearings to enhance sensor sensitivity via a Wheatstone circuit in a half-bridge configuration. The rest of the parts were either cut using wire electrical discharge machining (EDM) from 6061-T6 aluminum, or, in the

case of the brackets, were machined from the same material. The two parallel blade flexures behave as linear bearings in that they guide a translational motion along the beam's axis while constraining all other directions. As the flexure bearings deform over their full range, however, they manifest a slight arching parasitic motion, which was considered in the selection and mounting of the voice coil actuator. Care was taken to make sure that the coil portion of the actuator could never make contact with or rub against the outer magnet portion of the actuator so that noise, friction, and hysteresis would be avoided. The brackets were mounted to the beam's body via bolts and a hard stop (labeled in Fig. 2.3A) was cut into the body to prevent the flexure bearings from yielding by preventing them from deforming beyond ±2.5 mm in either direction. The beam's body can attach to the modular node parts, labeled in Fig. S1B, via slide-on dove-tail joints, which are then locked in place by opposing wedges that are pressed together. The resulting joint effectively fuses the beam's body to the modular-node parts, thus preventing slip-induced friction and hysteresis while also allowing the body to be disassembled and reassembled quickly for debugging or calibration purposes. The utility of this feature is more clearly recognized in the context of the full MNN lattice. Each modular-node part uses two angled blade flexures to permit rotational deformations about the axis where the planes of the blade flexures intersect (i.e., at the center of the small cylinder shown) while constraining deformations in all other directions. A hard stop (Fig. 2.3B) is used to prevent excessive rotational deformations that would yield the blade flexures. Note that although the tunable beams used to demonstrate the concept of MNNs in this work (Fig. 2.3A) were designed such that only their axial stiffness could be changed, beams that can have their stiffness independently tuned along multiple directions (e.g., axial, transverse, and bending) would likely enhance MNN learning further.

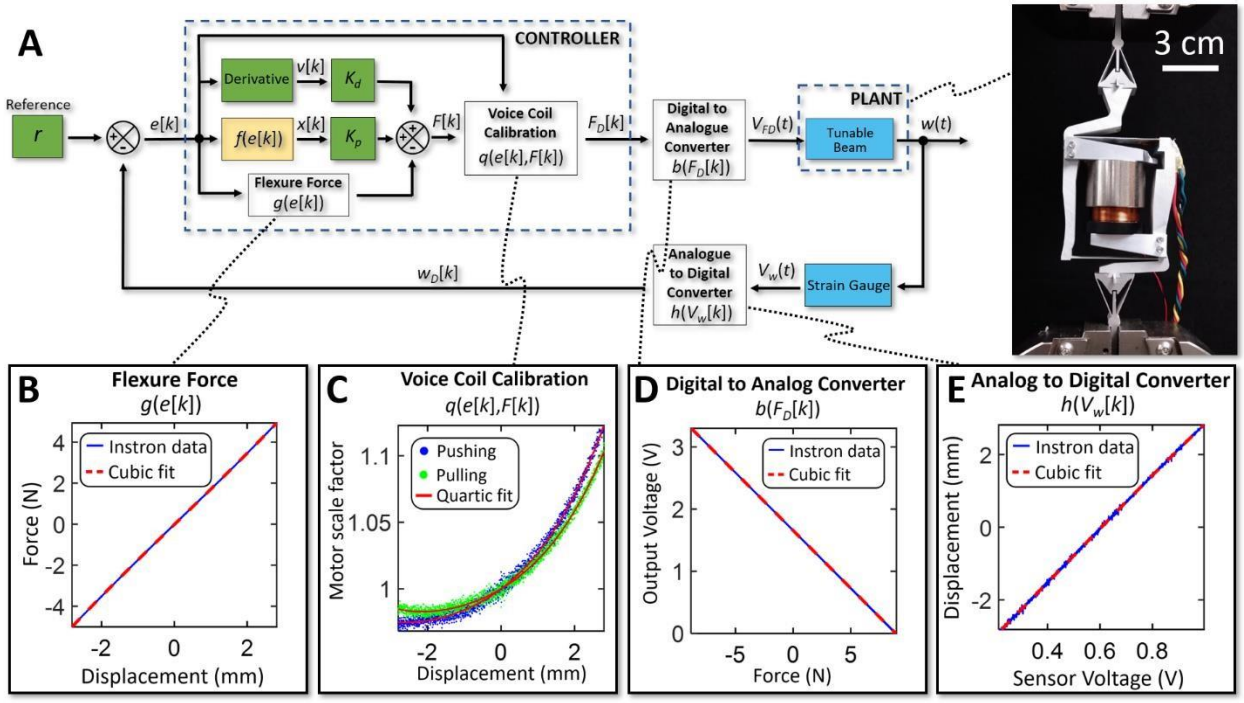

Figure 2.13 Each tunable beam uses proportional-derivative (PD) control to achieve prescribed axial force-displacement responses. An example beam's (A) closed-loop control diagram shown with its four measured calibration plots—(B) flexure force, (C) voice coil calibration, (**D**) digital to analog converter, and (**E**) analog to digital converter.

2.5.2 Tunable beam closed-loop controller

The closed-loop control diagram detailing how each beam within the MNN of this work uses proportional-derivative (PD) control to achieve tunable axial stiffness is provided in Fig. 2.13A. The digital displacement signal, $e[k]$, is the difference between a reference offset value, r, and the digital displacement feedback signal, $w_D[k]$. The derivative of $e[k]$ is a velocity signal, $v[k]$, which is multiplied by the controller's derivative gain, K_d, which behaves as a damping coefficient. For the purposes of this work, K_d was set to 650. The function, $f(e[k])$, can be set to determine the profile of the tunable beam's force-displacement response. Note that if $f(e[k])$ is set equal to $e[k]$, the beam's force-displacement response will be linear, but if it is set equal to $\tan(e[k])$, it will be a nonlinear tangent function. The output of $f(e[k])$, labeled $x[k]$ in Fig. 2.13A,

is multiplied by the controller's proportional gain, K_p. This proportional gain is set to equal the instantaneous axial stiffness of the beam (i.e., the stiffness of the beam before it is deformed). Note that the K_p values corresponding to each tunable beam within a MNN lattice are the variables that are adjusted during the learning process. Four calibration plots must be generated for each tunable beam in the lattice so that analytical functions can be fit to the measured data collected from an Instron testing machine and used within the control diagram. An example of the first calibration plot is provided in Fig. 2.13B. This plot, called flexure force $g(e[k])$, relates the extension or contraction of the tunable beam along its axis to the force required to deform the beam without control (i.e., the force-displacement response of the passive flexure bearings, labeled in Fig. 2.3B). The force, $F[k]$, (Fig. 2.13A) represents the required voice-coil output force to control the beam's axial stiffness as desired. Both this force and $e[k]$ are fed into the second calibration plot (Fig. 2.13C), called voice coil calibration $q(e[k], F[k])$, to generate a force, $F_D[k]$, which corrects for the nonlinearity of the voice coil actuator by multiplying $F[k]$ with a motor scale factor. The sign of $F[k]$ determines whether the pushing or pulling analytical fit function is used. The third calibration plot (Fig. 2.13D), called digital to analog converter $b(F_D[k])$, converts $F_D[k]$ into a voltage, $V_{FD}(t)$, which is fed to the voice coil actuator within the system's plant (i.e., the tunable beam). The beam responds by displacing an amount, $w(t)$, which causes the strain gauge sensors to produce a voltage, $V_w(t)$, which is then converted into $w_D[k]$ by the fourth calibration plot (Fig. 2.13E), called analog to digital converter $h(V_w[k])$.

2.5.3 MNN features, fabrication, and control electronics

The MNN of Fig. 3B consists of 21 tunable beams, which were joined together at nodes (colored purple in Fig. 3A). Each node consists of blade flexures, which permit rotational deformations at each node's center and thereby allow the MNN's lattice to freely deform as it is

loaded. The MNN's two input nodes (Fig. 3B) are each loaded by a pair of voice coil actuators that collectively allow their corresponding input node to be loaded with a force that points in any direction within the lattice's plane. These actuators are fixtured within decoupling flexures (shown green in Fig. 3A) that enable each input node to displace appreciable amounts without imparting transverse jamming loads on the actuators themselves. Hard stops are provided to prevent any flexure within the lattice from yielding as they deform. Two pairs of grounded nodes (Fig. 3A) are fixtured to a frame along the top and bottom of the two-layer deep MNN as shown in Fig. 3B. Two cameras are mounted on the same frame to directly measure the displacement of the two output nodes by tracking pins inserted at their center. Black felt is used to contrast the white color of the pin heads so that they stand out (Fig. 3B).

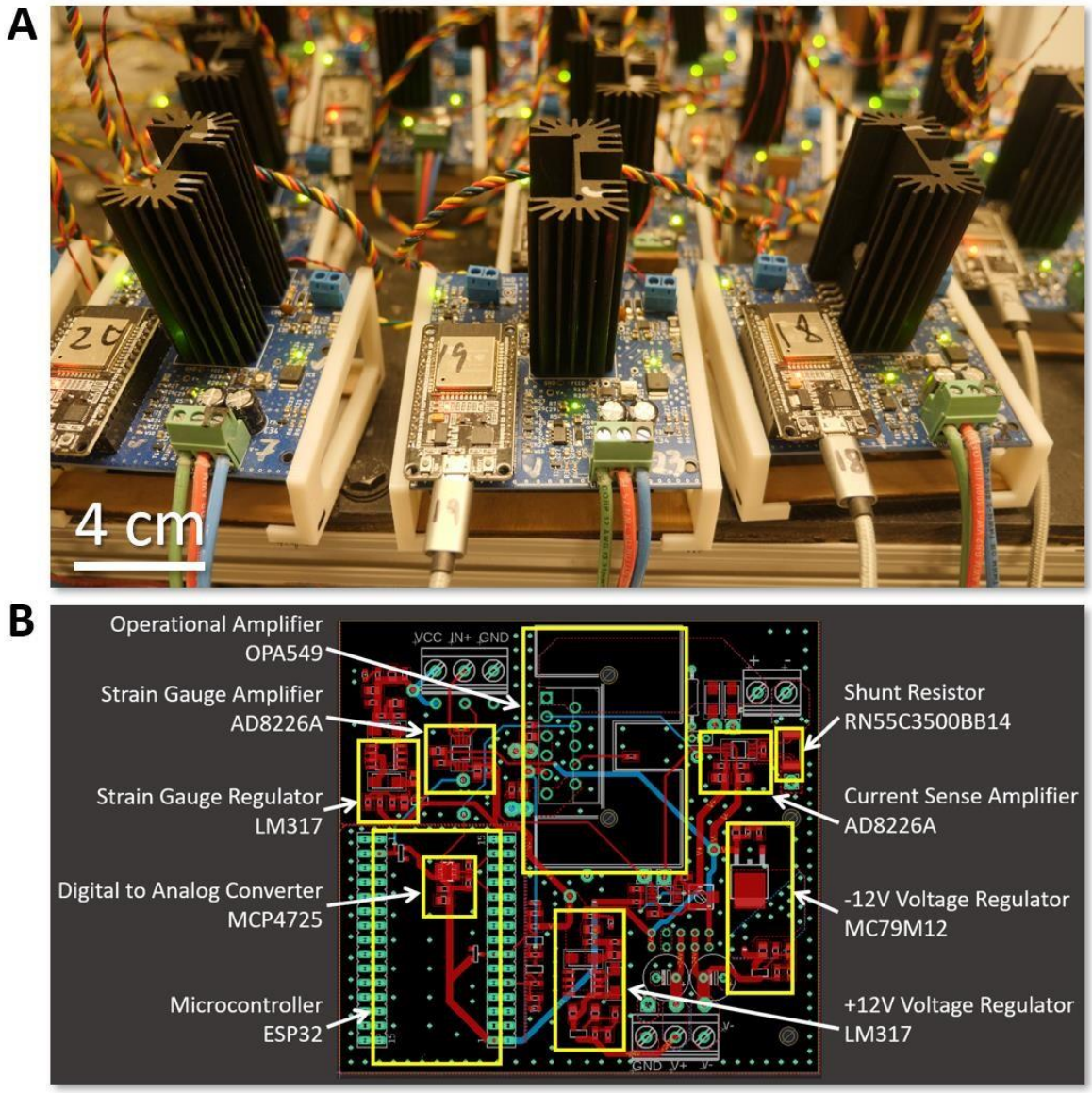

Figure 2.14 Electronics used to control the tunable beams of the mechanical neural network (MNN). (**A**) Photograph and (**B**) labeled schematic of the electronics used to control each tunable beam and input actuator within the MNN lattice of Fig. 3B.

In addition to the tunable beam parts (Fig. 2.3B) discussed previously, the MNN of Fig. 3B consists of other parts, which were also cut from 6061-T6 aluminum using wire EDM (i.e., the nodes, grounded nodes, and decoupling flexures labeled in Fig. 3A). The four input actuators fixtured within the decoupling flexures are Moticont linear voice coil motors (LVCM-038-038-02) and the two cameras (i.e., Adafruit 636 Digital Microscropes) were mounted to an 80/20 T-

slot aluminum frame using parts additively fabricated from Acrylonitrile Butadiene Styrene (ABS) with a Stratasys UPrint SE Plus three-dimensional (3D) printer. Wooden boards were used to support the electronics underneath the MNN.

A closeup photo of the electronics used to control the MNN of Fig. 3B is provided in Fig. 2.14A and a schematic of the circuit used to drive each tunable beam (i.e., all 21) and each input actuator (i.e., all 4) within the MNN is labeled in Fig. 2.14B. Within the circuit, which is current-controlled, a Microchip Technology MCP4725 digital to analog converter (DAC) produces a voltage proportional to the desired actuator current, which is supplied to the noninverting input of a Texas Instruments OPA549 operational amplifier. The OPA549 operates in an arrangement similar to a voltage follower and its output current passes through both the actuator and a Vishay RN55C3500BB14 shunt resistor before reaching ground. The voltage drop across the shunt resistor is amplified by an Analog Devices AD8226A instrumentation amplifier, which acts as a current-sense amplifier. The output of the AD8226A is in the same range as the DAC output, which is provided to the inverting input of the OPA549 operational amplifier for closed-loop control. To measure the displacement of the tunable beams, the circuit board has another AD8226A instrumentation amplifier, which acts as a strain gauge amplifier. An Espressif ESP32 microcontroller is used to set the DAC input voltage, read the strain gauge voltage, and shut down the OPA549. Stable supply voltages for the analog components are created using a Texas Instruments LM317 voltage regulator for the +12 supply and an ON Semiconductor MC79M12 voltage regulator for the -12V supply. Another LM317 voltage regulator is used for the strain gauge supply.

2.5.4 Mechanical neural network (MNN) calibration

Since each tunable beam must be assembled within the mechanical neural network (MNN) of Fig. 2.4, A and B after its four calibration plots (e.g., Fig. 2.13, B to E) are generated using an Instron testing machine, the MNN's two output nodes tend to displace arbitrary amounts every time different combinations of axial stiffness values are simply assigned to the tunable beams. Ideally, the output nodes of the MNN should only displace in response to its input nodes being loaded during learning—not simply in response to new stiffness combinations being assigned to the tunable beams. The unwanted output-node displacements resulting from the assignment of axial stiffness values occur because each tunable beam is slightly stretched or compressed when it is assembled within the lattice to a different length than how it was when it was calibrated in the Instron. Thus, the reference offset value, r, labeled in Fig. 2.13A must be adjusted for each tunable beam to reduce the unwanted displacements, which, if severe enough, can adversely affect learning. MNN calibration is the process of adjusting this offset value for each beam in a lattice so that the MNN can successfully learn.

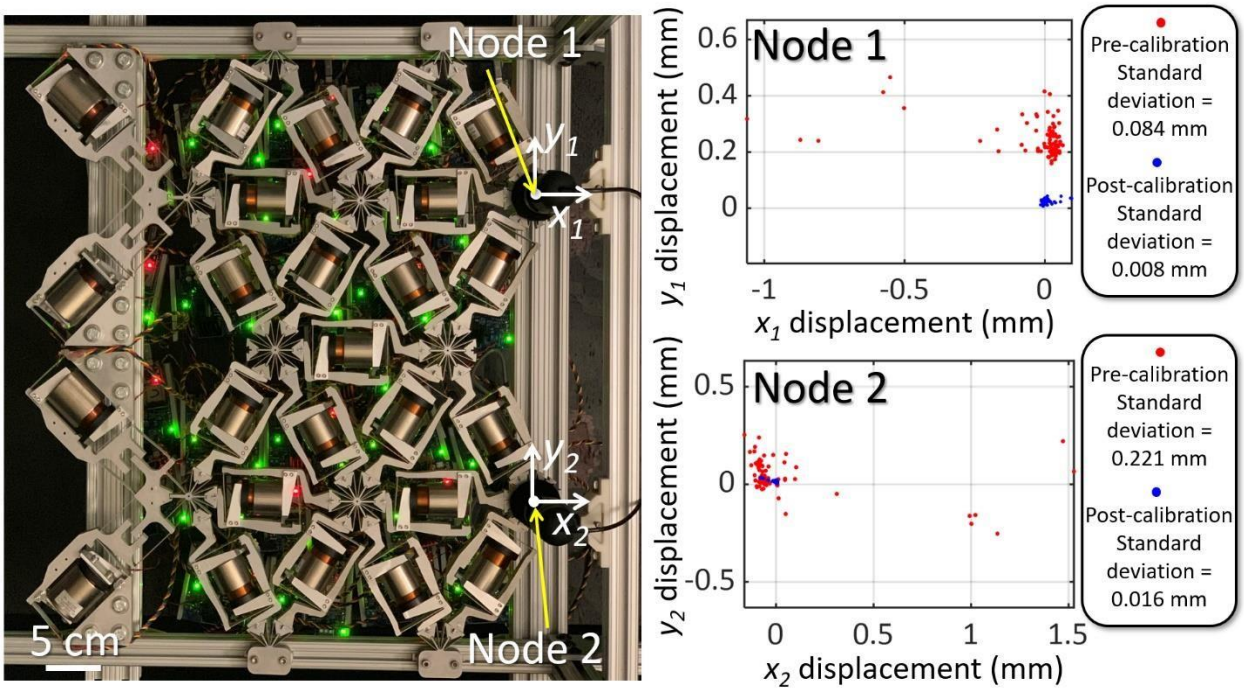

Figure 2.15 Pre- and post-calibration comparison of the mechanical neural network's (MNN's) unwanted output-node displacements resulting from assigned combinations of axial stiffness values. Eighty random but different combinations of axial stiffness values were assigned to the 21 tunable beams within the MNN of Fig. 3B and the resulting displacements of the output nodes (i.e., Node 1 and Node 2) are plotted before (red dots) and after (blue dots) calibrating each individual beam. Standard deviations of these unwanted displacements are also provided.

The output-node displacements (i.e., the displacements of Node 1 and Node 2 labeled in Fig. 2.15) resulting from the assignment of 80 random but different axial stiffness combinations to the tunable beams in the MNN of Fig. 2.4, A and B were measured and plotted before and after MNN calibration was performed as shown by the red and blue dots in the plots of Fig. 2.15 respectively. Note that the standard deviation (i.e., the spread) of the displacements improved by at least an order of magnitude after the reference offset values were calibrated (i.e., the standard deviation of Node 1's displacements dropped from 0.084 mm to 0.008 mm and the standard deviation of Node 2's displacements dropped from 0.221 mm to 0.016 mm). Moreover, note that the postcalibration blue dots cluster much more successfully around the plot origins compared with the pre-calibration red dots (i.e., the average post-calibration x_1 displacement and y_1

displacement of Node 1 are 0.0047 mm and 0.0228 mm respectively, and the average post-calibration x_2 displacement and y_2 displacement of Node 2 are -0.0029 mm and 0.0145 mm respectively, whereas the average pre-calibration x_1 displacement and y_1 displacement of Node 1 are -0.0456 mm and 0.2483 mm respectively, and the average pre-calibration x_2 displacement and y_2 displacement of Node 2 are 0.0414 mm and 0.0408 mm respectively). This observation indicates that, on average, the output nodes of a properly calibrated MNN don't displace significantly from their resting positions regardless of what combinations of axial stiffnesses values are assigned to the lattice's tunable beams.

2.5.5 Validation of strain-gauge approach using cameras

The cameras shown in Fig. 3B were used to validate the strain-gauge approach, which indirectly measures MNN output-node displacements by calculating them from the collective strain-gauge measurements of each tunable beam in the lattice as they simultaneously deform when the MNN input nodes are loaded. A pair of forces with randomly generated x- and y-axis components between ±1 N were applied to the two input nodes of the MNN of Fig. 3B after a random combination of axial stiffness values were assigned to the tunable beams. The resulting x1- and y1-axis displacements of the upper output node, labeled Node 1 in Fig. 2.15, are plotted over time in Fig. 2.16A as they are measured using both the camera and the strain-gauge approach. The resulting x2- and y2-axis displacements of the lower output node, labeled Node 2 in Fig. 2.15, are also plotted over time in Fig. 2.16B as they are measured using both the camera and the straingauge approach. Note that since the input forces are step functions, both output

nodes vibrate until they settle on a steady-state value. The steady-state displacements of Node 1 (Fig. 2.16A) measured by the camera along the x_1- and y_1-axis of Fig. 2.15 are x_{1c} and y_{1c}

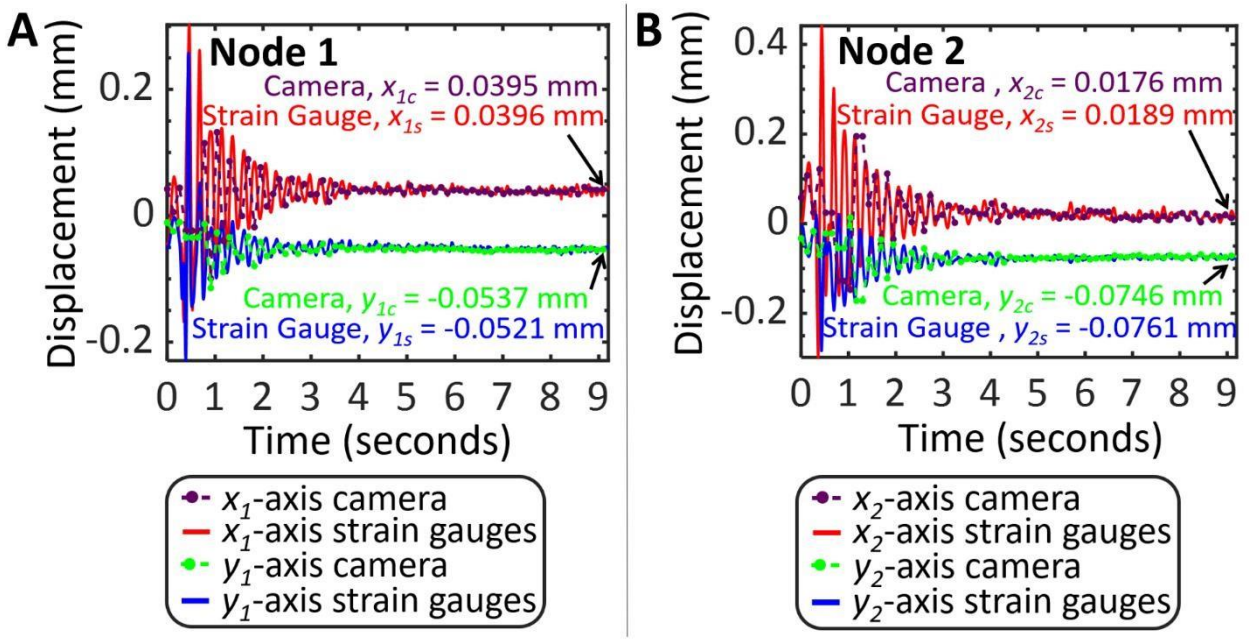

Figure 2.16 Camera validation of the strain-gauge approach for sensing the output nodes of the mechanical neural network (MNN). The *x*- and *y*-component displacements of the MNN's two output nodes, (**A**) Node 1 and (**B**) Node 2, plotted over time as measured directly by the cameras and indirectly by the tunable beams' strain gauges in response to a random pair of step forces imparted on each of the MNN's two input nodes.

respectively. The steady-state displacements of Node 1 measured by the strain-gauge approach along the x_1- and y_1-axis of Fig. 2.15 are x_{1s} and y_{1s} respectively. The steady-state displacements of Node 2 (Fig. 2.16B) measured by the camera along the x_2- and y_2-axis of Fig. 2.15 are x_{2c} and y_{2c} respectively. The steady-state displacements of Node 2 measured by the strain-gauge approach along the x_2- and y_2-axis of Fig. 2.15 are x_{2s} and y_{2s} respectively. A similar procedure was conducted 25 times using 25 different input force loads with randomly generated *x*- and *y*-axis components between ±1 N and 25 random but different combinations of axial stiffness values assigned to the tunable beams within the MNN. The resulting output-node steady-state displacements were measured and used to calculate 25 different scalar difference values, *E*, according to

$$E = \sum_{i=1}^{2} \sqrt{(x_{is} - x_{ic})^2 + (y_{is} - y_{ic})^2}. \qquad \text{(S1)}$$

The average and standard deviation of all 25 scalar difference values, *E*, corresponding to each of the 25 MNN loading tests were calculated to be 0.0074 mm and 0.003 mm respectively. These small values validate the strain-gauge approach because they demonstrate that the differences between the strain-gauge approach and the direct-measurement approach of the cameras are insignificant. Moreover, note how closely the strain-gauge data of Fig. 2.16, A and B matches the results of the camera data. The cameras don't appear to track the initial vibrations of the output nodes well because they only collected 7.5 frames per second at their required resolution. Thus, if the cameras themselves were used to train the MNN, the controller would need to wait upwards of 20 seconds per loading scenario so that the resulting dynamic vibrations would fully settle out to accurately record the final locations of the output nodes. Thus, the strain-gauge approach is not only sufficiently accurate but is also significantly faster. It enables MNNs to learn dynamic behaviors (e.g., wave propagation control) and predict the location where vibrating nodes will settle beforehand to dramatically reduce learning time. The strain-gauge approach was consequently applied to all experimental learning studies conducted for this work.

2.5.6 Optimization algorithm details

Optimization algorithms determine how combinations of stiffness values should be assigned to the tunable beams within a MNN for each loading scenario during the learning process. This work employed two optimization algorithms to train the MNN of Fig. 4A such that it learned the two shape-morphing behaviors detailed previously. The two optimization algorithms used were a genetic algorithm (GA) and partial pattern search (PPS).

The GA used for this work attempts 1,000 combinations of axial stiffness values per generation. The most promising combinations (i.e., those that were measured having the lowest mean squared error (MSE)) from each generation are then crossed according to MATLAB's 'ga' function to generate a new generation of 1,000 new combinations of axial stiffness values. The best combination of axial stiffness values (i.e., the one that is measured having the lowest MSE) from each generation is plotted and corresponds with each blue dot in the upper-most plot of Fig. 4B. The algorithm continues until new generations fail to produce combinations of axial stiffness values with lower MSEs at which point the algorithm terminates. Note that the upper-most plot of Fig. 4B resulted from 40 generations. Although the GA used for this work requires significant time and computational power to complete, the algorithm is very thorough and thus produces accurate results.

The PPS algorithm used for this work begins with all the tunable beams starting with the same stiffness value (i.e., 1.15 N/mm). A beam is randomly selected and its currently assigned stiffness value is added to and subtracted from a stiffness increment, which begins at 2.15 N/mm. If the two resulting combinations of stiffness values don't reduce the measured MSE, a different beam is randomly selected and the same process is repeated. If all the beams in the MNN are subjected to this process and the MSE never reduced for any of them, the current stiffness increment is multiplied by a reduction factor of 0.9 and the entire process repeats with the new, now smaller, stiffness increment. If adding or subtracting the stiffness increment to the current stiffness value assigned to any beam ever exceeds or falls below the stiffness limit achievable by the beam (i.e., 2.3 N/mm and -2 N/mm respectively according to Fig. 2C), the beam is assigned the stiffness limit that was surpassed. When a combination of stiffness values is identified that produces a measured reduction in the MNN's MSE, the entire process begins again until the

current stiffness increment is reduced below a specified threshold (i.e., 0.5 N/mm). Note from the upper-most plot of 4C that each blue dot corresponds to an event where the MNN's MSE was measured as being reduced, which for the specific learning example of Fig. 4C occurred 10 times until the algorithm terminated. Although PPS produces results that are not as accurate as the GA used for this work, it requires significantly less time and computational power.

Note that despite the fact that both algorithms are designed to identify combinations of stiffness values that produce progressively lower MSEs, the MSEs corresponding to some of the blue dots in the upper-most plots of Fig. 4, B and C increase in value compared to prior dots. These temporary increases in plotted MSE values are a result of system noise in the MNN (e.g., sensor noise). Finally note that before Nodes 1 and 2 are ever displaced during learning using either algorithm, both output nodes begin at the origin of the middle and lower-most plots of Fig. 4, B and C.

2.5.7 Computational tool assumptions

The computational tool, used to perform the simulation study of this work, assumes that its simulated beams exhibit the same characteristics as the physical beam of Fig. 2A so that the study's results accurately predict the capabilities of a MNN similar to the design of Fig. 3B but that consists of many more of its tunable beams. Specifically, the tool assumes that its simulated beams are the same length as the physical beam of Fig. 2A (i.e., 6 inches from node to node).

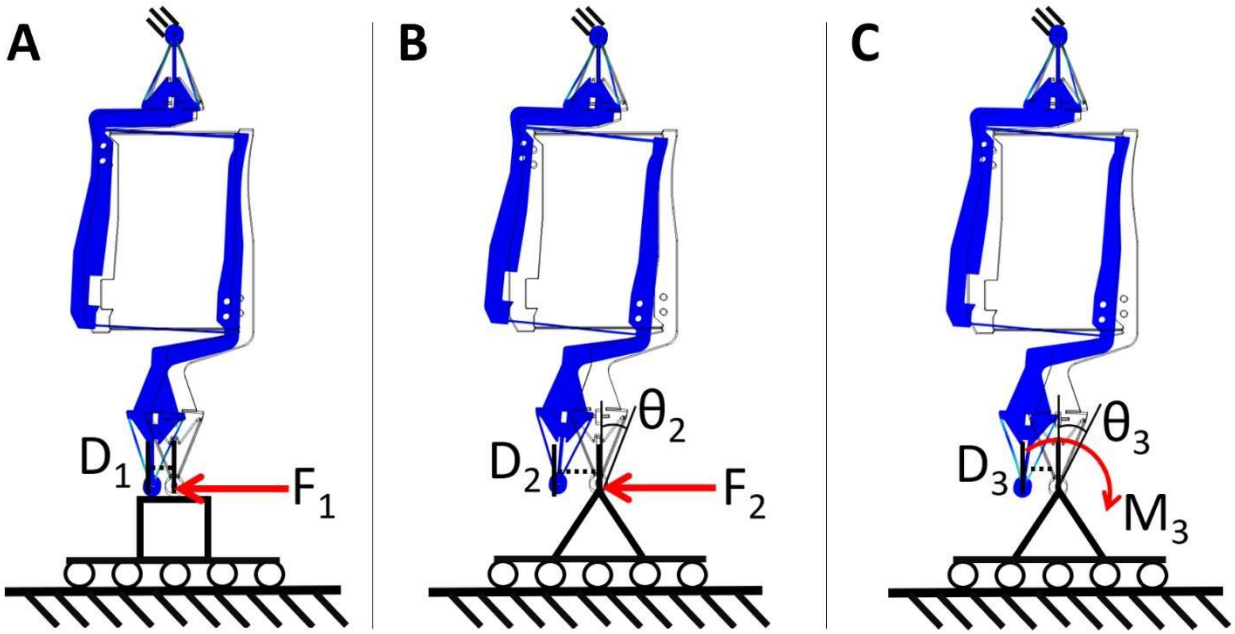

Figure 2.17 Finite element analysis (FEA) used to calculate the passive stiffness values of the tunable beam in Fig. 2.3A along its nonaxial directions. (**A**) Fixed sliding with force loading, (**B**) pinned sliding with force loading, and (**C**) pinned sliding with moment loading scenarios used to calculate the beam's passive stiffness values along nonaxial directions. These values were used to inform the computational tool that generated the simulation studies of this work.

The tool mimics the physical beam's hard stops, labeled in Fig. 2.1A, by preventing its simulated beams from being able to extend or contract beyond ±2.5 mm in either direction. It assumes that the in-plane passive stiffness values of each beam along nonaxial directions are also the same as the physical beam fabricated in Fig. 2A. These values were calculated using finite element analysis (FEA). Computer-aided-design (CAD) models of the body of each beam were fixtured and loaded according to the conditions specified in Fig. 2.17, A to C, using 6061-T6 aluminum properties to calculate three passive stiffness values required for the computational tool. The first stiffness value, K_1, was calculated from the displacement, D_1, resulting from a shearing force, F_1, imparted on the beam with one end fixed and the other end fixed to a sliding prism joint (Fig. 2.17A) according to $K_1=F_1/D_1$, which was found to be 0.2371 N/mm. The second stiffness value, K_2, was calculated from the displacement, D_2, and the bending angle, θ_2,

resulting from a shearing force, F_2, imparted on the beam with one end fixed and the other end pinned with a revolute joint to a sliding prism joint (Fig. 2.17B) according to $K_2=(F_2-D_2K_1)/\theta_2$, which was found to be 2.11 N/rad. The third stiffness value, K_3, was calculated from the displacement, D_3, and the bending angle, θ_3, resulting from a moment, M_3, imparted on the beam with one end fixed and the other end pinned with a revolute joint to a sliding prism joint (Fig. 2.17C) according to $K_3=(M_3-D_3K_2)/\theta_3$, which was found to be 25,300 Nmm/rad. The tool also assumes that these passive stiffness values remain constant over large deformations to avoid the computational cost of nonlinear considerations. It also assumes that axial stiffness values can be assigned to the beams with linear force-displacement responses between any prescribed maximum and minimum value (including negative stiffness values) to mimic the actively controlled beams of Fig. 2A. The tool assumes that all lattice nodes along the top and bottom of every simulated MNN are fixed to grounded rigid bodies (e.g., Fig. 2.1B) and that all the simulated MNNs are twodimensional (i.e., their tunable beams are all constrained to lie on a common plane). The tool simulates MNN learning by using the approach described in the main text, but rather than using a GA or PPS to assign combinations of axial stiffness values to its tunable beams during learning, the computational tool uses a MATLAB optimization algorithm, called ‘fmincon.’ The ‘fmincon’ algorithm is a powerful gradient-based approach that leverages derivatives to rapidly minimize a desired value (e.g., MSE), and thus, the algorithm can be applied to simulated MNNs, which are modeled using stiffness equations, but are difficult to apply to physical MNNs, which don’t inherently possess equations to accurately differentiate and are subject to noise.

The computational tool also assumes that the principle of force scaling is applied when simulating MNN learning so that the best version of any behavior (i.e., the one that achieves the

lowest MSE) can be identified regardless of how many layers (Fig. 2.1B) constitute the MNN. The shape-morphing behaviors discussed in this work are characterized by forces, which are applied to input nodes that map to desired output node displacements with specified directions and magnitudes. Although the directions and relative magnitudes of the input forces are also important in achieving a desired behavior, MNNs with different numbers of layers would need their input force vectors to be multiplied by an optimal scaling factor to allow the MNN to best achieve the behavior with the smallest possible MSE for fair comparison. MNNs with fewer layers are inherently more compliant than MNNs with many layers and thus fewer-layer MNNs would require smaller force-magnitude scaling factors so that their input forces don't deform the entire MNN far beyond the desired output node displacements of the intended behavior. Moreover, MNNs with many layers would require larger force-magnitude scaling factors than MNNs with fewer layers because the input forces of deep-layer MNNs must be sufficiently large to transmit through the many layers to displace the output nodes at all let alone with the displacements necessary to achieve the intended behavior. Since the computational tool assumes stiffness linearity as discussed previously, the optimal force-magnitude scale factor can be directly solved analytically for any desired input-force-to-output-displacement behavior applied to any simulated MNN lattice assigned any combination of beam stiffness values. Thus, as the computational tool simulates MNN learning, optimal scale factors are calculated and multiplied by the desired behavior's input forces every time a new combination of axial stiffness values is assigned to the tunable beams of the MNN to achieve the lowest MSE for any scenario.

Note that although the principle of force scaling could be applied to physical MNNs to achieve reduced MSEs, force scaling was not applied to the experimental study of Fig. 4 since that study did not compare MNNs of different sizes (i.e., different numbers of layers or input and

output nodes). Moreover, reasonable input-force magnitudes were selected that could achieve the output-displacement magnitudes of the desired behaviors being learned and the same force magnitudes were applied across comparative studies. If the principle of force scaling were ever applied to physical MNNs, a force-magnitude scale factor would need to be multiplied by the input actuators' signals and swept over a range of values until an optimal value is identified that achieves the lowest MSE for any scenario. Such a sweep would be important since calculating the optimal scale factor directly is difficult for physical scenarios. Thus, applying force scaling to physical MNNs would require substantially more time to learn behaviors.

2.5.8 Computational tool verification

The computational tool used to generate the simulation study of this work was verified using FEA. The passive nonaxial stiffness values (i.e., K_1, K_2, and K_3, defined previously) of every tunable beam used within the computational tool's simulation studies were informed via FEA performed on the tunable beam design of Fig. 2.3A without its voice-coil actuator or brackets (Fig. 2.17) as discussed previously. The passive axial stiffness value of the same beam design (i.e., its axial stiffness without active control) was also calculated using FEA for 6061-T6 aluminum properties and was found to be 1.81 N/mm. This value was also used to inform the computational tool's tunable beams for the purposes of the tool's verification. A CAD model of the 21-beam MNN lattice of Fig. 4A without its voice coil actuators or brackets is shown in Fig. 2.18A. FEA was performed on this model using linear deformations and linear material properties of 6061-T6 aluminum to computationally compare various loading conditions with the same loading conditions applied to the same 21-beam lattice simulated by the computational tool. Twenty-five different force combination attempts, each with *x*- and *y*-components that were

selected randomly between ±1 N, were applied to the two input nodes and the resulting displacements of the two output nodes (i.e., Node 1 and Node 2) were calculated and plotted in Fig. 2.18, B to E using both FEA and the computational tool of this work. Note that x_1 and y_1 are the displacements of Node 1 as measured relative to the origin of x_1 and y_1, labeled in Fig. 2.18A, which is located where Node 1 is prior to lattice deformation. Additionally note that x_2 and y_2 are the displacements of Node 2 as measured relative to the origin of x_2 and y_2, labeled in Fig. 2.18A, which is located where Node 2 is prior to lattice deformation. The first force combination attempt shown in the plots of Fig. 2.18, B to E correspond to the FEA results shown in Fig. 2.18A. The fact that the results of the 25 different force combination attempts generated by both the computational tool and FEA correspond well verifies the computational tool's ability to accurately predict the response of general lattice configurations and sizes when subjected to general loading scenarios.

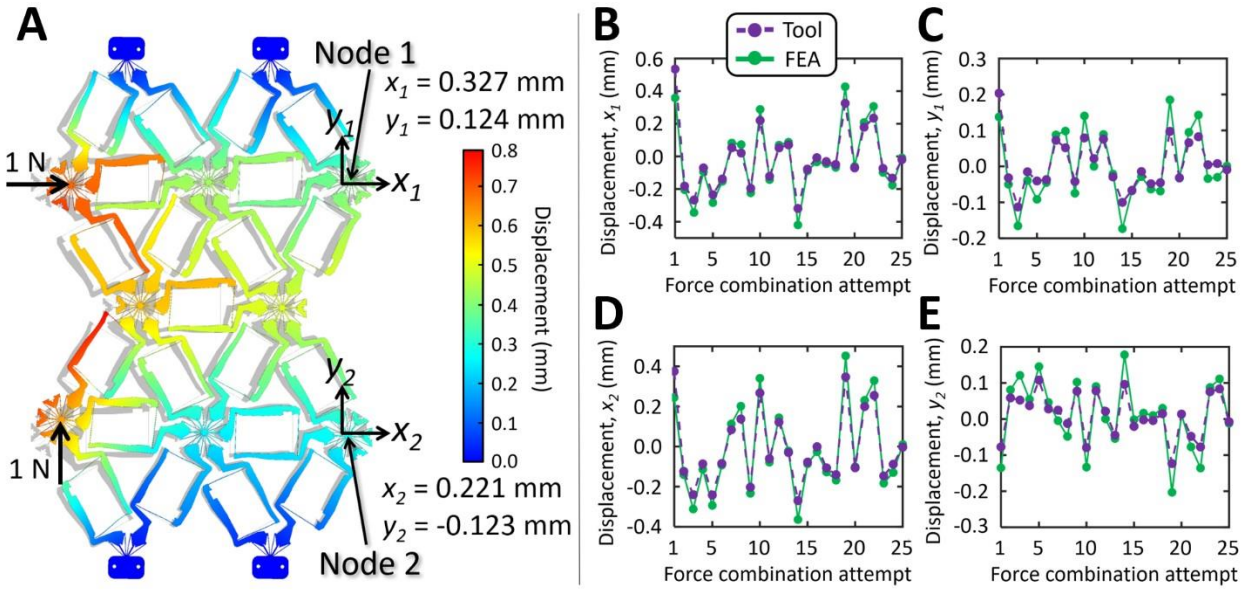

Figure 2.18 Computational tool verification using finite element analysis (FEA). (**A**) FEA results of a 21-beam lattice being loaded with a force combination attempt that involved loading the lattice's top input node with a horizontal force of 1 N and loading its bottom input node with a vertical force of 1 N. (**B**) Node 1's *x*-component displacements, (**C**) Node 1's *y*-component displacements, (**D**) Node 2's *x*-component displacements, and (**E**) Node 2's *y*-component displacements that result from 25 random but different force combination attempts calculated using FEA (green) and using the computational tool (purple) for comparison.

2.5.9 How the computational tool generated the example of Fig. 2.1, D and E

The computational tool generated the learning results of the MNN lattice shown in Fig. 2.1, D and E. The amplitudes of both behaviors' sinusoidal contours on which their target displacements lie were set to 2 mm. With a final scaled force magnitude of 0.5 N applied to every input force of both behaviors, Solution 1 (Fig. 2.1D) and Solution 2 (Fig. 2.1E) achieved both behaviors with a MSE of 0.0047 mm^2 and 0.0008 mm^2 respectively. Note that the nodal displacements of both solutions are all shown with an exaggeration factor of 25 to visually enhance the lattice's behavior. Moreover, note that although the lattice's tunable beams were subjected to axial, shearing, and bending deformations, which were taken into account during the calculations performed by the computational tool, the deformed beams shown in Fig. 2.1, D and E are graphically depicted as overly simplified straight lines, which directly join the final node locations together.

2.6 References

[1] LeCun, Y., Bengio, Y., and Hinton, G., 2015, "Deep Learning," Nature, **521**(7553), pp. 436–444.

[2] Tait, A. N., de Lima, T. F., Zhou, E., Wu, A. X., Nahmias, M. A., Shastri, B. J., and Prucnal, P. R., 2017, "Neuromorphic Photonic Networks Using Silicon Photonic Weight Banks," Sci Rep, **7**(1), p. 7430.

[3] Gao, H., Cheng, B., Wang, J., Li, K., Zhao, J., and Li, D., 2018, "Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment," IEEE Transactions on Industrial Informatics, **14**(9), pp. 4224–4231.

[4] McCulloch, W. S., and Pitts, W., 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, **5**(4), pp. 115–133.

[5] Rosenblatt, F., 1958, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psychological Review, **65**(6), pp. 386–408.

[6] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., and Kiru, M. U., 2019, "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," IEEE Access, **7**, pp. 158820–158846.

[7] Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks Are Universal Approximators," Neural Networks, **2**(5), pp. 359–366.

[8] Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B., 2015, "Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors," Nature, **521**(7550), pp. 61–64.

[9] Zhang, H.-T., Park, T. J., Islam, A. N. M. N., Tran, D. S. J., Manna, S., Wang, Q., Mondal, S., Yu, H., Banik, S., Cheng, S., Zhou, H., Gamage, S., Mahapatra, S., Zhu, Y., Abate, Y., Jiang, N., Sankaranarayanan, S. K. R. S., Sengupta, A., Teuscher, C., and Ramanathan, S., 2022, "Reconfigurable Perovskite Nickelate Electronics for Artificial Intelligence," Science, **375**(6580), pp. 533–539.

[10] Lee, S., Kim, H., Lee, S.-T., Park, B.-G., and Lee, J.-H., 2022, "SiO2 Fin-Based Flash Synaptic Cells in AND Array Architecture for Binary Neural Networks," IEEE Electron Device Letters, **43**(1), pp. 142–145.

[11] Han, R., Huang, P., Xiang, Y., Liu, C., Dong, Z., Su, Z., Liu, Y., Liu, L., Liu, X., and Kang, J., 2019, "A Novel Convolution Computing Paradigm Based on NOR Flash Array With High Computing Speed and Energy Efficiency," IEEE Transactions on Circuits and Systems I: Regular Papers, **66**(5), pp. 1692–1703.

[12] Dillavou, S., Stern, M., Liu, A. J., and Durian, D. J., 2022, "Demonstration of Decentralized, Physics-Driven Learning."

[13] Shen, Y., Harris, N. C., Skirlo, S., Prabhu, M., Baehr-Jones, T., Hochberg, M., Sun, X., Zhao, S., Larochelle, H., Englund, D., and Soljačić, M., 2017, "Deep Learning with Coherent Nanophotonic Circuits," Nature Photon, **11**(7), pp. 441–446

[14] Zhang, H., Gu, M., Jiang, X. D., Thompson, J., Cai, H., Paesani, S., Santagati, R., Laing, A., Zhang, Y., Yung, M. H., Shi, Y. Z., Muhammad, F. K., Lo, G. Q., Luo, X. S., Dong, B., Kwong, D. L., Kwek, L. C., and Liu, A. Q., 2021, "An Optical Neural Chip for Implementing Complex-Valued Neural Network," Nat Commun, **12**(1), p. 457.

[15] Lin, X., Rivenson, Y., Yardimci, N. T., Veli, M., Luo, Y., Jarrahi, M., and Ozcan, A., 2018, "All-Optical Machine Learning Using Diffractive Deep Neural Networks," Science, **361**(6406), pp. 1004–1008.

[16] Wu, Z., Zhou, M., Khoram, E., Liu, B., and Yu, Z., 2020, "Neuromorphic Metasurface," Photon. Res., PRJ, **8**(1), pp. 46–50.

[17] Furuhata, G., Niiyama, T., and Sunada, S., 2021, "Physical Deep Learning Based on Optimal Control of Dynamical Systems," Phys. Rev. Applied, **15**(3), p. 034092.

[18] Füchslin, R. M., Dzyakanchuk, A., Flumini, D., Hauser, H., Hunt, K. J., Luchsinger, R. H., Reller, B., Scheidegger, S., and Walker, R., 2013, "Morphological Computation and Morphological Control: Steps Toward a Formal Theory and Applications," Artificial Life, **19**(1), pp. 9–34.

[19] Boyd, S., and Chua, L., 1985, "Fading Memory and the Problem of Approximating Nonlinear Operators with Volterra Series," IEEE Transactions on Circuits and Systems, **32**(11), pp. 1150–1161.

[20] Maass, W., Natschläger, T., and Markram, H., 2002, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations," Neural Computation, **14**(11), pp. 2531–2560.

[21] Hauser H, Ijspeert A J, Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W., 2011, "Towards a Theoretical Foundation for Morphological Computation with Compliant Bodies," Biol Cybern, **105**(5), pp. 355–370.

[22] Coulombe, J. C., York, M. C. A., and Sylvestre, J., 2017, "Computing with Networks of Nonlinear Mechanical Oscillators," PLOS ONE, **12**(6), p. e0178663.

[23] Caluwaerts, K., D'Haene, M., Verstraeten, D., and Schrauwen, B., 2013, "Locomotion Without a Brain: Physical Reservoir Computing in Tensegrity Structures," Artificial Life, **19**(1), pp. 35–66.

[24] Caluwaerts, K., Despraz, J., Işçen, A., Sabelhaus, A. P., Bruce, J., Schrauwen, B., and SunSpiral, V., 2014, "Design and Control of Compliant Tensegrity Robots through Simulation and Hardware Validation," Journal of The Royal Society Interface, **11**(98), p. 20140520.

[25] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W., 2012, "The Role of Feedback in Morphological Computation with Compliant Bodies," Biol Cybern, **106**(10), pp. 595–613.

[26] Nakajima, K., Hauser, H., Li, T., and Pfeifer, R., 2015, "Information Processing via Physical Soft Body," Sci Rep, **5**(1), p. 10487.

[27] Hermans, M., Burm, M., Van Vaerenbergh, T., Dambre, J., and Bienstman, P., 2015, "Trainable Hardware for Dynamical Computing Using Error Backpropagation through Physical Media," Nat Commun, **6**(1), p. 6729.

[28] Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., and McMahon, P. L., 2022, "Deep Physical Neural Networks Trained with Backpropagation," Nature, **601**(7894), pp. 549–555.

[29] Schaedler, T. A., and Carter, W. B., 2016, "Architected Cellular Materials," Annual Review of Materials Research, **46**(1), pp. 187–210.

[30] Zangeneh-Nejad, F., Sounas, D. L., Alù, A., and Fleury, R., 2021, "Analogue Computing with Metamaterials," Nat Rev Mater, **6**(3), pp. 207–225.

[31] Zuo, S., Wei, Q., Tian, Y., Cheng, Y., and Liu, X., 2018, "Acoustic Analog Computing System Based on Labyrinthine Metasurfaces," Sci Rep, **8**(1), p. 10103.

[32] Hughes, T. W., Williamson, I. A. D., Minkov, M., and Fan, S., "Wave Physics as an Analog Recurrent Neural Network," Science Advances, **5**(12), p. eaay6946.

[33] Stern, M., Hexner, D., Rocks, J. W., and Liu, A. J., 2021, "Supervised Learning in Physical Networks: From Machine Learning to Learning Machines," Phys. Rev. X, **11**(2), p. 021045.

[34] Stern, M., Arinze, C., Perez, L., Palmer, S. E., and Murugan, A., 2020, "Supervised Learning through Physical Changes in a Mechanical System," Proceedings of the National Academy of Sciences, **117**(26), pp. 14843–14850.

[35] Brown, E., Rodenberg, N., Amend, J., Mozeika, A., Steltz, E., Zakin, M. R., Lipson, H., and Jaeger, H. M., 2010, "Universal Robotic Gripper Based on the Jamming of Granular Material," Proceedings of the National Academy of Sciences, **107**(44), pp. 18809–18814.

[36] Poon, R., and Hopkins, J. B., 2019, "Phase-Changing Metamaterial Capable of Variable Stiffness and Shape Morphing," Advanced Engineering Materials, **21**(12), p. 1900802.

[37] Kuppens, P. R., Bessa, M. A., Herder, J. L., and Hopkins, J. B., 2021, "Monolithic Binary Stiffness Building Blocks for Mechanical Digital Machines," Extreme Mechanics Letters, **42**, p. 101120.

[38] Wei, K., Bai, Q., Meng, G., and Ye, L., 2011, "Vibration Characteristics of Electrorheological Elastomer Sandwich Beams," Smart Mater. Struct., **20**(5), p. 055012.

[39] Blanc, L., Delchambre, A., and Lambert, P., 2017, "Flexible Medical Devices: Review of Controllable Stiffness Solutions," Actuators, **6**(3), p. 23.

[40] Whitley, D., 2001, "An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls," Information and Software Technology, **43**(14), pp. 817–831.

[41] Dolan, E. D., Lewis, R. M., and Torczon, V., 2003, "On the Local Convergence of Pattern Search," SIAM J. Optim., **14**(2), pp. 567–583.

CHAPTER 3. COMPARING MECHANICAL NEURAL-NETWORK LEARNING ALGORITHMS

3.1 Abstract

The purpose of this work is to compare learning algorithms to identify which is the fastest and most accurate for training mechanical neural networks (MNN)s. MNNs are a unique class of lattice-based artificial-intelligence (AI) architected materials that learn their mechanical behaviors with repeated exposure to external loads. They can learn multiple behaviors simultaneously in-situ and re-learn desired behaviors after being damaged or cut into new shapes. MNNs learn by tuning the stiffnesses of their constituent beams similarly to how artificial neural networks (ANN)s learn by tuning their weights. In this work, we compare the performance of six algorithms (i.e., genetic algorithm, full pattern search, partial pattern search, interior point, sequential quadratic progression, and Nelder-Mead) applied to MNN leaning. A computational model was created to simulate MNN learning using these algorithms with experimentally measured noise included. 3,900 runs were simulated. The results were validated using experimentally collected data from a physical MNN. We identify algorithms like Nelder-Mead that are both fast and able to reject noise. Additionally, we provide insights into selecting learning algorithms based on the desired balance between accuracy and speed, as well as the general characteristics that are favorable for training MNNs. These insights will promote more efficient MNN learning and will provide a foundation for future algorithm development.

3.2 Introduction

Artificial intelligence (AI) has enabled the automation of many complex tasks [1-3] through the extraordinary ability of learning. Although artificial neural networks (ANN)s were

developed in the 1950s [4,5], and had rich potential as universal approximators [6], ANNs were not initially adopted broadly. They proved challenging and computationally expensive to train even for the best algorithms of their time. The modern recurrence of ANN-based solutions is largely due to the application of new techniques such as backpropagation [7], which allow ANNs to use more powerful gradient-based learning algorithms to efficiently adjust their weights during the learning process [8, 9].

More recently, physical neural networks (PNNs) [10-13] have been developed due to advantages such as high-power efficiency [14], simple digital to analog conversion [15], and the ability to learn physical properties [16]. Despite these advantages, PNNs are not currently used broadly due largely to the same problems that initially faced ANNs when they were first proposed (i.e., PNNs lack effective techniques and algorithms to efficiently train them). In physical systems, gradients are not always easily calculable, making gradient-based algorithms hard to use. When gradient methods are applicable, they often come at the expense of future adaptability [17] or simplicity by requiring masking signals to control unwanted system dynamics [15]. Other techniques avoid using a gradient such as the No-Prop algorithm [18], which enables learning from in-situ oscillating systems [19-21] but does so computationally rather than by changing the system's physical properties. Physics-driven learning (PDL) is another gradient-free approach for physical learning that can quickly train physical systems with AI abilities [16,22] by gradually modifying a system through external perturbations towards desired states. However, PDL requires the ability to externally manipulate the system's outputs [23] to function.

Most recently, mechanical neural networks (MNNs) were proposed [24] as PNN architected materials that learn their behaviors and properties by tuning the stiffness of their

constituent beams similarly to how an ANN tunes its weights. This framework allows a single architected material to modify its physical properties in-situ and learn a wide variety of behaviors simultaneously using simple gradient-free algorithms. Although the work of Lee et al. [24] demonstrates MNN learning, the speed and accuracy of the algorithms used to guide the process of determining a working combination of beam stiffness values that achieved desired behaviors were not sufficiently practical.

In this paper, six learning algorithms are compared using both simulation and experimentation using the same physical MNN presented in Lee et al. [24] to identify the fastest and most accurate algorithm in general for making MNNs a practical reality. The sensitivity of these six algorithms to system noise is also compared. This study provides insights into the process for selecting learning algorithms based on the desired balance between the accuracy and speed of learning general behaviors using MNNs as well as insights regarding the general characteristics that are favorable for training such MNNs.

3.2.1 Mechanical neural network (MNN) learning approach

This section describes how MNNs learn desired behaviors. As mentioned previously, MNNs are lattices consisting of beams with adjustable axial stiffness. The adjustable beams used in Lee et al. [24] (shown in Fig. 3.1a) are electromechanical compliant mechanisms that use voice coil actuators, flexure bearings, and strain gauge sensors to adjust their axial stiffness via closed-loop control. The beams are assembled within a regular triangular lattice and join together at nodes, which are composed of rotary flexures (Fig. 3.1a). These flexures allow the lattice to accommodate deformations as they extend and contract along their axes. To load the MNN, pairs of voice coil actuators connected to a set of decoupling flexures (Fig. 3.1b) are

placed on each of the lattice's input nodes (Fig. 3.1c), which allow the actuators to independently apply forces to the input nodes.

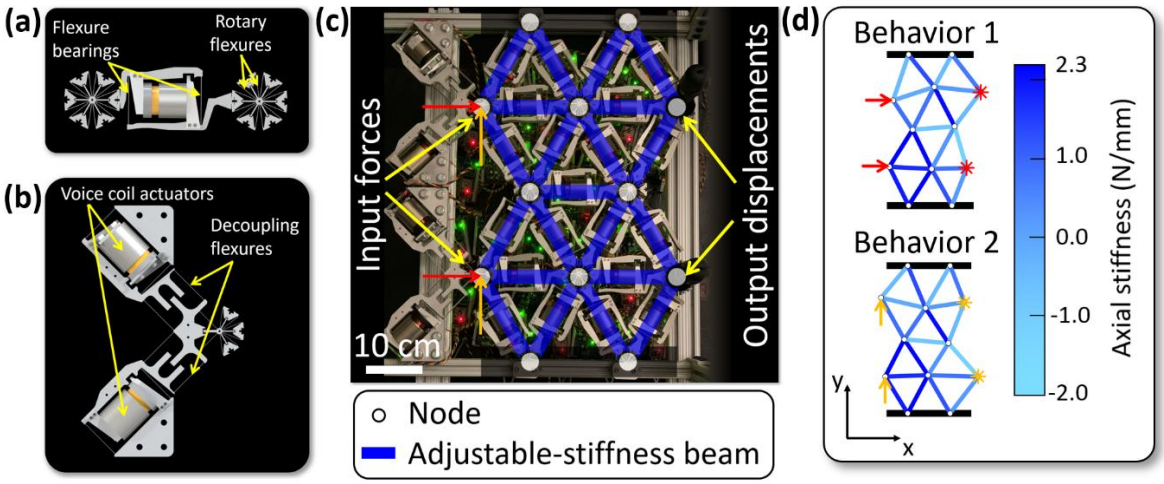

Figure 3.1 (a) The tunable stiffness beams that constitute the mechanical neural network (MNN) of this study. (b) The actuators and decoupling flexures that enable the input nodes of the MNN to be driven in any in-plane direction. (c) The fabricated MNN of this study shown with blue lines drawn on top of each tunable beam. (d) Two shape-morphing behaviors that the simulated MNN achieved by finding a working combination of axial stiffness values using a learning algorithm.

The 21 beams within the physical MNN of Fig. 3.1c are shown as simplified blue beam lines in the lattice. The MNN has two input nodes on its left side and two output nodes on its right side. Shape morphing behaviors are achieved when the lattice's output nodes displace to desired target displacements in response to its input nodes being loaded by desired input forces. The lattice learns its behaviors simultaneously by first assigning each beam in the lattice with random stiffness values. The lattice is then loaded with the desired input forces of each behavior and its resulting output-node displacements are then subtracted from the desired target displacements of the corresponding behavior. A mean-squared error (MSE) is then calculated by averaging the square of these difference values for all the behaviors simultaneously. A learning

algorithm then assigns a new combination of axial stiffness values to the beams of the lattice and the process repeats in an effort to minimize the calculated MSE until the desired behaviors are all simultaneously achieved.

A computational tool used this approach to simulate the learning of the two example behaviors shown in Fig. 3.1d. The first behavior was achieved when the lattice's top output node displaced 0.25 mm to the right and when its bottom output node displaced 0.25 mm to the left in response to its input nodes being loaded to the right with the same force magnitude. The second behavior was achieved when the lattice's top output node displaced 0.25 mm to the left and when its bottom output node displaced 0.25 mm to the right in response to its input nodes being loaded up with the same force magnitude. For visual clarity, the lattices' displacements shown in Fig. 3.1d were multiplied 100 fold. Note from Fig. 3.1d that the learning algorithm used (i.e., Sequential Quadratic Programming) successfully identified a combination of axial stiffness values, depicted as different shades of blue, that enabled the lattice to simultaneously achieve both behaviors with an impressively small MSE of 1.4^{-4} mm^2.

This paper compares six of the most promising learning algorithms applied to MNNs to identify which can most rapidly identify a combination of beam axial stiffness values that enable their lattices to achieve desired behaviors as accurately as possible. Here learning accuracy is measured using MSE. The lower the algorithm can make the MNNs final MSE, the more accurately the MNN learns its desired behaviors. Learning speed is measured using the number of iterations that the algorithm requires to arrive at the final MSE. Here one iteration is achieved each time an MSE is calculated during the learning process. Iterations roughly scale with learning time and are thus a fair way to compare the speed of different algorithms that are applied with different time scales (e.g., in simulation and experimental learning scenarios).

3.3 Learning Algorithms

This section describes the six learning algorithms that were compared for the MNN studies of this paper. The approach of each algorithm is briefly summarized and the hyperparameters that were used for this study are also provided.

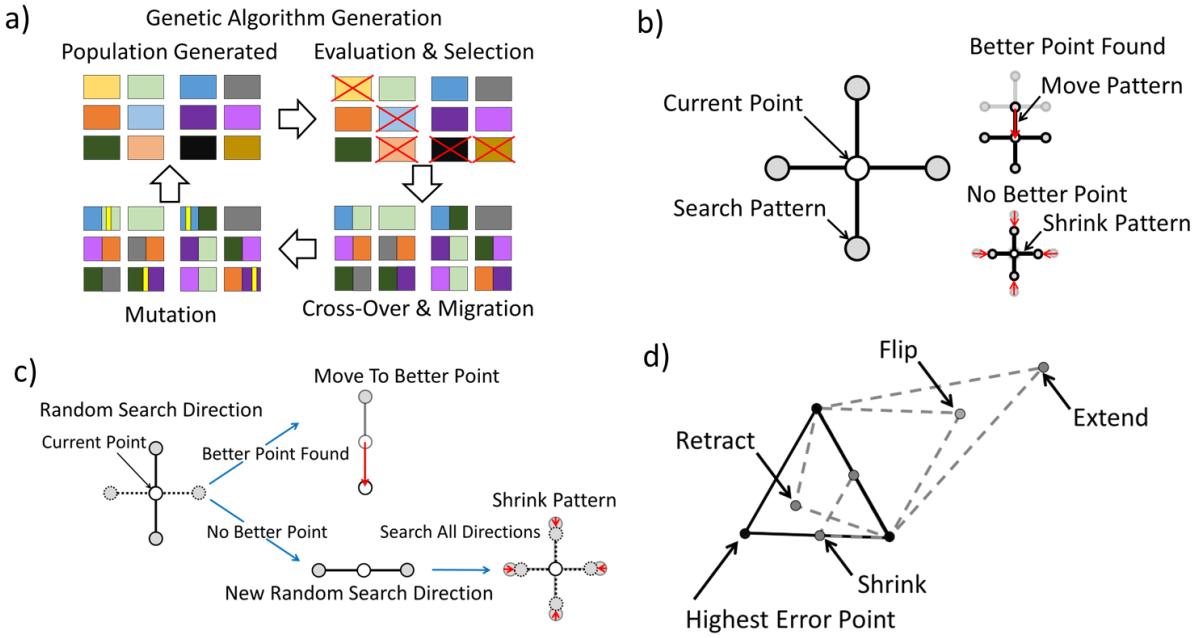

Figure 3.2 Update cycles for the a) Genetic Algorithm, b) Full Pattern Search, c) Partial Pattern Search, and d) Nelder-Mead algorithms.

3.3.1 Genetic Algorithm (GA)

The genetic algorithm (GA) optimizes systems using a process similar to natural evolution [25]. As illustrated in Fig 3.2a, every GA has a set of candidate configurations, called a population. Once the population is generated, the error for each configuration is evaluated, and a subset of the population is selected to create a new generation of candidates favoring members with low error to keep favorable traits in the population. The new population is generated with some combination of copying members (migration), exchanging traits between individuals (cross-over), and random changes (mutations) [26,27].

The GA implementation in this paper, uses the default functions and hyperparameters for migration, cross-over, and mutation created by Matlab. The population size is set to 500 members, and the initial population is generated randomly for each attempt with axial-stiffness varying from -2 N/mm to 2.3 N/mm. The algorithm converges when the minimum error is stagnant for 50 generations.

3.3.2 Full Pattern (FP)

The pattern search algorithm refers to a class of search-based optimization algorithms conceived in Hooke et al. [28]. This paper will refer to the pattern search algorithm as the full pattern (FP) search to distinguish it from the partial pattern search used in earlier MNN work by Lee et al. [24].

FP attempts to reduce the number of function evaluations needed to find a minimum [29] by sampling a set of points around a given pattern center, instead of searching in every direction. Points in the search pattern, as illustrated in Figure 3.2b, correspond to positive or negative deltas for the independent variables [30]. If one of these points has an error value that is lower than the pattern center, the center is moved to the best point. If none of the new points produces a decrease in error, then the delta is decreased. FP repeats these two steps until the delta reaches a minimum threshold, where the algorithm settles to a minimum. From this initial inception, augmented Lagrangian evaluations [31] and more complicated search patterns [32] have improved the efficiency of FP.

The implementation used in this paper follows the more basic structure of FP described above, but its functionality is tweaked to include additional polling methods [33]. The

implementation for this paper uses the general hyperparameters from Matlab but enforces a minimum pattern size of $5x10^{-2}$ N/mm.

3.3.4 Partial Pattern (PP)

Partial pattern (PP) search is the name given to the algorithm described in Lee et al. [24]. It uses a modified pattern search algorithm, as illustrated in Fig. 3.2c, PP has three main features separating it from a simple pattern search. The first modification is in the polling method used. Instead of searching the entire patten before selecting a new center point, PP evaluates the pattern in a random order and moves to a new center as soon as a point with lower error is found. The other two modifications work to stabilize the algorithm for noisy error values. PP performs multiple iterations for a given point, which decreases the speed of its readings but increasing their precision. PP also periodically reevaluates the error of the pattern center to ensure that pattern comparisons are accurate. As in the FP algorithm, the PP shrinks its pattern once if no lower points exist in the pattern.

For this paper, PP starts with a pattern width of 2.15 N/m truncating any values that extend beyond the valid stiffness combinations (-2 N/mm to 2.3 N/mm) and the pattern is shrunk to 90% of its initial value if no better error point exists in the pattern.

3.3.5 Interior Point (IP)

Interior point (IP) refers to a class of algorithms that optimize constrained systems by implementing logarithmic barrier functions in the Lagrangian. IP methods are used by Jarre et al. [34] in the optimization of truss configurations to respond to a given load. IP improves upon the Lagrange multiplier method as formalized by Fiaco et al. [35] for constrained optimization, where the objective function and its domain are subject to a set of mathematical constraints [36].

$$\text{Equation 4.1: } L(x, \lambda, \mu) = f(x) + \lambda g(x)$$

For Lagrange multipliers (Eq. 4.1), the Lagrangian function acts as a summation of the objective function, $f(x)$ with each of the equality constraints, $g(x)$, scaled by Lagrange multipliers λ. These variables create a system of equations with an equal number of equations and unknowns, thus allowing for a solution. Using this formulation, any given objective function can be minimized while satisfying any set of mathematical constraints by finding a stationary point for the Lagrangian and ensuring its concavity [37].

The IP algorithm modifies the Lagrangian into a logarithmic barrier function, B, with logarithmic terms, $\log c_i(x)$, to capture each inequality constraint for the problem (Eq. 4.2). In this formulation, $L(x, \lambda)$ is the original formulation of the Lagrangian.

$$\text{Equation 4.2: } B(x, \lambda, \mu) = L(x, \lambda) + \mu \sum \log(\, c_i(x)\,) \quad where\ c_i(x) \geq 0$$

These barrier functions can decrease computational cost for computing Jacobians and Hessian matrices, reducing computational time by speeding up the evaluation of each iteration [38].

For computational efficiency, this paper uses Matlab's implementation of IP. The Matlab IP adds a merit function with the logarithmic barrier function as well as linearized constraints to increase computation speed. The matrix used in the Lagrangian is constructed to include additional second-order parameters to ensure that the first-derivative terms converge towards a minimum. Each step of the algorithm consists of a symmetrized matrix inversion [39], eventually converging to a stationary point. The general purpose hyperparameters determined by Matlab are used in this paper.

3.3.6 Sequential Quadratic (SQ)

Sequential quadratic (SQ) programming, like IP, is an optimization algorithm that improves upon the traditional method of Lagrange multipliers for constrained optimization problems (Eq. 4.3) by operating on second-order Taylor-series approximations centered around the desired solution, $f(x_k)$, which is updated every iteration. This is done to simplify computation to obtain stable convergence [40]. The second order approximation method is effective for large-scale nonlinear optimization, where matrix operations are a rate-limiting step, and where linear approximations converge slowly. The formulation for this process is shown in Eq. 4.3. The term d describes the relative error between consecutive iterations of the algorithm and it becomes zero when x_k settles to the final solution. The term μ_k and the function $h(x_k)$ respectively describe Lagrange multipliers and functions for inequality constraints.

$$\text{Equation 4.3: } Q(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx}[L(x_k, \lambda_k) + \mu_k h(x_k)]d$$

This paper uses Matlab's implementation of SQ programming, which uses a positive definite Hessian matrix as the quadratic coefficient; this parameter, similarly to the Lagrange multiplier terms, is updated at every iteration to ensure fast computation until a stationary point is reached [39]. No hyperparameters are changed from the standard implementation suggested in Matlab.

3.3.7 Nelder-Mead (NM)

Developed by Nelder and Mead, the Nelder-Mead (NM) algorithm uses the properties of simplex elements to locate minima for a given objective function [41]. Simplices are an extension of triangles to varying dimensional space (e.g., lines in one dimension, triangles in two dimensions, and tetrahedrons in three dimensions). The method takes the form of a search

algorithm that compares the relative values of a simplex element's vertices, to move and deform the shape, until it settles to a minimum with as few function evaluations as possible [42]. The movements involved in this process are shown in Fig. 3.2d. The first transformation used is flipping, where the simplex is flipped over the face opposite the vertex with the highest error, a 'worst point.' If the flipped vertex is not the new best point, it is extended further from the flipping edge to search for a lower point. If these two transformations do not find a better point, then the point is retracted closer to the face used as a reference for the reflection. Finally, if the simplex achieves no improvements with the prior operations, all vertices are shrunk inwards. The NM algorithm repeats these transformations to create a converging series that eventually reaches a local minimum for a given constrained problem. Work by Marandi et al. shows use of the Nelder-Mead algorithm to optimize sensor networks data collection [43].

This paper uses the Nelder-Mead simplex algorithm that follows the sequence of transformations as stated above to search for a minimum [44]. The NM used for this paper has an error tolerance of 0.01 mm^2 and a minimum error difference of 0.005 N/mm, the remaining hyperparameters use the generalized values determined by Matlab.

3.4 Simulation Studies

This section provides the results of this paper's simulation studies, which compare the learning speed and accuracy produced by the MNN of Fig. 3.1c when the six algorithms provided in Section 2 are applied during the learning process. The simulation tool introduced in Lee et al. [24] was used for 21 beams arranged in a triangular lattice to simulate the MNN of Fig. 3.1c. The nodes along the top and bottom of the lattice were held fixed as shown in Fig. 3.1d. The beams' length (152.4 mm), range of tunable axial stiffness (-2 N/mm to 2.3 N/mm), allowable axial displacement (±2.5 mm), and off-axis passive stiffness values (provided in Lee et

al. [24]) were all set to mimic those of the fabricated MNN in Fig. 3.1c. The beams were assumed to be linear and the principle of force scale was applied as described in Lee et al. [24].

Moreover, to help the simulation more closely mimic the fabricated MNN of Fig. 3.1c, the sensor noise thresholds of the fabricated MNN were directly measured and incorporated into the simulation studies of this section to capture the variability of each algorithm due to system noise.

For each simulation the procedure, detailed in Lee et al. [24] for generating different sets of random behaviors, is used. Specifically, the MNN is trained to achieve different sets of two random behaviors. For each behavior, two randomly oriented input forces with magnitudes between ±2 N are applied to the input nodes and random target displacements between ±0.35 mm along the x and y axes (defined in Fig. 3.1d) are assigned to each output node. To ensure that the behaviors in a set are distinct, the MSEs between each behavior's loading forces and target displacements are calculated, and only pairs of behaviors with more than 0.6 N^2 and 0.1 mm^2 for force and displacement MSE are permitted. Once the pairs of two random behaviors are generated, learning is simulated with the same behaviors using each algorithm.

The first simulation study attempts to train the MNN so that it learns 5 different sets of two random behavior pairs per algorithm. This is done with 5 different initial conditions (i.e., different starting combinations of axial beam stiffness values) and the entire process is repeated 10 different times resulting in a total of 250 runs per algorithm. The results are provided in Fig. 2.

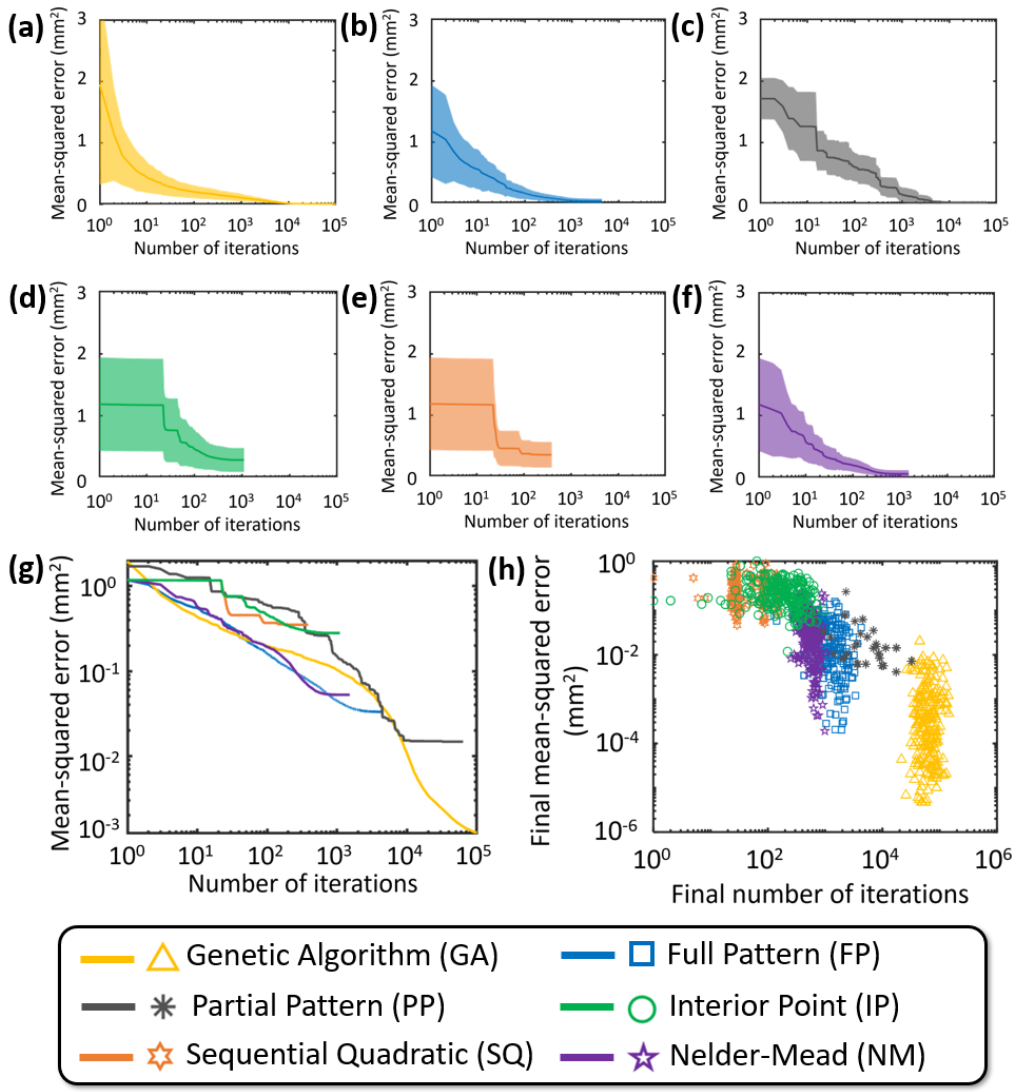

Figure 3.3. The average mean-squared error (MSE) of 250 different simulated runs plotted against the number of iterations using (a) genetic algorithm, (b) full pattern, (c) partial pattern, (d) interior point, (e) sequential quadratic, and (f) Nelder-Mead. The shaded error regions represent one standard deviation. (g) The average MSE plot of all six algorithms plotted together. (h) The final MSE plotted against the final number of iterations for each of the algorithm's 250 runs.

Figs. 3.3a-f provide plots of the lowest MSE identified versus the number of corresponding iterations for each of the six algorithms of Section 2. The solid line in each plot represents the average MSE of all 250 runs. The shaded regions represent one standard deviation

from the average and indicate how much MSE variability exists in the system due in part to system noise, differences in initial conditions, and differences in behaviors learned. Note that the GA plot of Fig. 3.3a shows significant variability and begins with a high MSE that drops rapidly with its first generation and then gradually levels off with subsequent generations until it achieves its final lowest MSE. The trend and spread of the FP plot in Fig. 3.3b both decrease steadily and converge moderately fast. The PP plot in Fig. 3.3c has an MSE that gradually decreases in a stepwise fashion with a spread that is largely consistent from iteration to iteration. The IP plot in Fig. 3.3d begins with a MSE plateau that then decreases with a large spread between runs. The SQ plot in Fig. 2e has many similar characteristics to the IP plot largely because both algorithms rely on Lagrange multipliers. The NM plot in Fig. 3.3f shows a constant, smooth decrease in both average MSE and spread. Note that algorithms with defined starting points (e.g., FP, IP, SQ, and NM), except for PP, have similar spread in the first iteration.

Fig. 3.3g plots the average MSE of all the algorithms on top of each other versus the number of iterations using a log-log scale. For low numbers of iterations, GA, NM, and FP have similarly low MSE whereas PP, IP, and SQ have a similarly high MSE. For large numbers of iterations, GA converges to the lowest MSE followed by PP. The trends for the FP and NM plots are similar although their algorithms are substantially different. Although the IP and SQ plots are nearly coincidental at the beginning, after the initial iterations, the SQ plot decreases in MSE more rapidly than the IP plot, but the IP plot converges to a lower MSE.

Fig. 3.3h. provides the final lowest MSE determined for every run performed by each algorithm (i.e., 250 runs per algorithm), plotted against the final number of iterations at which the algorithms converged. From this data, it is clear that the IP and SQ algorithms produce high MSE (i.e., low accuracy) but converge in relatively few iterations (i.e., high speed). The PP

algorithm produces moderate MSEs in moderate numbers of iterations. Although the NM and FP algorithms converge in moderate numbers of iterations (NM is slightly faster than FP), they both achieve impressively low MSE. The GA algorithm consistently converges to the lowest MSE but requires orders of magnitude more iterations (i.e., time) compared with the other algorithms. The spread of the data runs plotted in Fig. 3.3h for each algorithm provides an indication of the algorithm's repeatability.

The data for the NM, FP, and GA algorithms are all positioned in narrow but tall clusters, which indicate that these algorithms consistently converge in a similar amount of time but produce less consistent accuracy. Algorithms that produce fewer visible run points on the plot indicate highly consistent performance between different runs. Note that the PP algorithm, which tends to stabilize the final MSE, has many overlapping data points spread across an area that is relatively short along the y-axis but wide along the x-axis.

The most efficient algorithms will achieve runs with the lowest final MSE in the fewest final number of iterations (i.e., data points closer to the bottom-left corner of Fig. 3.3h). Thus, although GA is the most accurate (i.e., achieves the lowest final MSE), NM and FP achieve sufficiently low MSE in much less time (i.e., fewer iterations). Thus, if accuracy is the only priority, GA is the clear algorithm of choice. But if learning speed is important so that the MNN can learning in a practical amount of time, NM is likely the best choice since it is a bit faster than FP but it achieves a sufficiently low final MSE.

The second simulation study attempts again to train the MNN of Fig. 3.1c but so that it learns only one set of two random behaviors with only one initial condition repeated 10 times (i.e., a total of 10 runs) per algorithm to isolate how system noise affects the spread. Thus, this

study excludes effects on spread due to differences in learned behaviors and initial conditions. The results of the second study are provided in Fig. 3.4a-f. The solid lines in the plots are the average of all 10 runs and the colored error region represents their standard deviation. Note that the spread of the plots of Figs. 3.4a and 3.4c are larger than the other plots because, in addition to being a result of system noise, they are also due to the inherent variability of the GA and PP algorithms. Note that as iterations progress in the plots of Figs. 3.4b, 3.4d, and 3.4e, the spread due to system noise accumulates. Finally, note that the NM algorithm is the most resistant to system noise in that it exhibits very little spread in the plot of Fig. 3.4f.

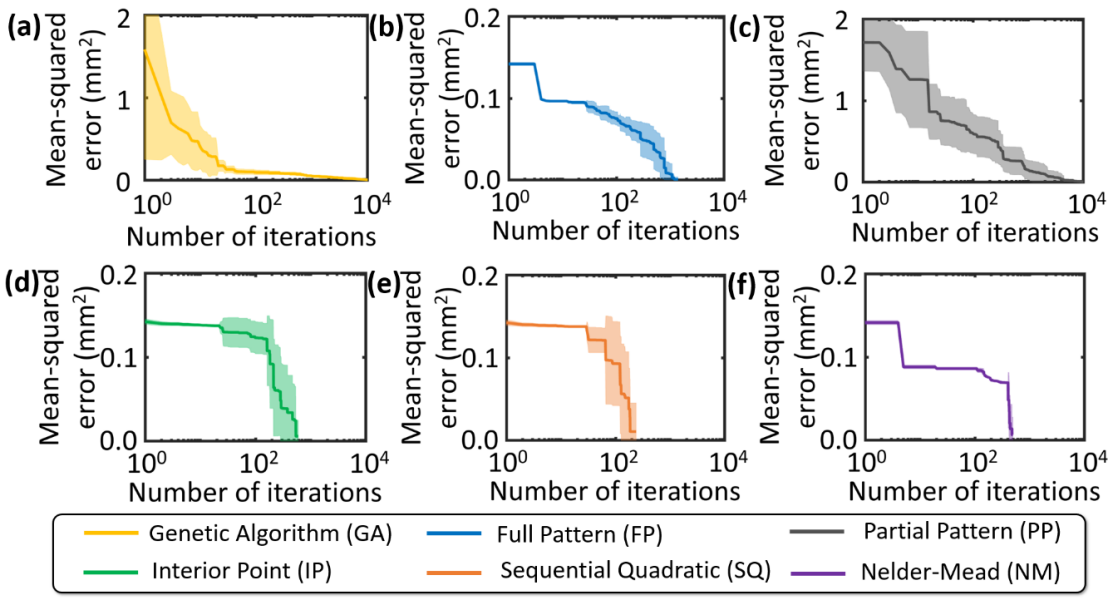

Figure 3.4. The average mean-squared error (MSE) of 10 different simulated runs for the same two random behaviors and the same initial condition plotted against the number of iterations using (a) genetic algorithm, (b) full pattern, (c) partial pattern, (d) interior point, (e) sequential quadratic, and (f) Nelder-Mead. The shaded error regions represent one standard deviation and predominantly represent the spread due to system noise.

The third simulation study attempts to train the MNN of Fig. 3.1c so that it learns 100 different sets of two random behavior pairs using two different initial conditions repeated twice per behavior (i.e., a total of 400 runs). This simulation study was conducted to identify how the

observed trends of Figs. 3.3g and 3.3h would change when the MNN learns a greater diversity of random behavior pairs. The results are provided in Fig. 3.5. The plot of Fig. 3.5a provides the average lowest MSE of all 400 runs for each of the six algorithms plotted on top of each other against the number of iterations using a log-log scale. Fig. 3.5b. provides the final lowest MSE determined for all 400 runs performed by each algorithm, plotted against the final number of iterations at which the algorithms converged. Note that compared with Figs. 3.3g and 3.3h, Figs. 3.5a and 3.5b have remarkably similar and consistent trends. The most noticeable difference is observed in the performance of the PP algorithm. The data points corresponding to each of the PP algorithm runs in Fig. 3.5b show even more consistency in the final MSE identified with even more variability in the final number of iterations required to converge on a solution compared with the results of Fig. 3.3h.

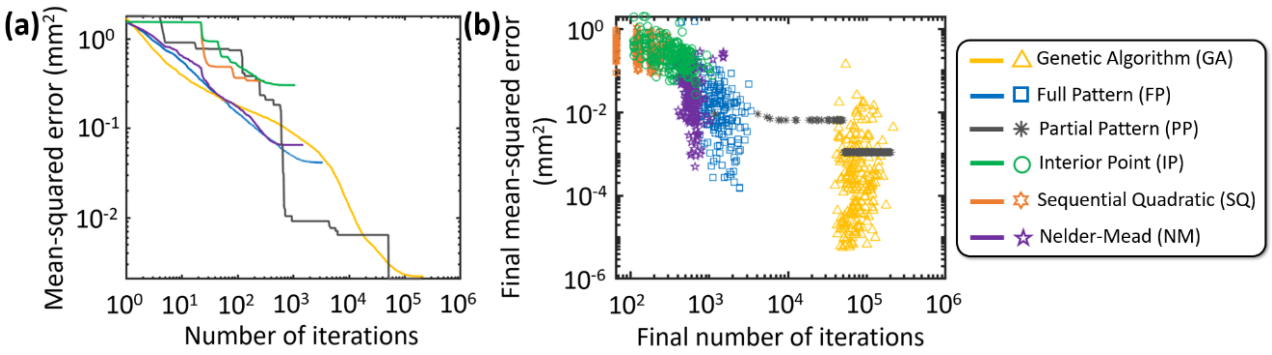

Figure 3.5. (a) The average mean-squared error (MSE) generated by all 6 algorithms simulated with 2 different runs for 100 different random behavior pairs and 2 different initial conditions plotted on top of each other. (b) The final MSE plotted against the final number of iterations for each algorithm's 400 runs.

3.5 Experimental Studies

Although the fabricated MNN of Fig. 3.1c is not capable of generating learning data as rapidly as the simulation tool, it was able to generate enough data to compare the learning

capabilities of the six algorithms and validate the results of the simulation studies. The results of the experimentally collected data differ from the simulated results in that force scaling [24] was not applied during the learning process and reality considers many effects that are not considered by the simulation tool (e.g., large deformation nonlinearities, dynamic effects, and thermal effects).

The physical MNN of Fig. 3.1c was trained to learn two different sets of two random behaviors with one initial condition for each of the six learning algorithms during a six-month study (i.e., a total of 2 runs per algorithm). The results are provided in Fig. 3.6. The plot of Fig. 6a provides the average lowest MSE of the 2 runs for each of the six algorithms plotted on top of each other against the number of iterations using a log-log scale. Fig. 3.6b. provides the final lowest MSE determined for each of the 2 runs performed by each algorithm, plotted against the final number of iterations at which the algorithms converged. Note that although the results don't compare as well as the simulated plots of Figs. 3.6g and 3.6h compare with the simulated plots of Figs. 3.5a and 3.5b, the general trends do match. Although the IP and SQ algorithms were the fastest, they did not learn with sufficiently accuracy to use in practical MNN applications. At the other extreme, although the GA and PP algorithm learned with the most impressive accuracy, they required an impractical amount of time to learn (the GA algorithm occupied most of the six-month study). Thus, for practical applications the NM and FP algorithms were able to learn sufficiently quickly and accurately and are thus most suited for MNN learning in general. Furthermore, note that the NM and FP algorithms seemed to learn faster in the real MNN.

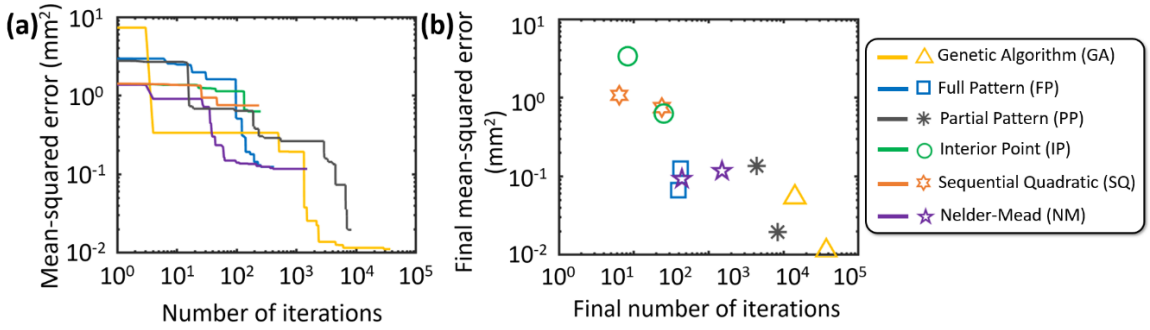

Figure 3.6. (a) The average mean-squared error (MSE) generated by all 6 algorithms simulated for 2 different random behavior pairs and 1 initial condition plotted on top of each other. (b) The final MSE plotted against the final number of iterations for each algorithm's 2 runs.

To illustrate the correlation between the learning trials of the simulated and experimental data, the bar chart of Fig. 3.7 was generated. The simulation values of this chart were calculated by dividing the final average MSE achieved by each algorithm from the simulation plot of Fig. 3.4a by the initial average MSE of the corresponding algorithm from the same plot. The experimental values of this chart were similarly calculated by dividing the final average MSE achieved by each algorithm from the experimental plot of Fig. 3.6a by the initial average MSE of the corresponding algorithm from the same plot. Note from Fig. 3.7 that although all of the final MSE to initial MSE ratios of the experimental data are consistently larger than the corresponding ratios of the simulation data, the relative performance of both the experimental and simulation approach are similar for each algorithm, thus, validating the results of the simulation approach. The fact that the experimental ratios are consistently larger than the corresponding simulation ratios indicates that, on average, the simulated MNN learned with greater accuracy than the fabricated MNN. These ratios would likely even out though if similar numbers of runs could be performed experimentally as they were performed in simulation. Recall that that the simulation

data shown in Fig. 3.7 is derived from 400 runs per algorithm whereas the experimental data shown in the same figure is derived from only 2 runs per algorithm.

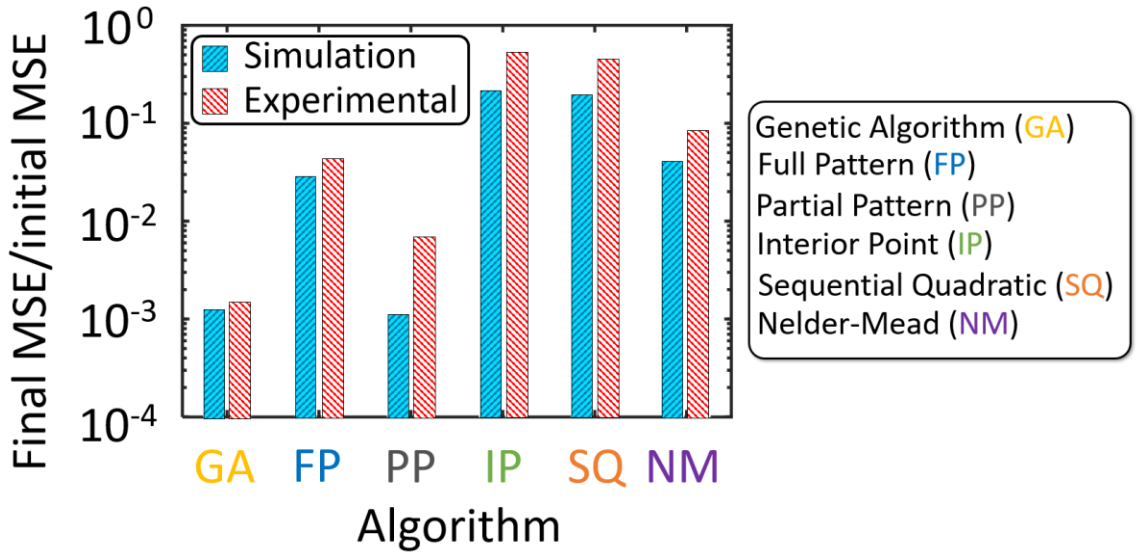

Figure 3.7. A chart showing the ratio between the final average mean-squared error (MSE) and the initial average MSE of each algorithm calculated from the simulation data plotted in Fig. 4a and the experimental data plotted in Fig. 5a.

3.6 Conclusion

In this paper, we compared the efficiency (i.e., speed and accuracy) of six different learning algorithms applied to mechanical neural networks (MNNs) that attempted to learn behaviors via simulation and experimentation. We found that although fast, Lagrangian methods like sequential quadratic (SQ) programing and interior point (IP) did not learn with sufficient accuracy (i.e., they converged to unacceptably high mean-squared error (MSE)). The genetic algorithm (GA) learned with the highest accuracy but required an unreasonably along amount of time to learn. Partial pattern (PP) search also tended to require too much time and didn't perform as accurately as the GA algorithm. The most promising algorithms for MNN learning were found

to be the full pattern (FP) and Nelder-Mead (NM) algorithms since they learned with impressive accuracy and in short enough times to be practical. On average, the NM algorithm seems to be a bit faster than FP and is the most resistant algorithm to MNN system noise by far. Thus, the NM algorithm appears to be the most suited for practical MNN learning applications.

3.7 References

[1] LeCun, Y., Bengio, Y., and Hinton, G., 2015, "Deep Learning," Nature, **521**(7553), pp. 436–444. DOI: 10.1038/nature14539

[2] Gao, H., Cheng, B., Wang, J., Li, K., Zhao, J., and Li, D., 2018, "Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment," IEEE Transactions on Industrial Informatics, **14**(9), pp. 4224–4231. DOI: 10.1109/TII.2018.2822828

[3] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., and Kiru, M. U., 2019, "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," IEEE Access, **7**, pp. 158820–158846. DOI: 10.1109/ACCESS.2019.2945545

[4] McCulloch, W. S., and Pitts, W., 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, **5**(4), pp. 115–133. DOI: 10.1007/BF02478259

[5] Rosenblatt, F., 1958, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psychological Review, **65**(6), pp. 386–408. DOI: 10.1037/h0042519

[6] Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks Are Universal Approximators," Neural Networks, **2**(5), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8

[7] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986, "Learning Representations by Back-Propagating Errors," Nature, **323**(6088), pp. 533–536. DOI: 10.1038/323533a0

[8] LeCun, Y., Bengio, Y., and Hinton, G., 2015, "Deep Learning," Nature, **521**(7553), pp. 436–444. DOI: 10.1038/nature14539

[9] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X., 2019, "Gradient Descent Finds Global Minima of Deep Neural Networks," *Proceedings of the 36th International Conference on Machine Learning*, PMLR, pp. 1675–1685.

[10] Fernando, C., and Sojakka, S., 2003, "Pattern Recognition in a Bucket," *Advances in Artificial Life*, W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J.T. Kim, eds., Springer, Berlin, Heidelberg, pp. 588–597. DOI: 10.1007/978-3-540-39432-7_63

[11] Lv, Z., Liu, P., and Pei, Y., 2020, "Temporal Acoustic Wave Computational Metamaterials," Appl. Phys. Lett., **117**(13), p. 131902. DOI: 10.1063/5.0018758

[12] Zuo, S., Wei, Q., Tian, Y., Cheng, Y., and Liu, X., 2018, "Acoustic Analog Computing System Based on Labyrinthine Metasurfaces," Sci Rep, **8**(1), p. 10103. DOI: 10.1038/s41598-018-27741-2

[13] Hughes, T. W., Williamson, I. A. D., Minkov, M., and Fan, S., "Wave Physics as an Analog Recurrent Neural Network," Science Advances, **5**(12), p. eaay6946. DOI: 10.1126/sciadv.aay6946

[14] Coulombe, J. C., York, M. C. A., and Sylvestre, J., 2017, "Computing with Networks of Nonlinear Mechanical Oscillators," PLOS ONE, **12**(6), p. e0178663. DOI: 10.1371/journal.pone.0178663

[15] Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., and McMahon, P. L., 2022, "Deep Physical Neural Networks Trained with Backpropagation," Nature, **601**(7894), pp. 549–555. DOI: 10.1038/s41586-021-04223-6

[16] Stern, M., Arinze, C., Perez, L., Palmer, S. E., and Murugan, A., 2020, "Supervised Learning through Physical Changes in a Mechanical System," Proceedings of the National Academy of Sciences, **117**(26), pp. 14843–14850. DOI: 10.1073/pnas.2000807117

[17] Furuhata, G., Niiyama, T., and Sunada, S., 2021, "Physical Deep Learning Based on Optimal Control of Dynamical Systems," Phys. Rev. Applied, **15**(3), p. 034092. DOI: 10.1103/PhysRevApplied.15.034092

[18] Widrow, B., Greenblatt, A., Kim, Y., and Park, D., 2013, "The No-Prop Algorithm: A New Learning Algorithm for Multilayer Neural Networks," Neural Networks, **37**, pp. 182–188. DOI: 10.1016/j.neunet.2012.09.020

[19] Boyd, S., and Chua, L., 1985, "Fading Memory and the Problem of Approximating Nonlinear Operators with Volterra Series," IEEE Transactions on Circuits and Systems, **32**(11), pp. 1150–1161. DOI: 10.1109/TCS.1985.1085649

[20] Nakajima, K., Hauser, H., Li, T., and Pfeifer, R., 2015, "Information Processing via Physical Soft Body," Sci Rep, **5**(1), p. 10487. DOI: 10.1038/srep10487

[21] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W., 2012, "The Role of Feedback in Morphological Computation with Compliant Bodies," Biol Cybern, **106**(10), pp. 595–613. DOI: 10.1007/s00422-012-0516-4

[22] Stern, M., Hexner, D., Rocks, J. W., and Liu, A. J., 2021, "Supervised Learning in Physical Networks: From Machine Learning to Learning Machines," Phys. Rev. X, **11**(2), p. 021045. DOI: 10.1103/PhysRevX.11.021045

[23] Dillavou, S., Stern, M., Liu, A. J., and Durian, D. J., 2022, "Demonstration of Decentralized, Physics-Driven Learning," Phys. Rev. Applied, **18**(1), p. 014040.DOI: 10.1103/PhysRevApplied.18.014040

[24] Lee, R. H., Mulder, E. A. B., and Hopkins, J. B., 2022, "Mechanical Neural Networks: Architected Materials That Learn Behaviors," Science Robotics, **7**(71), p. eabq7278. DOI: 10.1126/scirobotics.abq7278

[25] Paterni, P., Vitet, S., Bena, M., and Yokoyama, A., 1999, "Optimal Location of Phase Shifters in the French Network by Genetic Algorithm," IEEE Transactions on Power Systems, **14**(1), pp. 37–42. DOI: 10.1109/59.744481

[26] Lambora, A., Gupta, K., and Chopra, K., 2019, "Genetic Algorithm- A Literature Review," *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255

[27] Katoch, S., Chauhan, S. S., and Kumar, V., 2021, "A Review on Genetic Algorithm: Past, Present, and Future," Multimed Tools Appl, **80**(5), pp. 8091–8126. DOI: 10.1007/s11042-020-10139-6

[28] Hooke, R., and Jeeves, T. A., 1961, "``Direct Search'' Solution of Numerical and Statistical Problems," J. ACM, **8**(2), pp. 212–229. DOI: 10.1145/321062.321069

[29] Güneş, F., and Tokan, F., 2010, "Pattern Search Optimization with Applications on Synthesis of Linear Antenna Arrays," Expert Systems with Applications, **37**(6), pp. 4698–4705. DOI: 10.1016/j.eswa.2009.11.012

[30] Findler, N. V., Lo, C., and Lo, R., 1987, "Pattern Search for Optimization," Mathematics and Computers in Simulation, **29**(1), pp. 41–50. DOI: 10.1016/0378-4754(87)90065-6

[31] Lewis, R. M., and Torczon, V., 2002, "A Globally Convergent Augmented Lagrangian Pattern Search Algorithm for Optimization with General Constraints and Simple Bounds," SIAM J. Optim., **12**(4), pp. 1075–1089. DOI: 10.1137/S1052623498339727

[32] Torczon, V., and Trosset, M. W., 1998, "From Evolutionary Operation to Parallel Direct Search: Pattern Search Algorithms for Numerical Optimization," Computing Science and Statistics, **29**, pp. 396–401.

[33] "How Pattern Search Polling Works - MATLAB & Simulink" [Online]. Available: https://www.mathworks.com/help/gads/how-pattern-search-polling-works.html. [Accessed: 12-Oct-2022].

[34] Jarre, F., Kocvara, M., and Zowe, J., 1998, "Optimal Truss Design by Interior-Point

Methods," SIAM J. Optim., **8**(4), pp. 1084–1107. DOI: 10.1137/S1052623496297097

[35] Fiacco, A. V., and McCormick, G. P., 1990, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, SIAM.

[36] Bussotti, P., 2003, "On the Genesis of the Lagrange Multipliers," Journal of Optimization Theory and Applications, **117**(3), pp. 453–459. DOI: 10.1023/A:1023952102705

[37] Montoya, O. D., Gil-González, W., and Garces, A., 2019, "Sequential Quadratic Programming Models for Solving the OPF Problem in DC Grids," Electric Power Systems Research, **169**, pp. 18–23. DOI: 10.1016/j.epsr.2018.12.008

[38] Vanderbei, R. J., and Shanno, D. F., 1999, "An Interior-Point Algorithm for Nonconvex Nonlinear Programming," Computational Optimization and Applications, **13**(1), pp. 231–252. DOI: 10.1023/A:1008677427361

[39] "Constrained Nonlinear Optimization Algorithms - MATLAB & Simulink" [Online]. Available: https://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html. [Accessed: 12-Oct-2022].

[40] Hager, W. W., 1999, "Stabilized Sequential Quadratic Programming," *Computational Optimization*, J.-S. Pang, ed., Springer US, Boston, MA, pp. 253–273. DOI: 10.1007/978-1-4615-5197-3_13

[41] Nelder, J. A., and Mead, R., 1965, "A Simplex Method for Function Minimization," The Computer Journal, **7**(4), pp. 308–313. DOI: 10.1093/comjnl/7.4.308

[42] Olsson, D. M., and Nelson, L. S., 1975, "The Nelder-Mead Simplex Procedure for Function Minimization," Technometrics, **17**(1), pp. 45–51. DOI: 10.1080/00401706.1975.10489269

[43] Marandi, P. J., Mansooriazdeh, M., and Charkari, N. M., 2008, "The Effect of Re-Sampling on Incremental Nelder-Mead Simplex Algorithm: Distributed Regression in Wireless Sensor Networks," *Wireless Algorithms, Systems, and Applications*, Y. Li, D.T. Huynh, S.K. Das, and D.-Z. Du, eds., Springer, Berlin, Heidelberg, pp. 420–431. DOI: 10.1007/978-3-540-88582-5_40

[44] "Fminsearchbnd, Fminsearchcon" [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd-fminsearchcon. [Accessed: 12-Dec-2022].

CHAPTER 4. MECHANICAL NEURAL-NETWORK METAMATERIALS THAT LEARN VIA BINARY-STIFFNESS BEAMS

4.1 Abstract

This work introduces the concept of applying binary-stiffness beams within a lattice to achieve a mechanical neural-network (MNN) metamaterial that learns its behaviors and properties with prolonged exposure to unanticipated ambient loading scenarios. Applying such beams to MMN metamaterials greatly increases their learning speed and simplifies the actuation demands, control circuitry, and optimization algorithms required by previously proposed concepts. A binary-stiffness beam design is proposed that uses principles of constraint manipulation and stiffness cancelation to achieve two switchable and discrete states of stiffness (i.e., binary stiffness) along its axis. The beam achieves a near-zero low-stiffness state and a large difference in stiffness between its high and low-stiffness states, which are both shown to be desirable attributes for learning mechanical behaviors. Simulations are conducted to characterize the effect of lattice size, the difference in stiffness between the constituent beam's high and low-stiffness states, the magnitude of its low-stiffness state, and the number of simultaneously learned behaviors on MNN learning using binary-stiffness beams. Thus, this work provides a necessary step toward enabling practical artificial intelligent (AI) metamaterials.

4.1 Introduction

Inspired by the speed and efficiency of biological brains, engineers have sought to create physical neural networks [1] that learn. Although most of these physical networks are classified as electrical [2-6] or optical [7-11], some concepts have recently emerged that are best classified as mechanical [1,12-15].

Most recently, a mechanical neural-network (MNN) lattice was proposed [16] as an artificial intelligent (AI) architected material [17] (a.k.a., a mechanical metamaterial) that can learn desired behaviors (e.g., shape morphing, mechanical computation, and acoustic wave propagation) and properties (e.g., shear and Young's modulus, Poisson's ratio, and density) with continued exposure to unanticipated and changing ambient loading conditions. The concept is a direct physical analog to the purely mathematical formulation of an artificial neural network (ANN) [18-20]. Whereas ANNs learn by tuning their weight values [21], which are graphically represented by lines that feed into the various layers of neurons [21] to fit input data to output data, the newly proposed MNN concept [16] learns by tuning the axial stiffness values of its interconnected beams, which propagate mechanical stress waves through various layers of nodes, to fit input loads to desired output displacements.

In addition to introducing the concept, that paper [16] provided a 21-beam triangular lattice design, shown in Fig. 4.1a, that was used to experimentally demonstrate the simultaneous learning of two shape morphing behaviors. Each constituent beam within the lattice consisted of a voice-coil actuator, which could be driven to deform a pair of parallel blade flexures that acted as bearings to guide a translational displacement along the beam's axis. The magnitude and direction of the beam's axial displacements were sensed by strain gauges mounted at the base of the beam's flexure bearings, which also stiffly constrained the beam to not appreciably deform in any other nonaxial direction. Thus, using these actuators, flexure bearings, and sensors, the axial stiffness of each beam was prescribed with any value between an upper and a lower limit during the MNN learning process using active closed-loop control to achieve the desired mechanical behaviors.

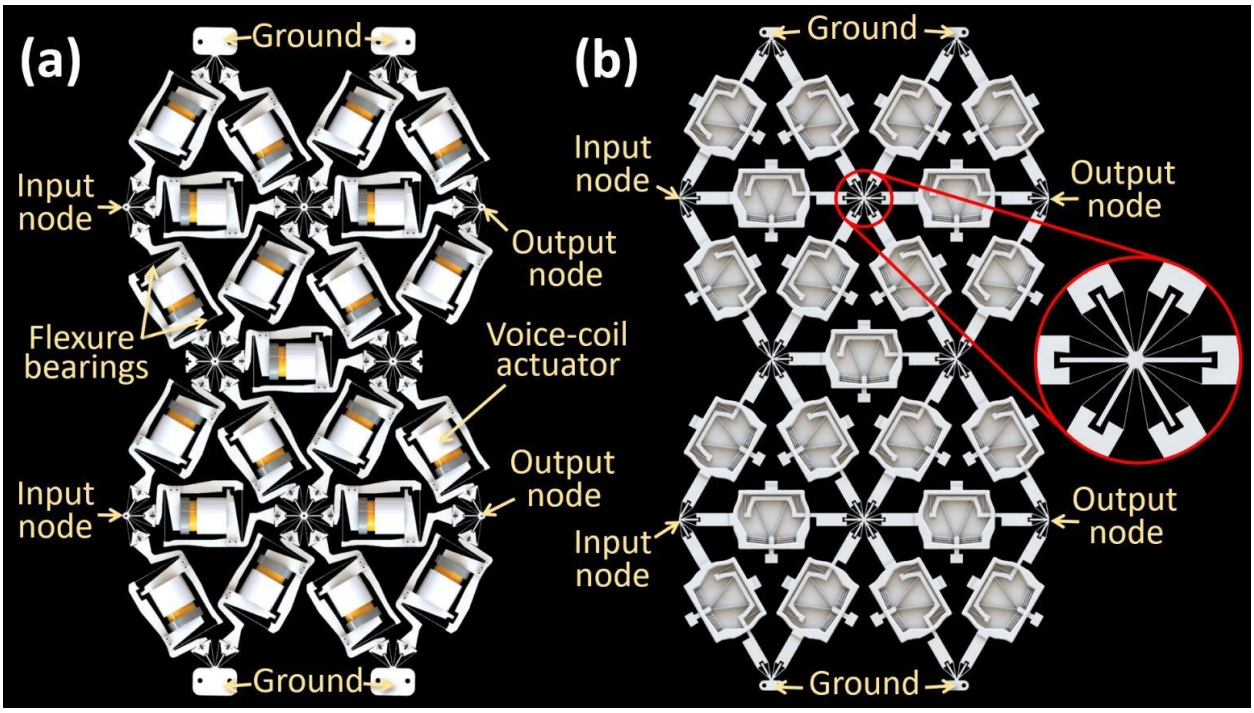

Figure 4.1. **(**a) A previously proposed mechanical neural-network (MNN) design [16] that uses closed-loop control within each of its beams to achieve any prescribed value of axial stiffness between an upper and a lower limit to learn desired behaviors. (b) The new MNN concept proposed here learns behaviors using binary-stiffness beams that can be switched to achieve two different states of axial stiffness only. Rotary flexures connect the beams to the lattice's nodes to accommodate deformations.

Here we introduce a new kind of MNN lattice (e.g., the design shown in Fig 4.1b) that uses binary stiffness beams [22], which can be switched between two different states of stiffness only (i.e., a high and a low value) to learn desired mechanical behaviors. The advantage of this binary stiffness approach is that it greatly simplifies the learning process and dramatically increases its speed. Instead of needing a fast high-resolution actuator to accurately prescribe each beam's axial stiffness via active closed-loop control (e.g., the design shown in Fig. 4.1a), the binary-stiffness approach would only need crude actuators that can push and pull with forces that can exceed the threshold necessary to trigger their beam's bistable switch. And, since the two states of stiffness are passively achieved by the beam's flexure topology, no closedloop control is necessary to achieve and maintain the desired axial stiffness values. Thus, the circuitry necessary

for MNN learning is greatly reduced and simplified. Moreover, since the beams can only achieve two states of stiffness instead of an infinite number of values between an upper and lower limit, the optimization algorithm for selecting the best combination of axial beam stiffness values for achieving a desired set of behaviors is greatly simplified and can thus be achieved with orders of magnitude greater speed. Finally, unlike the actively controlled modular beams of Fig. 4.1a that need to be individually calibrated and assembled within their lattice [16], the binary-stiffness beams of Fig. 4,1b can be made within a single monolithic lattice that requires no calibration. Thus, fabricating large binary-stiffness lattices that consist of many beams at multiple scales (including the micro-scale) is much easier and more practical.

Although MNN designs that achieve a continuous range of stiffness between an upper and lower limit (e.g., Fig. 4.1a) can typically learn more behaviors with higher accuracy than binary-stiffness MNN designs (e.g., Fig 4.1b) consisting of the same number of beams, binary-stiffness MNNs provide a more practical solution for learning behaviors faster as long as such designs can learn enough behaviors with sufficient accuracy. In this work, the learning accuracy and speed of binary-stiffness MNNs are characterized for MNN lattices of different sizes and for different numbers of behaviors learned. The effect of their beam's low-stiffness value and the difference between their high and low-stiffness values on the overall learning capabilities of such MNNs are also quantified.

4.2 Method

4.2.1 Binary-stiffness beam design

The binary-stiffness MNN design used to demonstrate the concept of learning mechanical behaviors via beams that achieve only two states of stiffness is provided in Fig. 4.1(b). An example beam [22] was fabricated out of polylactic acid (PLA) using a Prusa i3 MK3S printer as

shown in Fig. 4.2. The design uses principles of constraint manipulation and stiffness cancelation to achieve two dramatically different states of stiffness. Constraint manipulation occurs when flexible elements that constrain certain directions are deformed in controlled ways so that those directions are no longer constrained as desired. Stiffness cancelation occurs when flexible elements that exhibit positive stiffness are arranged in parallel with other flexible elements that exhibit negative stiffness [23-25]. When these positive and negative stiffness values fully cancel each other so that zero stiffness is manifest [26], the system is said to be statically balanced [27-31]. A number of mechanisms have had their stiffness adjusted over large ranges by deforming flexible elements until they exhibit negative stiffness that then cancels the positive stiffness of other flexible elements in the mechanism to achieve static balancing [32-35]. Some compliant mechanisms [36-39] have leveraged bi-stability [40-42] to deform and undeform certain flexible elements so that they exhibit negative and positive stiffness respectively using a bistable switch.

The binary-stiffness beam design of Fig. 4.2 employs the latter approach to achieve its two states of stiffness. The design consists of two positive-stiffness flexure bearings (Fig. 4.2a) that guide the beam's shuttle with a translational degree of freedom (DOF) along the beam's axis, shown by the double-sided thick black arrow in Fig. 4.2b. When the bistable flexures are directed outward as shown in Fig. 4.2a, the V-shaped flexures impart additional positive stiffness to the shuttle and stiffly constrain it so that it can't move without large forces. When, however, the bistable switch is triggered inward as shown in Fig 4.2b, the once V-shaped flexures deform into a curved configuration and thus, lose their ability to constrain the beam's shuttle. Moreover, the now preloaded flexures exhibit a negative stiffness on the shuttle that cancels the positive stiffness of its flexure bearings and statically balances the system with a near-zero stiffness. An

Instron testing machine was used to measure the beam's axial stiffness in both its high and low-stiffness states [22]. These values are 6.70 N/mm and 0.08 N/mm respectively.

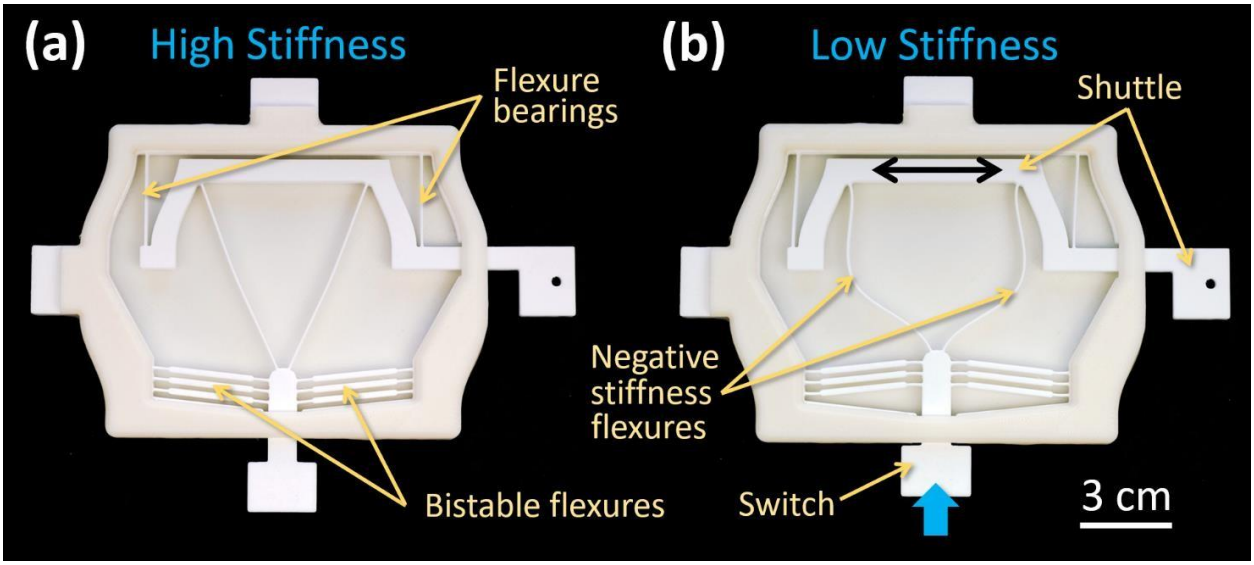

Figure 4.2. A fabricated binary-stiffness beam [22] shown in its (a) high and (b) low-stiffness states. A bistable switch triggered inward deforms V-shaped flexures so that they impart a negative stiffness on the beam's shuttle, which then cancels with the positive stiffness of flexure bearings to enable the beam to manifest near-zero stiffness along its axis. Thus, by triggering the bistable switch in and back out again, the beam can be switched between a low and high-stiffness state respectively.

4.2.2 Nonaxial stiffness values for enabling simulation

To characterize the learning capabilities of different MNN lattices consisting of the binary-stiffness beam design of Fig 4.2, it's necessary to also identify the beam's stiffness along nonaxial directions, which do not and cannot be changed during the learning process. Note from the magnified portion of Fig. 4.1b that the beams in both MNN designs of Fig. 4.1 use blade flexures at their ends that intersect at the center of each node where the beam's join together. These flexures allow a rotational DOF at the nodes so that the lattice can freely deform without jamming as the beams expand or contract along their axes as the lattice is loaded. Hard stops are also included in the nodes to prevent the blade flexures from rotating to their yielding point.

These rotary flexures must also be considered in identifying the nonaxial stiffness values of the beam design as shown in Fig. 4.3a.

COMSOL was used to calculate the beam's three nonaxial stiffness values (i.e., K1, K2, and K3), necessary to simulate binary-stiffness MNN learning for desired scenarios. The material properties used for this finite element analysis (FEA) are those of PLA (i.e., a Young's modulus of 4.107 GPa and a Poisson's ratio of 0.3). The first nonaxial stiffness, K1, was calculated using the displacement, D1, that resulted from the shearing force, F1, imparted on the beam with one end fixed and the other end fixed to a sliding prism joint (Fig 4.3b) according to K1=F1/D1, which was found to be 0.702 N/mm. The second stiffness, K2, was calculated from the displacement, D2, and the bending angle, θ2, that resulted from a shearing force, F2, imparted on the beam with one end fixed and the other end pinned with a revolute joint to a sliding prism joint (Fig 4.3c) according to K2=(F2-D2K1)/θ2, which was found to be 2.94 N/rad. The third stiffness, K3, was calculated from the displacement, D3, and the bending angle, θ3, that resulted from a moment, M3, imparted on the beam with one end fixed and the other end pinned with a revolute joint to a sliding prism joint (Fig 4.3d) according to K3=(M3-D3K2)/θ3, which was found to be 36,320 Nmm/rad.

The beam's calculated nonaxial stiffness values along with its two switchable axial stiffness states, which were directly measured from the fabricated beam, were used within the simulation tool introduced and verified previously [16] to characterize learning scenarios for MNN lattices consisting of the binary-stiffness beams of Fig. 4.3a. The length of the beam measured node to node (i.e., 360 mm), the total allowable axial displacement of the beam (i.e., ±7 mm), and the desired lattice configuration (i.e., a triangular tessellation) were also used to inform the simulation tool for the study of this work.

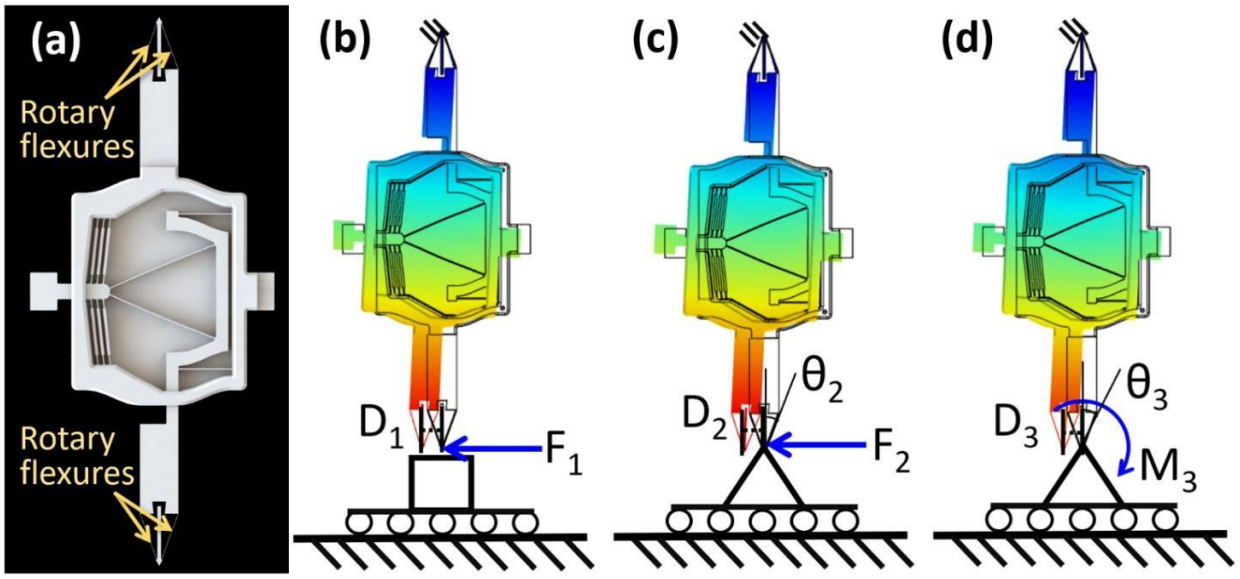

Figure 4.3. (a) The full binary-stiffness beam design including its rotary blade flexures. Its nonaxial stiffness values were calculated using finite element analysis (FEA) for (b) fixed sliding with force loading, (c) pinned sliding with force loading, and (d) pinned sliding with moment loading scenarios. The rainbow of colors shown imposed on the beams represent various displacements where red colors represent maximum displacements and blue colors represent minimum displacements.

4.3 Learning Approach

The computational tool simulates MNN learning according to the approach introduced and demonstrated previously [16] but with an adapted optimization algorithm suited to MNN lattices consisting of binary-stiffness beams. To understand the learning approach, consider the 2 layer-deep triangular lattice with 2 input and 2 output nodes from Fig. 4.1b, which is graphically depicted by the simulation tool as the simplified lattice shown in Fig. 4.4a. Each binary-stiffness beam within the lattice is represented by a single blue line and each node is represented by a white circle. The black bars along the top and bottom of the lattice represent grounded bodies that are held fixed.

Suppose it is desired that the lattice learns one shape morphing behavior, which is labeled ‘Behavior 1’ and shown colored red in Fig. 4.4b. This behavior is achieved when the lattice’s

upper input node, labeled ‘i1,’ is pushed down and to the right with a 45^{o} angle and its lower input node, labeled ‘i2,’ is pushed up and to the right with a 45^{o} angle using the same force magnitude as the force on the upper input node, and the resulting upper output node, labeled ‘o1,’ responds by displacing along the x-axis 0.1 mm and along the y-axis 0.04 mm while the lower output node, labeled ‘o2,’ responds by displacing along the x-axis 0.15 mm and along the y-axis -0.06 mm. Note from Fig 4.3b that the input-node forces and the desired output-node displacement targets of the behavior are represented as red arrows and red asterisks respectively.

To learn this behavior in the midst of changing and random loading scenarios at the input nodes, strain gauge sensors would need to be attached to the base of each beam’s flexure bearings, labeled in Fig 4.2a. These strain gauge sensors would determine how much each beam has contracted or expanded along the beam’s axis as a consequence of the lattice being loaded so that the displacements of all of its nodes could be calculated. And since the axial stiffness of each beam is always known (since it is assigned), the input loads that caused the displacements of the nodes could be indirectly calculated so that the lattice would always know how it is being externally loaded.

A random combination of axial stiffness values (i.e., either the beam’s high or low-stiffness state) would initially be assigned to each beam within the lattice. When the lattice senses that it is being loaded with the forces of the desired behavior, the resulting displacements of the output nodes would be used to calculate a mean squared error (MSE) by subtracting their displacements from the target displacements of the desired behavior and averaging the resulting differences squared.

The beams would then be assigned a different combination of axial stiffness values according to an optimization algorithm so that when the process of loading, measuring, and calculating the MSE is repeated, the MSE is minimized until a working combination of beam stiffness values is identified that allows the lattice to achieve the desired behavior. Note from Fig. 4.4b that a working combination of axial stiffness values was identified that enabled the lattice to achieve Behavior 1 with a final MSE of 0.00036 mm2. Each of the two shades of blue shown in Fig. 4.4b represent the two states of axial stiffness that the beam of Fig. 4.2 can achieve. An exaggeration factor of 750 was multiplied by the displacements of all the nodes in the lattice of Fig. 4.4b to help make the behavior more visibly noticeable.

It's important to recognize that there are multiple different combinations of beam stiffness values in addition to the one shown in Fig 4.4b that can achieve the same behavior. Thus, the same learning approach can enable MNN lattices to learn new behaviors in addition to the previous behaviors learned.

Suppose it is desired that a new combination of beam stiffness values is identified that achieves both Behavior 1 (e.g., Fig 4.4c) and a new behavior, which is labeled 'Behavior 2' and shown colored purple in Fig. 4.4d. This new behavior is achieved when both input nodes, labeled 'i1' and 'i2,' are pushed up with vertical forces of equal magnitude without either output nodes displacing at all. Note from Fig 4.4d that the input-node forces and the desired output-node displacement targets of the behavior are represented as purple arrows and purple asterisks respectively.

To achieve both behaviors simultaneously, the combination of axial stiffness values that was identified for achieving only Behavior 1 from the previous learning attempt (i.e., the

combination shown in Fig. 4.4b) would be used as the new starting point (i.e., the initial stiffness combination assignment). The beams would then be assigned different combinations of axial stiffness values according to the same optimization algorithm so that when the process of loading, measuring, and calculating the MSE is repeated, the MSE is minimized until a new working combination of beam stiffness values is identified that allows the lattice to achieve both behaviors simultaneously. It's important that the MSE being calculated and minimized include the squared differences of the output-node displacements and the corresponding target displacements resulting from all the behaviors averaged together (i.e., both Behavior 1 and Behavior 2).

A possible solution is shown with the same exaggeration factor of 750 in Fig. 4.4c and 4.4d. Note that the same combination of axial stiffness values assigned to the lattice successfully achieved the two desired behaviors simultaneously with a final MSE of 0.0036 mm2. This final MSE is an order of magnitude larger than the previous final MSE generated when only Behavior 1 was learned because learning more behaviors is generally more demanding than learning fewer behaviors. And it's harder to find a working combination of axial stiffness values, which produces output displacements that accurately achieve the target displacements of multiple behaviors simultaneously, especially when only two states of stiffness are available for each beam and only 21 beams are present. Note also that the stiffness combination found in Fig. 4.4c and 4.4d is different from the original combination found in Fig 4.4b but both combinations successfully achieve Behavior 1.

The optimization algorithm used to assign combinations off axial stiffness values to the beams within the MNNs of this work was specially customized to help binary-stiffness lattices learning quickly and effectively. The algorithm randomly chooses a beam in the lattice and

changes its axial stiffness to its alternate state. It then checks to see if loading the input nodes with the desired behaviors decreases the MSE of the output nodes given the new state. If it does not decrease the MSE, other beams are randomly assigned different states in turn to check whether the resulting MSE decreases. Any time the MSE decreases, the beam that made the difference remains switched to its new stiffness state and the entire process repeats. The algorithm continues until it finds a combination of axial stiffness values that can't produce a lower MSE when the stiffness state of every beam is individually switched.

Finally, it's important to recognize that when MNN learning is simulated to compare different scenarios, fair comparisons are most effectively achieved when the principle of load scaling [16] is applied. Once the lattice has identified that it is being loaded with the forces specified in one of its desired behaviors (i.e., the input nodes are being loaded in the desired direction and with the desired magnitude ratios), an optimal scale factor is calculated that, when multiplied to all the input forces, produces the smallest MSE for every loading attempt in the learning process. This load scaling approach ensures that learning comparisons between lattices with different numbers of layers in particular are fairer because lattices with more layers typically need larger load magnitudes to push through the layers and displace the output nodes appreciable amounts. Thus, load scaling was applied to all the simulated results and comparisons of this work.

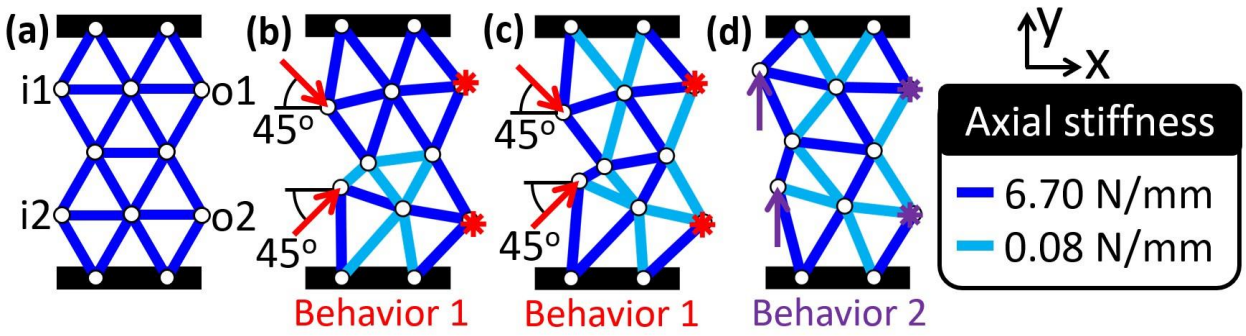

Figure 4.4. (a) An undeformed 2-layer-deep lattice made of binary-stiffness beams that would behave similarly to the beam of Fig. 4.3(a) with 2 input (i.e., i1 and i2) and 2 output (i.e., o1 and o2) nodes. (b) The lattice can learn a single behavior, colored red, and (c) in addition to the initial red behavior can (d) simultaneously learn an additional behavior, colored purple, using a different combination of axial stiffness values. The lattices shown in all parts of this figure were graphically generated by the computational tool of this work for simulating the learning process of various binary-stiffness mechanical neural network (MNN) scenarios.

4.4 Results and Discussion

Despite the many advantages discussed in the introduction, the downside of binary-stiffness MNN lattices like the kind in Fig. 4.1(b) (i.e., the one simulated in Fig. 4.4) compared to actively-controlled MNN lattices like the kind in Fig. 4.1a, is that it's more difficult for binary-stiffness lattices to learn as many behaviors with sufficient accuracy (i.e., a low enough final MSE so that the lattice actually achieves the desired behaviors). Although the lattice of Fig. 4.1b could learn the behaviors detailed in the simulation of Fig. 4.4 with impressive accuracy, it would be difficult for the lattice to learn many other behaviors since it can only use 21 beams and each beam can only achieve two discrete states of stiffness, which is one reason why the full lattice of Fig. 4.1b was not fabricated and tested experimentally. However, large MNN lattices consisting of many binary-stiffness beams do show substantial promise for learning mechanical behaviors.

This section employs the simulation tool discussed previously to determine *(i)* how large binary-stiffness MNN lattices would need to be, *(ii)* how large the differences between the high

and low-stiffness states of their beams would need to be, and (iii) how small the low-stiffness states of their beams would need to be to successfully learn behaviors with sufficient accuracy. A case study is also provided to determine when a binary-stiffness MNN lattice has learned the maximum number of behaviors that is possible for it to learn. A measure for how long the learning process takes for each study is also provided. All the binary-stiffness beams that constitute the lattices simulated in the studies of this section are assumed to have the same nonaxial stiffness values, beam length, axial beam displacement limits, and packing configuration (i.e., triangular) as detailed in Section 2.2.

4.4.1 Study 1 – Number of layers and stiffness difference

The first study examined how binary-stiffness MNN learning is affected by the number of lattice layers and the difference between the high and low-stiffness values of the binary-stiffness beams that constitute the lattices, which also consist of 8 input and 8 output nodes. The low-stiffness state of the binary-stiffness beams were assumed to be the same as the design of Fig. 4.2 (i.e., 0.08 N/mm). The lattices of this study simulated MNN learning for two simultaneous behaviors. The first behavior, shown red in Fig. 4.5a, was achieved when the lattice's input nodes were all pushed to the right with horizontal forces of the same magnitude and the output nodes displaced in response to target locations that lie along a sinusoidal curve with an amplitude of 1.25 mm. The second behavior, shown purple in Fig. 4.5b, was achieved when the lattice's input nodes were all pushed up with vertical forces of the same magnitude and the output nodes displaced in response to target locations that lie along an inverted sinusoidal curve with an amplitude of 1.25 mm.

Many different MNN lattices from 1 to 25 layers deep and with 0 to 20 N/mm differences in stiffness between the high and low-stiffness state of the constituent binary-stiffness beams

attempted to learn these two behaviors via simulation. One example 16-layer lattice with a 16.96 N/mm difference in axial stiffness successfully learned the two desired behaviors simultaneously with a final MSE of 0.0077 mm2 as shown with an exaggeration factor of 40 in Fig. 4.5a and 4.5b. Note that although the lattice successfully achieved both behaviors using the same combination of axial stiffness values, the input nodes needed to be significantly displaced to achieve the desired displacement magnitude of the output nodes. This effect can be reduced and binary-stiffness MNN learning can typically be improved using rotary flexures (Fig. 4.1b) that exhibit greater compliance about their rotational axis.

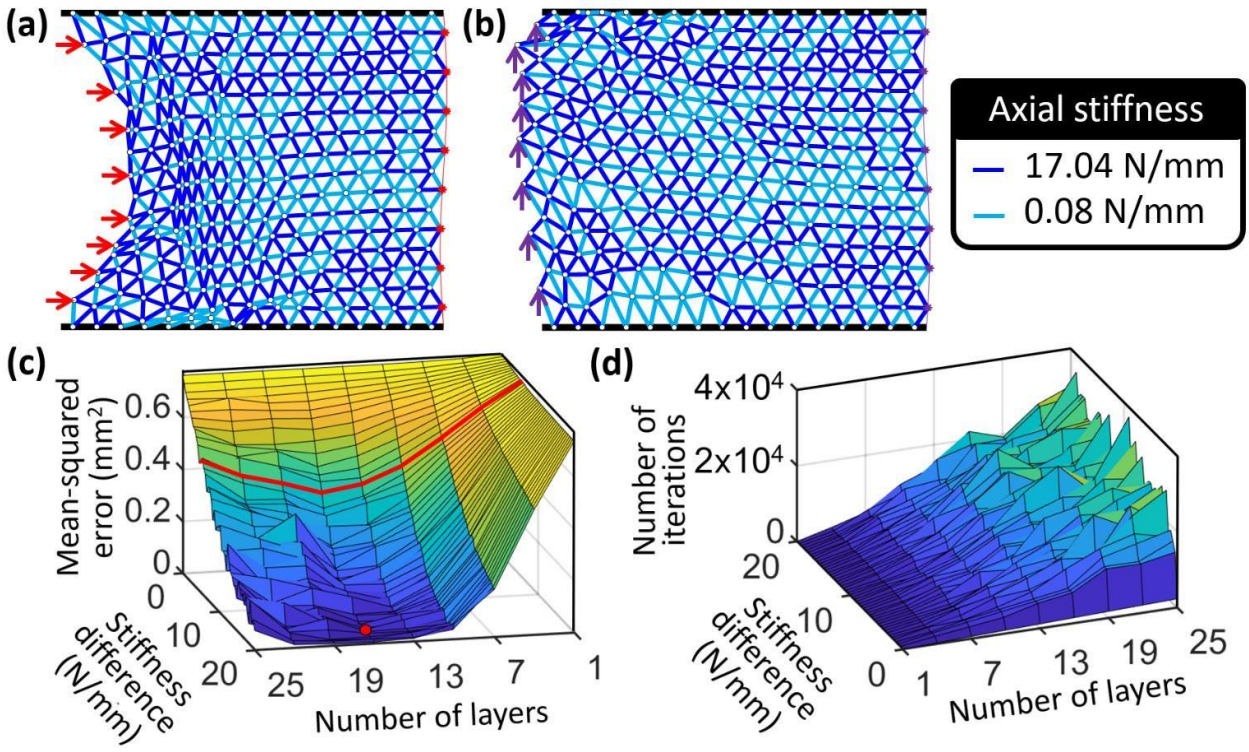

Figure 4.5. An example 16-layer lattice with a 16.96 N/mm difference in axial stiffness successfully learned the (a) first and (b) second desired sinusoidal behavior simultaneously. Plots of the (c) final mean-squared error (MSE) and (d) number of iterations generated by the simulation tool for different numbers of layers and differences in axial stiffness values.

The resulting plot of the first study is provided in Fig. 4.5c. This plot shows the lowest (i.e., best) final MSE of 3 different learning attempts using 3 randomly generated starting assignments (i.e., combinations of beam axialstiffness values). For scenarios with stiffness differences larger than 10 N/mm, the lowest MSE of 8 starting assignments were used to refine the optimal region. The red dot, corresponds with the example provided in Fig. 4.5a and Fig. 4.5b. Note that this example lies within the optimal region of the plot. The red cross-section curve represents all the final MSE's that could be achieved by binary-stiffness MNNs with different numbers of layers consisting of the beam design of Fig. 4.2 (i.e., an axial stiffness difference of 6.62 N/mm). Note from the plot that binary-stiffness MNNs can generally learn with higher accuracy (i.e., lower MSE) the more layers they possess and the larger the difference in stiffness states that their binary-stiffness beams can achieve. This trend makes sense since MNNs consisting of more binary-stiffness beams should typically allow for more combinations of axial stiffness values that achieve the same behaviors. Moreover, the larger the difference in axial stiffness states that each beam can achieve, the more differently the lattice can be made to behave. Note, however, that MNN lattices can begin losing the ability to learn accurately with too many layers. Even with the principle of load scaling discussed in Section 2.3, lattices can become too deep for the loads on the input nodes to cause the output nodes to displace at all. This limitation arises in part because the beams are limited to contract and extend by a fixed amount (i.e., ±7 mm in this study) and they can't collide or pass through one another in practice. Thus, input-node forces can only penetrate so deep even if they are optimized to achieve the lowest achievable MSE.

The number of iterations for each of the scenarios simulated in the plot of Fig. 4.5c are also provided in the plot of Fig. 4.5d. An iteration is defined as every time the lattice's input

nodes are loaded to determine the resulting displacements of the lattice's output nodes during the learning process. Thus, this number multiplied by how long it would take to load the input nodes of a physical MNN and then measure its resulting output-node displacements is how much time it would take for a fabricated MNN of the kind studied here to learn the two sinusoidal behaviors. Note from the trend in Fig. 4.5d that binary-stiffness MNNs with more layers require more time to learn since there are more beams with stiffness states that need to be switched during the optimization algorithm.

4.4.2 Study 2 - Stiffness difference and low-stiffness state

The second study examined how binary-stiffness MNN learning is affected by the difference between the high and low-stiffness values of the beams that constitute the lattices and the value of the low-stiffness state of the beams. The lattices of this study consisted of 7 layers, 8 input nodes, and 8 output nodes. MNN learning was simulated on these lattices for the same two sinusoidal behaviors described in Section 3.1.

The resulting plot of the second study is provided in Fig. 4.6a. This plot shows the lowest (i.e., best) final MSE of 3 different learning attempts using 3 randomly generated starting assignments (i.e., combinations of beam axial stiffness values). The red dot, corresponds with the final MSE that would be achieved by a MNN that consists of the beam design of Fig. 4.2 (i.e., an axial stiffness difference of 6.62 N/mm and a low-stiffness state of 0.08 N/mm). Note from the plot that binary-stiffness MNNs can generally learn with higher accuracy (i.e., lower MSE) the larger the difference in stiffness states that their binary-stiffness beams can achieve and the smaller their low-stiffness state is. Thus, the larger the ratio a binary-stiffness beam's stiffness difference to low-stiffness state is, the better the beam is for facilitating MNN learning.

The number of iterations (defined in Section 3.1) for each of the scenarios simulated in the plot of Fig. 4.6a are also provided in the plot of Fig. 4.6b. Note from the trend in the plot of Fig. 4.6b that binary-stiffness MNNs with larger differences in stiffness tend to require more time to learn since beams that achieve larger changes in stiffness are more likely to produce a significant system change during the learning process and thus would require more iterations to settle on a solution. This trend is less obvious to see in the plot of Fig. 4.5d since increased layer numbers increases the learning time significantly more than the difference between beam stiffness states.

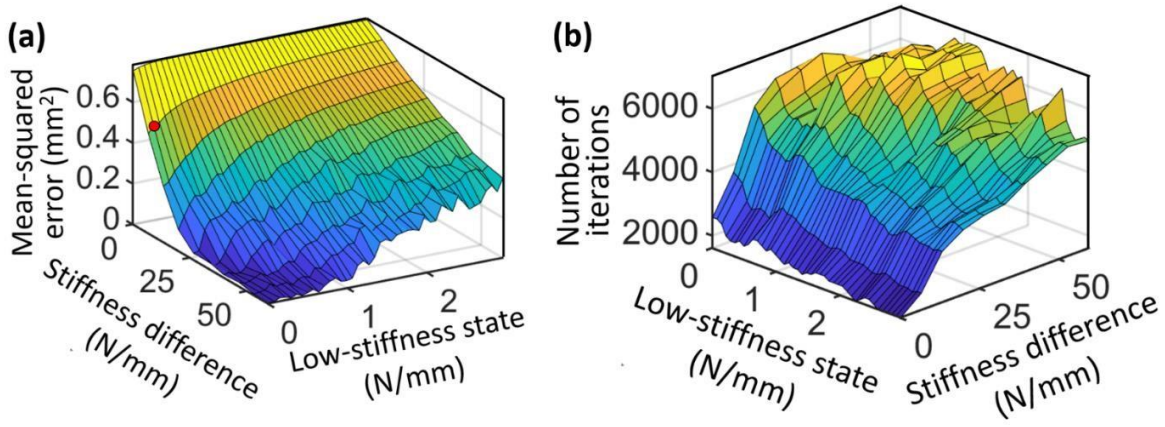

Figure 4.6. Plots of the (a) final mean-squared error (MSE) and (b) number of iterations generated by the simulation tool for various differences in axial stiffness values and different low-stiffness states for a 7-layer binary-stiffness MNN that learns the two sinusoidal behaviors of Fig. 4.5a and b.

4.4.3 Study 3 – Number of random behaviors

The third study examined how binary-stiffness MNN learning is affected when a lattice attempts to learn different numbers of random behaviors. The lattice of this study consists of 7 layers, 8 input nodes, and 8 output nodes. Its binary stiffness beams simulate those of Fig. 4.2 in that they achieve the same high and low-stiffness states (i.e., 6.70 N/mm and 0.08 N/mm).

The computational tool was used to simulate the lattice attempting to learn different numbers of randomly generated behaviors. Random behaviors were generated by selecting input-node forces and output-node displacements with randomly generated x- and y-axis components between ±1 N and ±0.5 mm respectively. To ensure that each new behavior was sufficiently different from the previously generated behaviors, a MSE was calculated for each previous behavior by averaging the difference between the previous and new behavior's input forces squared. As long as the MSEs that were calculated from each of the previously generated behaviors all exceeded 0.3 N2, the new behavior was deemed sufficiently different. Once 1 to 100 sufficiently different behaviors were generated, three additional unique sets of different behaviors were generated for the lattice to learn. The lattice then attempted to simultaneously learn each unique set of behaviors three different times and the lowest final MSE was averaged with the lowest final MSEs generated by learning the other unique sets of behaviors. The resulting MSE average was plotted in Fig. 4.7a.

This plot demonstrates that the 7-layer lattice, made of beams from Fig 4.3a, would not be able to accurately learn more than ~10 different behaviors of the kind generated. The plot plateaus at the worst MSE that the lattice produces when it is not able to learn the behaviors attempted. A binary stiffness MNN lattice with more layers that consists of beams with larger differences in axial stiffness and smaller low stiffness states could successfully learn substantially more behaviors (particularly if the behaviors were easier to learn with output node displacement limits that are smaller). The number of iterations (defined in Section 3.1) generated for each attempt at learning the random behaviors simulated in the plot of Fig 4.7a are also provided in the plot of Fig. 4.7b. Note from the trend of the plot that it takes the binary stiffness MNN lattice more time to learn more behaviors because the input nodes need to be loaded and

the resulting displacements of the output nodes need to be measured for every behavior attempted in the learning process.

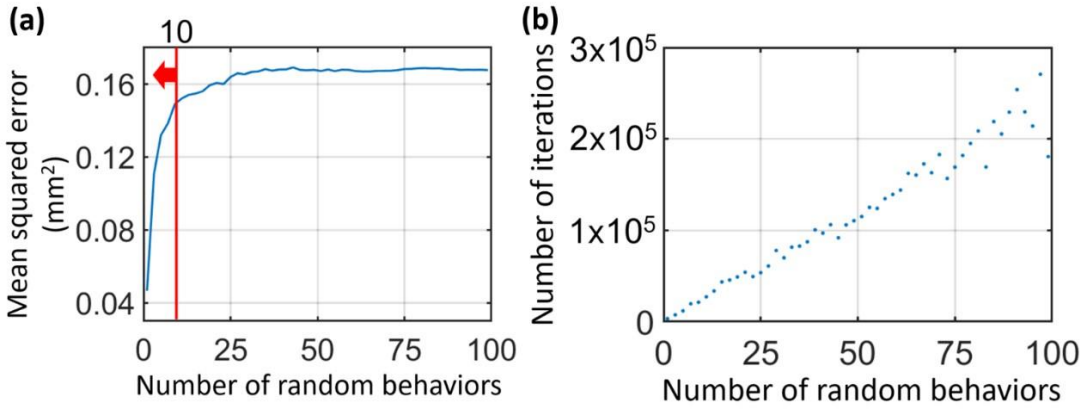

Figure 4.7. Plots of the (a) final mean-squared error (MSE) and (b) number of iterations generated by the simulation tool for a 7-layer lattice consisting of beams from Fig. 4.3a that attempt to learn different numbers of random behaviors.

4.5 Conclusion

This work studies how well mechanical neural-network (MNN) metamaterials that consist of binary-stiffness beams can learn desired shape-morphing behaviors. A binary stiffness beam design is provided as an example and is used to inform the simulations of a computational tool used to conduct the studies. In general, it was determined that MNNs consisting of more layers of binary-stiffness beams that achieve larger differences in stiffness states with smaller low stiffness states can learn more behaviors with higher accuracy but require more time to learn. Such binary-stiffness MNNs can generally learn many behaviors simultaneously and with sufficient accuracy that their dramatic increase in learning speed coupled with their simplified design, fabrication requirements, and approach to learning justify them as a more practical solution to pursue for most MNN applications compared to other MNN approaches presented previously.

Although this study was restricted to characterizing binary stiffness MNNs that learned shape-morphing behaviors, such MNNs could learn many other kinds of quasi-static and dynamic behaviors using the same learning approach presented here. Thus, this work could enable a host of applications including armor that learns to redirect shock waves most effectively to minimize damage to what is being protected, buildings that learn to minimally shake during earthquakes of unanticipated and changing kind and magnitude, and aircraft wings that learn to optimally change their shape in response to fluctuating wind conditions to optimize fuel efficiency and maneuverability.

4.6 References

[1] Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., and McMahon, P. L., 2022, "Deep Physical Neural Networks Trained with Backpropagation," Nature, **601**(7894), pp. 549–555.

[2] Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B., 2015, "Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors," Nature, **521**(7550), pp. 61–64.

[3] Han, R., Huang, P., Xiang, Y., Liu, C., Dong, Z., Su, Z., Liu, Y., Liu, L., Liu, X., and Kang, J., 2019, "A Novel Convolution Computing Paradigm Based on NOR Flash Array With High Computing Speed and Energy Efficiency," IEEE Transactions on Circuits and Systems I: Regular Papers, **66**(5), pp. 1692–1703.

[4] Zhang, H.-T., Park, T. J., Islam, A. N. M. N., Tran, D. S. J., Manna, S., Wang, Q., Mondal, S., Yu, H., Banik, S., Cheng, S., Zhou, H., Gamage, S., Mahapatra, S., Zhu, Y., Abate, Y., Jiang, N., Sankaranarayanan, S. K. R. S., Sengupta, A., Teuscher, C., and Ramanathan, S., 2022, "Reconfigurable Perovskite Nickelate Electronics for Artificial Intelligence," Science, **375**(6580), pp. 533–539.

[5] Lee, S., Kim, H., Lee, S.-T., Park, B.-G., and Lee, J.-H., 2022, "SiO2 Fin-Based Flash Synaptic Cells in AND Array Architecture for Binary Neural Networks," IEEE Electron Device Letters, **43**(1), pp. 142–145.

[6] Dillavou, S., Stern, M., Liu, A. J., and Durian, D. J., 2022, "Demonstration of Decentralized, Physics-Driven Learning."

[7] Shen, Y., Harris, N. C., Skirlo, S., Prabhu, M., Baehr-Jones, T., Hochberg, M., Sun, X., Zhao, S., Larochelle, H., Englund, D., and Soljačić, M., 2017, "Deep Learning with Coherent Nanophotonic Circuits," Nature Photon, **11**(7), pp. 441–446.

[8] Wu, Z., Zhou, M., Khoram, E., Liu, B., and Yu, Z., 2020, "Neuromorphic Metasurface," Photon. Res., PRJ, **8**(1), pp. 46–50.

[9] Furuhata, G., Niiyama, T., and Sunada, S., 2021, "Physical Deep Learning Based on Optimal Control of Dynamical Systems," Phys. Rev. Applied, **15**(3), p. 034092.

[10] Zhang, H., Gu, M., Jiang, X. D., Thompson, J., Cai, H., Paesani, S., Santagati, R., Laing, A., Zhang, Y., Yung, M. H., Shi, Y. Z., Muhammad, F. K., Lo, G. Q., Luo, X. S., Dong, B.,

Kwong, D. L., Kwek, L. C., and Liu, A. Q., 2021, "An Optical Neural Chip for Implementing Complex-Valued Neural Network," Nat Commun, **12**(1), p. 457.

[11] Lin, X., Rivenson, Y., Yardimci, N. T., Veli, M., Luo, Y., Jarrahi, M., and Ozcan, A., 2018, "All-Optical Machine Learning Using Diffractive Deep Neural Networks," Science, **361**(6406), pp. 1004–1008.

[12] Hermans, M., Burm, M., Van Vaerenbergh, T., Dambre, J., and Bienstman, P., 2015, "Trainable Hardware for Dynamical Computing Using Error Backpropagation through Physical Media," Nat Commun, **6**(1), p. 6729.

[13] Stern, M., Hexner, D., Rocks, J. W., and Liu, A. J., 2021, "Supervised Learning in Physical Networks: From Machine Learning to Learning Machines," Phys. Rev. X, **11**(2), p. 021045.

[14] Hughes, T. W., Williamson, I. A. D., Minkov, M., and Fan, S., 2019, "Wave Physics as an Analog Recurrent Neural Network," Science Advances, **5**(12), p. eaay6946.

[15] Stern, M., Arinze, C., Perez, L., Palmer, S. E., and Murugan, A., 2020, "Supervised Learning through Physical Changes in a Mechanical System," Proceedings of the National Academy of Sciences, **117**(26), pp. 14843–14850.

[16] Lee, R. H., Mulder, E. A. B., and Hopkins, J. B., 2022, "Mechanical Neural Networks: Architected Materials That Learn Behaviors," Science Robotics, **7**(71), p. eabq7278.

[17] Schaedler, T. A., and Carter, W. B., 2016, "Architected Cellular Materials," Annual Review of Materials Research, **46**(1), pp. 187–210.

[18] McCulloch, W. S., and Pitts, W., 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, **5**(4), pp. 115–133.

[19] Rosenblatt, F., 1958, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psychological Review, **65**(6), pp. 386–408.

[20] Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks Are Universal Approximators," Neural Networks, **2**(5), pp. 359–366.

[21] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., and Kiru, M. U., 2019, "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," IEEE Access, **7**, pp. 158820–158846.

[22] Kuppens, P. R., Bessa, M. A., Herder, J. L., and Hopkins, J. B., 2021, "Monolithic Binary Stiffness Building Blocks for Mechanical Digital Machines," Extreme Mechanics Letters, **42**, p. 101120.

[23] Van Eijk, J., and Dijksman, J. F., 1979, "Plate Spring Mechanism with Constant Negative Stiffness," Mechanism and Machine Theory, **14**(1), pp. 1–9.

[24] Ren, C., Yang, D., and Qin, H., 2018, "Mechanical Performance of Multidirectional Buckling-Based Negative Stiffness Metamaterials: An Analytical and Numerical Study," Materials, **11**(7), p. 1078.

[25] Tan, X., Chen, S., Wang, B., Tang, J., Wang, L., Zhu, S., Yao, K., and Xu, P., 2020, "Real-Time Tunable Negative Stiffness Mechanical Metamaterial," Extreme Mechanics Letters, **41**, p. 100990.

[26] Schenk, M., and Guest, S. D., 2014, "On Zero Stiffness," Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, **228**(10), pp. 1701–1714.

[27] Gallego, J. A., and Herder, J. L., 2011, "Criteria for the Static Balancing of Compliant Mechanisms," American Society of Mechanical Engineers Digital Collection, pp. 465–473.

[28] Tolou, N., Estevez, P., and Herder, J. L., 2012, "Collinear-Type Statically Balanced Compliant Micro Mechanism (SB-CMM): Experimental Comparison Between Pre-Curved and Straight Beams," American Society of Mechanical Engineers Digital Collection, pp. 113–117.

[29] Wang, P., and Xu, Q., 2018, "Design and Modeling of Constant-Force Mechanisms: A Survey," Mechanism and Machine Theory, **119**, pp. 1–21.

[30] Herder, J., 2001, "Energy-Free Systems; Theory, Conception and Design of Statically Balanced Spring Mechanisms."

[31] Hoetmer, K., Herder, J. L., and Kim, C. J., 2010, "A Building Block Approach for the Design of Statically Balanced Compliant Mechanisms," American Society of Mechanical Engineers Digital Collection, pp. 313–323.

[32] Zhao, H., Han, D., Zhang, L., and Bi, S., 2017, "Design of a Stiffness-Adjustable Compliant Linear-Motion Mechanism," Precision Engineering, **48**, pp. 305–314.

[33] Secord, T. W., and Asada, H. H., 2010, "A Variable Stiffness PZT Actuator Having Tunable Resonant Frequencies," IEEE Transactions on Robotics, **26**(6), pp. 993–1005.

[34] Churchill, C. B., Shahan, D. W., Smith, S. P., Keefe, A. C., and McKnight, G. P., 2016, "Dynamically Variable Negative Stiffness Structures," Science Advances, **2**(2), p. e1500778.

[35] Dahiya, A., and Braun, D. J., 2017, "Efficiently Tunable Positive-Negative Stiffness Actuator," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1235–1240.

[36] Pluimers, P. J., Tolou, N., Jensen, B. D., Howell, L. L., and Herder, J. L., 2012, "A Compliant On/Off Connection Mechanism for Preloading Statically Balanced Compliant Mechanisms," *2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC)*, pp. 373–377.

[37] Lassooij, J., Tolou, N., Tortora, G., Caccavaro, S., Menciassi, A., and Herder, J. L., 2012, "A Statically Balanced and Bi-Stable Compliant End Effector Combined with a Laparoscopic 2DoF Robotic Arm," Mechanical Sciences, **3**(2), pp. 85–93.

[38] Chen, G., and Zhang, S., 2011, "Fully-Compliant Statically-Balanced Mechanisms without Prestressing Assembly: Concepts and Case Studies," Mechanical Sciences, **2**(2), pp. 169–174.

[39] Kuppens, P. R., Bessa, M. A., Herder, J. L., and Hopkins, J. B., 2021, "Compliant Mechanisms That Use Static Balancing to Achieve Dramatically Different States of Stiffness," Journal of Mechanisms and Robotics, **13**(2).

[40] Chen, G., Gou, Y., and Zhang, A., 2011, "Synthesis of Compliant Multistable Mechanisms Through Use of a Single Bistable Mechanism," Journal of Mechanical Design, **133**(8).

[41] Chen, G., Gou, Y., and Yang, L., 2010, "Research on Multistable Compliant Mechanisms: The State of the Art," Proceedings of the 9th International Conference on Frontiers of Design and Manufacturing (ICFDM 2010), pp. 17–19.

[42] Wilcox, D. L., and Howell, L. L., 2005, "Fully Compliant Tensural Bistable Micromechanisms (FTBM)," Journal of Microelectromechanical Systems, **14**(6), pp. 1223–1235.

CHAPTER 5. CONCLUSIONS

This dissertation presents advances in architected materials by experimentally demonstrating a method for creating architected materials that can learn new properties and behaviors by adjusting the stiffnesses of the material's constituent elements. These materials are defined to be Mechanical Neural-networks (MNN) due to their learning approach that have many similarities to more traditional Artificial Neural-networks (ANNs).

Chapter 2 proves the functionality of these materials by teaching several shape morphing behaviors to MNNs in simulation as well as experimentally using a macroscale MNN demonstration machine. These studies explored lattices with a range of parameters (e.g. depth, height, layout, linearity) and determined the height of the MNN, did not have much of an impact on the material's ability to learn. MNN with many layers of linear beams in an triangular lattice has a high potential for accurately learning many simultaneous behaviors.

Chapter 3 compares effects of six different learning algorithms applied to simulated MNNs and experimentation on their speed and accuracy. These studies found that Lagrangian methods like sequential quadratic (SQ) programing and interior point (IP) converged quickly to a solution, but did not learn with sufficient accuracy (i.e., they converged to unacceptably high mean-squared error (MSE)). The studies showed that the genetic algorithm (GA) learned with the highest accuracy but required one to two orders of magnitude more time to learn than the next best algorithm Partial pattern (PP) search. PP search preformed similarly to GA, requiring a very high number of iterations to converge, but was not as accurate. The most promising algorithms for MNN learning were found to be the full pattern (FP) search and Nelder-Mead (NM) algorithms as they learned with impressive accuracy and in short enough times to be practical. On average, the NM algorithm is five times faster than FP and is the most resistant

algorithm to MNN system noise. Thus, the NM algorithm appears to be the most suited for practical MNN learning applications.

Chapter 4 explores the potential of MNNs constructed of beams with only two discreet stiffness states. To test the potential of these MNNs, shape-morphing behaviors are taught to a simulated MNN material. In general, it was determined that MNNs consisting of more layers of binary-stiffness beams that achieve larger differences in stiffness states with smaller low stiffness states can learn more behaviors with higher accuracy. However, lattices with these parameters require more time to learn. Such binary-stiffness MNNs can generally learn many behaviors simultaneously and with sufficient accuracy that their dramatic increase in learning speed coupled with their simplified design, fabrication requirements, and approach to learning justify them as a more practical solution to pursue for most MNN applications compared to other MNN approaches presented previously.