7-1-2021

Reinforcement learning applied to metamaterial design

Tristan Shah
Eastern Michigan University

Linwei Zhuo
San Jose State University

Peter Lai
San Jose State University

Amaris De La Rosa-Moreno
San Jose State University

Feruza Amirkulova
San Jose State University, feruza.amirkulova@sjsu.edu

See next page for additional authors

Recommended Citation

Tristan Shah, Linwei Zhuo, Peter Lai, Amaris De La Rosa-Moreno, Feruza Amirkulova, and Peter Gerstoft. "Reinforcement learning applied to metamaterial design" *Journal of the Acoustical Society of America* (2021): 321-338. https://doi.org/10.1121/10.0005545

JULY 14 2021

Reinforcement learning applied to metamaterial design[a)]

Tristan Shah; Linwei Zhuo; Peter Lai; ... et. al

J Acoust Soc Am 150, 321–338 (2021)
https://doi.org/10.1121/10.0005545

JASA ARTICLE

Reinforcement learning applied to metamaterial design[a)]

Tristan Shah,[1] Linwei Zhuo,[2] Peter Lai,[2] Amaris De La Rosa-Moreno,[2] Feruza Amirkulova,[2,b)] and Peter Gerstoft[3,c)]
[1]*Data Science and Analytics, Eastern Michigan University, Ypsilanti, Michigan 48197, USA*
[2]*Mechanical Engineering Department, San Jose State University, San Jose, California 95192, USA*
[3]*Scripps Institution of Oceanography, University of California San Diego, La Jolla, California 92093, USA*

ABSTRACT:
This paper presents a semi-analytical method of suppressing acoustic scattering using reinforcement learning (RL) algorithms. We give a RL agent control over design parameters of a planar configuration of cylindrical scatterers in water. These design parameters control the position and radius of the scatterers. As these cylinders encounter an incident acoustic wave, the scattering pattern is described by a function called total scattering cross section (TSCS). Through evaluating the gradients of TSCS and other information about the state of the configuration, the RL agent perturbatively adjusts design parameters, considering multiple scattering between the scatterers. As each adjustment is made, the RL agent receives a reward negatively proportional to the root mean square of the TSCS across a range of wavenumbers. Through maximizing its reward per episode, the agent discovers designs with low scattering. Specifically, the double deep Q-learning network and the deep deterministic policy gradient algorithms are employed in our models. Designs discovered by the RL algorithms performed well when compared to a state-of-the-art optimization algorithm using *fmincon*. © *2021 Acoustical Society of America*. https://doi.org/10.1121/10.0005545

(Received 9 March 2021; revised 22 May 2021; accepted 17 June 2021; published online 14 July 2021)
[Editor: James F. Lynch] Pages: 321–338

I. INTRODUCTION

Acoustic metamaterials are sub-wavelength sized structures that, when formed into a continuous material, can exhibit unusual mechanical properties, such as negative effective mass density, negative refraction, high transmission, or absorption. These unique properties of acoustic metamaterials enable control over wave propagation energy within a medium. Many exciting applications have emerged, such as cloaking,[1] sound focusing,[2] wave steering,[3] and beam forming.[4] However, to achieve unconventional material property values required for acoustic wave manipulation, meticulously designed metamaterials are necessary. A novel approach in designing metamaterials can be found within the field of machine learning (ML). Review papers on ML for molecular design,[5,6] inverse designs of inorganic solid materials,[7] and inverse designs of nanophotonics[8–10] suggest that deep learning (DL), reinforcement learning (RL), and generative modeling assisted inverse design models can, by far, exceed human capability.

RL is a sub-field of ML that studies how agents take actions based on trial and error. RL agents are capable of learning optimal policies in an environment that maximize a reward. In previous works, these agents have mastered complex games, such as chess or go, with the only information supplied being the rules of the game.[11,12] We can use these agents and repurpose them to optimize design parameters of a device. By giving a reward signal to the agent that reflects how well the design satisfies our criteria, it will produce optimal designs. Badloe *et al.*[13] used this technique to successfully optimize the design parameters of a photonic absorption device.

Our motivation is to eventually create an acoustic cloaking device. Once built, this device will render objects invisible to incoming waves in the bandwidth for which it is optimized. The design parameters for our cloaking device are the positioning of multiple cylindrical scatterers on a two-dimensional (2D) grid (position adjustment) or the radius of each cylindrical scatterer on a 2D grid (radius adjustment). Our aim is to compare the performance of two different RL algorithms for each of these parameters. One algorithm chooses from a set of finite discrete actions; the other chooses from continuous actions. We compare the optimal designs that the algorithms produced for the positional and radius design parameters.

A. Related work

In recent years, we have been experiencing the rebirth of ML. While the most mature works are in computer science, we observe the application of innovative data-driven ML models in genomics, natural sciences, physical sciences, engineering, and art. The potential that DL[14] and RL[15] offer for the discovery of new devices and functionalities is drawing attention from a growing community of researchers.[9]

1. ML in acoustics

In acoustics, neural networks (NNs) were employed in the late 1990s and are currently experiencing a revival.[16–18] Jenison[16,19] used spherical basis function NNs for approximating the acoustic scattering of a rigid scatterer. Hesham and El-Gamal[20] solved an integral equation of acoustic

[a)]This paper is part of a special issue on Machine Learning in Acoustics.
[b)]Electronic mail: feruza.amirkulova@sjsu.edu, ORCID: 0000-0002-6348-4941.
[c)]ORCID: 0000-0002-0471-062X.

scattering using wavelet basis and NNs. Bianco *et al.*[21] provided a review of recent applications of ML in acoustics, including the applications of support vector machine (SVM), K-means techniques, dictionary learning, autoencoders, and DL. They specifically discussed recent ML applications in bioacoustics, source localization in ocean acoustics, speaker localization and tracking, signal processing, speech modeling, source separation and enhancement, etc. Cueto and Hadithi[22] designed acoustic metamaterials by means of NNs for the correction of skull-induced aberrations on the ultrasonic beam for neurological applications. Meng *et al.*[23] explored the inverse acoustic scattering problem that reconstructs the obstacle shape with far-field information using NNs. Fan *et al.*[24] studied acoustic scattering by 2D geometries, including circular and elliptical cylinders, and convex prisms as an image-to-image regression problem using convolutional neural networks (CNNs). Fan *et al.*[25] also studied an inverse acoustic scattering problem using CNNs, which predict the object, given the total acoustic field. Liu *et al.*[18] proposed indoor sound source localization algorithm assisted by CNNs for a small-sized microphone array. Komen *et al.*[26] showed that CNNs trained on synthetic spectrograms have the potential to make predictions on the seabed type and source parameters. Morgan *et al.*[17] employed convolutional long short-term memory networks to solve speech emotion regression tasks.

2. DL approaches to inverse design of metamaterials

Various data-driven inverse designs of spinodoid, mechanical, optical, and acoustic metamaterials were recently proposed. Kumar *et al.*[27] proposed a data-driven inverse design of spinodoid metamaterials. Gao and Zhu[28] proposed an inverse design method for acoustic metamaterials. Finol *et al.*[29] showed that CNNs can massively outperform traditional fully connected NNs in solving eigenvalue problems for one-dimensional (1D) and 2D phononic crystals. Pornaras *et al.*[30] presented a method to simulate acoustic multiple scattering by a configuration of cylinders using NNs, where NNs were trained to approximate the total scattering cross section (TSCS) and to solve inverse problems. Wu *et al.*[31] optimized bandgaps of mechanical metamaterials using NNs and genetic algorithms. Gurbuz *et al.*[32] designed acoustic metamaterials for broadband sound insulation using conditional generative adversarial networks (GANs) combined with finite element simulations performed in COMSOL. DL has been used in the inverse design of photonic devices,[33–35] waveguides,[36] and metastructures[22,28,37–39] and in obstacle shape reconstructions,[23] mass transport cloaking,[40] and molecular simulations[41] with superior performance. Deep generative models have been applied in inverse design of molecular components,[5,42–44] metasurfaces,[37,45–48] optical cloak,[49] filters,[50] power splitters,[51] and material microstructure.[52] These generative models have the undesirable property of only being able generate parameters within the limits of the training data. RL is successful in finding the extreme limits of a specific design[53] as discussed next.

3. RL in inverse design

RL has been applied in many disciplines, including robotics,[54] transportation and traffic control,[55] multi-agent systems,[56] quantum physics,[57] and genetic algorithms.[58] Guimaraes *et al.*[59] developed a method (ORGAN) that combines GANs and RL to bias the generation toward desirable metrics. Putin *et al.*[43] introduced NNs for the design of novel small-molecule organic structures based on RL and the GAN paradigm. Sajedian *et al.*[53] studied the colour generation by dielectric nanostructures and optimized the colour generation finding geometrical properties predicted by RL, i.e., Q-learning algorithm. Sajedian *et al.*[60] obtained the optimal geometrical design parameters and material type for metasurface holograms using a double deep Q-learning network (DDQN). Badloe *et al.*[13] used a DDQN algorithm to optimize the parameters of ultra-broadband, moth-eye structure absorbers for several different metallic materials. By exploring and exploiting important regions of the parameter space, their model quickly optimizes the absorption of the structure and chooses the appropriate materials for the substrate and spacer layers to find the highest average absorption over the specified wavelength range. In their design,[13] the authors considered a simpler model. Their system contains fewer parameters, and the absorption coefficients are optimized when the device location is fixed.

B. Proposed model

Our model is more complex as the parameter space is of a higher order. In addition, our model entails a constrained non-convex optimization problem such that the cylinders are free to move within a defined planar region while maintaining a spacing of equal or greater value than the allowable minimal distance. The source codes are for our deep RL models are available on GitHub depository at https://github.com/gladisor/Reinforcement-Learning-Applied-To-Metamaterial-Design.

We discuss the following in the remaining sections: Sec. II A introduces RL methods. Section II B gives an overview of the environment that we have used for RL and provides the definition of the multiple scattering problem. Section II C presents the mechanisms of optimal control. Section II D outlines our proposed gradient assisted approach for inverse design. Section II E addresses our DDQN approach for discrete actions. Section II F discusses our deep deterministic policy gradient (DDPG) approach for continuous actions. In Sec. III, we present the application of the closed form for the gradient of TSCS to acoustic cloak design. Single and broadband scattering suppression effects are illustrated using multiple reconfigurable cylinders as the cloaking mechanism by adjusting positions or radii of each scatterer. Section IV concludes our study and discusses related future work.

https://doi.org/10.1121/10.0005545

II. METHODS AND PROBLEM FORMULATION

A. RL

The main idea of RL is to create a self-improving algorithm that learns an optimal policy in an environment through its own experience or offline data. The entity that selects actions and learns is called an agent. The environment comprises everything outside the agent. As the agent interacts with the environment, it generates data from which it can learn to improve its value function or policy. There is a cycle between policy evaluation and improvement that continues until an optimal policy is discovered. There are two branches of RL algorithms: Q-learning[12] and policy gradients.[61] These two methods differ in the way they represent the action space (set of all possible actions) in the environment; however, they are both driven by the Markov decision process (MDP) described by Sutton and Barto (Ref. 15, pp. 47–71). The MDP provides a mathematical framework for policy iteration and improvement.

The agent interacts with the environment one step at a time as shown in Fig. 1. Starting from an initial state S_t, the agent selects an action A_t and based on this receives a reward R_{t+1} and a state S_{t+1}. This sequence is repeated until a terminal state has been reached, resulting in an episode trajectory[15]

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots. \tag{1}$$

In this setting, the agent's goal is maximizing the sum of rewards received throughout the episode. The sum of rewards from t onward is defined as G_t

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T, \tag{2}$$

where T is the time of termination. In the definition above, the agent values rewards received long after the current time step t just as much as rewards very close to it. By introducing a discount factor parameter γ, $0 \le \gamma \le 1$, we can control how much the agent will value future rewards[15]

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k. \end{aligned} \tag{3}$$

The discounted sum of rewards G_t defines the value of the state that the agent is in or the value of the action taken in that state. Most RL algorithms have a value function or action value function Q_* that is trained to estimate G_t. Below is the Bellman optimality equation for Q_*[15]

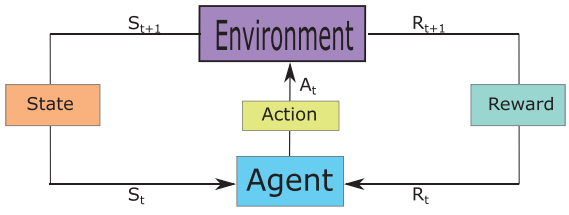

FIG. 1. (Color online) Agent interacting with environment in a MDP.

$$Q_*(s,a) = E\left[R_{t+1} + \gamma \max_a Q_*(S_{t+1}, a) | S_t = s, A_t = a\right], \tag{4}$$

where the expectation is under the distribution of states visited with the current policy. Thus, the Q_* estimates the value function of the optimal policy when repeatedly applied as an update.

B. Acoustic multiple scattering

We consider multiple scattering in the context of the acoustic time harmonic wave equation in two dimensions following Amirkulova and Norris[62] and also Appendix A. The governing equation for the acoustic pressure $p(\mathbf{x})$, $\mathbf{x} \in \mathcal{R}^2$, is the Helmholtz equation

$$\nabla^2 p + k^2 p = 0, \tag{5}$$

where $k = \omega/c$ is the wavenumber, c is the acoustic speed, and ω is the frequency. The total field $p(\mathbf{x})$ is defined as the sum of incident p_{inc} and scattered p_{sc} pressure fields

$$p = p_{inc} + p_{sc}. \tag{6}$$

Let σ denote the TSCS. The TSCS is directly related by the optical theorem to the scattering amplitude in the forward direction, i.e., the direction of propagation of the incident plane wave, here assumed to be $\mathbf{e}_1$ or the x direction. Thus,[63]

$$\sigma = -2\mathrm{Re} f(0), \tag{7}$$

where the far-field amplitude form function $f(\theta) = f(\theta, \mathbf{r}_1, \ldots, \mathbf{r}_M)$, $\theta = \arg(\mathbf{x})$ is defined by the scattered pressure p_{sc} in the far-field given by Eq. (A16), where $\mathbf{r}_1, \ldots, \mathbf{r}_M$ denote the positions of each scatterer depicted in Fig. 16. We chose these scatterers to be circular rigid cylinders for the simplicity of implementation of model and cylindrical thin elastic shells as our eventual goal is to design an acoustic cloaking device made of isotropic materials available in nature. Choosing symmetric shape scatterers allows the computation of TSCS and its gradients with respect to the scatterer positions in a closed form. Providing analytical formulas of gradients to the model enhances the performance of both gradient-based optimization[62] and deep RL algorithms as will be shown in this work.

The TSCS can then be expressed,[62]

$$\sigma(\mathbf{r}_{jm}, ka) = -\frac{4}{k}\mathrm{Re}\mathbf{a}^\dagger \mathbf{b}, \tag{8}$$

where $\mathbf{a}^\dagger$ is the Hermitian transpose, the vectors $\mathbf{a}$ and $\mathbf{b}$ are defined by Eq. (A14), and $\mathbf{r}_{jm} = \mathbf{r}_j - \mathbf{r}_m$ is a position vector of multipole O_m with respect to multipole O_j depicted in Fig. 16. The incident field p_{inc} is a plane wave that interacts with a given configuration of M separate scatterers. The plane wave is propagating from left to right and directed in $\mathbf{e}_1 = (1, 0)$ direction. For simplicity, we take these to be

circularly cylindrical scatterers that may be either rigid cylinders or thin elastic shells. Our objective is to reduce the scattering by rearranging the scatterers. As a measure of the scattering, we use TSCS. We define the root mean square (RMS) of a set of TSCSs over some range of normalized wavenumbers $k_i a\ (i = 1, 2, \ldots, N_k)$

$$\sigma_{RMS}(\mathbf{r}_{jm}) = \left[\frac{1}{N_k}\sum_{i=1}^{N_k}(\sigma(k_i a_j, \mathbf{r}_{jm}))^2\right]^{1/2}, \quad (9)$$

where a_j is the radius of each cylindrical scatterer. Specifically, we consider a range of normalized wavenumbers $k_i a \in [0.35, 0.45]$, where $i = 1, \ldots, 11$. The RL algorithms were able to reduce scattering outside this range at higher wavenumbers. However, the time needed to train them increased to unfeasible levels. The critical quantity that we use in the process is the reward function, which will be defined next in terms of RMS of TSCS, i.e., σ_{RMS}.

C. Optimal control

Our goal is to show that a RL agent can discover design parameters of M scatterers that suppress TSCS across a range of wavenumbers from $k_i a \in [0.35, 0.45]$, $(i = \overline{1, 11})$ to an optimal or near-optimal level. To achieve this goal, we implemented an environment in which an agent can change the design parameters of M scatterers through selecting an action at each time step. The x, y plane that contains the scatterers spans $x, y \in [-5.0, 5.0]m$. To avoid overlapping, the distance between the centers of cylinders/shells is constrained by the allowable minimal distance.[62] The environment is able to reset to an initial random configuration of design parameters. From the starting configuration, the agent selects an action, which changes the parameters and receives a reward from the environment. The cycle of state, action, reward repeats until the terminal state is reached, the episode is complete, and the environment is reset. In this environment, the terminal state occurs after a fixed number of actions are taken. To use a NN to approximate the value of a state-action pair, both must be represented numerically. The representation of actions depends on the RL algorithm in question, while the state representation of the environment is common between both algorithms we use in this paper.

The state vector consists of L design parameters d_i of all the scatterers in the form $[d_1, \ldots, d_L]^T$, the TSCS at N_k wavenumbers, the RMS of the TSCS, and the current time step of the environment scaled between 0 and 1. The dimension of the state vector is $L + N_k + 2$. The final component of the environment is the reward function, which determines how our agent is rewarded for taking actions within the environment

$$R_t = \begin{cases} -\sigma_{RMS}(r_{jm})_t & \text{legal state}, \\ -\sigma_{RMS}(r_{jm})_{t-1} - 1.0 & \text{else}. \end{cases} \quad (10)$$

A legal state requires all cylinder centers to lie within the grid with none overlapping. If the state of the environment is illegal, it reverts to the previous state with a penalty deducted from the reward. This reward function incentivizes the agent to take actions that bring the reward function close to zero quickly to minimize the accumulation of negative rewards while also avoiding illegal states.

A tactic to improve sample efficiency of data in RL is to use a replay buffer to store transition data generated from the environment.[64] Transitions are added to the buffer as the agent interacts with the environment and are sampled in batches to perform gradient updates. We used one-step transitions of the form $S_t, A_t, R_{t+1}, S_{t+1}$. To improve sample efficiency, we employed a prioritized replay buffer in both Q-learning and policy gradient algorithms.[65] Prioritized replay buffers function similarly to standard replay buffers, except the sampling probability is weighted by a priority δ rather than uniformly. The priority for new transitions added to the buffer is set to the maximum to ensure that they are sampled. After each batch update, the sampling priority δ of each transition is defined as[65]

$$\delta = |r + \gamma \max_{a'} Q(s', a') - Q(s, a)|, \quad (11)$$

where Q is an action value function that predicts the discounted sum of future rewards based on the current state s and action a, where s' and a' correspondingly represent the next state and the maximum valued action in that state. The value of δ measures the agent's error in predicting the action value.

To correct for the change in distribution of gradient updates due to prioritized sampling, Schaul *et al.*[65] calls for multiplying each sample update by an importance-sampling weight w_i

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^{\beta}, \quad (12)$$

where hyperparameter $\beta\ (0 < \beta < 1)$ controls how aggressively to correct for the change in distribution, N is the number of samples in the replay buffer, and $P(i)$ is the probability of sampling transition i. Hyperparameters for the prioritized replay buffer in our experiments come from the suggested values given by Schaul *et al.*[65] Values of β outside the recommended ones were not explored.

We consider three design spaces for the RL algorithms. The first design space is the positioning of M cylinders/shells of radius a on a bounded plane. The design parameters in this case are the x and y coordinate of each scatterer. To represent the design parameter vector, we concatenated the M coordinates together and allowed the agent to change the coordinates at each time step. The design parameter vector is $\mathbf{x}^M = [x_1, y_1, \ldots, x_M, y_M]^T$.

The second design space is for large numbers of M scatterers $(M \geq 6)$. It is employed only to the DDPG agent, since a single action moves all cylinders simultaneously. The challenge here is the chance of encountering illegal states, which dramatically increases as the number of scatterers grows. This leads to the environment not generating

https://doi.org/10.1121/10.0005545

useful data in the replay buffer. Instead of reverting the illegal state back to the previous state, we propose to recall the illegal cylinder adjustments only. In this new environment, we check the condition of each adjustment individually. If the resulting configuration is legal, we allow it in the action. This cycle continues until all the adjustments are either applied or withdrawn. Once this process is complete, we apply the action to the environment. This method helps the environment to always output new states for each time step, improving the convergence for large numbers of scatterers; the limitation of this method is the increase in computation time and resources. In our previous environment, only one loop is required to apply an action, whereas the new environment needs one iteration for each cylinder adjustment in the action. Since the new environment always generates a legal state, we can replace the penalty by the number of invalid actions $N_{invalid}$ in the reward function. The new reward function for larger M is defined as

$$R_t = -\sigma_{RMS}(r_{jm})_t - w_p(N_{invalid}), \text{ for } M \geq 6, \tag{13}$$

where w_p is a tunable parameter that adjusts the weight of $N_{invalid}$ in the reward function.

The third design space is the radii of 19 scatterers in two concentric rings around a core unchanging scatterer (see Fig. 15) that is being cloaked. The design parameter for each scatterer is the current radius a, and the parameter vector is $\mathbf{x}^M = [a_1, a_2, \ldots, a_M]^T$. Under this design space, the x and y coordinates of each cylinder, including the core, are held constant. The core radius is also kept constant.

D. Gradient assisted inverse design

The procedure for the gradient assisted inverse design is as follows: we evaluate the TSCS and its gradients with respect to the cylinder coordinates and provide them in the state vector. We define the broadband gradient vectors $\mathbf{q}_j$ with respect to position vectors $\mathbf{r}_j$[62]

$$\mathbf{q}_j = \frac{\partial \sigma_{RMS}(\mathbf{r}_{jm})}{\partial \mathbf{r}_j}, \; j = 1, 2, \ldots M. \tag{14}$$

The explicit formulas for gradient vectors $\mathbf{q}_j$ are given by Eq. (A22). After inserting the gradients $\mathbf{q}_j$ $(j = \overline{1, M})$ into the state vector, its dimension becomes $2L + N_k + 2$. The RL agent minimizes the TSCS by evaluating its derivative with respect to the scatterer coordinates and taking actions accordingly. The gradient of a function points in the direction of greatest increase; likewise, the negative gradient points in the opposite direction. If the scatterer coordinates were adjusted purely based on the gradients as in standard optimization methods, there would be a risk of converging to a local minimal configuration.[62] RL systems, however, consider reward on subsequent actions rather than just the best action in the current state. This means that they consider the gradient but learn to move the scatterers in a different direction, which leads to higher reward in the long term.

E. Q-learning-based formulation

In the Q-learning approach, we use DDQN with several modifications from the rainbow deep Q-learning network approach,[66] including a prioritized replay buffer, a target Q NN, and a dueling network architecture. The DDQN uses two NNs: a Q and a target Q (see Fig. 2). Both of these NNs take in the state vector defined in Sec. II C as input and output the expected discounted sum of rewards (q value) for each discrete action by Eq. (3).

Actions are selected by taking an argmax function on the output of the Q NN, given a state vector. To ensure adequate random exploration of the parameter space, the highest valued action is rejected in favor of a randomly selected

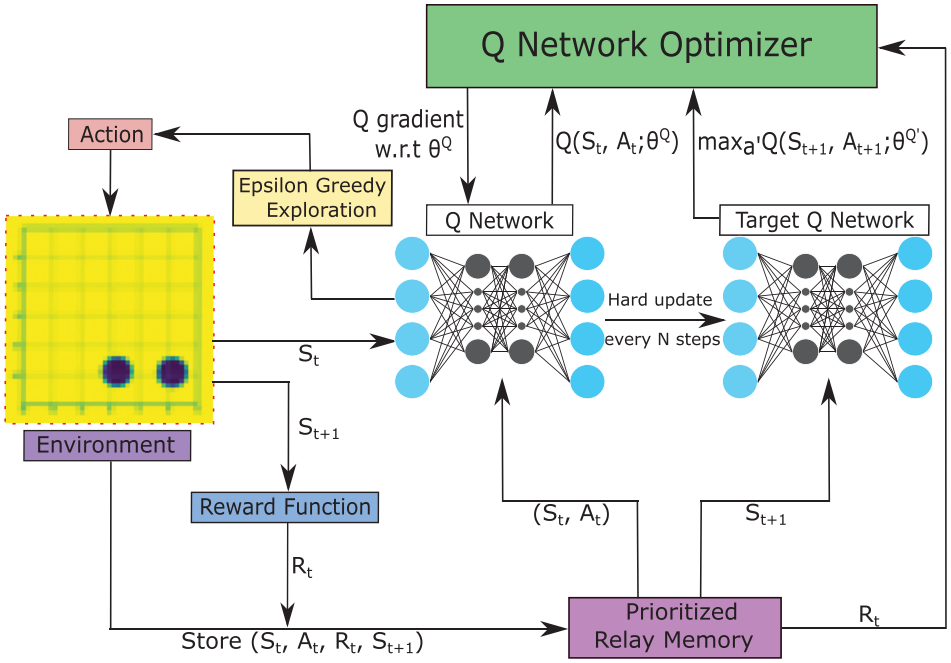

FIG. 2. (Color online) Diagram of DDQN agent interacting with the environment by adjusting design parameters.

TABLE I. Design actions of DDQN for the positional adjustment of M scatterers. Each row corresponds to one output of the Q network.

Scatterer	Action
1	$x - 0.5$ m
1	$y + 0.5$ m
1	$x + 0.5$ m
1	$y - 0.5$ m
M	$x - 0.5$ m
M	$y + 0.5$ m
M	$x + 0.5$ m
M	$y - 0.5$ m

action with probability ϵ for each time step. After each episode, we reduce the probability of selecting a random action ϵ linearly until it reaches a non-zero minimum. This ensures continued exploration while still taking mostly optimal actions. This process is known as ϵ–greedy action selection.[66]

The QNN selects actions to take in the environment. After each action, one update to the NN weights is applied. A single update involves sampling a batch of transitions from the prioritized replay buffer and minimizing the Huber loss[67] between the predicted q value and the target given by Eq. (4). The target NN is used to compute the maximum valued action in the next state. At a regular interval, the parameters of the Q NN θ^Q overwrite the parameters of the target Q NN $\theta^{Q'}$.

DDQNs learn a discrete action space shown in Tables I and II. When using the positions of scatterers as the design parameters, we chose to assign each action to moving one scatterer in one cardinal direction by a distance of 0.5 m. The 0.5 m for position adjustment was chosen empirically. The number of actions is $4M$, where M is the total number of scatterers. For the radius adjustment design, there are two actions for each design parameter: increase or decrease radii. The radius adjustments are made in increments of 0.04 m. This value is derived by linearly scaling the adjustment from the positional design down to the range of allowable radii. The number of actions is fixed at $2(M-1) = 38$, since $M = 20$ is constant and core cylinder radii are not changing. All hyperparameters used for the DDQN algorithm are shown in Table V.

F. Policy gradient-based formulation

To give an agent more fine grain control over the environment than the DDQN, we used an algorithm that can learn a continuous action space. When the agent is far from optimal design, it can make large adjustments and then make sensitive adjustments to settle into a precise design. A DDPG[61] is a policy gradient algorithm with the ability to learn multiple continuous actions (see Fig. 3).

TABLE II. Design actions of DDQN for radius adjustment of 19 scatterers. Each row corresponds to one output of the Q network.

Scatterer	Action
1	Radii + 0.04 m
1	Radii − 0.04 m
19	Radii + 0.04 m
19	Radii − 0.04 m

The DDPG algorithm uses four NNs: an actor and Q, and a target NN for each. The actor NN receives the state vector as input and outputs an action vector. The action vector is the adjustment made to each design parameter. To bind the action vector to a reasonable range, the actor NN has a tanh function on the final layer, which restricts actions between –1 and 1. We multiplied the output by 0.5 to restrict the actions to the same range as DDQN. The Q NN takes in the concatenated state and action vector and predicts the expected discounted sum of rewards (q value) for that state-action pair. The Q NN is trained similarly to the DDQN Q NN by sampling a batch of transitions from the replay buffer and minimizing the Huber loss[67] between the predicted q values and the target q values given by Eq. (4). As the Q NN learns accurate q values, it is used to update the actor NN to output actions that maximize the Q NN output, given a state vector.

After each gradient update to the base NN, a portion of the parameters $(\theta^{\mu}, \theta^{Q})$ are copied over to the target NN parameters $(\theta^{\mu'}, \theta^{Q'})$ using a soft update,[61] $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$. This soft update procedure greatly improves training stability by having the target networks slowly track the base networks.[61] The value of τ used in our experiments was given in the paper by Lillicrap *et al.*,[61] which introduced the DDPG algorithm. According to the paper, smaller values of τ lead to slower but more stable training. By increasing τ, we may be able to decrease convergence times; however, in this research, we only used the suggested value.

Exploration of new states is achived by adding a normally distributed noise to the output of the actor network during training with $\mu = 0$ and $\sigma = 1$. The noise injected action was clipped to be within the proper range. We scaled the noise by a noise scale factor, which is reduced linearly over the training process.

The action space for DDPG is shown in Tables III and IV. When using the DDPG for the positional design, each action output of the DDPG moves one scatterer along one axis simultaneously; therefore, the number of actions is $2M$. For the radius adjustment design, there is a single continuous action for each scatterer. With the number of design scatterers $M = 19$, the total number of outputs is 19. All hyperparameters for the DDPG algorithm were the same across all runs and can be found in Table VI.

III. NUMERICAL EXPERIMENTS AND RESULTS

In this section, we present the results of the RL algorithms after training. Numerical simulations are performed by calling the MATLAB engine from PYTHON libraries. The environment was run on a central processing unit (CPU), while the action selection and deep learning updates for the

https://doi.org/10.1121/10.0005545

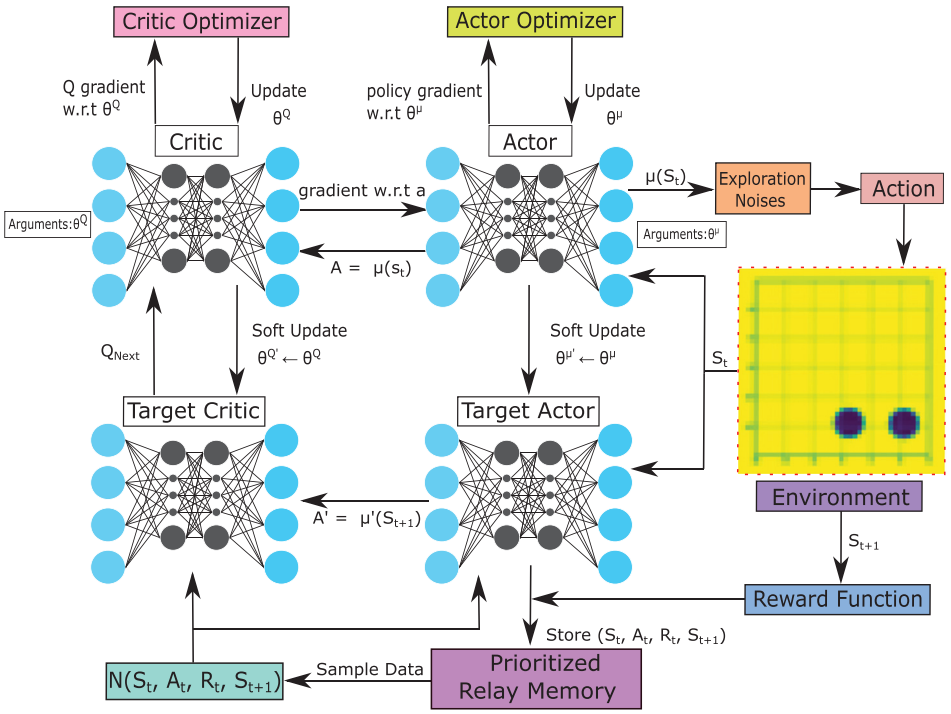

FIG. 3. (Color online) Diagram of DDPG agent interacting with the environment by adjusting cylinder position.

agents were computed on a graphics processing unit (GPU) using PyTorch. For $M \leq 4$ rigid cylinder environments, an Nvidia (Santa Clara, CA) GTX 10 606GB was used. Both algorithms converged in similar times (100 time steps per episode): for the two scatterer environment 200 episodes (14 min), three scatterers 1500 (140 min), and four scatterers 8000 (14 h). For the $M \leq 4$ thin elastic shells environment, an Nvidia RTX 20 708GB was used. Similar to the rigid cylinder environments, episodes were 100 time steps. The agents for thin elastic shells were trained for the same number of episodes (with similar training times) for each M as rigid cylinders. For the design space with $M \geq 4$ (6, 8, 10, and 12 scatter) environments, an Nvidia P-100 was used from the COE HPC cluster[68] of San Jose State University. All training cycles ran with 100 time steps per episode and $wp = 0$, 0.1, 0.01, and 0.001. After experimenting with different weights, we used the following setup for our results: (1) for $M = 6$: $wp = 0$ and episodes $= 7000$, (2) for $M = 8$: $wp = 0.1$ and episodes $= 6500$, (3) for $M = 10$: $wp = 0.1$ and episodes $= 7000$, (4) for $M = 12$: $wp = 0$ and episodes $= 6500$. Compared to $M \leq 4$, the computation time increased linearly with changes in M: from 40 h for six scatterers up to 80 h for 12 scatterers. For the DDPG algorithm, the starting noise scale is an important parameter. For positional designs, the initial noise scale is set to 1.2 and decays to 0.02 over the entire training period, while the radii noise scale starts at 0.25 and decays to 5×10^{-5}.

We consider rigid cylinders submerged in a medium with the acoustic properties of water: $\rho_0 = 1000$ kg/m^3, $c_0 = 1480$ m/s. Additionally, we consider suppression of scattering by configuration of empty thin elastic cylindrical shells situated in the same medium, varying the material properties of the cylinders to compare the resulting optimal TSCS. Separate runs of optimization are done for each material. The material properties we experiment with are nickel and titanium.

A. Adjusting positions of each scatterer for $M \leq 4$

In this section, we illustrate the effect of suppressing plane wave scattering from configurations of rigid and elastic scatterers by modifying the position of each scatterer. We compare the discovered configurations by the DDQN and DDPG algorithms starting from the same initial configurations. Each algorithm was trained on $M = 2$, 3, and 4 scatterer setups. To show the performance of the RL models, we also optimized the RMS of TSCS σ_{RMS} by employing

TABLE III. Design actions of DDPG for positional adjustment of M cylinders. Each row corresponds to one output of the actor network. Each adjustment is bounded $[-0.5, 0.5]m$ per action. The legal range of the axis is $[-5.0, 5.0]m$.

Scatterer	Action
1	x axis adjustment
1	y axis adjustment
M	x axis adjustment
M	y axis adjustment

TABLE IV. Design actions of DDPG for radius adjustment of M scatterers. Each row corresponds to one output of the actor network. Each adjustment is bounded from $[-0.04, 0.04]m$ per action. The legal range of each cylinder radii is $[0.2, 1.0]m$.

Scatterer	Action
1	Radius adjustment
M	Radius adjustment

TABLE V. Hyperparameters of DDQN.

Name	Value	Description
γ	0.9	Controls how much the agent values future reward
ϵ-end	0.1	Final exploration rate at end of training
Target update	10	How many gradient updates between syncing weights with policy net
Memory size	$1e6$	Maximum number of transitions in priority replay buffer
α	0.6	How much to use prioritized sampling
β	0.4	How aggressively to apply importance-sampling weights
Optimizer	SGD	Optimizer for Q network
Learning rate	$5e-4$	Learning rate of optimizer
Momentum	0.9	Momentum of gradient updates
Hidden size	128	Number of neurons in hidden layers
Hidden layers	1	Number of hidden layers of neurons
Activation	ReLU	Nonlinear activation function used between layers
Batch size	256	Number of samples per batch of gradient descent

MATLAB optimization solver *fmincon*[69] with state-of-the-art sequential quadratic (SQP) algorithms. Both the RL algorithms and *fmincon* are very sensitive to initial conditions. Therefore, starting the optimization process from the same initial configuration is crucial to ensure accurate performance evaluation. Computations are performed for a plane wave incident from left to right starting with the same initial random configurations. Experiments for rigid cylinders are presented in Figs. 4–6. Numerical results for elastic thin shells are depicted in Figs. 7–10 (Figs. 7 and 9 for nickel shells and Figs. 8 and 10 for titanium shells).

1. Results for rigid cylinders

The benefit of using the gradient assisted inverse design approach described in Sec. II D is illustrated in Fig. 4 for a configuration of $M=4$ rigid cylinders. The performance of DDQN and DDPG with and without the gradients provided in the state vectors is demonstrated in Figs. 4(a) and 4(b), respectively. Each curve is averaged over three runs. In these runs, a 25 (DDQN) and 5 (DDPG) point rolling median filter was applied to provide additional smoothing. As Fig. 4 shows, both DDQN and DDPG algorithms experience a noticeable increase in their convergence speed and stability. The optimal configurations discovered by the gradient assisted models and with and without providing gradients are similar. However, the gradient assisted models converge faster during training. Decreasing the number of episodes to train is helpful, since training times increase steeply for larger numbers of scatterers.

The variation of TSCS (σ) with the non-dimensional wavenumber ka is shown in Fig. 5. A comparison of suppression of σ curves for configurations of $M=2$, 3, and 4 rigid cylinders is depicted in Fig. 6 using DDQN and DDPG algorithms contrasting with *fmincon* solver using SQP algorithms. Here, we performed a broadband minimization of TSCS (σ) for 11 discrete values of non-dimensional wavenumber $k_i a \in [0.35, 0.45]$, $i = 1, \ldots, 11$. Interestingly, the reduced TSCS values fall not only within but also outside the optimized range of wavenumbers as shown in Fig. 5. For configurations of $M=2$ and 3 cylinders, performance of all three algorithms is similar within the range of wavenumbers

TABLE VI. Hyperparameters of DDPG.

Name	Value	Description
γ	0.9	Controls how much the agent values future reward
Initial noise scale	Position: 1.2, radii: 0.25	Scale of normally distributed noise
Final noise scale	Position: 0.02, radii: 0.00005	Final scale of noise at end of training
τ	0.001	Rate at which the target networks are updated
Memory size	$1e6$	Maximum number of transitions in priority replay buffer
α	0.7	How much to use prioritized sampling
β	0.5	How aggressively to apply importance-sampling weights
Optimizer	Adam	Optimizer used for both networks
Actor learning rate	$1e-4$	Learning rate of actor optimizer
Critic learning rate	$1e-3$	Learning rate of critic optimizer
Critic weight decay	$1e-2$	Regularization term to prevent overfitting
Actor hidden size	128	Number of neurons in hidden layers
Actor hidden layers	2	Number of hidden layers of neurons
Normalization	Layer norm	Normalizes inputs across features
Activation	ReLU	Nonlinear activation function used between layers
Batch size	64	Number of samples per batch of gradient descent

https://doi.org/10.1121/10.0005545

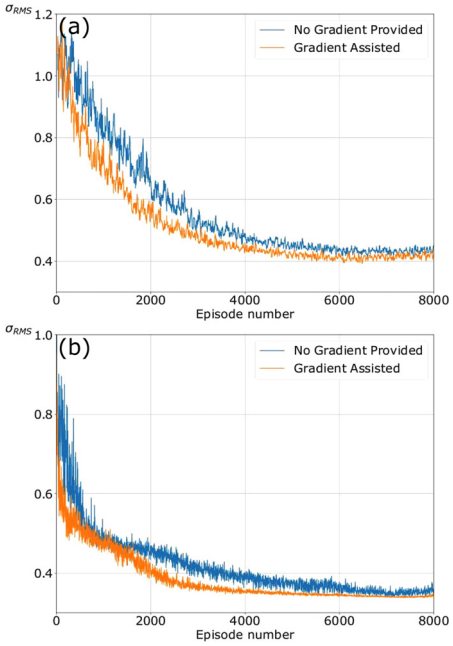

FIG. 4. (Color online) Performance of DDQN (a) and DDPG (b) with and without gradient provided in the state vector for $M = 4$ rigid scatterers.

at which the configurations are optimized, and beyond this range RL algorithms reduce TSCS slightly better. Importantly, for a configuration of $M = 4$ cylinders, both RL algorithms discover a design that produces lower scattering and better results than *fmincon*.

Figure 6 illustrates the final optimal configurations predicted by DDQN, DDPG, and *fmincon* algorithms for $M = 2$, 3, and 4 rigid cylinders. Due to the incident wave coming in the direction $\mathbf{e}_1 = (1, 0)$, configurations shifted up or down produced similar responses. The final optimal configurations discovered by both RL algorithms are very similar to that of *fmincon* for $M = 2$ and 3. For $M = 2$, the values of σ_{RMS} are 0.22 for DDQN, DDPG, and *fmincon*. Likewise, for $M = 3$, values of σ_{RMS} are 0.31 for DDQN, while DDPG and *fmincon* designs are slightly lower at 0.30. These optimal configurations share similar design features in the way the cylinders are positioned as shown in Figs. 6(a)–6(f). They are structured in a row that is perpendicular to the incident wave. All algorithms have determined that the optimal solution is to allow one cylinder to interact with the incident wave while the others are shielded behind it. The discovered designs for $M = 4$ are shown in Figs. 6(g)–6(i). The values of σ_{RMS} are 0.35 and 0.34 for DDQN and DDPG, respectively, and 0.43 for *fmincon*. Both RL algorithms split the four cylinders into two rows parallel to the incident wave at the extremes of the grid. This mimics the design of $M = 2$ but also pushes the rows as far apart as possible to minimize scattering. The *fmincon* design produces more scattering with a diamond shaped configuration as more scatterers are exposed to the plane wave.

2. Results for thin elastic cylindrical shells

Next, we illustrate the TSCS suppression for empty thin cylindrical shells in water. This type of scatterer is quite different from the rigid cylinder because it can display a different scattering pattern compared with rigid cylinders. Numerical results are demonstrated considering configurations of empty thin elastic cylindrical shells of outer radii $a = 1$ m and thickness $h = 0.1a$ with the following mechanical properties correspondingly for nickel shells: density $\rho = 8850\,\text{kg/m}^3$, and longitudinal wave speed $cp = 5480$ m/s, and for titanium shells: $\rho = 4500\,\text{kg/m}^3$, $c_p = 5046.1$ m/s. We compare optimal discovered configurations of thin elastic shells using DDQN and DDPG RL algorithms to our chosen baseline: *fmincon* using SQP algorithms. We varied the material properties of the thin elastic shells to compare the resulting optimal TSCS. At these considered values of wavenumber *ka*, these curves show lower amounts of scattering for both nickel and titanium; however, the initial random configurations are also lower than for rigid cylinders.

Broadband optimization performed for $M = 2$, 3, and 4 is shown in Fig. 7 for thin nickel shells comparing both RL algorithms and *fmincon*. The corresponding optimal configurations discovered by the DDQN and DDPG algorithms are shown in Fig. 9. Notably, for $M = 2$, the DDPG algorithm's configuration outperforms *fmincon* for all *ka* as illustrated in Fig. 7(a). Figure 7(b) compares the suppression for $M = 3$ thin nickel shells, where DDPG outperforms *fmincon* for all *ka* in the optimization range $ka \in [0.35, 0.45]$ with *fmincon*

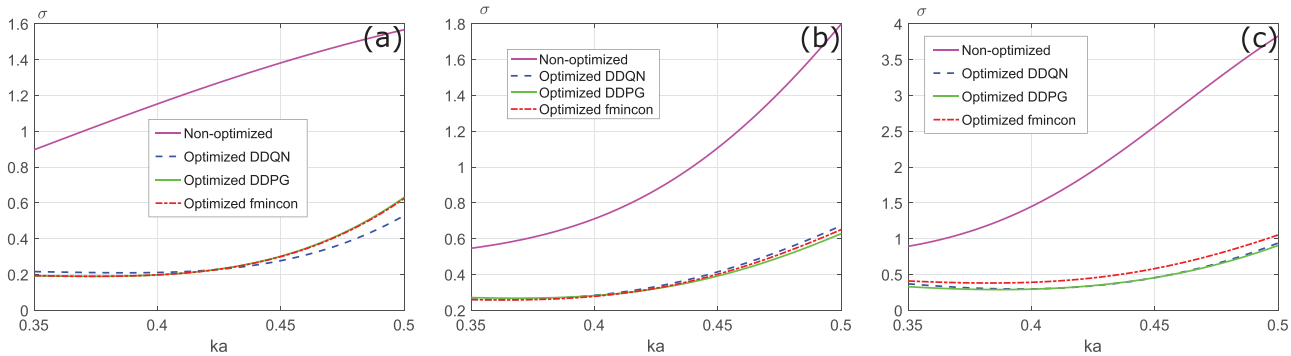

FIG. 5. (Color online) Variation of TSCS vs non-dimensional wavenumber *ka* comparing the suppression of curves for $M = 2$, 3, and 4 rigid cylinders shown in Fig. 6. Since all algorithms discovered similar configurations, the optimal curves for TSCS are overlapping for DDPG and *fmincon* (Ref. 69).

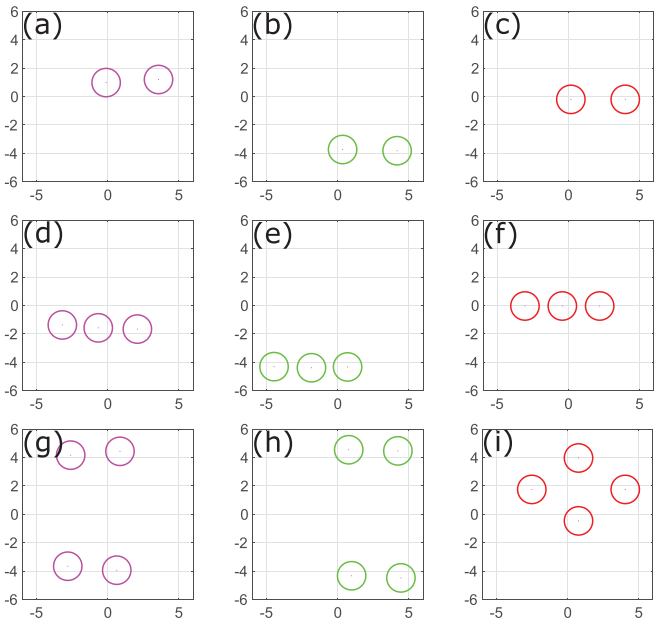

FIG. 6. (Color online) Final optimal configurations proposed by DDQN (pink color), DDPG (green color), and *fmincon* (red color) algorithms and shown in the left, center, and right columns respectively, for $M = 2$, 3, and 4 rigid cylinders.

producing less scattering at the highest *ka*. Figure 7(c) illustrates the suppression curves for $M = 4$ thin nickel shells. We can see that DDPG and *fmincon* discover a configuration with very similar scattering, while the DDQN performs worse.

Figure 8 illustrates DDQN, DDPG, and *fmincon* optimized TSCS vs *ka* for $M = 2$, 3, and 4 titanium thin shells, and the corresponding optimal configurations obtained by these algorithms are shown in Fig. 10. The optimal designs discovered by RL algorithms are very similar to *fmincon* for $M = 2$ and 3 thin titanium shells. For $M = 2$, DDPG and *fmincon* designs produce similar levels of scattering, and DDQN performs slightly worse as shown in Fig. 8(a). All designs are in a triangular configuration; however, neither RL algorithm is able to suppress TSCS more than *fmincon* as shown in Fig. 8(b). Interestingly, for $M = 4$ thin titanium shells, unlike for nickel shells, DDPG surpasses *fmincon* and is able to produce a design that reduces TSCS more than one produced by *fmincon* as shown in Fig. 8(c).

Figure 9 illustrates DDQN and DDPG optimized final configurations for $M = 2$, 3, and 4 scatterer configurations using nickel thin shell material properties starting with the same initial configuration. The configurations discovered by the RL algorithms and shown in Figs. 9(a) and 9(b) move the scatterers apart on the *y* axis rather than the row configurations when using rigid scatterers. The *fmincon* configuration depicted in Fig. 9(c) maintains the row design. The configurations generated by the RL algorithms given in Figs. 9(d) and 9(e) are, once again, different from ones produced by *fmincon* given in Fig. 9(f). Both DDQN and DDPG generate a triangular design with two scatterers in a row at the top and a single scatterer at the bottom. The *fmincon* discovers a row configuration that produces more scattering than the DDPG. Figure 9(g) shows that the DDQN configuration is different from the DDPG and *fmincon* configurations depicted in Figs. 9(h) and 9(i).

Figure 10 presents DDQN and DDPG optimized final configurations for $M = 2$, 3, and 4 titanium thin shell. Designs for $M = 2$ are shown in Figs. 10(a)–10(c). The $M = 3$ designs are also very similar for each algorithm shown in Figs. 10(d)–10(f). Despite the designs for DDQN and DDPG being visually similar as shown in Figs. 10(g) and 10(h), the DDQN design does not perform as well as *fmincon* [Fig. 10(i)].

B. Adjusting positions of rigid cylinders for $6 \leq M \leq 12$

Figures 11–13 demonstrate our findings for DDPG following the procedure described in Sec. II C at a larger number of scatterers *M* and using the reward function defined by Eq. (13). In addition, all DDPG runs in this section are done with gradient assisted design described in Sec. II D.

With the modifications discussed in Secs. II C and II D, the DDPG algorithm was able to discover configurations on a similar order to *fmincon* for $M = 6$ and 8. The DDQN was tested but not able to successfully converge for $M > 4$ scatterers. Most importantly, in the $M = 6$ environment, the DDPG algorithm was able to outperform *fmincon* as shown by the TSCS vs *ka* curves in Fig. 11(a). Although the DDPG

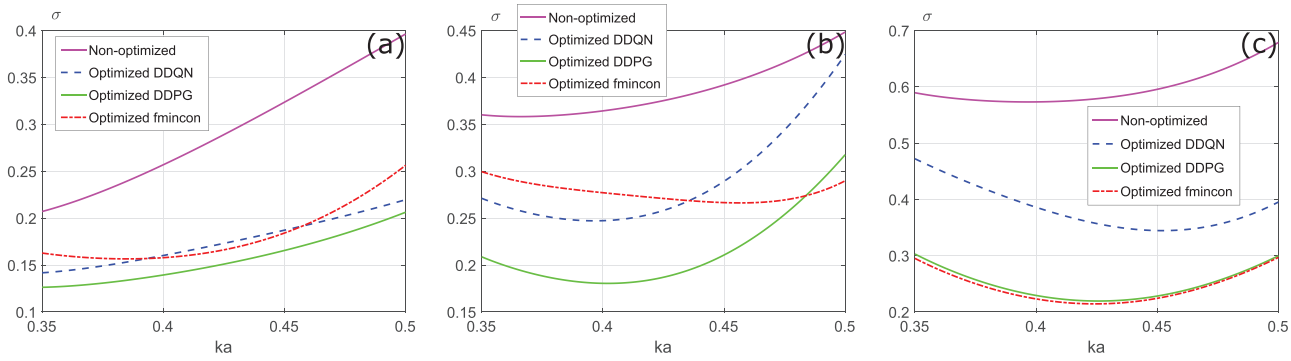

FIG. 7. (Color online) Variation of TSCS vs *ka* comparing suppression of curves for $M = 2$, 3, and 4 scatterers for thin nickel shells. For configuration of $M = 2$ and 3 thin nickel shells, DDPG surpasses *fmincon* and performs better at the range of wavenumbers at which the configurations are optimized and beyond this range.

https://doi.org/10.1121/10.0005545

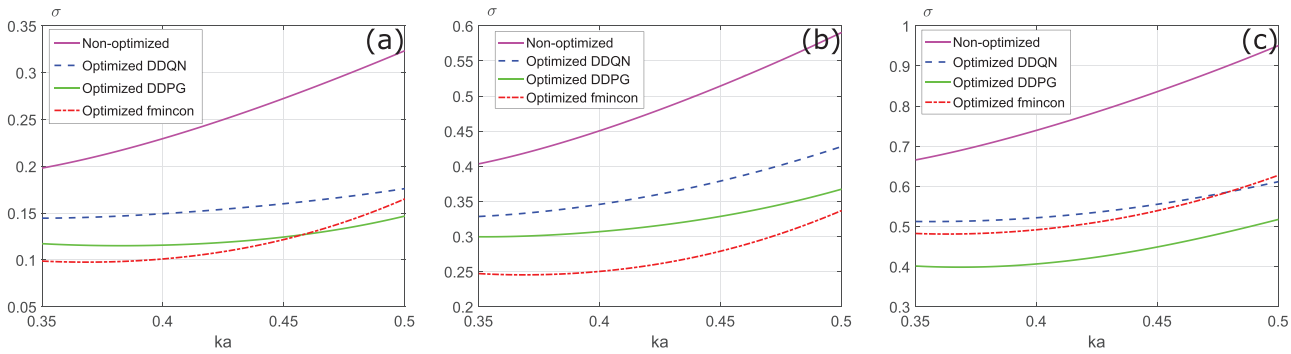

FIG. 8. (Color online) Optimized TSCS vs *ka* results comparing elastic titanium thin shells for $M = 2$, 3, and 4 scatterers. All optimized configurations exhibit a final suppressed TSCS response. For configuration of $M = 4$ thin titanium shells, DDPG outperforms *fmincon* within and beyond the range of wavenumbers at which the structure is optimized.

and *fmincon* algorithms discovered a design in the same local minimal configuration, minor differences cause the *fmincon* design to have a lower TSCS for larger *ka* shown in Fig. 11(b). Results from the $M = 10$ environment show that the DDPG algorithm struggles to compete with *fmincon* at this *M*. Figure 11(c) shows TSCS values for DDPG are higher than *fmincon* for all *ka*, diverging more as *ka* increases. Finally, for the $M = 12$ design, the $5.0m$ grid size was no longer large enough to ensure adequate exploration for the DDPG. In this environment, we increased the grid size to $8.0m$. The resulting configurations for DDPG and *fmincon* are shown in Figs. 12(k) and 12(l) to be discussed next.

Figure 12 illustrates initial and final optimized configurations for a larger number of scatterers, $M \geq 6$, with the left column figure panels depicting initial random configurations for $M = 6$, 8, 10, and 12. Figure 12(b) splits the six scatterers into two rows of three as far apart as possible. This design performs better than that of *fmincon* given in Fig. 12(c), which splits the cylinders into a row of four and a row of two. For the $M = 8$ environment, DDPG and *fmincon* have very similar designs as illustrated in Figs. 12(e) and 12(f). Due to the larger number of scatterers, clustering them in straight rows no longer appears to be the best strategy. Comparing the optimized configurations in Figs. 12(h) and 12(i), we see different designs. The design produced by the DDPG algorithm splits the cylinders into three rows, with the bottom row containing four staggered cylinders. The *fmincon* algorithm uses a similar configuration for the top cluster in $M = 8$ reflected on the *x* axis. Finally, for the $M = 12$ design, the $5.0m$ grid size was no longer large enough to ensure adequate exploration for the DDPG. In this

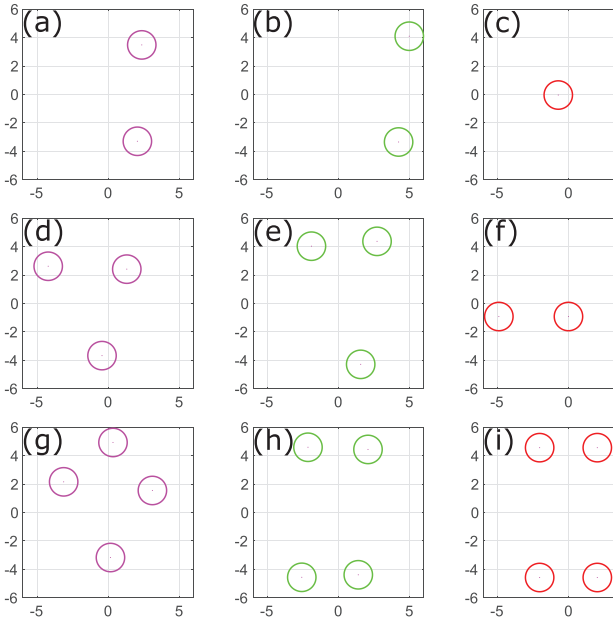

FIG. 9. (Color online) Optimal configurations for thin elastic nickel shells discovered by various algorithms for $M = 2$, 3, and 4. The left, center, and right columns are for DDQN (pink color), DDPG (green color), and *fmincon* (red color), respectively. The initial configurations are the same as in Fig. 6.

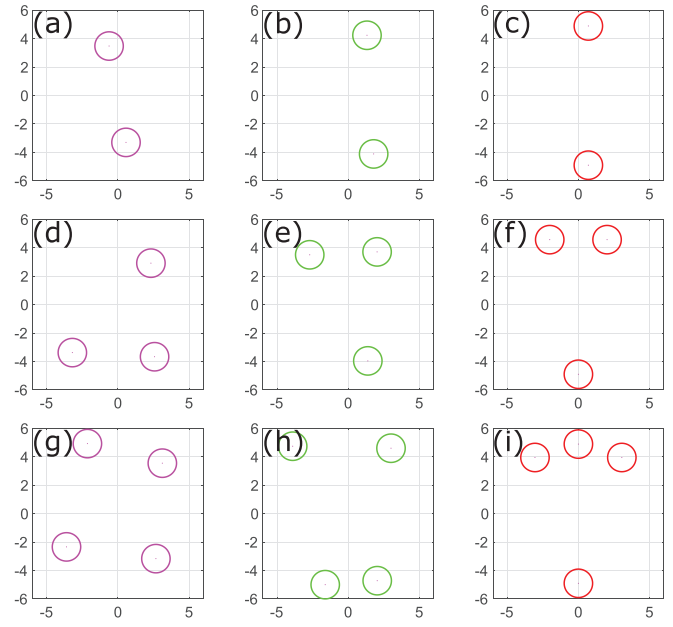

FIG. 10. (Color online) Final optimal configurations proposed by DDQN, DDPG, and *fmincon* algorithms for thin titanium shells for $M = 2$, 3, and 4. The left, center, and right columns are for DDQN (pink color), DDPG (green color), and *fmincon* (red color), respectively.

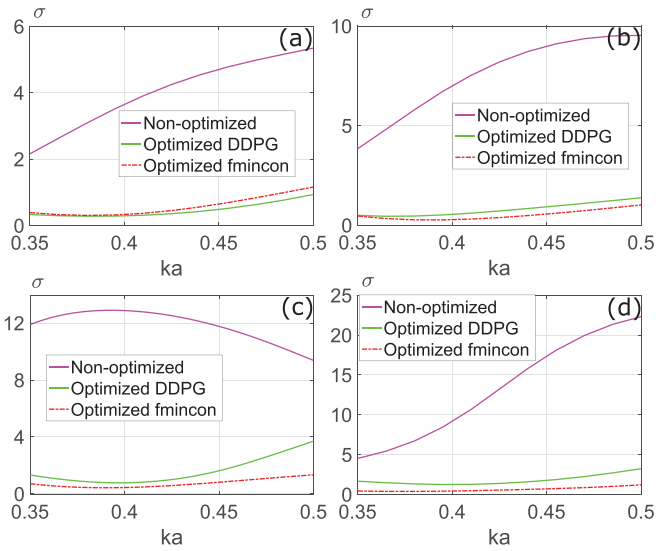

FIG. 11. (Color online) Variation of TSCS vs ka comparing suppression of curves for $M = 6$, 8, 10, and 12 rigid scatterers following the procedure described in Sec. II C for DDPG at a larger number of scatterers and using a reward function defined by Eq. (13).

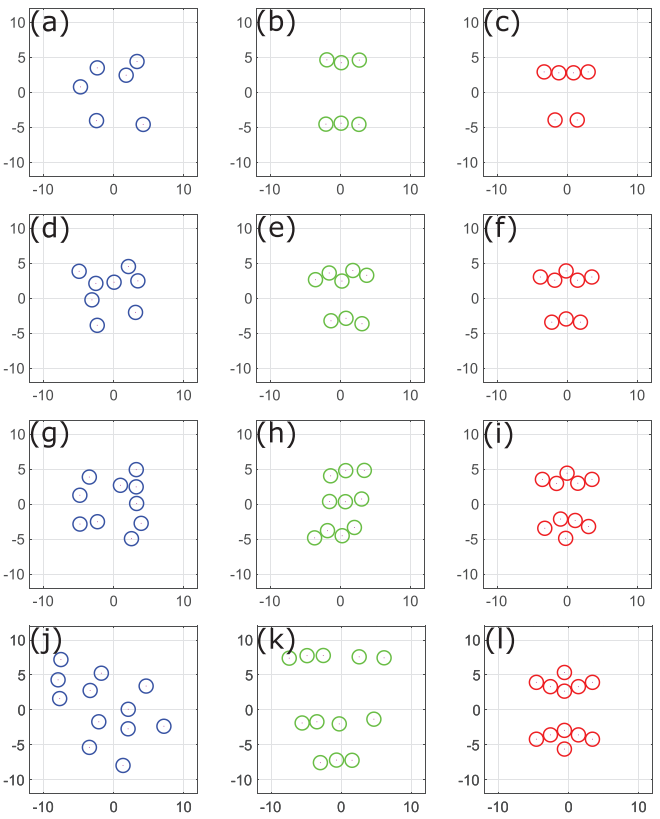

FIG. 12. (Color online) Initial non-optimized configurations (blue color, left column) and final optimal configurations of DDPG (green color, center column) and *fmincon* (red color, right column) for $M = 6$, 8, and 10 rigid cylinders. Values of σ_{RMS} are within the optimization range: $k_i a \in [0.35, 0.45]$, $i = 1, \ldots 11$. For $M = 6$, initial, optimal DDPG, and optimal fmincon σ_{RMS} values are 3.65, 0.34, and 0.42, respectively. For $M = 8$, σ_{RMS} values are 6.96, 0.63, and 0.38. For M = 10, σ_{RMS} values are 12.48, 1.09, and 0.58. Last, for M = 12, σ_{RMS} values are 10.81, 1.44, and 0.49.

environment, we increased the grid size to $8.0m$. The resulting configurations for DDPG and *fmincon* are shown in Figs. 12(k) and 12(l). Due to the larger grid size, the DDPG configuration is able to cluster cylinders in three straight rows, while *fmincon* produces a star-like configuration.

Figure 13 illustrates the total acoustic pressure distribution, at normalized wavenumber $ka = 0.45$, for different configurations of $M = 10$ rigid cylinders depicted in Fig. 12(h). Figures 13(a) and 13(b) depict, correspondingly, the real part of total pressure field p at single value of wavenumber $ka = 0.45$ for a random initial configuration [Fig. 12(g)] and final optimal configuration [Fig. 12(h)] found by the DDPG algorithm. The real part of total acoustic pressure field, Rep, exhibits a suppressed scattering response and a less disturbed wave propagation for the final optimal configuration depicted in Fig. 13(b). Figures 13(c) and 13(d) show, respectively, the absolute total pressure, $|p|$, at $ka = 0.45$ for an initial [Fig. 12(g)] and final [Fig. 12(h)] optimal configuration predicted by DDPG. Figure 13(c) exhibits very strong backscattering from the initial configuration. Figure 13(d) illustrates different scattering characteristics and suppression of backscattering effect as well as showing the overall reduction of absolute pressure amplitude for the optimized configuration.

C. Adjusting radii of each scatterer

Finally, we show the ability of the RL agents to discover design parameters of radii that suppress TSCS using rigid cylindrical scatterers. Initial random radii and optimized radii configurations are demonstrated as well as their corresponding TSCS curves across wavenumbers $k_i a \in [0.35, 0.45]$, $i = \overline{1, 11}$, where $a = a_{\max} = \max(a_j)$, a_j is the radius of j-cylinder, and $j = \overline{1, M}$. Coordinates of all scatterers are fixed, and the core cylinder radius is constant $a_c = 1.6 = a_{\max}$. All other cylinder radii besides the core are allowed to change within the bounds of minimum and maximum radii defined in Sec. II C.

The TSCS curves in Fig. 14 show that both RL algorithms produce a suppression of σ for all wavenumbers for a configuration of $M = 20$ rigid cylinders. The initial random configuration and final optimal configurations predicted by RL models are depicted in Fig. 15. The DDPG produces a configuration that causes more reduction within the optimization range, while the DDQN performs slightly worse in that range. Since the formula for the gradient of σ_{RMS} with respect to each scatterer radius has not yet been derived, we have not compared the RL algorithms to *fmincon*, which will be done elsewhere.

Figure 15 illustrates the real part of total acoustic pressure Rep at normalized wavenumber $ka = 0.4$ (left column) and $ka = 0.72$ (right column) for different configurations of $M = 20$ rigid cylinders of various radii, i.e., random initial and final optimal configurations predicted by the DDPG and DDQN algorithms. Figures 15(c) and 15(d) show the optimal design discovered by the DDQN, while Figs. 15(e) and 15(f)

https://doi.org/10.1121/10.0005545

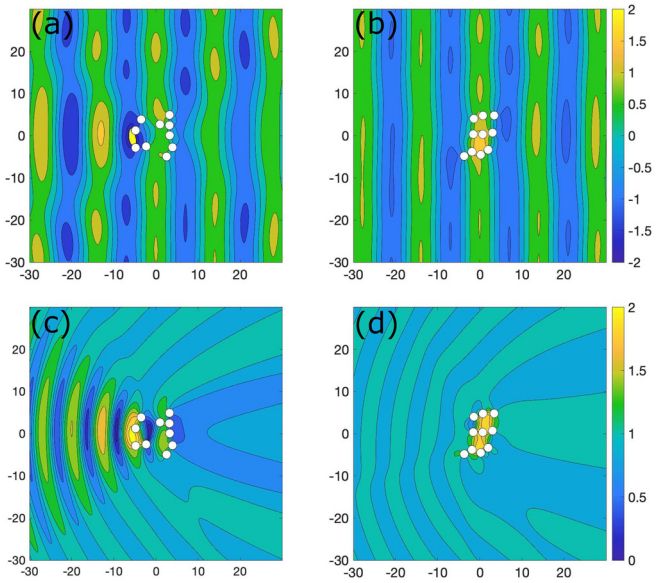

FIG. 13. (Color online) The real part of total acoustic pressure field, Rep, and the absolute total pressure $|p|$ at normalized wavenumber $ka = 0.45$ for different configurations of $M = 10$ rigid cylinders for a random initial configuration [(a) and (c)] and final optimal configuration predicted by the DDPG algorithm [(b) and (d)]. The Rep for the final optimal configuration exhibits a suppressed scattering response. It also illustrates a less disturbed wave propagation for the optimal configuration. (c) and (d) show the reduction of absolute pressure amplitude $|p|$ for the final optimal configuration.

show the discovered optimal design for DDPG. Both algorithms started at the same initial random radii [Figs. 15(a) and 15(b)]. The DDPG agent discovered a design that is slightly different and more effective than the DDQN's. Interestingly, enlarging the scatterers closest to the incident plane wave produces better suppression. There are two common features between the designs discovered by the algorithms. First, the cylinders farthest from the wave are increased to near the maximum allowable radii. This suggests some wave interaction occurs that is beneficial, rather than reducing the radii to the minimum and allowing the wave to pass through. Second, scatterers in the middle of the x axis are brought close to the minimal allowable radii.

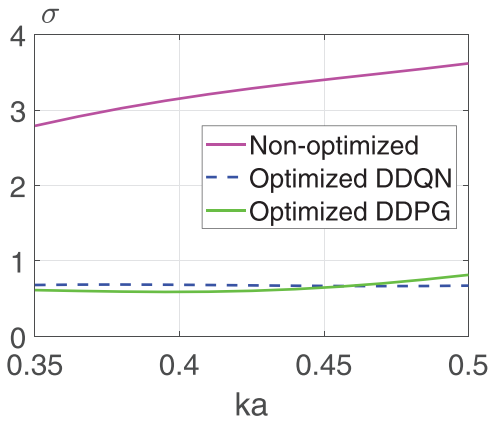

FIG. 14. (Color online) TSCS against normalized wavenumber ka for initial, DDQN, and DDPG optimized configurations of $M = 20$ rigid cylinders of radii a_j, where $ka = k_{\max}(a_j)$ and $j = 1, \ldots, M$.

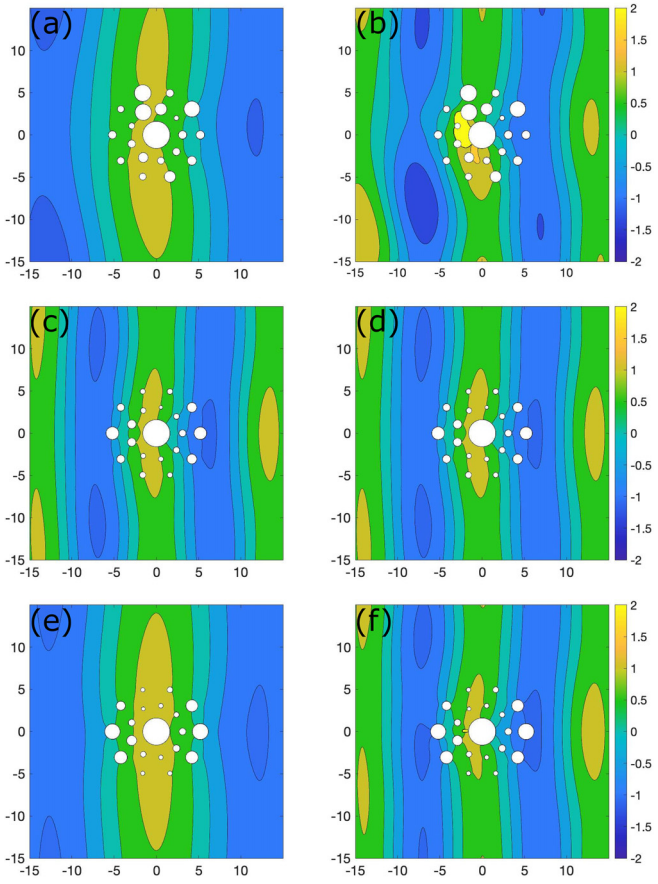

FIG. 15. (Color online) The real part of the total acoustic pressure for configurations of rigid cylindrical scatterers of different radii. Top, middle, and bottom rows represent initial, DDQN optimized, and DDPG optimized configurations, respectively. The left column panels (a), (c), and (e) show results for the wavenumber $ka = 0.4$ at which the configuration was optimized, and the right column figures show Rep for the wavenumber $ka = 0.72$, which is beyond the range of wavenumbers at which the configuration was optimized, i.e., $ka \in [0.35, 0.45]$.

The algorithms have discovered that these cylinders are not useful in suppressing σ_{RMS}.

The initial random radii configuration with $M = 20$ (a) produced $\sigma_{RMS} = 3.13$. The DDQN algorithm discovered a configuration (c) with $\sigma_{RMS} = 0.68$. Last, the DDPG discovered a configuration (e) with slightly better results with $\sigma_{RMS} = 0.60$. Although the optimal radii discovered by the DDPG produced lower σ_{RMS} than the DDQN, there is evidence to suggest it could suppress it even further. Figure 18(a) shows the lowest σ_{RMS} discovered by the DDPG algorithm per episode during training. During the training process, the lowest σ_{RMS} decreased to 0.50, although the convergence was not stable. Attempts to stabilize the performance to that level were unsuccessful.

IV. CONCLUSION AND OUTLOOK

We demonstrated that the RL algorithms DDQN and DDPG are capable of discovering positional and radius design parameters of scatterers that suppress the RMS of TSCS to a local minimum. We also introduced a novel

method for increasing convergence speed and stability of RL algorithms by using the gradient of the objective function with respect to design parameters. As far as we know, our paper is the first description of this approach, and RL algorithms are attempted for the first time to suppress acoustic scattering. We discovered that DDPG almost always outperforms DDQN for any number of scatterers and performed very well compared to *fmincon* with SQP algorithms.

Our results show that using RL agents for design optimization is viable in acoustics and engineering. This enables new approaches for modeling acoustic devices. A limitation of these RL algorithms is that as the number of scatterers increases, the training time increases very rapidly. It was shown[62] that an optimal configuration of 79 scatterers suppressed TSCS at high frequency to near zero; at lower frequencies, 16 cylinders were sufficient and produced a reasonable cloaking effect. Perhaps if we were able to scale RL algorithms to efficiently handle many cylinders, we could achieve a similar result. Additionally, we can experiment with different frequency ranges. We initially trained the algorithms at high frequencies; however, this dramatically increased training times as well. A potential method of scaling to many cylinders and higher frequencies could involve a multi-agent approach[56,70] that would split the design parameters between agents working cooperatively to maximize a shared reward while reducing the complexity.

A solution to the large training times could involve splitting the inference and training of the model into multiple threads as was shown by Espeholt *et al.*[71] This approach allowed the training to be paralellized across multiple CPUs and greatly decreased convergence times for RL algorithms.

APPENDIX A: MULTIPLE SCATTERING

1. Formulation

Consider acoustic scattering by M obstacles, which for simplicity are taken to be cylinders $\mathrm{S}^{(m)}$ $(m=\overline{1,M})$ centered at $\mathbf{r}_m$. A schematic configuration of cylindrical elastic shells is given in Fig. 16. The incident wave of unit amplitude is in direction $\mathbf{e}_1=(1;0)$ and in the neighborhood of cylinder $\mathrm{S}^{(m)}$, it can be written as

$$p_{inc}^{(m)}(\mathbf{x})=\sum_{n=-\infty}^{\infty}A_n^{(m)}U_n^{+}(\mathbf{x}_m), \tag{A1}$$

with the coefficients $A_n^{(m)}=\mathrm{i}^n e^{\mathrm{i}kx_m}$ and $U_n^{+}(\mathbf{x})=J_n(k|\mathbf{x}|)$ $e^{\mathrm{i}n\arg\mathbf{x}}$, are the regular solutions associated with the radiating functions V_n^{+} defined as

$$V_n^{+}(\mathbf{x})=H_n^{(1)}(k|\mathbf{x}|)e^{\mathrm{i}n\arg\mathbf{x}}, \tag{A2}$$

$$V_n^{+'}(\mathbf{x})=H_n^{(1)'}(k|\mathbf{x}|)e^{\mathrm{i}n\arg\mathbf{x}}. \tag{A3}$$

Here, $\arg\mathbf{x}\in[0,2\pi)$, and $\mathbf{x}_m$ is a position vector of point P with respect to the centers of multipoles at O_m (see Fig. 16): $\mathbf{x}_m=\mathbf{x}-\mathbf{r}_m$. The total scattered field p_{sc}, considered as a superposition of the fields scattered by all cylinders, may be expanded as a sum of multipoles

$$p_{sc}(\mathbf{x})=\sum_{m=1}^{M}p_{sc}^{(m)}(\mathbf{x}), \tag{A4a}$$

$$p_{sc}^{(m)}(\mathbf{x})=\sum_{n=-\infty}^{\infty}B_n^{(m)}V_n^{+}(\mathbf{x}_m), \tag{A4b}$$

where $p_{sc}^{(m)}$ is the wave scattered by cylinder m, and $B_n^{(m)}$ are unknown coefficients. To apply boundary conditions on the surface of each cylinder, we express the total field using Graf's theorem [Ref. 72, Eq. (9.1.79)]

$$V_l^{+}(\mathbf{x}-\mathbf{y})=\sum_{n=-\infty}^{\infty}\begin{cases}V_n^{+}(\mathbf{x})\,U_{n-l}^{-}(\mathbf{y}), & |\mathbf{x}|>|\mathbf{y}|,\\ U_n^{+}(\mathbf{x})\,V_{n-l}^{-}(\mathbf{y}), & |\mathbf{x}|<|\mathbf{y}|,\end{cases} \tag{A5}$$

where

$$U_n^{-}(\mathbf{x})=J_n(k|\mathbf{x}|)e^{-\mathrm{i}n\arg\mathbf{x}}=(-1)^nU_{-n}^{+}(\mathbf{x}),$$

$$V_n^{-}(\mathbf{x})=H_n^{(1)}(k|\mathbf{x}|)e^{-\mathrm{i}n\arg\mathbf{x}}=(-1)^nV_{-n}^{+}(\mathbf{x}).$$

The $U_n^{\pm}$ and $V_n^{\pm}$ functions therefore satisfy

$$W_n^{+}(\mathbf{x})=W_{-n}^{-}(-\mathbf{x}),\ W=U,\ V. \tag{A6}$$

Using Graf's theorem for $|\mathbf{x}_j|<l_j$, where $l_j=\min|\mathbf{r}_{jm}|$, we obtain the total incident field impinging on the cylinder $\mathrm{S}^{(j)}$ in the form[62,73]

$$p_{inc}^{(j)}+\sum_{\substack{m=1\\ m\neq j}}^{M}p_{sc}^{(m)}=\sum_{n=-\infty}^{\infty}\left\{A_n^{(j)}+\sum_{\substack{m=1\\ m\neq j}}^{M}\sum_{l=-\infty}^{\infty}P_{nl}(\mathbf{r}_{jm})B_l^{(m)}\right\}U_n^{+}(\mathbf{x}_j), \tag{A7}$$

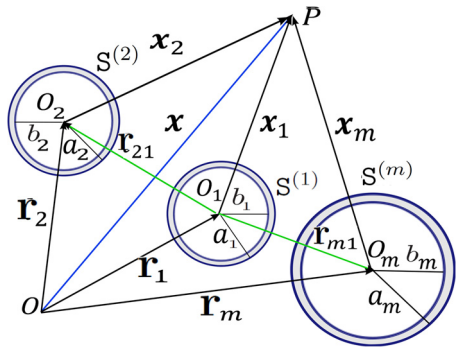

FIG. 16. (Color online) An arbitrary planar configuration of M cylinders $\mathrm{S}^{(m)}$ with outer radius a_m and inner radius b_m, $m=\overline{1,M}$.

https://doi.org/10.1121/10.0005545

where $\mathbf{r}_{jm} = \mathbf{r}_j - \mathbf{r}_m$, and

$$P_{ql}(\mathbf{r}_{jm}) = H_{l-q}^{(1)}(kr_{jm})e^{\mathrm{i}(l-q)\arg\mathbf{r}_{jm}} \tag{A8}$$

can be identified as $V_{l-n}^{+}(\mathbf{x})$. Here, $H_n^{(1)}$ is the Hankel function of the first kind of order n. The response of cylinder $\mathbb{S}^{(j)}$ to the incident field (A7) can be obtained by incorporating the boundary conditions at the interface and the transition matrix elements $T_{nq}^{(j)}$ of cylinder $\mathbb{S}^{(j)}$[74,75]

$$p_{sc}^{(j)} = \sum_{n=-\infty}^{\infty}\sum_{q=-\infty}^{\infty} T_{nq}^{(j)}\left\{A_q^{(j)} + \sum_{\substack{m=1\\ m\neq j}}^{M}\sum_{l=-\infty}^{\infty} P_{ql}(\mathbf{r}_{jm})B_l^{(m)}\right\} V_n^{+}(\mathbf{x}_j). \tag{A9}$$

Equations (A4) and (A9) yield a linear system of equations for the unknowns $B_l^{(m)}$

$$\sum_{m=1}^{M}\sum_{l=-\infty}^{\infty} X_{jnml}B_l^{(m)} = A_n^{(j)}, \quad j = \overline{1,M},\ n \in \mathbb{Z}, \tag{A10a}$$

$$X_{jnml} = \begin{cases} T_{nl}^{(j)-1}, & j = m, \\ -P_{nl}(\mathbf{r}_{jm}), & j \neq m', \end{cases} \tag{A10b}$$

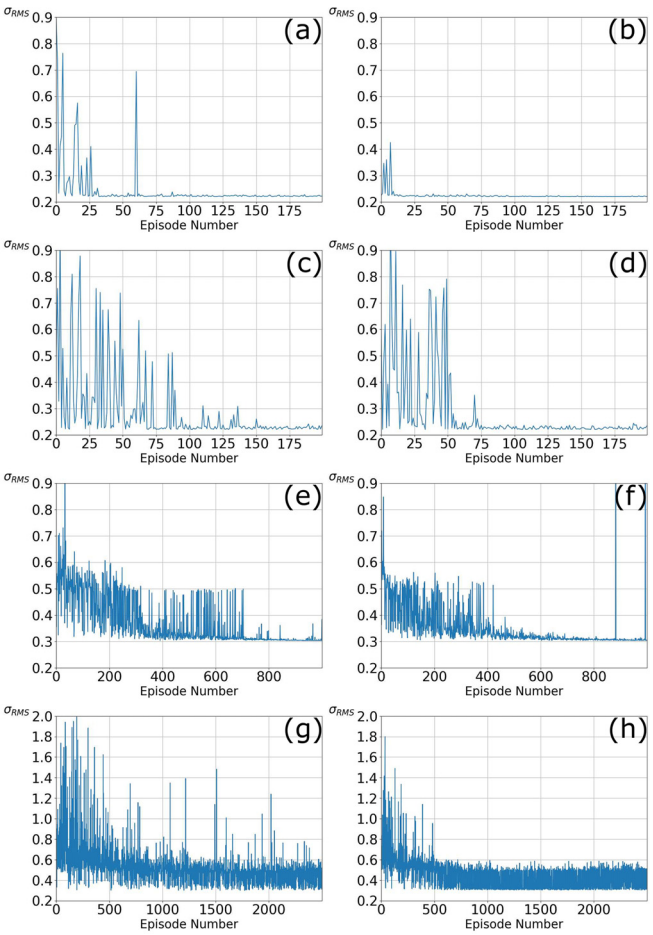

FIG. 17. (Color online) The lowest σ_{RMS} discovered per episode during training. These are results by varying the algorithms used, number of scatterers, and use of gradient assisted inverse design.

where $T_{nq}^{(j)-1}$ are elements of the inverse of the jth T-matrix. The truncated version of the infinite sum in Eq. (A10a) yields a finite algebraic system of equations[62,71]

$$\sum_{m=1}^{M}\sum_{l=-N}^{N} X_{jnml}B_l^{(m)} = A_n^{(j)}, \quad n \in (-N,N), \tag{A11}$$

for $j = 1, 2, \ldots M$, or in matrix form

$$\mathbb{X}\,\mathbf{b} = \mathbf{a}, \tag{A12}$$

where $\mathbb{X}$, $\mathbf{b}$, and $\mathbf{a}$ are of the forms

$$\mathbb{X} = \begin{bmatrix} \mathbf{T}^{(1)-1} & -\mathbf{P}^{1,2} & -\mathbf{P}^{1,3} & \ldots & -\mathbf{P}^{1,M} \\ -\mathbf{P}^{2,1} & \mathbf{T}^{(2)-1} & -\mathbf{P}^{2,3} & \ldots & -\mathbf{P}^{2,M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\mathbf{P}^{M,1} & -\mathbf{P}^{M,2} & -\mathbf{P}^{M,3} & \ldots & \mathbf{T}^{(M)-1} \end{bmatrix}, \tag{A13}$$

and

$$\mathbf{a} = \begin{pmatrix} \mathbf{a}^{(1)} \\ \vdots \\ \mathbf{a}^{(M)} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \mathbf{b}^{(1)} \\ \vdots \\ \mathbf{b}^{(M)} \end{pmatrix}, \tag{A14a}$$

$$\mathbf{a}^{(j)} = \begin{pmatrix} A_{-N}^{(j)} \\ A_{-N+1}^{(j)} \\ \vdots \\ A_{N}^{(j)} \end{pmatrix}, \mathbf{b}^{(j)} = \begin{pmatrix} B_{-N}^{(j)} \\ B_{-N+1}^{(j)} \\ \vdots \\ B_{N}^{(j)} \end{pmatrix}. \tag{A14b}$$

Figure 17 shows the training curves of $M = 2$ and 3 for DDQN and DDPG for positional adjustment of rigid cylinders. Each algorithm is compared without gradients (left column) and with gradients (right column). All the same hyperparameters were used for corresponding runs. Using our proposed gradient assisted method results in faster convergence for each pair of runs. Despite improving performance for $M = 3$ DDQN in Figs. 17(g) and 17(h), the algorithm was still not able to converge nearly as stably as the DDPG algorithm shown in Figs. 17(e) and 17(f).

Finally, the scattered field p^{sc} of Eq. (A4) in the far-field, $k|\mathbf{x}| \gg 1$, becomes

$$p^{sc} = f(\theta)\sqrt{\frac{k}{\mathrm{i}2\pi|\mathbf{x}|}}e^{\mathrm{i}k|\mathbf{x}|}\left[1 + O\left(\frac{1}{k|\mathbf{x}|}\right)\right], \tag{A15}$$

where the far-field amplitude function is

$$f(\theta) = \frac{2}{k}\sum_{m=1}^{M} e^{-\mathrm{i}k|\mathbf{r}_m|\cos(\theta-\arg(\mathbf{r}_m))} \times \sum_{n=-\infty}^{\infty}(-\mathrm{i})^n B_n^{(m)} e^{\mathrm{i}n\theta}. \tag{A16}$$

2. Analytical form of the gradient vectors $\mathbf{s}_j$ at single *ka*

Define the gradient vectors

$$\mathbf{s}_j = \frac{\partial \sigma}{\partial \mathbf{r}_j},\ j = 1, 2, \ldots M. \tag{A17}$$

Following Eqs. (8) and (A12), the TSCS can be expressed as

$$\sigma = -\frac{4}{k}\mathrm{Re}\mathbf{a}^{\dagger}\mathbb{X}^{-1}\mathbf{a}. \tag{A18}$$

The gradient of the TSCS, $\mathbf{s}_j$, is found by combining the last equation [Eq. (A18)] for σ with the definition [Eq. (A17)] and may be written as[62]

$$\mathbf{s}_j = -\frac{4}{k}\mathrm{Re}\left[\frac{\partial \mathbf{a}^{\dagger}}{\partial \mathbf{r}_j}\mathbf{b} + \mathbf{a}^{\dagger}\mathbb{X}^{-1}\left(\frac{\partial \mathbf{a}}{\partial \mathbf{r}_j} - \frac{\partial \mathbb{X}}{\partial \mathbf{r}_j}\mathbf{b}\right)\right], \tag{A19}$$

where $\mathbf{a}^{\dagger}$ is the Hermitian transpose and the vectors $\mathbf{a}$ and $\mathbf{b}$ are defined by Eq. (A14),

$$\frac{\partial X_{inml}}{\partial \mathbf{r}_j} = \begin{cases} O_{nl}\delta_{ij}, & i = m, \\ \delta_{jm}\dfrac{\partial P_{nl}}{\partial \mathbf{r}_j}(\mathbf{r}_{ji}) - \delta_{ji}\dfrac{\partial P_{nl}}{\partial \mathbf{r}_j}(\mathbf{r}_{jm}), & i \neq m, \end{cases} \tag{A20}$$

and O_{nl} are components of the zero matrix. The gradients in Eq. (A20) follow from

$$\begin{aligned}\frac{\partial P_{nl}}{\partial \mathbf{r}_j}(\mathbf{r}_{jm}) = &\frac{k}{r_{jm}}V_{l-n}^{+\prime}(\mathbf{r}_{jm})\,\mathbf{r}_{jm} \\ &+ \frac{\mathrm{i}(l-n)}{r_{jm}^2}V_{l-n}^{+}(\mathbf{r}_{jm})\,\mathbf{e}_3 \times \mathbf{r}_{jm},\end{aligned} \tag{A21}$$

where V_n^+ and the derivative function $V_n^{+\prime}$ are defined by Eq. (A2). The gradient of the components of $\mathbf{a}$ with respect to the position of the jth scatterer is $\partial \mathbf{a}^{(m)}/\partial \mathbf{r}_j = \delta_{jm}\mathbf{a}^{(m)} \otimes \mathrm{i}k\,\mathbf{e}_1 \in \mathcal{C}^{M\times(2N+1)} \times \mathcal{C}^2$.

3. Broadband gradient vectors $\mathbf{q}_j$

We defined our cost function to be minimized as the RMS of a set of TSCSs over some range of normalized wavenumbers $k_i a\,(i = 1, \ldots, N_k)$, i.e., $\sigma_{RMS}(\mathbf{r}_{jm})$ [Eq. (9)]. Broadband gradient vectors $\mathbf{q}_j$ are defined by Eq. (14). The explicit form of gradients can be found in terms of the individual single frequency gradients as

$$\mathbf{q}_j = \frac{1}{\sigma_{RMS}}\frac{1}{N_k}\left[\sigma(k_1 a)\mathbf{s}_j(k_1 a) + \cdots + \sigma(k_{N_k} a)\mathbf{s}_j(k_{N_k} a)\right], \tag{A22}$$

where $\mathbf{s}_j(k_i a)$ are evaluated at normalized wavenumbers $k_i a\,(i = 1, \ldots, N_k)$ by using Eq. (A19).

APPENDIX B: TRAINING PROCESS DETAILS

1. Training curves

Figure 18 compares the DDPG and DDQN algorithms' training curves for radius adjustment of rigid cylinders. The

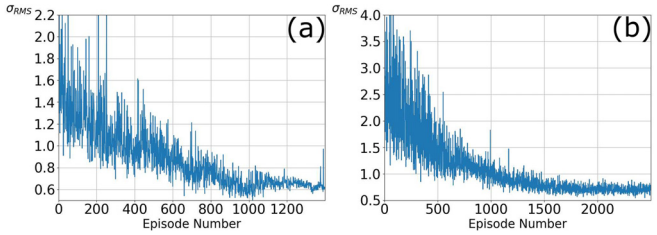

FIG. 18. (Color online) The lowest σ_{RMS} discovered per episode during training for a configuration of $M = 20$ rigid cylinders by adjusting the radii of each scatterer in rings while keeping core cylinder radius unchanged and varying the algorithms: (a) DDQN and (b) DDPG without using the gradient information.

DDPG algorithm converges in fewer episodes (1400) as shown in Fig. 18(a), but it is much more unstable than the DDQN. This is surprising because for most other environments the DDPG consistently found a better configuration and converged more reliably than the DDQN. Further tuning of hyperparameters could potentially help the DDPG discover better designs.

2. Hyperparameters

In machine learning, hyperparameters are settings of an algorithm that must be chosen before training a model. They are usually not learnable parameters like the weights and biases of the model itself, but rather initial conditions that influence how the model learns. Tables V and VI show the hyperparameters used by the DDQN and DDPG algorithms. These settings were used for all runs of these algorithms. The only changes made for environments are the size of the input and output layers of the NNs.

[1]A. N. Norris, "Acoustic cloaking theory," Proc. R. Soc. A **464**, 2411–2434 (2008).
[2]A. Climente, D. Torrent, and J. Sánchez-Dehesa, "Sound focusing by gradient index sonic lenses," Appl. Phys. Lett. **97**, 104103 (2010).
[3]P. Packo, A. Norris, and D. Torrent, "Metaclusters for the full control of mechanical waves," arXiv:2009.13376 (2020).
[4]V. Popov, F. Boust, and S. N. Burokur, "Beamforming with metagratings at microwave frequencies: Design procedure and experimental demonstration," IEEE Trans. Antennas Propag. **68**(3), 1533–1541 (2020).
[5]K. Butler, D. Davies, H. Cartwright, O. Isayev, and A. Walsh, "Machine learning for molecular and materials science," Nature **559**(7715), 547–555 (2018).
[6]D. Elton, Z. Boukouvalas, M. Fuge, and P. Chung, "Deep learning for molecular design—A review of the state of the art," Mol. Syst. Des. Eng. **4**(4), 828–849 (2019).
[7]J. Noh, G. H. Gu, S. Kim, and Y. Jung, "Machine-enabled inverse design of inorganic solid materials: Promises and challenges," Chem. Sci. **11**(19), 4871–4881 (2020).
[8]R. S. Hegde, "Deep learning: A new tool for photonic nanostructure design," Nanoscale Adv. **2**(3), 1007–1023 (2020).
[9]S. So, T. Badloe, J. Noh, J. Rho, and J. Bravo-Abad, "Deep learning enabled inverse design in nanophotonics," Nanophotonics **9**(5), 1041–1057 (2020).
[10]S. Campbell, D. Sell, E. Jenkins, R. Whiting, J. Fan, and D. Werner, "Review of numerical optimization techniques for meta-device design," Opt. Mater. Express **9**(4), 1842–1863 (2019).
[11]D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," arXiv:1712.01815 (2017).

[12]V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," arXiv:1312.5602v1 (2013).
[13]T. Badloe, I. Kim, and J. Rho, "Biomimetic ultra-broadband perfect absorbers optimised with reinforcement learning," Phys. Chem. Chem. Phys. **22**(4), 2337–2342 (2020).
[14]I. Goodfellow, Y. Bengio, and, and A. Courville, *Deep Learning* (MIT, Cambridge, MA, 2016).
[15]R. S. Sutton and A. G. Barto, *Reinforcement Learning* (MIT, Cambridge, MA, 2018).
[16]R. Jenison, "A spherical basis function neural network for approximating acoustic scatter," J. Acoust. Soc. Am. **99**(5), 3242–3245 (1996).
[17]M. M. Morgan, I. Bhattacharya, R. J. Radke, and J. Braasch, "Classifying the emotional speech content of participants in group meetings using convolutional long short-term memory network," J. Acoust. Soc. Am. **149**(2), 885–894 (2021).
[18]N. Liu, H. Chen, K. Songgong, and Y. Li, "Deep learning assisted sound source localization using two orthogonal first-order differential microphone arrays," J. Acoust. Soc. Am. **149**(2), 1069–1084 (2021).
[19]R. L. Jenison, "Models of direction estimation with spherical-function approximated cortical receptive fields," in *Central Auditory Processing and Neural Modeling*, edited by P. Poon and J. Brugge (Plenum, New York, 1998), pp. 161–174.
[20]M. Hesham and M. El-Gamal, "Neural network model for solving integral equation of acoustic scattering using wavelet basis," Commun. Numer. Methods Eng. **24**, 183–194 (2006).
[21]M. Bianco, P. Gerstoft, J. Traer, E. Ozanich, M. A. Roch, S. Gannot, and C.-A. Deledalle, "Machine learning in acoustics: Theory and applications," J. Acoust. Soc. Am. **146**(5), 3590–3628 (2019).
[22]C. Cueto and B. Hadithi, "Cancelling out skull-induced aberrations: Analysis of acoustic metamaterials using neural networks," IEEE Latin Am. Trans. **15**(10), 1948–1959 (2017).
[23]P. Meng, L. Su, W. Yin, and S. Zhang, "Solving a kind of inverse scattering problem of acoustic waves based on linear sampling method and neural network," Alex. Eng. J. **59**(3), 1451–1462 (2020).
[24]Z. Fan, V. Vineet, H. Gamper, and N. Raghuvanshi, "Fast acoustic scattering using convolutional neural networks," in *Proceedings of ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain (May 4–8, 2020).
[25]Z. Fan, V. Vineet, C. Lu, T. W. Wu, and K. McMullen, "Prediction of object geometry from acoustic scattering using convolutional neural networks," arXiv:2010.10691 (2021).
[26]D. Komen, T. B. Neilsen, D. B. Mortenson, M. C. Acree, D. P. Knobles, M. Badiey, and W. S. Hodgkiss, "Seabed type and source parameters predictions using ship spectrograms in convolutional neural networks," J. Acoust. Soc. Am. **149**(2), 1198–1210 (2021).
[27]S. Kumar, S. Tan, L. Zheng, and D. M. Kochmann, "Inverse-designed spinodoid metamaterials," NPJ Comput. Mater. **6**(1), 73–83 (2020).
[28]H. Gao and J. Zhu, "Inverse design method for acoustic metamaterials," J. Acoust. Soc. Am. **146**(4), 2828–2828 (2019).
[29]D. Finol, Y. Lu, V. Mahadevan, and A. Srivastava, "Deep convolutional neural networks for eigenvalue problems in mechanics," Numer. Methods Eng. **118**(5), 258–275 (2019).
[30]D. Pornaras, W.-C. Wang, Y. Chen, G. Kwak, F. Amirkulova, and E. Khatami, "Broadband suppression of total multiple scattering cross section using neural networks," J. Acoust. Soc. Am. **146**(4), 2876–2877 (2019).
[31]L. Wu, L. Liu, Y. Wang, Z. Zhai, H. Zhuang, D. Krishnaraju, Q. Wang, and H. Jiang, "A machine learning-based method to design modular metamaterials," Extreme Mech. Lett. **36**, 100657 (2020).
[32]C. Gurbuz, F. Kronowetter, C. Dietz, M. Eser, J. Schmid, and S. Marburg, "Generative adversarial networks for the design of acoustic metamaterials," J. Acoust. Soc. Am. **149**(2), 1162–1174 (2021).
[33]Y. Peurifoy, J. Shen, L. Jing, Y. Yang, F. Cano-Renteria, B. DeLacy, J. Joannopoulos, M. Tegmark, and M. Soljačić, "Nanophotonic particle simulation and inverse design using artificial neural networks," Sci. Adv. **4**(6), eaar4206 (2018).
[34]E. Bor, O. Alparslan, M. Turduev, Y. S. Hanay, H. Kurt, S. Arakawa, and M. Murata, "Integrated silicon photonic device design by attractor selection mechanism based on artificial neural networks: Optical coupler and asymmetric light transmitter," Opt. Express **26**(22), 29032 (2018).
[35]Z. Liu, Y. Tan, E. Khoram, and Z. Yu, "Training deep neural networks for the inverse design of nanophotonic structures," ACS Photonics **5**, 1365–1369 (2018).
[36]T. Zhang, J. Wang, Q. Liu, J. Zhou, J. Dai, X. Han, Y. Zhou, and K. Xu, "Efficient spectrum prediction and inverse design for plasmonic waveguide systems based on artificial neural networks," Photonics Res. **7**(3), 368 (2019).
[37]J. Jiang and J. A. Fan, "Global optimization of dielectric metasurfaces using a physics-driven neural network," Nano Lett. **19**(8), 5366–5372 (2019).
[38]S. Inampudi and H. Mosallaei, "Neural network based design of metagratings," Appl. Phys. Lett. **112**(24), 241102 (2018).
[39]G. Oliveri and J. T. Overvelde, "Inverse design of mechanical metamaterials that undergo buckling," Adv. Funct. Mater. **30**(12), 1909033 (2020).
[40]R. Khodayi and M. Zavlanos, "Deep learning for robotic mass transport cloaking," IEEE Trans. Robot. **36**(3), 967–974 (2020).
[41]R. Barrett, M. Chakraborty, D. Amirkulova, H. Gandhi, and A. White, "A GPU-accelerated machine learning framework for molecular simulation: HOOMD-blue with TensorFlow," ChemRxiv:8019527 (2019).
[42]B. Sanchez-Lengeling and A. Aspuru-Guzik, "Inverse molecular design using machine learning: Generative models for matter engineering," Science **361**(6400), 360–365 (2018).
[43]E. Putin, A. Asadulaev, Y. Ivanenkov, V. Aladinskiy, B. Sanchez-Lengeling, A. Aspuru-Guzik, and A. Zhavoronkov, "Reinforced adversarial neural computer for de novo molecular design," J. Chem. Inf. Model. **58**(6), 1194–1204 (2018).
[44]Y. Dan, Y. Zhao, X. Li, S. Li, M. Hu, and J. Hu, "Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials," NPJ Comput. Mater. **6**(1), 84–91 (2020).
[45]J. Fan, "Freeform metasurface design based on topology optimization," MRS Bull. **45**(3), 196–201 (2020).
[46]J. Jiang and J. Fan, "Simulator-based training of generative neural networks for the inverse design of metasurfaces," Nanophotonics **9**(5), 1059–1069 (2019).
[47]J. Jiang, D. Sell, S. Hoyer, J. Hickey, J. Yang, and J. Fan, "Free-form diffractive metagrating design based on generative adversarial networks," ACS Nano **13**(8), 8872–8878 (2019).
[48]F. Wen, J. Jiang, and J. A. Fan, "Robust freeform metasurface design based on progressively growing generative networks," ACS Photonics **7**, 2098–2104 (2020).
[49]A. Blanchard-Dionne and O. Martin, "Successive training of a generative adversarial network for the design of an optical cloak," OSA Contin. **4**(1), 87–95 (2021).
[50]X. Han, Z. Fan, Z. Liu, C. Li, and L. J. Guo, "Inverse design of metasurface optical filters using deep neural network with high degrees of freedom," InfoMat **3**, 432–442 (2021).
[51]Y. Tang, K. Kojima, T. Koike-Akino, Y. Wang, P. Wu, M. Tahersima, D. Jha, K. Parsons, and M. Qi, "Generative deep learning model for a multi-level nano-optic broadband power splitter," in *Proceedings of Optical Fiber Communication Conference 2020*, San Diego, CA (March 8–12, 2020).
[52]R. Tan, N. Zhang, and W. Ye, "A deep learning–based method for the design of microstructural materials," Struct. Multidis. Optim. **61**(4), 1417–1438 (2020).
[53]I. Sajedian, T. Badloe, and J. Rho, "Optimisation of colour generation from dielectric nanostructures using reinforcement learning," Opt. Expr. **27**(4), 5874–5883 (2019).
[54]A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, "End-to-end robotic reinforcement earning without Reward engineering," arXiv:1904.07854 (2019).
[55]B. Abdulhai and L. Kattan, "Reinforcement learning: Introduction to theory and potential for transport applications," Can. J. Civil Eng. **30**(6), 981–991 (2003).
[56]R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," arXiv:1706.02275 (2017).
[57]J. Yao, M. Bukov, and L. Lin, "Policy gradient based quantum approximate optimization algorithm," Proc. Machine Learning Res. **107**, 605–634 (2020).
[58]F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive

alternative for training deep neural networks for reinforcement learning," arXiv:1712.06567 (2017).
[59]G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, "Objective-reinforced generative adversarial networks (organ) for sequence generation models," arXiv:1705.10843v3 (2017).
[60]I. Sajedian, H. Lee, and J. Rho, "Double-deep Q-learning to increase the efficiency of metasurface holograms," Sci. Rep. **9**(1), 10899 (2019).
[61]T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971 (2015).
[62]F. A. Amirkulova and A. N. Norris, "The gradient of total multiple scattering cross-section and its application to acoustic cloaking," J. Theor. Comput. Acoust. **28**, 1950016 (2020).
[63]A. N. Norris, "Acoustic integrated extinction," Proc. R. Soc. A **471**(2177), 20150008 (2015).
[64]S. Zhang and R. S. Sutton, "A deeper look at experience replay," arXiv:1712.01275 (2017).
[65]T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv:1511.05952 (2015).
[66]M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," arXiv:1710.02298 (2017).
[67]P. J. Huber, "Robust estimation of a location parameter," Ann. Math. Statist. **35**(1), 73–101 (1964).
[68]"The College of Engineering high performance computing system," San Jose State University, http://coe-hpc-web.sjsu.edu (Last viewed 4 July 2021).
[69]"When the solver fails. MathWorks MATLAB documentation," https://www.mathworks.com/help/optim/ug/when-the-solver-fails.html (Last viewed 4 July 2021).
[70]M. Wooldridge, *An Introduction to MultiAgent Systems* (Wiley, New York, 2009).
[71]L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, "SEED RL: Scalable and efficient deep-RL with accelerated central inference," arXiv:1910.06591 (2019).
[72]M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (Dover, New York, 1974).
[73]F. A. Amirkulova, "Acoustic and elastic multiple scattering and radiation from cylindrical structures," Ph.D. thesis, Rutgers University, Piscataway, NJ, 2014.
[74]V. K. Varadan and V. V. Varadan, eds., *Acoustic, Electromagnetic and Elastic Wave Scattering - Focus on the T-Matrix Approach* (Pergamon, New York, 1980).
[75]F. Amirkulova and A. Norris, "Acoustic multiple scattering using fast iterative techniques," in *Proceedings of 2017 ASME International Mechanical Engineering Congress and Exposition (IMECE 2017)*, Tampa, FL (November 3–9, 2017), Paper No. IMECE2017/72249.