Supplementary Information

Structural Optimization of a One-Dimensional Freeform Metagrating Deflector via Deep Reinforcement Learning

Dongjin Seo[1,2†], Daniel Wontae Nam[2†], Juho Park[1], Chan Y. Park[2], and Min Seok Jang[1]**

[1] School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, Republic of Korea

[2] KC Machine Learning Lab, Seoul 06181, Republic of Korea

* chan.y.park@kc-ml2.com

* jang.minseok@kaist.ac.kr

[†] These authors contributed equally to this work.

The Supplementary file includes:

S1. Algorithm Summary

Table S1. Algorithm tables of our work.

Algorithm 1 Agent-Environment Interaction

```
1:  Given initial device structure $s_0 = \{+1, -1, \ldots, +1\}$, network parameters
    $\omega$, target network parameters $\omega^-$, environment, learning rate $\alpha$, discount
    factor $\gamma$
2:  Set Current device structure $s = s_0$
3:  Initialize target network parameters $\omega^- \leftarrow \omega$
4:
5:  for number of episodes = {0, ..., K-1} do
6:      while episode not done do                                  ▷ Running an episode
7:          $a = \begin{cases} \arg\max_{a'} Q^{\omega}(s, a') & 1-\epsilon \\ a' \sim \mathcal{U}\{0, \mid A \mid -1\} & \epsilon \end{cases}$      ▷ $\epsilon$-greedy, unit cell index to flip
8:          $s'[a] \leftarrow -s[a]$                                 ▷ Update device structure
9:          $r = \eta^3$                                    ▷ Calculate reward using efficiency
10:         store transition $(s, a, r, s')$ in Experience Replay
11:
12:         if Train stage then
13:             Randomly sample $(s, a, r, s')$ from Experience Replay
14:             $y = r + \gamma \max_{a'} Q^{\omega^-}(s', a')$                    ▷ Bellman Target
15:             $L = \text{Huber}(y, Q^{\omega}(s, a))$
16:             $\omega \leftarrow \text{Adam}(\omega)$                             ▷ Adam Optimization
17:
18:
19:         if Update target network then
20:             $\omega^- \leftarrow \omega$                                      ▷ Copy parameters
```

Algorithm 2 Greedy algorithm

```
1:  Given initial device structure $s_0$, environment, step $t$, max depth of greedy
    algorithm $d_{max}$, cell number $N$
2:  Set Current device structure $s = s_0$
3:  while algorithm not done do
4:      for search depth $d = \{0, ..., d_{max} - 1\}$ do
5:          for cell index $i = \{0, ..., N - 1\}$ do
6:              Take action $a = i$
7:              Collect $\eta$
8:      Choose $a_{\{0,\ldots,d-1\}} = \arg\max_{a'_{\{0,\ldots,d-1\}}} \eta$
```

S2. Results of Optimization

Table S2. Tabular Data of Fig4: percentage values of maximum, mean, and standard deviation of discovered deflection efficiency distribution derived from each algorithm. For RL, 3 random network initializations were used for statistics. For RL, bold letter of maximum efficiency represents the condition where RL outperforms the others.

Maximum value [%]		Greedy Algorithm		Adjoint-based	GLOnet	RL
Wavelength [nm]	Angle [°]	Depth=1	Depth=2			
900	50	46.8	81.0	93	98	**98.7**
	60	68.2	99.2	93	97	**99.7**
	70	77.3	79.7	92	98	**98.3**
1000	50	41.4	96.5	95	96	**96.9**
	60	73.4	99.0	92	98	98.8
	70	67.2	58.1	84	93	**94.7**
1100	50	30.2	71.2	91	91	**98.4**
	60	29.9	79.8	79	80	**87.1**
	70	50.8	58.7	84	84	78.5

Mean value [%]		Greedy Algorithm		Adjoint-based	GLOnet	RL
Wavelength [nm]	Angle [°]	Depth=1	Depth=2			
900	50	46.8	81.0	64	90	97.6
	60	68.2	99.2	59	73	99.5
	70	77.3	79.7	59	83	98.0
1000	50	41.4	96.5	55	85	96.3
	60	73.4	99.0	56	85	98.7
	70	67.2	58.1	62	76	94.4
1100	50	30.2	71.2	49	77	98.3
	60	29.9	79.8	52	59	86.9
	70	50.8	58.7	59	65	78.3

Standard deviation [%]		Greedy Algorithm		Adjoint-based	GLOnet	RL
Wavelength [nm]	Angle [°]	Depth=1	Depth=2			
900	50	0	0	16	10	1.05
	60	0	0	18	18	0.2
	70	0	0	13	14	0.3
1000	50	0	0	16	12	0.5
	60	0	0	14	17	0.1
	70	0	0	12	18	0.4
1100	50	0	0	10	11	0.05
	60	0	0	15	17	0.2
	70	0	0	14	14	0.1

Table S3. Optimized structures for 9 input conditions. 1 represents Si and 0 represents air. The text information of each structure can be changed from {+1, 0} to {+1, -1} and directly applied to RETICOLO simulation program powered by MATLAB. The structure files are also accessible as numpy structure file in the '/structures' folder of github with link: https://github.com/dongjin-seo2020/1DFreeFormDQN.

Wavelength [nm]	Angle [°]	Structure
900	50	11111101011111111100111111111100 01010000000010101101011101010001
	60	11010010110011110010111001110011 11111101111111110010111111111111
	70	11111011101101011101110110111011 11111011111111110110111111111111
1000	50	11111110111111111111000110010001 00010111101000101101011111111011
	60	10001101110011111111011111111110 01111111111111111100000010110110
	70	11011010000000001101111111110111 11111110100111111111111111111100
1100	50	11110000010010000001111110011100 11011111111101111111111111111111
	60	11110000001011010000000010111110 11111011111111101111111111111111
	70	00010000100000101011111111011101 11111110111111111111111111111000

S3. Implementation Details

Designing RL experiments often requires simultaneous alteration of a large number of hyperparameters. In Table S4, we provide the list and values of parameters we optimized and used for our algorithm. Furthermore, we run an ablation study on selected hyperparameters that are likely to have an impact on the results. The results are plotted and presented in Figure S1. It shows that change of parameters, within a logical range, does not affect the performance at a significant degree.

Table S4. Hyperparmameter list and setup

Variable Name	**Outline**	**Value**
ncells	number of unit cells for one grating period	64
nG	diffraction order of RCWA algorithm	40
buf	size of replay buffer	1,000,000
gamma	discount factor	0.99
stepnum	overall learning step number	2,000,000
eps_greedy_period	step period that decreases epsilon	1,000,000
minimum_epsilon	minimum value of epsilon for epsilon greedy algorithm during training	0.01
epilen	overall step number of one episode	128
train_start_memory_size	step number that training of neural network begins	5,000
lr	learning rate of training	0.001
batch_size	batch size of training	512
train_num	number of training when the function *train*() is called	1

train_step	step period that training happens	2
merge_step	step period that merging from Q network to Target network happens	20,000
tau	merging proportion of Q network and target network (τ:1-τ)	0.1
val_num	number of episodes in the validation stage	10
printint	episode period that the intermediate results of learning are printed	50

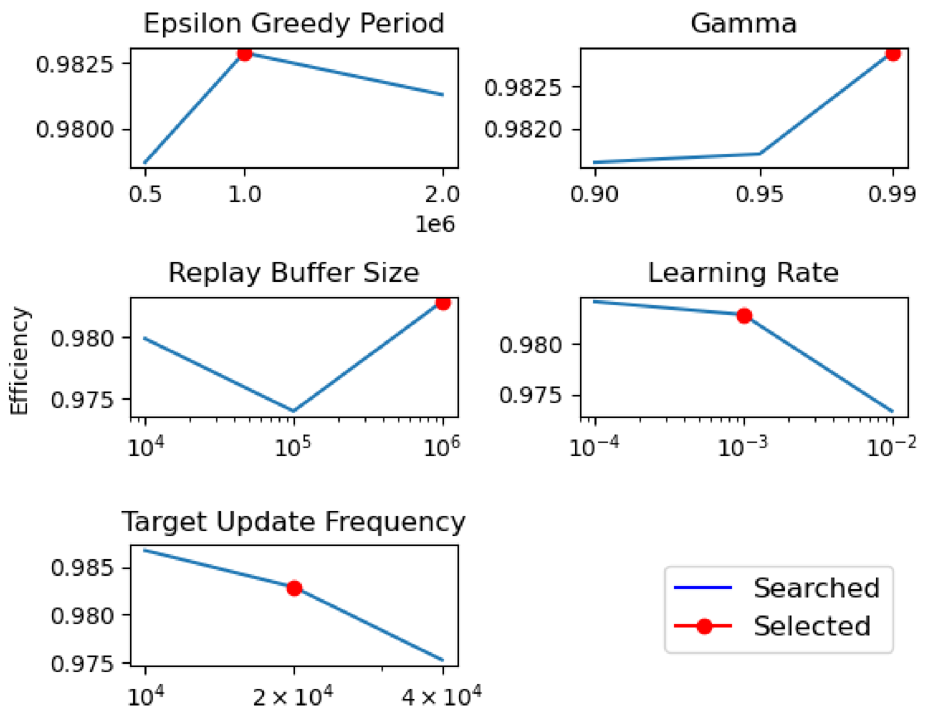

Figure S1. Different ablation studies on the hyperparameters. The recorded values are an average of two individual runs.

S4. Validation of Optimization

In Fig. S2, we show two curves plotted over the entire learning steps for the validation of learning process depicted in Fig. 3(a). Figure S2(a) shows the maximum efficiency value of devices acquired during the inference time, in which we periodically freeze the network and run 10 episodes for recording the maximum efficiency found during each episode with exploration epsilon of $\epsilon = 0.01$. The mean and standard deviation values of the 10 episodes are recorded. Figure S2(b) shows the mean of the maximum efficiency over the 10 episodes from inference with the variance taken over different network initializations. Three different network initializations were used for statistics in Fig. S2(b). In the early stage of learning, all networks fall into discovering only low efficiency devices (1). In the intermediate stage, each seed takes different search routes following what they have learned (2), and finally all networks converge to the generation of devices with very similar efficiencies as shown in (3). The plot indicates that regardless of different random initializations, learning processes of our proposed RL method converge to a very similar device efficiency value.

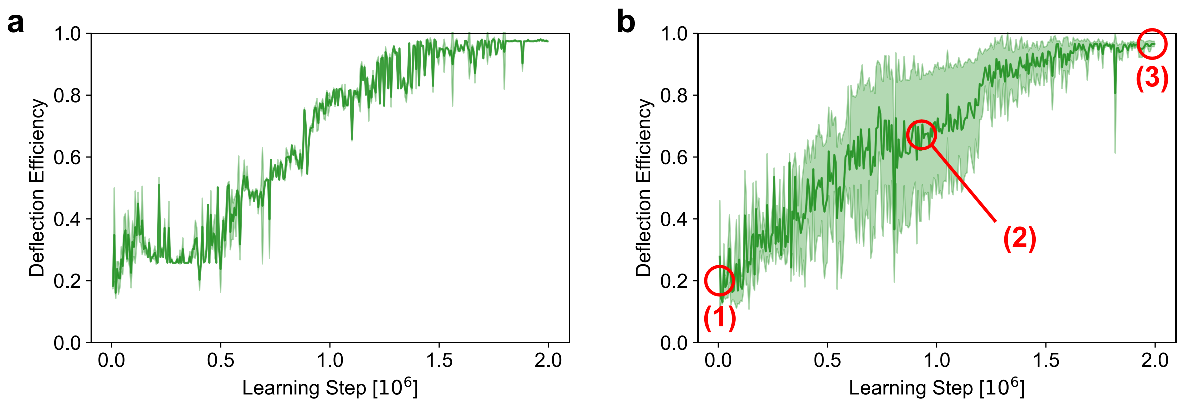

Figure S2. Validation statistics of the deep Q network during the learning process for input condition of λ_0 = 1100 nm, θ = 50°. **(a)** mean and standard deviation of maximum efficiency found during each of 10 episodes for inference **(b)** mean and standard deviation of maximum efficiency where variance is taken over three different random network initializations.

S5. High-impact Unit Cell

Table S5. High impact cell structure in Fig 5. 1 represents Si and 0 represents air. The underlined and bold character denotes the changed cell position. The text information of each structure can be changed from {+1, 0} to {+1, -1} and directly applied to RETICOLO simulation program powered by MATLAB.

Wavelength [nm]	Angle [°]	Structure
900	50	1111110101111111110011**1**111111100 01010000000010101101011101010001
		1111110101111111110011**0**111111100 01010000000010101101011101010001

S6. Effect of Random Structural Initialization

We experiment the effect of random structural initialization to our network inference to provide better understanding of our algorithm. For the condition of {wavelength: 1100nm, deflection angle: 50 degree}, the inference results from 10,000 random initial designs of the 3 saved pre-trained networks are shown in the figure below:

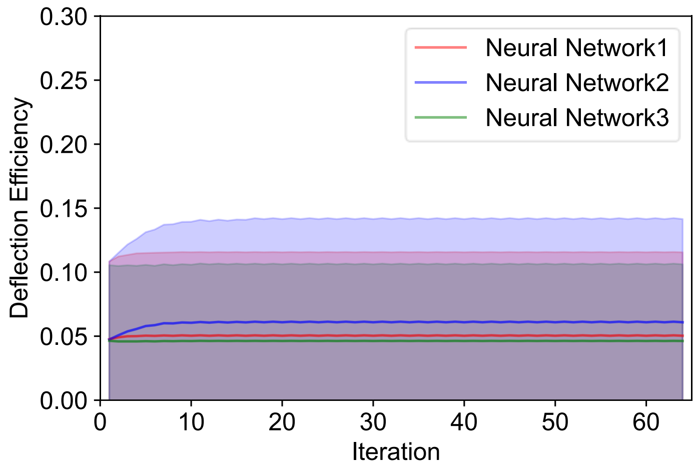

Figure S3. Inference results of 10,000 random initializations where the solid line represents the mean, and the shaded region corresponds to one standard deviation from the mean. Here, the mean and standard deviation values are calculated from the statistics of random initializations for each network.

The result shows that the network cannot convert a random initial design to a good one. We claim that this is an expected phenomenon from the nature of algorithm design. In our work, we kept the algorithm and network structure as simple as possible to isolate the applicability of RL in freeform inverse design on an intuitive scale. We distinguish this from the application of deep learning techniques that focuses on simply enhancing the performance of the algorithm. Therefore, we only used multi-layered perceptrons (MLP) with arguably small capacity and avoided generalization technology (i.e., Dropout or dimension reduction such as CNN) except exploration supported by the epsilon greedy algorithm. Essentially, the lack of generalization makes the network vulnerable to unseen conditions and may cause ‘catastrophic forgetting’ (oblivion of data which are out of distribution along the trajectory of optimal policy) during the exploitation part of the learning. However, while lacking in generalization, we focused on training the network to perform a single goal of “making one good design”.

Meanwhile, optimization from any random initialization is an interesting topic. It makes the problem more challenging since the network cannot exploit the same initial state, increasing the search space of the agent. We leave the further improvement of the suggested RL method through the aforementioned generalization techniques as possible future works.