Extended Abstract

Motivation Inverse-design methodologies, applied to the context of meta-material design, involve reshaping design regions (consisting of meta-materials) using computational methods to achieve pre-defined objectives such as redirecting electromagnetic input waves. Tasks such as these are notoriously difficult to design flexible materials for, due to the complexity of the environment dynamics. Rather than explicitly creating a configuration to satisfy a specific task, we seek to train an agent that implicitly understands the dynamics of the environment. The hope is to eventually train a multifaceted agent that can, at runtime, be specified a task (potentially outside of its training experience), and its understanding of the underlying dynamics will allow it to succeed.

Method We constructed a custom environment around the Ceviche package Fan (2019) to model the wave demultiplexer task. Both states and actions are represented as matrices $\mathbb{R}^{60\times 60}$ such that $s_{t+1} = a_t$, modeling the change of state. The reward was the objective value of the state subtracted by the objective value of the best state discovered. The objective was computed using J, discussed in the Introduction. J is positively correlated with wavelengths that overlap with desired sections of the design region and negatively correlated with wavelengths that overlap with incorrect sections of the design region. The two training algorithms employed were Deep Deterministic Policy Gradient (DDPG) Lillicrap et al. (2019) and Proximal Policy Optimization (PPO) Schulman et al. (2017) with design originally inspired by Barbhate (2021).

Implementation The environment utilized the Ceviche package, which was interfaced with by a custom wrapper for standard Gymnasium environments. A pipeline was constructed (the rl-laboratory) to streamline experiments, which is where DDPG was implemented. PPO experiments were facilitated by a much cruder Jupyter notebook run on Google Colab (motivating the need for the pipeline, which can now be used for future experiments).

Results Eight types of models were trained across 2 different algorithms, DDPG and PPO. Across all models, PPO was consistently the superior algorithm. For depth sizes of 01 and 50, the PPO algorithm that trained a convolutional net constructed design regions that meaningfully achieved the task; this is explicitly depicted in the visualizations of the wave propagatio . Models trained by the DDPG algorithm failed to construct any meaningful design regions; however, the convolutional net with depth 50 crafted very striking symmetric design regions that, although suboptimal, were unique among all other results. DDPG, therefore, crafted regions with the clearest structure.s

Discussion PPO achieved superior results than DDPG across a majority of the models. The convolutional net with a depth horizon of 50 steps performed the best overall, which is consistent with the spatial interdepencies at play in the environment. However, the states discovered by PPO (although consistently better) are not visually very interpretable. DDPG, although performing worse in its covolutional 50 step model, produces symmetric configurations and convergence is achieved relatively quickly. This may be an artifact of how the noise is generated. The specification of the task may have also been a specific weakness for DDPG.

Conclusion Although DDPG's objective values were not that impressive, its unexpectedly interpretable structures and PPO's success prove that potential lies in this approach. Future work will need to perform far more experiments with varying hyper parameters and reinterpretting the reward function. We hope to explore more diverse environments than the two-channel demultiplexer (like the N-channel demultiplexer, AND gates, and OR gates).

Comparative Performance and Stability Analysis of PPO and DDPG Agents on Inverse Design of Meta-materials for Demultiplexing

Selin Ertan
Department of Computer Science
Stanford University
szertan@stanford.edu

Matthew Villescas
Department of Computer Science
Stanford University
mattjv22@stanford.edu

Abstract

The following study explores linear and convolutional neural network approaches to PPO and DDPG-based reinforcement learning tasks for the inverse-design of metamaterials for demultiplexing, and concludes that convolutional methods with PPO agents are more applicable to the task due to the agent's stability and inclusion of the spatial aspect of the design space. With the convolutional neural network policy paired with PPO agent, a highest isolation objective of 7.511 was achieved, whereas the highest isolation for the linear neural network policy has remained at 2.016. DDPG agent repeatedly underperformed in both linear and convolutional settings compared to the PPO agent, but some unusual pattern-generating behavior was observed through, which is to be explored in upcoming studies.

1 Introduction

Inverse-design methodologies, applied to the context of meta-material design, involve reshaping design regions (consisting of meta-materials) using computational methods to achieve pre-defined objectives such as redirecting electromagnetic input waves. For meta-materials with electromagnetic properties, inverse-design methodologies often use algorithmic processes to generate design configurations to fulfill predetermined objectives (such as demultiplexing overlapping input waves, or altering the design region to create computational units such as AND/OR gates). Focusing specifically on plasma-optical computers, the mete-material refers to an environment consisting of plasmonic crystals that interact with and refract electromagnetic waves. As such, an environment with such properties may be inverse designed to refract light in specialized ways that allows us to achieve computational objectives.

Under traditional design techniques involving machine learning, the approach would include designing an agent to learn form the environment, and the final product would be the agent rather than the environment. In the case of plasma-optical computers, however, the agent would correspond to the incoming electromagnetic waves, since the waves interact with the environment directly the same way an agent would. It is, however, impossible to control the input wave's behavior directly, since the wave would behave in deterministic ways based on the active environmental properties, and "training" the input wave to "learn" a specific kind of behavior renders impossible. Therefore, rather than modeling an agent to learn from the environment and meet predetermined objectives, we choose to use inverse-design techniques that allow us to define an agent that alters the environment's components instead. By doing so, we train our agent to alter the properties of the design region so that the newly designed environment interacts with the input wave precisely such that the input wave's behavior aligns with our objective. Generally speaking, this approach is especially useful for designing complex physical systems where changing the input is impractical or when the environment is deterministic yet too complicated for manual design.

Studying the developmental processes of inverse-design methods offers significant opportunities across several domains. First, it is crucial for advancing alternative computational methods beyond traditional architectures, notably in neuromorphic computers (which plasma-optical computers happen to fall under), where hardware properties can be reconfigured by tracing back the design constraints necessary to perform a task. Second, it provides a framework for understanding self-coupled systems—such as fusion reactors or optical computers—where interdependent design choices complicate predictive analysis. Third, inverse-design methods enable the transfer of high-precision techniques to fields that depend on specialized equipment, like advanced chip design.

Optical computers are an example of neuromorphic systems that can be programmed in real time by learning from environmental feedback. Expanding on this, a plasma-optical computer leverages plasma-wave interactions to achieve computational tasks, such as wave-guiding, frequency demultiplexing, and implementing logic gates (AND, OR, XOR). Plasmas are uniquely suited for wave-based computation because they both refract and absorb electromagnetic waves, making them ideal for optimizing electromagnetic configurations through inverse design. Insights from this work may also apply to fusion reactors, where inverse design has been instrumental—such as at Livermore—in achieving breakthroughs like the first fusion ignition at the National Ignition Facility.

In this study, we apply reinforcement learning to inverse-design methodologies for plasma-optical computing. We encode the electromagnetic properties of the meta-material design region (which are directly correlated with the material density of independent discretized design regions) as states and train an agent to modify these properties, with the final objective of redirecting the input wave to embed a demultiplexing operation onto the design region. In doing so, we conduct a comparative analysis of a PPO (Proximal Policy Optimization) and DDPG (Deep Deterministic Policy Gradient) to understand how different agents interact with the design region, as well as gaining insight as to why they behave the way they do.

Specifically, we use a FDFD (finite-domain forward differentiation) package, Ceviche, Fan (2019) to construct our design space and output channels. Through Ceviche, we model our design space as a meta-material whose regional densities are subject to alterations and model the design region's interactions with incoming electromagnetic waves at two different frequencies ($\overline{w}_1 = 1$ and $\overline{w}_{1.1} = 1.1$). As part of our demultiplexing task, we will be programming the design space to have two input waves with distinct frequencies along with two output channels for target signal isolation. The goal of this task is to design an environment where an input wave consisting of multiple signal channels can be split and isolated into two distinct channels for further analysis, which would have direct applications for multi-bandwidth telecommunication channels as well as neuromorphic chip design/optimization.

For the demultiplexing task, we define our objective function as follows:

$$J(w_1, w_{1.1}) = \left(\int \mathbf{E}_{\omega 1} \cdot \mathbf{E}^*_{m=1} \, dl_{\omega 1 \text{ exit}} \right) \times \left(\int \mathbf{E}_{\omega 1.1} \cdot \mathbf{E}^*_{m=1} \, dl_{\omega 1.1 \text{ exit}} \right)$$
$$- \left(\int |\mathbf{E}_{\omega 1}|^2 \, dl_{\omega 1.1 \text{ exit}} \right) \left(\int |\mathbf{E}_{\omega 1.1}|^2 \, dl_{\omega 1 \text{ exit}} \right) \text{ Rodriguez and Abdalla (2021)}$$

The first term in our objective function rewards our model for redirecting the first input frequency w_1 to $exit_{w_1}$. Similarly, the second term in our objective function rewards the model for redirecting the second input frequency $w_{1.1}$ to $exit_{w_{1.1}}$. Finally, the third term in our objective function punishes the model for redirecting input wave w_1 to $exit_{w_{1.1}}$ and redirecting input wave $w_{1.1}$ to $exit_{w_1}$. The product of the two sub-terms allows us to punish for incorrect refractive indexes simultaneously. The objective of our agents, therefore, is to maximize the reward function through altering the states and actions (which are defined in detail in Methods section).

The primary objective of this study is comparing the performance of PPO and DDPG agents in maximizing the aforementioned objective function J. Additionally, this study also aims to conduct a behavioral analysis and comparison of the given two agents in terms of their stability by directly comparing the outputs for the design region. Our research questions, therefore, are as follows:

1. What are the maximal signal isolations we can obtain through the agents PPO and DDPG?
2. How do PPO and DDPG's performances compare to each other? What might be the reason why for these differences?

3. How do PPO and DDPG's stabilities compare to each other? What might be the reason why for these differences?

PPO and DDPG agents are specifically chosen for a number of reasons. First, since PPO puts a constraint on the maximal change allowed to be imposed onto the policy in a given iteration and DDPG does not, this provides us with an opportunity to observe the impact this restriction has on the overall stability of the agent's design predictions and performance. Second, given that PPO trains a state-value function $V(s)$ and DDPG trains a state-action value function $Q(s, a)$, comparing the two agents allows us to observe the impact of including the action space in the decision process of optimizing the design region. Lastly, comparing the final design suggestions of the agents directly allows us to further understand agent-state interactions and gives us insights on what kinds of agent designs must be prioritized for the problem at hand.

2 Related Work

For training and testing purposes, a FDFD (finite-domain-frequency-differentiation) simulation package named Ceviche by Fan (2019) has been used in the past studies described below.

For the demultiplexing problem, where the objective is to take two input signals at different frequencies and inverse-design a meta-material for maximum signal isolation, there are two primary non-reinforcement-based implementations worth referencing. The first is the work conducted by Rodriguez and Abdalla (2021), where the two input frequencies on a demultiplexer setting ($\overline{\omega}_1 = 1$ and $\overline{\omega}_2 = 1.1$) have been isolated into two distinct outlets, with the purpose of distinct signal isolation. In this study, the optimization algorithm used the Adam optimizer, implemented by default within the Ceviche package. The objective function's value (measuring the cross-compared signal isolations of the two input signals to the correct outlets, which is the same objective function used in the rest of the experiments in demultiplexing category) achieved a maximum value of 30 in this study after 1250 iteration cycles. It must be noted, however, that since the algorithm uses a gradient-based approach to search the design space (using the Jacobian matrix), there have been multiple instances we observed (while replicating the same experiments) where a maximum value of 30 for the objective function has been obtained multiple times, after which the objective fell down to near zero values and climbed back up once again. This behavior could be explained by the design of the Adam optimizer, where alterations to the design space after a maximum has already been achieved might lead the objective function to fall rapidly from a local maxima to a local minima due to insufficiently small step sizes. As such, the training time was not used efficiently during this study, in addition to not having any fail-safe mechanisms built in place to recover best obtained values. The optimizer relied primarily on a naive search of the gradient-space of the objective function, which has demonstrably led to unstable behaviors at local maxima points, indicating that the algorithm does not generate any generalizable insights that might apply real-time optimization tasks.

These frequency values, however, have been chosen arbitrarily by the authors of the study, and differ from the default values Ceviche packages operate on. Specifically, the demultiplexer experiments designed and implemented by Ceviche have frequency values ($\overline{\omega}_1 = 1.3$ and $\overline{\omega}_2 = 1.5$). For the default Ceviche optimizer, which is the same optimizer Rodriguez and Abdalla (2021) uses, the best value obtained by the same objective function stabilizes around 11 after 100 iteration cycles. Once again, however, the rapid loss observed in the study conducted by Rodriguez and Abdalla (2021) repeated in our experimental replications of the default demultiplexer implementation of the Ceviche package, despite the altered frequency values and iteration cycle lengths. This indicates the need for an improved implementation of the optimization methodology, since the design spaces in both cases seem to be responding in analogous ways.

In this study, we propose the implementation of policy-based training methods through which agents learn to design the meta-material spaces for demultiplexing input signals, followed by the comparative performance analysis of these methods, neither of which have been implemented thus far in the past studies. The details for the implementation of the agent are included in the Methods section below.

3 Method

The aforementioned design region is modeled as a 60x60 grid. Each element in the grid stores a continuous value $\rho_{ij} \in [0, 1]$. This represents the material density of the rod, which is directly related

to the permittivity and permeability (electromagnetic properties of the region) of the design space. A "configuration" (or state, used interchangably) ρ for the design region consists of a matrix with each entry representing the density value of a discretized design region. As such, each configuration consists of a 60x60 matrix with each entry corresponding to the density of one of the design regions.

The state space $\mathcal{S}$ is precisely $\rho \in [0,1]^{60\times 60}$. The continuous nature of each density entry in our design configurations require the use of policy-based methods due to the lack of discretization. The action space, $\mathcal{A} = [0,1]^{60\times 60}$, corresponds to all possible changes an agent might make to a given state ρ by updating a subset of its density values, which is chosen to be 10% of all discrete design regions in this case. One step for a given agent constitutes changing the space ρ_t to correspond to a_t, i.e. $\rho_{t+1} = a_t$; namely, an action taken by an agent returns the next state upon taking the action a_t, and does not represent any values to be added or subtracted from the ρ_t.

The objective for the demultiplexing task, as defined in 1, is taking in two input signals w_1 and $w_{1.1}$, and redirecting them to their corresponding outlet channels. $exit_{w_1}$ corresponds to the lower outlet placed to the right of the design region, and $exit_{w_{1.1}}$ corresponds to the upper outlet placed to the right of the design region, as it may be seen in Figure 1. The uniform configuration of $\rho_{\text{default}} = [0.5^{60\times 60}]$ favors no particular channel and disperses the waves uniformly across the design region. In particular, the agent seeks to find the state with the highest J within D steps, where D is defined as the finite depth and a hyperparameter; for the experiments to be presented below, we have a constant $H = 8000$ steps, and all agents have been tested within the same training range.

To incentivize further exploration from the previously found local optima, the best ρ_{best} found in the collected set of trajectories $\{\tau_i\}_{i=1}^{T}$ is used to evaluate the next state at timestep t. In particular, we define the following reward function r_t through our objective function J:

$$r_t = J(\omega_1(\rho_{\text{best}}), \omega_{1.1}(\rho_{\text{best}})) - J(\omega_1(\rho_{t+1}), \omega_{1.1}(\rho_{t+1}))$$

That is, an agent's newly created state is compared to the best state found thus far, ensuring that the agent can recover the best state found thus far in case it moves to a worse state. This was to incentivize exploration and maintain state progress; in particular, the dynamic ρ_{best} computed so far is used as the new initial state for future episodes of length H. We acknowledge, however, that the definition we chose for the reward function might lead the agents to get stuck in local optima and converge into suboptimal states depending on the maximum allowed stochasticity defined onto a given agent.

To model our agent, we used two training methods: Proximal Policy Optimization (PPO) as presented in Algorithm 2 Schulman et al. (2017) and Deep Deterministic Policy Gradient (DDPG) as presented in Algorithm 1 Lillicrap et al. (2019). Both PPO and DDPG are chosen as our agents due to being policy-based methods that can be used for training the agents in continuous action spaces. DDPG, specifically, has been chosen in place of Deep Q-Network (DQN) to avoid arbitrary finite enumeration of the action space.

The algorithm is comparable to discrete DQN. However, it is not clear how an action that maximizes the current Q estimates is can be sampled in a continuous space with DQN. DDPG solves this problem by employing an actor-critic paradigm, modeling the optimal action for a given Q function as its own neural net parameterized by θ^{μ}. The Q function takes the role of the critic parameterized by θ^{Q}. The weakness of this approach lies in its lack of exploration: Consequently, any action generated by the actor is perturbed by noise sampled from a normal distribution $\mathcal{N}(0, \sigma)$. The action executed in the environment is precisely the sum of the output of the actor network and noise generated from the normal distribution. This distribution could be parameterized in future experiments, was kept as a constant throughout this study.

The second algorithm used for training was Proximal Policy Optimization, as described in Barbhate (2021). The algorithm is presented as Algorithm 2. PPO was chosen for its ability to manage continuous action spaces. The ratio $r_t(\theta)$ represents the degree to which the new policy has been altered from the previous policy iteration. The clipped objective prevents this ratio from becoming too large or too small, modulating the magnitude of the updates that can be asserted onto the policy updates. The actions are also sampled from a multivariable normal distribution with independent parameters, where the standard deviation is a hyperparameter and the mean is the output of the chosen neural network.

It must be noted that an important and very strong assumption made throughout this study is the independent and decoupled nature of the design regions our agent updates across the training cycles.

Algorithm 1 Deep Deterministic Policy Gradient (DDPG)

Initialize actor network $\mu(s|\theta^{\mu})$ and critic network $Q(s, a|\theta^{Q})$
Initialize target networks: $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize replay buffer $\mathcal{D}$
for episode = 1 to $\frac{H}{D}$ **do**
    Initialize state $s_0 = \rho_{\text{best}}$
    for t = 1 to D **do**
        Select action with exploration noise: $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$
        Execute action a_t and observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Sample minibatch of N transitions from $\mathcal{D}$
        Compute target:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

        Update critic by minimizing loss:

$$L = \frac{1}{N} \sum_i \left(Q(s_i, a_i|\theta^{Q}) - y_i\right)^2$$

        Update actor using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{a=\mu(s)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})$$

        Soft update target networks:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \quad \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

    end for
end for

In reality, the design regions themselves interact with each other in complicated and coupled ways that are hard to model for a project with a scope of our kind; as such, we assume that the regions are fully independent and alterable. Similarly, we also assume that our agent has access to all specified design regions and can alter their material densities freely; this, once again, might not be (and usually is not) the case when it comes to material design, due to production limitations. As such, if the agents demonstrated in this study are to be adapted to model real world phenomena, these assumptions would have to be addressed.

Algorithm 2 Proximal Policy Optimization (PPO)

Initialize policy parameters θ_0, value function parameters ϕ_0
for iteration $= 1, 2, \ldots, \frac{H}{D}$ **do**
  Collect set of trajectories $\mathcal{D} = \{\tau_i\}$ by running policy π_θ in the environment for D steps.
  for each trajectory τ_i in $\mathcal{D}$ **do**
    Compute advantages $\hat{A}_t$ using GAE
    Compute rewards-to-go $\hat{R}_t$ as targets for value function
  end for
  for epoch $= 1$ to K **do**
    for each minibatch of transitions **do**
      Compute the ratio: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
      Compute the surrogate loss:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

      Update θ via gradient ascent on L^{CLIP}
      Update ϕ by minimizing value loss: $\left(V_\phi(s_t) - \hat{R}_t \right)^2$
    end for
  end for
end for

4 Experimental Setup

All our experiments consisted of modeling a design space using Ceviche through which the agents altered the densities of the design region to alter the wave-material interactions. Our experiments included training the agent for the demultiplexing objective with a variety of neural networks, namely linear and convolutional neural network with varying search depths with a constant number of steps $H = 8000$. Eight different models were trained throughout our experiments: $\{(model, D) : model \in \{linear, conv\}$ and $D \in \{1, 5, 10, 50\}\}$, where D represents the search depth of the model (the number of action updates the agent consecutively performs on the design space per iteration cycle). Roughly $200,000$ hyperparameters were used by both the linear and convolutional (60x60 input) neural networks. Actor learning rate was set to 0.0001 and critic learning rate was set to 0.001 and kept as constants throughout the experiments. All training cycles were started from a default state of $\rho_{\text{start}} = [0.5^{60\times60}]$. As a baseline, the objective value of the initial state $J(\omega_1(\rho_{\text{start}}), \omega_{1.1}(\rho_{\text{start}}))$ was -1.03, which is a unitless measure we calculate by taking the projection of the forward-propagating components of the input waves, summed over the target regions for the wave based on its input frequency.

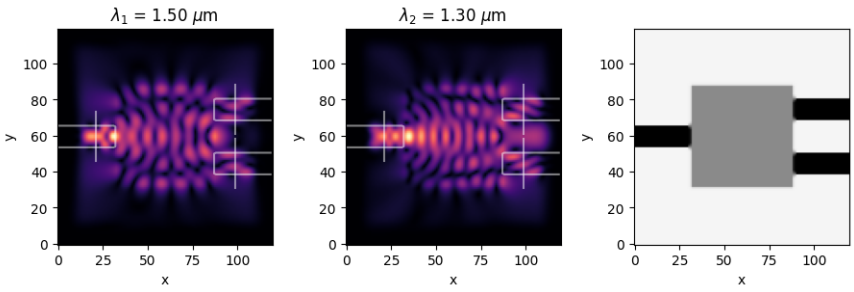
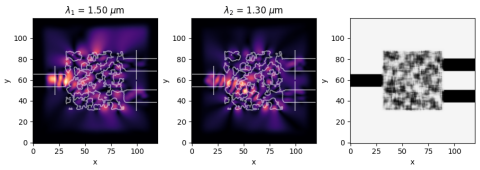


Figure 1: Default Configuration

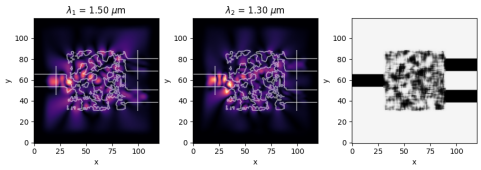
5 Results

5.1 Qualitative Analysis

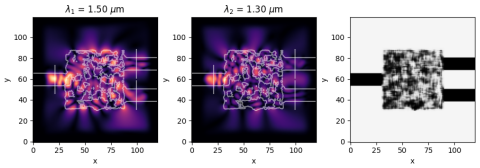
We will first start by analyzing the qualitative behavior of the PPO agent paired with a linear neural network, as described in the Methods section. As it may be seen in Figure 2, the PPO agents with search depths 01 and 50 generated design regions that split the input signals into their prospective outlets more clearly. In both (a) and (d), we observe two input signals being directed towards the two outlets, with both signals having considerable losses due to being directed outside of the design region and both signals being directed to both outlets instead of preserving clear signal isolation we would have ideally liked to observe. Yet, both the 01 and 50 step depth search provide considerably more isolation than 05 and 10 step depth searches, indicating that the greedy (one-step maximizing) approach and long-term planning in designing the region both work well, whereas the 05 and 10 step depth searches do not provide enough meaningful insights about the design region to achieve significant signal isolation.



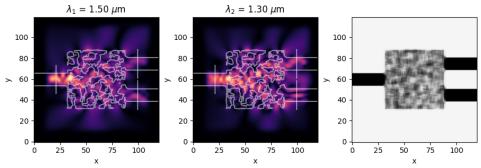
(a) Agent: PPO, Net: Lin, Depth: 01



(b) Agent: PPO, Net: Lin, Depth: 05



(c) Agent: PPO, Net: Lin, Depth: 10

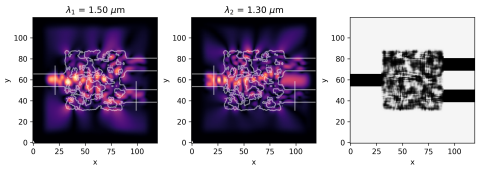


(d) Agent: PPO, Net: Lin, Depth: 50

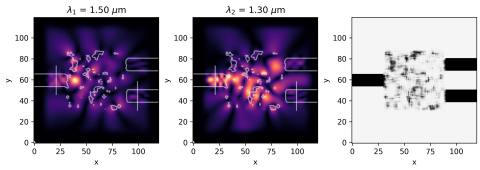
Figure 2: Design Regions for PPO (Linear Network, Variable Search Depths)

Comparing the qualitative behavior of the PPO agent with a linear network with the DDPG agent with a linear network, we observe that DDPG seems to produce some highly inconsistent behavior that does not seem to correspond to our objective function as much as PPO does. As it may be seen in Figure 3, the DDPG agents with search depths 01 and 10 generated design that seem to be sending both input signals to the same outlet. In both (a) and (c), we observe two input signals being directed towards the outlet below or outside of the design region, with (a) having considerable losses on both signals due to directing the signals to the region in between the outlets. We also observe that the search depths 05 and 50 seem to be interacting with the input signals more dramatically. In both (b) and (d), the first input signal is almost completely disrupted into meaningless noise and the second signal is directed distinctly to the upper outlet. These results indicate that there might be some local converging points the DDPG agent is unable to escape from. It also indicates that DDPG is a lot more unstable in generating design regions when compared to the PPO agent, likely due to its increased sensitivity to highly unstable states with action pairs.

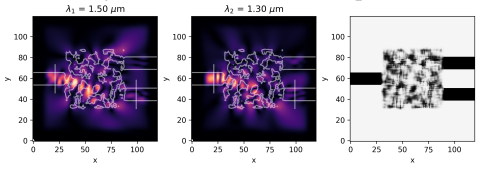
Next, we move onto the qualitative behavior of the PPO agent with a convolutational neural network. Results similar to those of the PPO agent with the linear neural network are also observed with the PPO agent paired with a convolutional neural network. As it may be seen in Figure 4, the PPO agents with search depths 01 and 50 generated design regions that split the input signals into their prospective outlets more clearly, acknowledging that the agent with search depth 10 seemed to outperform the corresponding agent from the linear network. These results are to be expected due to the nature of the design region, since the spatial element of the design and interactions between the discrete design regions in manipulating the wave were ignored in the linear network, but are now taken into consideration in the convolutional network. In both (a) and (d), we observe two input signals being directed towards the two outlets, with both signals having smaller losses due to being directed outside of the design region despite using the same reward function, which is another benefit of using a convolutional network. We also see that the signal isolation is a lot more clear now, with signals clearly being directed to separate outlets with barely any overlap. Still, both the 01 and 50 step depth



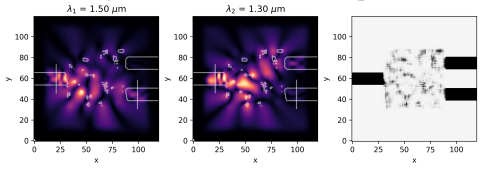
(a) Agent: DDPG, Net: Lin, Depth: 01



(b) Agent: DDPG, Net: Lin, Depth: 05



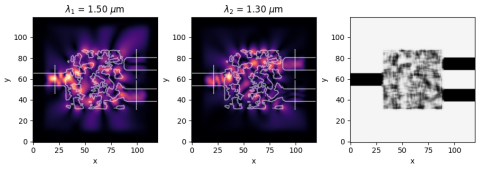
(c) Agent: DDPG, Net: Lin, Depth: 10



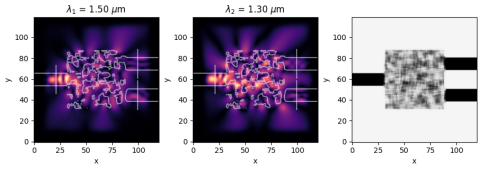
(d) Agent: DDPG, Net: Lin, Depth: 50

Figure 3: Design Regions for DDPG (Linear Network, Variable Search Depths)

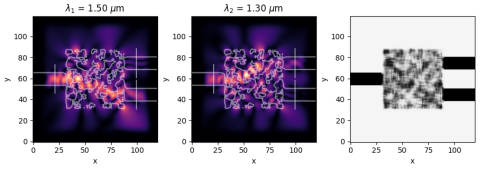
search provide considerably more isolation than 05 and 10 step depth searches, indicating that the greedy (one-step maximizing) approach and long-term planning in designing the region still both work well, whereas the 05 and 10 step depth searches do not provide enough meaningful insights about the design region to achieve significant signal isolation.



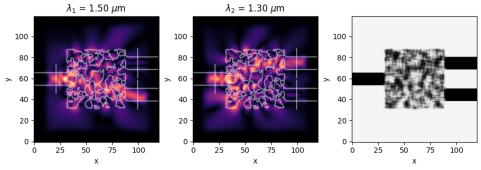
(a) Agent: PPO, Net: Conv, Depth: 01



(b) Agent: PPO, Net: Conv, Depth: 05



(c) Agent: PPO, Net: Conv, Depth: 10

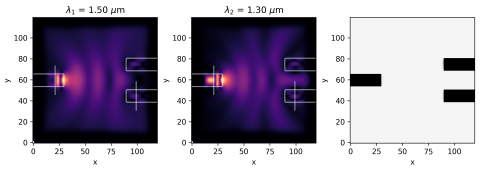


(d) Agent: PPO, Net: Conv, Depth: 50

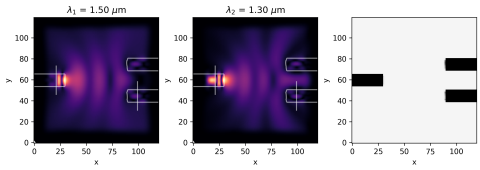
Figure 4: Design Regions for PPO (Convolutional Network, Variable Search Depths)

Finally, we evaluate the qualitative behavior of the DDPG agent with the convolutional neural network. As it may be seen in Figure 5, the DDPG agent fails to produce any meaningful designs for the search depths 01, 05, and 10 (graphs (a), (b), (c)). We suspect that this behavior is present due to the agent converging into some local inflection points and being unable to escape due to the highly sensitive nature of the agent's sensitivity to initial conditions and randomized actions not helping enough with moving out of the convergence regions. With the 50 step depth search showcased at (d), however, we observe that the agent comes up with a pattern based structure that we have never observed before. We find this result particularly enticing, as it indicates the agent's ability to come up with symmetrical/self-duplicating structures to manipulate the design region. We believe that this aspect might lead to various previously-unseen design patterns for our problem in the future upon conducting some detailed hyperparameter optimization on the agent, which will be part of our future work.

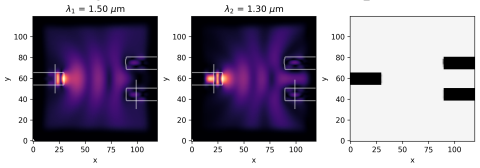
To conduct a small scale stability analysis of our agent, we trained four independent PPO agents with a convolutional neural network of search depth 50. These specific settings were chosen since PPO agents performed the best under those conditions. As it may be seen in all of the design regions provided in Figure 6, the PPO agent consistently came up with similar structures that sent the input signals to their corresponding outlets and provided decent signal isolation. This indicates that the PPO agents trained independently arrive at similar finalized designs, and the implementation behaves in stable and predictable ways when it comes to the design decisions.



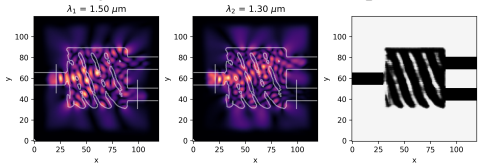
(a) Agent: DDPG, Net: Conv, Depth: 01



(b) Agent: DDPG, Net: Conv, Depth: 05

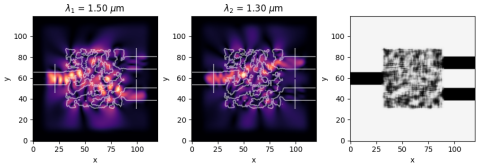


(c) Agent: DDPG, Net: Conv, Depth: 10

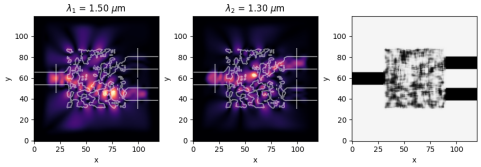


(d) Agent: DDPG, Net: Conv, Depth: 50

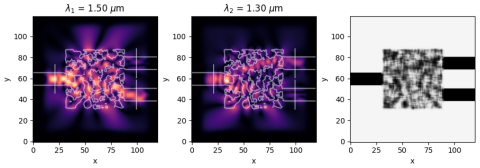
Figure 5: Design Regions for DDPG (Convolutional Network, Variable Search Depths)



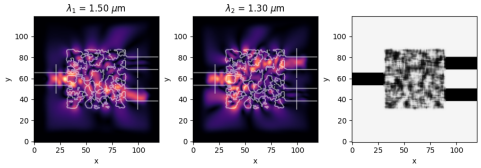
(a) Agent: PPO, Net: Conv, Depth: 50 (1)



(b) Agent: PPO, Net: Conv, Depth: 50 (2)



(c) Agent: PPO, Net: Conv, Depth: 50 (3)



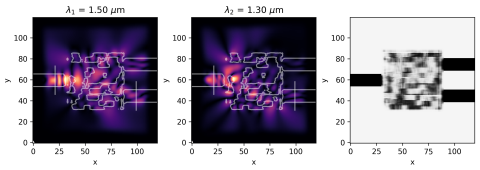
(d) Agent: PPO, Net: Conv, Depth: 50 (4)

Figure 6: Design Regions for PPO (Convolutional Network, Constant Search Depth (50), Consecutive Trials

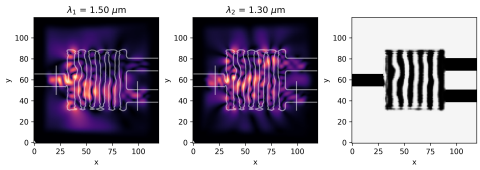
When it comes to training four DDPG agents with a convolutional network of a 50-step search depth, however, we see that the agents behave in highly unpredictable and erratic ways. In graphs (a) and (d), the agent is barely able to manipulate the design region enough to make any difference to the input signal. Especially in part (d), it ends up designing a region that completely blocks signal propagation, which is in complete conflict with the objective provided to the agent. In parts (b) and (c), however, we see that the agent comes up with similar repeating and symmetrical design patterns for wave propagation. Once again, these results are the most enticing, since they are the very first time we observe our trained agents to come up with design regions that use self-repeating structures in the design region.

5.2 Quantitative Evaluation

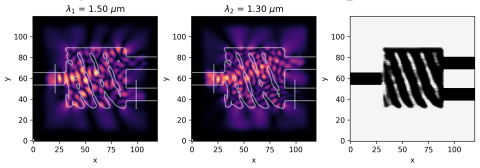
Consider Tables 1 and 2 for the performance analyses of the PPO and DDPG agents, respectively. As it may be seen in the numerical results, PPO agent repeatedly outperformed DDPG agent in all trials, which we believe is due to the on-policy paradigm of the PPO agent over the off-policy paradigm of the DDPG agent. Furthermore, the expressivity of PPO's source of randomness (i.e. sampling actions directly from a learned distribution) likely aided exploration, while the noise embedded in DDPG was likely not ufficiently expressive.



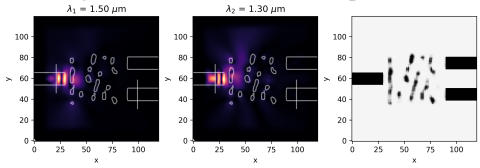
(a) Agent: DDPG, Net: Conv, Depth: 50 (1)



(b) Agent: DDPG, Net: Conv, Depth: 50 (2)



(c) Agent: DDPG, Net: Conv, Depth: 50 (3)



(d) Agent: DDPG, Net: Conv, Depth: 50 (4)

Figure 7: Design Regions for DDPG (Convolutional Network, Constant Search Depth (50), Consecutive Trials

Table 1: Performance Comparison for PPO

Agent	Search Depth	Linear (J)	Convolutional (J)
PPO	01	1.518	6.274
PPO	05	2.868	2.024
PPO	10	1.971	4.814
PPO	50	3.699	7.145

Table 2: Performance Comparison for DDPG

Agent	Search Depth	Linear (J)	Convolutional (J)
DDPG	01	1.385	-0.978
DDPG	05	1.116	-0.973
DDPG	10	0.820	-0.972
DDPG	50	1.465	2.016

Consider Tables 3 and 4 for the stability analyses of the PPO and DDPG agents, respectively. As it may be seen in the numerical results, PPO agent provided much more stable results compared to the DDPG agent, which we believe is also due to the on-policy paradigm of the PPO agent over the off-policy paradigm of the DDPG agent, and in alignment with the qualitative outcomes we described. Furthermore, dynamic shaping of the reward as a function of the number of trajectories entailed that most rewards were negative. This is to be expected since the action space for the agent is very large and hence a very small percentage of the chosen actions lead to increased returns on the reward function. The stability displayed by PPO implies an algorithmic flexibility that is conducive to the dynamically shaped reward. DDPG, evidently, was not as conducive compared to PPO, and its highly unstable behavior implies that a dynamically shaped reward might have hindered the agent's performance more than it helped.

Table 3: Stability Comparison for PPO

Agent	Trial	Search Depth	Convolutional (J)
PPO	01	50	5.935
PPO	02	50	5.414
PPO	03	50	7.511
PPO	04	50	7.145

Table 4: Stability Comparison for DDPG

Agent	Trial	Search Depth	Convolutional (J)
DDPG	01	50	-0.140
DDPG	02	50	0.049
DDPG	03	50	1.856
DDPG	04	50	2.016

6 Discussion

As far as raw objective values are concerned, the PPO implementation succeeded the most. The convolutional network with a 50-step search depth found the configuration associated with the greatest objective value, which was 7.511. Furthermore, virtually all of the models trained under PPO succeeded to a greater extend than those trained under DDPG. This may be due to the efficient exploration methods employed by PPO, by which the agent's updates get more constrained over time as the agent moves further into its training cycle. The stochastic exploration of DDPG, however, implements a constant update constraint that does not adapt to the training cycle of the agent, due to which the agent continuously makes cruder choices compared to the PPO agent.

Yet, it must also be acknowledged that the design configurations generated by the PPO agent are a lot less intuitive, more disorderly, and less interpretable as a consequence. Although the results are superior in terms of the objective values and consistency, there are no clear patterns emerging from the design's generated by the PPO agent, indicating that the agent might not be sufficiently harnessing the structure of the design space.

The states produced by DDPG, although inferior in objective values and consistency, converge to some clearly interpretable states with emergent design patterns. The convolutional network of depth 50 repeatedly converges to either practically empty design regions that do not let the input waves propagate through, or some strange designs where symmetrical emerging patterns are present. Given the stability analysis of DDPG we discussed above, the extreme variations we observed in model behavior are to be expected. However, the consistent convergence of the model to symmetric designs was not something we have expected, and is worth further exploration in the future studies. The fact that the DDPG agent's training loop reached converge much faster than PPO also motivates the further study of methods that will enforce exploration if the agent converges too quickly and/or gets stuck in a local optima.

Finally, there may be an asymmetry of success due to the way the task was specified. Rewarding an agent for the value of their objective directly seemed to be an inappropriate choice since the agents might get comfortable in suboptimal states that are close to worse states by changing the states very little. We, therefore, wanted a reward function that would avoid this comfort, which is why the reward adjusts to the best configuration found in the episode. Alternatively, rewards could be determined by the ending configuration of the agent, but then rewards would be very spare for models of very high depth. Reward shaping is a strategy that should be considered to resolve this. Furthermore, the objective function could be better refined by introducing terms that reflect the magnetic fields involved.

7 Conclusion

PPO was clearly superior in this set of experiments; however, we still believe DDPG has potential with modifications. Parameterizing its noise could go very far in its exploration strategy. Furthermore, there is clearly not much stability in the agents (something to be expected of randomness), but this implies that a solitary agent may not suffice. Indeed, ensemble methods that pool multiple agents and multiple critics may smooth out stability issues.

The success of PPO does prove there is potential to our approach. Further work needs to be done with tuning the task and exploring other algorithms such as the genetic algorithm. We hope to diversify the goal of the task, including a variety of wavelengths and a more generalized number of channels in the Demultiplexer task, and simulating the OR and AND gate tasks. We hope that diversifying the

environment will teach the agents more about the underlying dynamics of the metamaterials, which would be the ultimate goal

8 Team Contributions

- **Selin Ertan** Coded DDPG algorithm, conducted thorough result analysis, supervised environment construction
- **Matthew Villescas** Coded PPO algorithm, ported Ceviche dynamics into environment.

Changes from Proposal After realizing that DQN would not be a suitable method, we instead decided to implement DDPG. Due to time constraints, we were limited in scope. We did not get to build the environment to support the logic gate task or run the genetic algorithm; however, in developing this project, we constructed a flexible reinforcement learning pipeline that we can use to more easily develop models and test on more environments. Later work will make use of this.

References

Nikhil Barbhate. 2021. PPO-PyTorch. https://github.com/nikhilbarhate99/PPO-PyTorch.

Shanhui Fan. 2019. Ceviche Library. https://github.com/fancompute/ceviche.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG] `https://arxiv.org/abs/1509.02971`

Jesse Rodriguez and Ahmed Abdalla. 2021. Inverse Design of Plasma Metamaterial Devices for Optical Computing. *Review Applied 16, 014023* (2021).

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] `https://arxiv.org/abs/1707.06347`