Bridging the gap between Optimization and Reinforcement for Control Researchers

Zongqiang Pang,Liping Bai *Member, IEEE,*

***Abstract*—This paper is aimed at providing researchers in the field of control and optimization a bridge for incorporating deep reinforcement learning into their work. Considerable amount of progress has been made in deep learning and its associated compuatational infrastructure, which makes fast and accurate approximation a possiblity. Off the shelf implementations of various reinforcement learning agents written in either PyTorch or Tensorflow can be easily found. Yet, when control researchers venture into this subject, they would find out that there is still quite a lot caveats and pitfalls when it comes to applying neural network to optimization problems. They can't just formulate their optimization problem into reinforcement learning format and let those off the shelf agents have a go at it. We summarized how optimization problems differ from problems such as facial recognition and how the mathematical tools we use differ vary as a result. We also provide a summary of existing researches at the croassroad between optimization and reinforcement learning.**

***Index Terms*—Reinforcement Learning, Control Theory, Optimal Control, Optimization**

I. Introduction

REINFORCEMENT learning studies how actions should be chosen based on the observations of a dynamic system such that the expected reward along the trajectory is maximized. This objective is quite similar to that of Trajectory Optimization which is a subject studied in the field of control. Reinforcement learning can be seen as a "Black Box" approach to control, focusing on systematically shifting the policy distribution based on observed samples. Trajectory Optimization can be seen as a "White Box" approach to control, focusing on incorporating scripted constraints into computation. Neither approach is perfect. "Black Box" need copious amount of data while "White Box" need extriquisit equitions to capture the dynamics exactly.

Efforts has been made to make the black box approach more broadly applicable and more data efficient. For instance, Sergey Levine utilizes unsupervised learning to build an agent with intuitions of the physical world [1]; Chelsea Finn introduced Meta-Learning [2] such that the overaching structures embedded in similar tasks of different setting can be captured by the neuronet. This line of research is mostly done in the Computer Science department.

The question we tried to help researchers solve is how to take adavantage of the compuational advances made in the "Black Box" approaches and extend the reaches of the "White Box" approaches to dynamics that has so far eluded mathematical descriptions.

Nanjing Unversity of Posts and Telecommunications, College of Automation & College of Artificial Intelligence, Nanjing, Jiangsu,210000 China email:zqpang@njupt.edu.cn

If we observe biological creatures such as ourselves, it is obvious that our brain works by combining features from both black box and white box approach. We have some basic understandings of how the world works and how our actions would affect the world. Despite the fact that we can't write down the governing equation and boundary constraints of a control problem, we can make progress via trial and error. Unfortunately, these two subjects are studied by two communities of researchers who use different notations to discribe the same processes and they publish their research only in journals of their own disciplines. In this paper, we hope to provide a bridge for researchers in the field of control theories who would like to incorporate reinforcement learning into their works.

We are not the first ones to take up this challenge. Recently, there are interdiscipline conferences held specifially for the purpose of briding the gaps between these groups of people, for instance the L4DC (Learning for Dynamic Control) conference and Intersections between Control, Learning and Optimization Workshop.

In this paper, we build on the work of Benjamin Recht [3] to formulate reinforcelent learning in the language of optimization. The first barrier facing control researchers is consolidating the notation between control and reinforcement learning communities. Control communities use the notation system introduced by Lev Pontryagin. State is denoted $\mathcal{X}$, Action is denoted $\mathcal{U}$, which is the first letter of Russian for "Action", the dynamics and stocasticity is captured by physical model constraints $x_{t+1} = f(x_t, u_t, e_t)$ where e denote the noise of a system. The objective is usually to minimize the cost funtion $\mathcal{J}(.)$. Reinforcement Learning communities use the notation system introduced by Richard Bellman who studied dynamic programming. State is denoted $\mathcal{S}$, Action is denoted $\mathcal{A}$. The dynamics and stochaticity is captured via transition matrix $\mathcal{P}$ of a Markov Decision Process.The objective of RL is the maximize the reward function $\mathcal{R}(.)$. Yet, if we set the transition matrix to be identical to the noise, then it is clear that the underlying process behind these two notation system are exactly the same. The differences are nothing but style. Since the audience of our paper is the control community, we would cast reinforcement learning in the control notation system.

We would first introduce where to bring approximation into optimization and the mathematical tools we need to better formulate such approximations. We then elaborate on the enumerated path in section II and list all the papers in this area of research.

II. Where to Approximate

The constraints imposed in the Trajectory Optimization formulation manifest themselves directly in policy $\pi(x)$ and resulting in either a narrow band of trajectories or a single optimal solution. However, adjustments in the reinforecement learning formuation is not the policy per se but its distribution. Eventually, what we hope to acheive via learning is a policy distribution, either through policy gradient method or cost to go method, which would maximize the expected reward of a trajectory.

Reinforcement Learning is not a new subject, control researchers probably know it by the name Approximate Dynamic Programming. Yet the major progress in recent years is the enhanced compuation power which brought about the potential of neuronetworks as a universal approximator [4] to fruition. Most notably the series of wins reinforcement agents such as AlphaGo, AlphaStar forged against the best human players in the respective disciplines.

Broadly speaking, there are three revenues where neuronetwork based approximation can find its way into optimization as shown in 1. One is learning a dynamics model from samle; Second is policy gradient based learning; Third is approximation of cost to go functions such as value funtion and Q function. The details of the implementation would be specified in the subsequent sections.

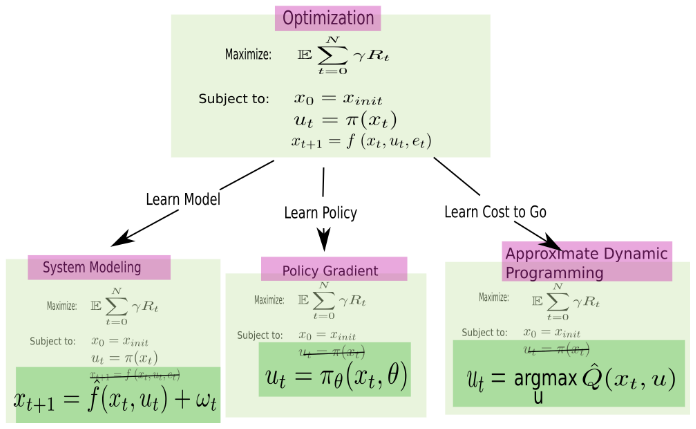

Optimization
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
$x_{t+1} = f(x_t, u_t, e_t)$
Learn Model
Learn Policy
Learn Cost to Go
System Modeling
$x_{t+1} = \hat{f}(x_t, u_t) + \omega_t$
Policy Gradient
$u_t = \pi_\theta(x_t, \theta)$
Approximate Dynamic Programming
$u_t = \underset{u}{\text{argmax}}\,\hat{Q}(x_t, u)$

Fig. 1: From Optimization to Learning

III. Additional Mathematical Tools

Before we dive into the detailed researches in each category, we'd like to introduce additional mathematical tools and measurements that has been proven useful in merging reinforcement learning with optimization. As we discussed in the previous section, the major breakthrough in the field of reinforcement learning is not necessarily the theory but the computational infrustruction. Today, reinforcement learning agents implemented with Tensorflow or PyTorch is widely available, yet if you simply formulate an optimization problem into reinforcement learning format and apply those off the shelf reinforcement learning agents, it would hardly work.

Neuronetwork based approximation in the context of optimization and control is fundamentally different from those we find in applications such as facial recognition or natual language processing. Additional mathematical tools and measurements are required to fit the structures required by control problems. While the topic of statistical learning theory is vast and profund, we selected two concepts that is most germane for bridging reinforcement learning to control theoriest. The exact use of these concept would be specified in the subsequent sections when we discuss the related papers.

A. *Relative Information*

The go to text book for reinforcement learning community is that of Richard Sutton. In ths system contstructed by Richard, the objective of an agent is to maximize discounted reward. This formulation is proven a sound goal in the context of video games. However, for optimizations with a physics underpining, reward maximization is not adequate since noise and disturbances is common place in control.

Empirically, entropy maximization has proven to work well in the control context.

B. *Bayesian Neuro Network (BNN)*

The venena formulation of neuronetwork is one where the weight of each neuron w_i is a single number, which is adjusted based on the backward propogation process. Baysian Neuro Network is a network where the weigh is subject to a parameterized distribution as shown in Fig. 2 The use of Bayesian Neuro Network is to measure the uncertainly of the current iteration.

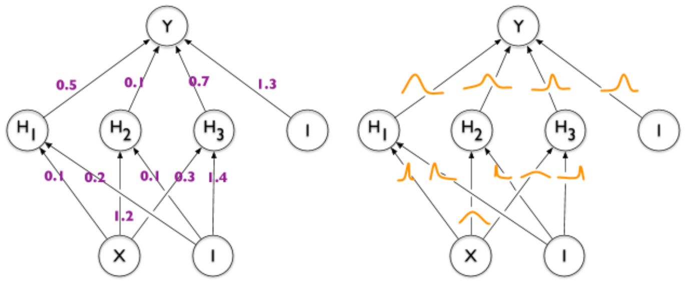
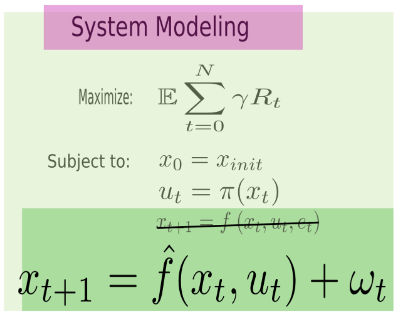


Fig. 2: Neuronet and BNN

IV. System Modelling


System Modeling
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
$x_{t+1} = f(x_t, u_t, e_t)$
$x_{t+1} = \hat{f}(x_t, u_t) + \omega_t$

For people who are trained in classical control theory, system modelling and system identification are the bread and butter of their field of study. One of the most obvious way to derive a model from learning is through the following althrithm:

Algorithm 1 Brute Force Modelling

```
1: run random policy, collect $\mathcal{D} = (x, u, x')_i$
2: while Model Has Not Converged do
3:     fit model to minimize $\sum_i \left\| \hat{f}(x_i, u_i) - x' \right\|$
4:     Model Predictive Control using $\hat{f}(x_i, u_i)$
5:     exercute control collect new data
6:     append new data to $\mathcal{D}$
7: end while
```

A. *Measure Model Uncertainty with BNN*

While in theory the aforementioned brute force modelling method should work, in practice it is proven to be far from satisfactory. [5]. One possible explanation is that any overfitting of the model would be hard to overcome, resulting in suboptimal performance. One solution is to measure uncertainly of the model with BNN.

asdfjajdflajsdlfjajdsfl;adfjlakjdflajsdflajkdfsjaldfja

B. *Embeded Optimization Layer and Residual Dynamics*

Writing governing equations of a physical system is something classical control theoriest trained on, yet there are always dynamics which are too intricate to be captured in full. One mental tool to deal with the uncaptured dynamics is to pack them all into a parameterized distribution which can be approximated with neuronetwork. There are two approaches in merging physics model with neuronetwork. One is to embed the model [6] [7] [8], two is to only approximate residual dynamics that can't be fully captured by the existing model [9].

C. *Safe Region*

V. Policy Gradient

A. *Trust Region Policy Iteration*

B. *Direct Policy Fitting*

VI. Approximate Dynamic Programming

A. *Value Iteration and Policy Iteration*

B. *Maximum Entropy Reinforcement Learning*

VII. Other Ways to Utilize Increased Computational Power

A. *Multiagent Formulation*

B. *Monte Carlo Tree Search*

C. *Random Shoot*

random text

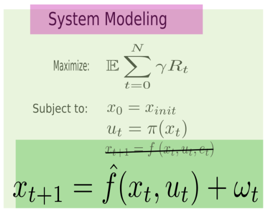

System Modeling
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
~~$x_{t+1} = f(x_t, u_t, e_t)$~~
$x_{t+1} = \hat{f}(x_t, u_t) + \omega_t$

Fig. 3: Optimization

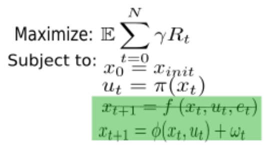

Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
~~$x_{t+1} = f(x_t, u_t, e_t)$~~
$x_{t+1} = \phi(x_t, u_t) + \omega_t$

Fig. 4: System Identification

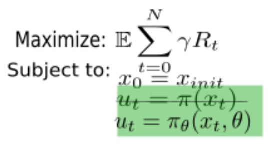

Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
~~$u_t = \pi(x_t)$~~
$u_t = \pi_\theta(x_t, \theta)$

Fig. 5: Policy Gradient

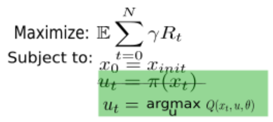

Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
~~$u_t = \pi(x_t)$~~
$u_t = \operatorname{argmax}_u Q(x_t, u, \theta)$

Fig. 6: Approximate Dynamic Programming

D. *Wireless Power Transfer*

The objective of energy beamforming control is to choose a beamforming code for each energy transmitting node such that the total received power is maximized while satisfying the minimum energy requirement of each energy receiver.

$$\begin{aligned} \underset{\mathbf{f}_p, \forall p}{\text{maximize}} \quad & \sum_{j \in \{1,2,\ldots,L\}} e_j \\ \text{subject to} \quad & \mathbf{f}_p \in \mathcal{F}, \forall p; \\ & e_j \geq e_{min} \end{aligned}$$

VIII. Reinforcement Learning

A. *problem setup*

Twritten in its recursive form, known as Bellman Equation, which is the basis for an iteratively implemented backward induction algorithm:

$$G_t = R_t + \gamma G_{t+1} \tag{1}$$

Transition matrix is intruduced to encode the stochastidy in the environmental dynamics. Transaction Matrix $\mathcal{P}$ is defined as:

$$\mathcal{P}^a_{ss'} := Pr\{S_{t+1} = s' | S_t = s, A_t = a\} \tag{2}$$

State/Action Function q(s,a) is definied as expected gain starting from state s by taking action a:

$$\begin{aligned} q(s, a) : &= \mathbb{E}\{G_t | S_t = s, A_t = a\} \\ &= \mathbb{E}\{\sum_{k=0}^{T-t} \gamma^k R_{t+k+1} | S_t = s, A_t = a\} \end{aligned} \tag{3}$$

Policy is defined as:

$$\pi(s, a) := Pr(A = a | S = s) \tag{4}$$

Optimal Policy is defined as:

$$\pi^*(s) := \arg\max_a q(s,a) \tag{5}$$

Value Function v(s) is defined as the expected gain starting from state s:

$$\begin{aligned} v(s) :&= \mathbb{E}\{G_t|S_t = s\} \\ &= \mathbb{E}\{\sum_{k=0}^{T-t} \gamma^k R_{t+k+1}|S_t = s\} \\ &= \sum_{a\in\mathcal{A}} \pi(s,a)q(s,a) \end{aligned} \tag{6}$$

B. Without Approximation

One obvious approach to learning is to statistically construct a model of the environment, which is called Model-Based Learning. The most primitive form of model-based learning is Bellman Equation based backward induction. Statistical tactics, such as maximum likelihood, Bayesian methods, etc., can be deployed to approximate the model with the least amount of sampling. However, since the environment is implicitly embedded in v(s) and q(s,a), the model building process can be circumvented entirely, hence Model-Free Learning. Depending on whether the iteration rules is policy dependent, model-free learning can be subdivided into on-policy learning and off-policy learning.

One hindrance to the implementation of the brute force backward induction is its memory requirement. A more effective approach is to update q value and v value after one episode, one step, or n steps. They are called Monte Carlo Method, Temporal Difference Method, and $\lambda(n)$ Method respectively.

For online learning, ϵ-greedy Policy $\pi_\epsilon(s)$ is frequently deployed to balance exploration and exploitation, such that the environment can be encoded most efficiently. ϵ is initiated set to 1 and then asymptotically goes to 0 as the episode counts increases.

$$\pi_\epsilon(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \arg\max_a q_\epsilon(s,a) \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

C. With Approximation

When the problem gets complex, state S becomes a rather large vector and function approximation with neuro networks can be utilized to facilitate learning. Reinforcement learning as a self-sustaining mathematical framework has been refined by Rich Sutton et al. since the 1980s. Only recently, the progress made with Deep Learning has been applied to the realm of Reinforcement Learning

Let the value function and state/action function be parameterized with $\mathbf{w}$: $\hat{v}(s,\mathbf{w}) \approx v(s)$ and $\hat{q}(s,a,\mathbf{w}) \approx q(s,a)$

Let the i^{th} iteration of parameter be denoted as w_i. The Loss Function $\mathcal{L}(w_i)$ is defined as the following:

$$\mathcal{L}(w_i) := \mathbb{E}\{[v(s) - \hat{v}(s,w_i)]^2\} \tag{7}$$

$$\mathcal{L}(w_i) := \mathbb{E}\{[q(s,a) - \hat{q}(s,a,w_i)]^2\} \tag{8}$$

While the real value of v(s) and q(s,a) are not knowable, it can be approximated:

$$v(s) \approx \sum_{a\in A} R(s,a) + \gamma v(s',\mathbf{w}) \tag{9}$$

$$q(s,a) \approx r + \gamma \arg\max_a q(s',a',\mathbf{w}) \tag{10}$$

The Gradient of weighing paramter **w** can be derived from 7 and 8 with the real values substituted by 9 and 10 respectively. By convention, constant is omitted. Parameter is updated following Gradient Descent:

$$\mathbf{w}_i = \mathbf{w}_{i-1} - \nabla_{w_{i-1}}\mathcal{L}(w_{i-1}) \tag{11}$$

D. Policy Gradient Methods

Policy $\pi(s)$ can be written as a function parameterized by θ with s as input and a smooth distribution overall all actions as output.By adjusting parameter θ we can adjust the distribution over action choices for different states. This style of learning is called policy gradient-based learning.

Let us register a path sequence taken by the agent as τ such that the sequence is denoted as $\{S_{\tau 0}, A_{\tau 0}, R_{\tau 0}...S_{\tau T}, A_{\tau T}, R_{\tau T}\}$. the gain of sequence τ is defined as the gain of this entire sequence of state, action, reward:

$$G(\tau) := \sum_{t=0}^{T} \gamma^t R_t \tag{12}$$

Denote $P(\tau,\theta)$ as the probability that path τ is travesed when the policy is parameterized by θ. The Objective Function can be defined in various ways. Here we adopt the definition as the following:

$$U(\theta) = \sum_\tau P(\tau,\theta)G(\tau) \tag{13}$$

The objective of the policy gradient method is to find the parameter θ to maximize the objective function.

The gradient of aforementioned utility function is:

$$\nabla_\theta U(\theta) = \nabla_\theta \sum_\tau P(\tau,\theta)G(\tau) \tag{14}$$

A mathematical sleight of hand called Importance Sampling is deployed to convert this theoretical expression of gradient into something that is algorithmically feasible.

$$\nabla_\theta U(\theta) \approx \frac{1}{N}\sum_{\tau=1}^{N}\sum_{t=0}^{T-1} \nabla_\theta ln\pi_\theta(s,a)|_{\theta_{old}}[q^{\pi_{\theta_{old}}}(s,a) - b] \tag{15}$$

We can use stochastic gradient descent(SGD) method to update θ:

$$\theta = \theta_{old} - \alpha\nabla_\theta ln\pi_\theta(s,a)|_{\theta_{old}}[q^{\pi_{\theta_{old}}}(s,a) - b] \tag{16}$$

Actor-Critic Method takes advantage of both policy gradient and function approximation to build a bootstrap structure that lead up to fast convergence. state/action function for policy $\pi_\theta(s)$ is approximated by $q^{\pi_\theta}(s, a, \mathbf{w})$.Baseline b is introduced into the bootstrap stracture to foster convergence. Different algorithms define baseline differently. In advantage Actor-Critic algorithm, baseline is defined as a value function based on π_θ.Because the SGD updating process does not rely on the ordering of things, it is obvious that some of the aforementioned computations can be done asynchronously. Asynchronous Advantage Actor-Critic (A3C) is proven one of the most effective agents for renforcement learning, and is the one we will use in this paper.

E. Multiagent Reinforcement Learning

$A_t = \{A_t^1, A_t^2, ..., A_t^M\}$ M is the number of agents. The action space is cartician product of action choices available to each agent. $A_t(s) = A_t^1(s) \times A_t^2(s) \times ... \times A_t^M(s)$, which grows exponentially as the number of agents grows.

Without intermidiate state rollout, the sequences of data collected is: $...S_t, A_t, R_t, S_{t+1}...$ as shown in Figure 7

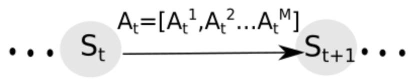
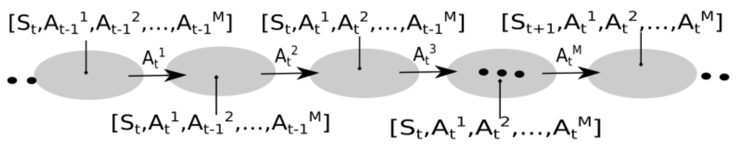


Fig. 7: Without Rollout

The intermediate states rollout technique converting action space complexity into state-space complexity by introducing intermediate states, denoted as S_t^k where k goes from 1 to M-1. The sequence of data is now: $...S_t, A_t^1, R_t^1, S_t^1, A_t^2, R_t^2, S_t^2, ..., S_t^{M-1}, A_t^M, R_t^M, S_{t+1}...$ as shown in Figure 8



$[S_t,A_{t-1}^1,A_{t-1}^2,...,A_{t-1}^M]$
$[S_t,A_t^1,A_t^2,...,A_{t-1}^M]$
$[S_{t+1},A_t^1,A_t^2,...,A_t^M]$
A_t^1
A_t^2
A_t^3
A_t^M
$[S_t,A_t^1,A_{t-1}^2,...,A_{t-1}^M]$
$[S_t,A_t^1,A_t^2,...,A_t^M]$

Fig. 8: With Rollout

suppose each agent has N choices. This formulation reduces the size action space from N^M to $N \times M$.

IX. Beamforming as a Multiagent Reinforcement Learning problem

A. Environment

The wirelessly powered communication network has L energy transmitting stations positioned at the corner of a 30m x 30m field. K energy receivers randomly scattered between 1m to 29m as illustrated by Figure 9. 0.5s of energy transfer is followed with 0.5s of information transfer. Assume no energy leftover at each cycle, such that at the beginning of the next energy transfer interval, the remaining power at each energy receiver is 0.

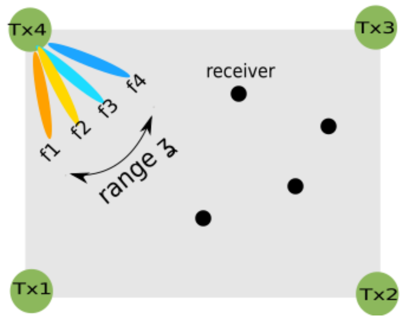


Fig. 9: Environment

Number of Energy Transmitting Nodes	L=4
Positions of Trasmitting Node	TX_1(0,0) TX_2(30,0) TX_3(30,30) TX_4(0,30)
Number of Radiation Elements per Trasmitting Node	M=64
Energy Carrier Frequency	8M Hz
Field of Energy Receivers	30m x 30m
Number of Energy Receivers	K
Energy Transfer Time	0.5s
Information Transfer Time	0.5s
Maximum Number of Steps	100

Observation Space:$\{e_1, e_2, ...e_K, c_1, c_2, c_3, c_4\}$ where e_j is the energy received at the j^{th} receiver, c_i is the codebook choice for the i^{th} energy emitting node.

Reward: If $e_j < e_min$, reward is deducted by 50 points each. If $e^n_{total}ew > e^n_{total}ew$, reward is increased by 100 points. If $e^n_{total}ew < e^n_{total}ew$, reward is deduced by 300 points.

B. A3C agent

Layers of Actor Network	3
Layers of Critic Network	3
Learning Rate for Actor	α_a=0.1
Learning Rate for Critic	α_c=0.1
Discount Rate	γ=0.9
Action Function	Softmax

C. Simulation Result

X. Conclusion

In this paper, we demonstrated the possibility to formulate WPT as a multiagent reinforcement learning problem, this lays the groundwork for further study towards fully locally computed control algorithms for wirelessly powered communication networks. Instead of group actions of all agents together, a multiagent rollout approach sees things sequentially, action taken by one agent becomes part of the state of another. This framework deduces the dimension of action space from exponential growth to multiplicative growth, and it can be applied to other problems. The most recent incarceration of beamforming technology is a passive reflective surface, or Intelligent Reflective Surface(IRS), where the reflective components are in the thousands. The multiagent approach proposed in this paper could be applied to IRS control as well, which should be a fruitful topic of future studies.

References

[1] C. Finn, I. J. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," *ArXiv*, vol. abs/1605.07157, 2016.

[2] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *ArXiv*, vol. abs/1703.03400, 2017.

[3] B. Recht, "A tour of reinforcement learning: The view from continuous control," *ArXiv*, vol. abs/1806.09460, 2018.

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

[5] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[6] F. de Avila Belbute-Peres, K. Smith, K. R. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *NeurIPS*, 2018.

[7] F. de Avila Belbute-Peres, T. D. Economon, and J. Z. Kolter, "Combining differentiable pde solvers and graph neural networks for fluid flow prediction," *ArXiv*, vol. abs/2007.04439, 2020.

[8] A. Agrawal, B. Amos, S. Barratt, S. P. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *ArXiv*, vol. abs/1910.12430, 2019.

[9] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790, 2019.