Model-Free Unsupervised Learning for Optimization Problems with Constraints

Chengjian Sun, Dong Liu, and Chenyang Yang

Beihang University, Beijing, China
Email: {sunchengjian, dliu, cyyang}@buaa.edu.cn

***Abstract*—In many optimization problems in wireless communications, the expressions of objective function or constraints are hard or even impossible to derive, which makes the solutions difficult to find. In this paper, we propose a model-free learning framework to solve constrained optimization problems without the supervision of the optimal solution. Neural networks are used respectively for parameterizing the function to be optimized, parameterizing the Lagrange multiplier associated with instantaneous constraints, and approximating the unknown objective function or constraints. We provide learning algorithms to train all the neural networks simultaneously, and reveal the connections of the proposed framework with reinforcement learning. Numerical and simulation results validate the proposed framework and demonstrate the efficiency of model-free learning by taking power control problem as an example.**

I. Introduction

Various resource allocation and transceivers in wireless networks, such as power allocation, beamforming, and caching policy, can be designed by solving optimization problems with constraints, say imposed by the maximal transmit power, cache size, and the minimal data rate requirement [1, 2].

Depending on the applications, the objective function, constraints and the policy to be optimized may vary in different timescales. If they are in the same timescale, the problem is variable optimization, and the policy to be optimized is a vector with finite dimension, e.g., optimizing beamforming based on small scale channel gains to maximize the instantaneous data rate subject to instantaneous transmit power constraint. If they are in different timescales, the problem is functional optimization [3], i.e., the policy to be optimized is a function, which can be interpreted as a vector with infinite elements. A classical example of functional optimization is finding the instantaneous power allocation to maximize the ergodic capacity under the average power constraint, whose solution is the classical water-filling power allocation [4].

Variable optimizations have been well studied. Efficient tools, such as interior point method [5], have been developed to find the numerical optimal solutions of convex optimizations and various approximation methods have been widely used in non-convex optimizations. The optimal solutions of functional optimizations are, however, generally not in closed-form and inefficient to be obtained numerically. One way for numerical searching is the finite element method [6], which converts the functional optimization into a variable optimization by only optimizing the values of function on a finite sampled points. However, such method suffers from the curse of dimensionality. To overcome this shortage, an unsupervised learning framework was developed in [7], which parameterizes the functions by neural networks and trains the network parameters with stochastic gradient descent (SGD).

To apply numerical searching methods, the expressions of objective function and constraints, i.e., the model, should be known. For methods like interior point method and SGD, the gradients of objective function and constraints with respect to the optimization variables or functions are further required. However, in many scenarios, the expressions of the objective or constraints are unavailable, or too complex to derive their gradients. Finite difference method can be used to estimate the gradients according to the observations of the objective and constraints. However, such a method is inefficient when the objective function and the constraints are in high dimensions, and hence is not applicable for functional optimization problems, which are with infinite dimensions.

In this work, we propose a model-free framework to solve functional optimizations with instantaneous and average constraints without the supervision of optimal solution. We begin with a model-based framework for unsupervised learning where a neural network (called as *policy network*) is used for parameterizing the function to be optimized. We recast the original constrained problem in the dual domain where the Lagrangian is served as the objective function and another neural network (called as *multiplier network*) is introduced to parameterize the Lagrange multiplier associated with the instantaneous constraint. In the model-free framework, we resort to neural networks (called as *value networks*) to approximate the unavailable expressions of objective function or constraints so that their gradients can be obtained for training the policy and multiplier networks. Then, we reveal the connections of the proposed framework with reinforcement learning and show how to extend the proposed framework into stochastic policy optimization, which is applicable for both continuous and discrete policy optimizations. We study a simple power control problem to illustrate how to apply the framework and show the effectiveness of the model-free unsupervised learning.

II. Unsupervised Learning for Optimizations

In this section, we introduce model-based and model-free unsupervised learning frameworks for solving general functional optimization problems. Since variable optimization can be treated as a special case of functional optimization, these frameworks are also applicable to variable optimization.

Let $\mathbf{h} \in \mathbb{R}^n$ denote a random vector reflecting environment status, e.g., channel gains. For each realization of $\mathbf{h}$, we aim to find a vector $\mathbf{x} \in \mathbb{R}^m$ to execute, e.g., transmit powers. Let function (called as a policy) $\mathbf{f}\colon \mathbb{R}^n \mapsto \mathbb{R}^m$ denote the mapping from $\mathbf{h}$ to $\mathbf{x}$, i.e., $\mathbf{x} = \mathbf{f}(\mathbf{h})$. The performance metric is a scalar function of $\mathbf{x}$ and $\mathbf{h}$ denoted by $J(\mathbf{x}, \mathbf{h})$, e.g., the instantaneous data rate. The goal is to design a function $\mathbf{f}$ that maximizes the performance metric averaged over $\mathbf{h}$, i.e., $\mathbb{E}_{\mathbf{h}}[J(\mathbf{x}, \mathbf{h})]$, subject to some constraints. This can be formulated as a general functional optimization problem as follows,

$$\mathsf{P1}: \quad \max_{\mathbf{f}(\mathbf{h})} \; \mathbb{E}_{\mathbf{h}}[J(\mathbf{f}(\mathbf{h}), \mathbf{h})] \tag{1a}$$

$$\text{s.t.} \;\; \mathbf{g}(\mathbf{f}(\mathbf{h}), \mathbf{h}) \preceq \mathbf{0} \tag{1b}$$

$$\mathbb{E}_{\mathbf{h}}[\mathbf{c}(\mathbf{f}(\mathbf{h}), \mathbf{h})] \preceq \mathbf{0} \tag{1c}$$

where (1b) and (1c) denote the instantaneous and the average constraints, respectively, and the curled inequality symbol "$\preceq$" (or "$\succeq$") denotes the element-wise inequality. It is noteworthy that optimization problems minimizing the objective function or having "$\succeq$" or "$=$" constraints (e.g., minimal data rate constraint) can be easily transformed into problem P1.

In general, problem P1 is hard to solve because it is a function $\mathbf{f}(\mathbf{h})$ that needs to optimize, which can be interpreted as vectors with infinite dimension when $\mathbf{h}$ is with infinity number of possible values.

A. Model-Based Unsupervised Learning

To tackle with the constraints, we reconsider problem P1 in its dual domain. The Lagrangian of P1 can be written as [8]

$$\begin{aligned}&\mathcal{L}(\mathbf{f}(\mathbf{h}), \boldsymbol{\lambda}(\mathbf{h}), \boldsymbol{\xi}) \\ &= \mathbb{E}_{\mathbf{h}}\left[J(\mathbf{f}(\mathbf{h}), \mathbf{h}) - \boldsymbol{\lambda}(\mathbf{h})^T \mathbf{g}(\mathbf{f}(\mathbf{h}), \mathbf{h}) - \boldsymbol{\xi}^T \mathbf{c}(\mathbf{f}(\mathbf{h}), \mathbf{h})\right]\end{aligned} \tag{2}$$

where $\boldsymbol{\lambda}(\mathbf{h})$ and $\boldsymbol{\xi}$ are the Lagrange multiplier associated with constraints (1b) and (1c), respectively. When strong duality condition holds [5], the original problem is equivalent to finding the saddle point of the Lagrangian as

$$\mathsf{P2}: \quad \min_{\boldsymbol{\lambda}(\mathbf{h}), \boldsymbol{\xi}} \max_{\mathbf{f}(\mathbf{h})} \; \mathcal{L}(\mathbf{f}(\mathbf{h}), \boldsymbol{\lambda}(\mathbf{h}), \boldsymbol{\xi}) \tag{3a}$$

$$\text{s.t.} \;\; \boldsymbol{\lambda}(\mathbf{h}) \succeq \mathbf{0} \tag{3b}$$

$$\boldsymbol{\xi} \succeq \mathbf{0} \tag{3c}$$

where $\boldsymbol{\lambda}(\mathbf{h})$ is also a function that needs to be optimized. Thanks to the universal approximation theorem [9], we can introduce two neural networks to approximate function $\mathbf{f}$ as $\mathbf{f}(\mathbf{h}) \approx \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f)$ (called as *policy network*) and approximate multiplier as $\boldsymbol{\theta}(\mathbf{h}) \approx \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda)$ (called as *multiplier network*), respectively, with arbitrary accuracy by finite-dimension parameter vectors $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_\lambda$. Then, problem P2 degenerates into the following variable optimization as

$$\mathsf{P3}: \quad \min_{\boldsymbol{\theta}_\lambda, \boldsymbol{\xi}} \max_{\boldsymbol{\theta}_f} \; \mathcal{L}(\tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f), \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda), \boldsymbol{\xi}) \tag{4a}$$

$$\text{s.t.} \;\; \tilde{\boldsymbol{\lambda}}(\mathbf{h}, \boldsymbol{\theta}_\lambda) \succeq \mathbf{0} \tag{4b}$$

$$\boldsymbol{\xi} \succeq \mathbf{0} \tag{4c}$$

To solve problem P3, we can adopt the primal-dual stochastic gradient method [10] that iteratively updates the primal variable $\boldsymbol{\theta}_f$, and the dual variables $\boldsymbol{\theta}_\lambda$ and $\boldsymbol{\xi}$ along the ascent and descent directions of sample-averaged gradients, respectively. The gradients of the Lagrangian (4a) with respect to $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\xi}$ can be derived as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_f} \mathcal{L} =& \mathbb{E}_{\mathbf{h}}\Big[\nabla_{\boldsymbol{\theta}_f} \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f)\big[\nabla_{\mathbf{x}} J(\mathbf{x}, \mathbf{h}) - \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}, \mathbf{h}) \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda) \\ &- \nabla_{\mathbf{x}} \mathbf{c}(\mathbf{x}, \mathbf{h}) \boldsymbol{\xi}\big]\big|_{\mathbf{x} = \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f)}\Big]\end{aligned} \tag{5a}$$

$$\nabla_{\boldsymbol{\theta}_\lambda} \mathcal{L} = -\mathbb{E}_{\mathbf{h}}\left[\nabla_{\boldsymbol{\theta}_\lambda} \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda) \mathbf{g}(\tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f), \mathbf{h})\right] \tag{5b}$$

$$\nabla_{\boldsymbol{\xi}} \mathcal{L} = -\mathbb{E}_{\mathbf{h}}\left[\mathbf{c}(\tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f), \mathbf{h})\right] \tag{5c}$$

where $\nabla_{\mathbf{x}} y = [\frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_n}]^T$ denotes the gradient, $\nabla_{\mathbf{x}} \mathbf{y} = [(\nabla_{\mathbf{x}} y_1), \cdots, (\nabla_{\mathbf{x}} y_m)]$ denotes the transpose of Jacobian matrix and $(\cdot)^T$ is the transpose operation. Both $\nabla_{\boldsymbol{\theta}_f} \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f)$ and $\nabla_{\boldsymbol{\theta}_\lambda} \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda)$ can be computed via back propagation.

Let $\mathcal{B}$ denote a batch of realizations of $\mathbf{h}$. Then, the primal and dual variables are updated by

$$\begin{aligned}\boldsymbol{\theta}_f^{(t+1)} \leftarrow& \boldsymbol{\theta}_f^{(t)} + \frac{\delta_f}{|\mathcal{B}|} \sum_{\mathbf{h} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}_f} \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f)\Big[\nabla_{\mathbf{x}} J(\mathbf{x}, \mathbf{h}) \\ &- \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}, \mathbf{h}) \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda^{(t)}) - \nabla_{\mathbf{x}} \mathbf{c}(\mathbf{x}, \mathbf{h}) \boldsymbol{\xi}^{(t)}\Big]\Big|_{\mathbf{x} = \tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f^{(t)})} \\ \boldsymbol{\theta}_\lambda^{(t+1)} \leftarrow& \boldsymbol{\theta}_\lambda^{(t)} + \frac{\delta_\lambda}{|\mathcal{B}|} \sum_{\mathbf{h} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}_\lambda} \tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda^{(t)}) \mathbf{g}(\tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f^{(t)}), \mathbf{h}) \\ \boldsymbol{\xi}^{(t+1)} \leftarrow& \left[\boldsymbol{\xi}^{(t)} + \frac{\delta_\xi}{|\mathcal{B}|} \sum_{\mathbf{h} \in \mathcal{B}} \mathbf{c}(\tilde{\mathbf{f}}(\mathbf{h}; \boldsymbol{\theta}_f^{(t)}), \mathbf{h})\right]^+\end{aligned} \tag{6}$$

where δ_f, δ_λ and δ_ξ are learning rates, and $[\mathbf{x}]^+ = [\max(x_1, 0), \cdots, \max(x_2, 0)]^T$. The operation $[\cdot]^+$ in (6) ensures constraint (4c). Constraints (4b) can be satisfied by properly chosen the activation function of the output layer of $\tilde{\boldsymbol{\lambda}}(\mathbf{h}; \boldsymbol{\theta}_\lambda)$, e.g., `ReLU`.

If the gradients $\nabla_{\mathbf{x}} J(\mathbf{x}, \mathbf{h})$, $\nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}, \mathbf{h})$ and $\nabla_{\mathbf{x}} \mathbf{c}(\mathbf{x}, \mathbf{h})$ can be computed, an approximated optimal solution of problem P3 can be obtained after the iterations in (6) converges.

B. Model-Free Unsupervised Learning

For many problems in wireless networks, one or all of the objective and constraint functions in problem P1 cannot be derived in closed-form, and we can only observe the values of these functions after executing $\mathbf{x}$ at a realization of $\mathbf{h}$ and then observe the values of $J(\mathbf{x}, \mathbf{h})$, $\mathbf{g}(\mathbf{x}, \mathbf{h})$, and $\mathbf{c}(\mathbf{x}, \mathbf{h})$. For example, we can measure the data rate $J(\mathbf{x}, \mathbf{h})$ after transmit with power $\mathbf{x}$ at channel state $\mathbf{h}$. For these scenarios, the gradients cannot be derived analytically. In what follows, we resort to model-free unsupervised learning that does not require the explicit expressions of these gradients.

Again according to the universal approximation theorem, we can approximate the objective function and constraints in problem P1 by neural networks as $J(\mathbf{x}, \mathbf{h}) \approx \tilde{J}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}_J)$, $\mathbf{g}(\mathbf{x}, \mathbf{h}) \approx \tilde{\mathbf{g}}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}_g)$, and $\mathbf{c}(\mathbf{x}, \mathbf{h}) \approx \tilde{\mathbf{c}}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}_c)$. With the approximated objective function and constraints (called as *value networks*), the gradients can be then computed.

The values of $J(\mathbf{x},\mathbf{h})$, $\mathbf{g}(\mathbf{x},\mathbf{h})$, and $\mathbf{c}(\mathbf{x},\mathbf{h})$ can be measured and recorded in a system, which can be used as labels for training. Then, the neural networks $\tilde{J}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_J)$, $\tilde{\mathbf{g}}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_g)$, and $\tilde{\mathbf{c}}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_c)$ can be trained by minimizing the L_2-norm loss function with stochastic gradient descent as

$$\begin{aligned}
\boldsymbol{\theta}_J^{(t+1)} &\leftarrow \boldsymbol{\theta}_J^{(t)} - \frac{\delta_J}{|\mathcal{B}|}\sum_{(\mathbf{x},\mathbf{h},J)\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_J}\Big[J(\mathbf{x},\mathbf{h}) - \tilde{J}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_J^{(t)})\Big]^2 \\
\boldsymbol{\theta}_g^{(t+1)} &\leftarrow \boldsymbol{\theta}_g^{(t)} - \frac{\delta_g}{|\mathcal{B}|}\sum_{(\mathbf{x},\mathbf{h},\mathbf{g})\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_g}\Big\|\mathbf{g}(\mathbf{x},\mathbf{h}) - \tilde{\mathbf{g}}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_g^{(t)})\Big\|^2 \\
\boldsymbol{\theta}_c^{(t+1)} &\leftarrow \boldsymbol{\theta}_c^{(t)} - \frac{\delta_c}{|\mathcal{B}|}\sum_{(\mathbf{x},\mathbf{h},\mathbf{c})\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_c}\Big\|\mathbf{c}(\mathbf{x},\mathbf{h}) - \tilde{\mathbf{c}}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_c^{(t)})\Big\|^2
\end{aligned} \tag{7}$$

where δ_J, δ_g and δ_c are learning rates, $\mathcal{B}$ denotes a batch of tuples whose elements are the realizations of $\mathbf{h}$, the corresponding vector $\mathbf{x}$ conditioned on $\mathbf{h}$, and the values of $J(\mathbf{x},\mathbf{h})$, $\mathbf{g}(\mathbf{x},\mathbf{h})$ and $\mathbf{c}(\mathbf{x},\mathbf{h})$ measured after executing $\mathbf{x}$.

In the following, we denote $\mathbf{y} \triangleq \mathbf{y}(\cdot)$ for notational simplicity, e.g., $J \triangleq J(\mathbf{x},\mathbf{h})$ and $\tilde{J} \triangleq \tilde{J}(\mathbf{x},\mathbf{h};\boldsymbol{\theta}_J)$. By substituting $J \approx \tilde{J}$, $\mathbf{g} \approx \tilde{\mathbf{g}}$, and $\mathbf{c} \approx \tilde{\mathbf{c}}$ into (6), we can obtain the update rule for $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\xi}$ as

$$\begin{aligned}
\boldsymbol{\theta}_f^{(t+1)} &\leftarrow \boldsymbol{\theta}_f^{(t)} + \frac{\delta_f}{|\mathcal{B}|}\sum_{\mathbf{h}\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_f}\tilde{\mathbf{f}}(\nabla_{\mathbf{x}}\tilde{J} - \nabla_{\mathbf{x}}\tilde{\mathbf{g}}\tilde{\boldsymbol{\lambda}} - \nabla_{\mathbf{x}}\tilde{\mathbf{c}}\boldsymbol{\xi})\big|_{\mathbf{x}=\tilde{\mathbf{f}}} \\
\boldsymbol{\theta}_\lambda^{(t+1)} &\leftarrow \boldsymbol{\theta}_\lambda^{(t)} + \frac{\delta_\lambda}{|\mathcal{B}|}\sum_{\mathbf{h}\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_\lambda}\tilde{\boldsymbol{\lambda}}\tilde{\mathbf{g}} \\
\boldsymbol{\xi}^{(t+1)} &\leftarrow \left[\boldsymbol{\xi}^{(t)} + \frac{\delta_\xi}{|\mathcal{B}|}\sum_{\mathbf{h}\in\mathcal{B}}\tilde{\mathbf{c}}\right]^+
\end{aligned} \tag{8}$$

Remark 1: When problem P1 has no constraints, our model-free unsupervised learning framework degenerates into a special case of reinforcement learning, where the policy $\mathbf{f}(\mathbf{h})$ does not affect the distribution of state $\mathbf{h}$. For the unconstrained problem, the gradient of the Lagrangian with respect to policy parameter $\boldsymbol{\theta}_f$ in (5a) degenerates into

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}_f}\mathbb{E}_{\mathbf{h}}\Big[J(\tilde{\mathbf{f}}(\mathbf{h};\boldsymbol{\theta}_f),\mathbf{h})\Big] \\
&\qquad = \mathbb{E}_{\mathbf{h}}\Big[\nabla_{\boldsymbol{\theta}_f}\tilde{\mathbf{f}}(\mathbf{h};\boldsymbol{\theta}_f)\nabla_{\mathbf{x}}J(\mathbf{x},\mathbf{h})\big|_{\mathbf{x}=\tilde{\mathbf{f}}(\mathbf{h};\boldsymbol{\theta}_f)}\Big]
\end{aligned} \tag{9}$$

which coincides with the deterministic policy gradient (DPG) theorem [11], where $J(\mathbf{x},\mathbf{h})$ is actually the action-value function (also known as *Q-function* or *critic*) and the policy network $\tilde{\mathbf{f}}(\mathbf{h};\boldsymbol{\theta}_f)$ is the *actor*. By replacing J in (9) with its approximation $\tilde{J}$, we can obtain the approximated policy gradient used for updating the actor in deep deterministic policy gradient (DDPG) algorithm [12].

Inspired by the great success of *actor-critic* approach in reinforcement learning, we can train the neural networks $\tilde{J}$, $\tilde{\mathbf{g}}$, $\tilde{\mathbf{c}}$, $\tilde{\mathbf{f}}$, and $\tilde{\boldsymbol{\lambda}}$ together via interactions with the environment. Each time after we observe the values of J, $\mathbf{g}$, and $\mathbf{c}$, we update parameters $\boldsymbol{\theta}_J$, $\boldsymbol{\theta}_g$, and $\boldsymbol{\theta}_c$ to obtain a better approximation of the Lagrangian. Meanwhile, we also update parameters $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\xi}$ to improve the policy. Because J, $\mathbf{g}$, and $\mathbf{c}$ are functions of $\mathbf{x}$, to better approximate J, $\mathbf{g}$, $\mathbf{c}$ and their gradients at $\mathbf{x}$, it is necessary to obtain the values of J, $\mathbf{g}$, and $\mathbf{c}$ in the neighbor of $\mathbf{x}$. To encourage such exploration, we add a noise term $\mathbf{n}^{(t)}$ that reduces over iterations to the output of policy network, i.e., $\mathbf{x} = \tilde{\mathbf{f}} + \mathbf{n}^{(t)}$. The detailed learning procedure is provided in Algorithm 1.

Algorithm 1 Model-Free Unsupervised Learning (Deterministic)

```
1: Initialize neural networks $\tilde{J}$, $\tilde{\mathbf{g}}$, $\tilde{\mathbf{c}}$, $\tilde{\mathbf{f}}$, $\tilde{\boldsymbol{\lambda}}$ with random parameters
   $\boldsymbol{\theta}_J$, $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_c$, $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_\lambda$ and initialize multiplier $\boldsymbol{\xi}$.
2: Initialize replay memory $\mathcal{D}$.
3: for $t = 1, 2, \cdots$ do
4:     Observe $\mathbf{h}^{(t)}$ from the environment.
5:     Execute $\mathbf{x}^{(t)} = \tilde{\mathbf{f}}(\mathbf{h}^{(t)};\boldsymbol{\theta}_f^{(t)}) + \tilde{\mathbf{n}}^{(t)}$.
6:     Observe values of $J^{(t)} = J(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$, $\mathbf{g}^{(t)} = \mathbf{g}(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$,
       and $\mathbf{c}^{(t)} = \mathbf{c}(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$ from the system.
7:     Store $\mathbf{e}^{(t)} = [\mathbf{h}^{(t)},\mathbf{x}^{(t)},J^{(t)},\mathbf{g}^{(t)},\mathbf{c}^{(t)}]$ in $\mathcal{D}$.
8:     Randomly sample a batch of training samples from $\mathcal{D}$ as $\mathcal{B}$.
9:     Update $\boldsymbol{\theta}_J$, $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_c$ by (7) and update $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_\lambda$, $\boldsymbol{\xi}$ by (8).
10: end for
```

So far, we have implicitly assumed that the policy to be learned is continuous (i.e., $\mathbf{f}(\mathbf{h})$ is a continuous function of $\mathbf{h}$), and learn its parameterized form $\tilde{\mathbf{f}}(\mathbf{h};\boldsymbol{\theta}_f)$ as a deterministic policy. In some scenarios, we need to find a discrete policy, e.g., for user scheduling, where parameterizing a deterministic policy is not applicable for both model-based and model-free unsupervised learning frameworks considered above.

Alternatively, we can parameterize a stochastic policy by neural network, which can be used to learn both continuous and discrete policies. Let $\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)$ denote the probability that we execute $\mathbf{x}$ conditioned on $\mathbf{h}$, and $\boldsymbol{\theta}_\pi$ is the network parameter. In this case, the parameterized form of problem P1 becomes

$$\begin{aligned}
\mathsf{P4}:\quad \max_{\boldsymbol{\theta}_\pi}\;& \mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}[J(\mathbf{x},\mathbf{h})] && \text{(10a)}\\
\text{s.t.}\;& \mathbf{g}(\mathbf{x},\mathbf{h}) \preceq \mathbf{0} && \text{(10b)}\\
& \mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}[\mathbf{c}(\mathbf{x},\mathbf{h})] \preceq \mathbf{0} && \text{(10c)}
\end{aligned}$$

where $\mathbf{x}\sim\pi$ denotes that random variable $\mathbf{x}$ is sampled from distribution $\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)$, the objective function and average constraints are also averaged over $\mathbf{x}$. We can obtain the Lagrangian and use neural network $\tilde{\boldsymbol{\lambda}}(\mathbf{h};\boldsymbol{\theta}_\lambda)$ to parameterize $\boldsymbol{\lambda}(\mathbf{h})$. Then, the gradient of Lagrangian with respect to $\boldsymbol{\theta}_\pi$ can be derived as

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}_\pi}\mathcal{L} &= \nabla_{\boldsymbol{\theta}_\pi}\mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}\Big[J - \tilde{\boldsymbol{\lambda}}^T\mathbf{g} - \boldsymbol{\xi}^T\mathbf{c}\Big] \\
&= \mathbb{E}_{\mathbf{h}}\left[\sum_{\mathbf{x}}\nabla_{\boldsymbol{\theta}_\pi}\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)(J - \tilde{\boldsymbol{\lambda}}^T\mathbf{g} - \boldsymbol{\xi}^T\mathbf{c})\right] && \text{(11a)}\\
&= \mathbb{E}_{\mathbf{h}}\left[\sum_{\mathbf{x}}\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)\frac{\nabla_{\boldsymbol{\theta}_\pi}\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)}{\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi)}(J - \tilde{\boldsymbol{\lambda}}^T\mathbf{g} - \boldsymbol{\xi}^T\mathbf{c})\right] \\
&= \mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}\Big[(J - \tilde{\boldsymbol{\lambda}}^T\mathbf{g} - \boldsymbol{\xi}^T\mathbf{c})\nabla_{\boldsymbol{\theta}_\pi}\log(\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi))\Big] && \text{(11b)}
\end{aligned}$$

The gradient of Lagrangian with respect to $\boldsymbol{\theta}_\lambda$ and $\boldsymbol{\xi}$ can be derived as

$$\nabla_{\boldsymbol{\theta}_\lambda}\mathcal{L} = -\nabla_{\boldsymbol{\theta}_\lambda}\mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}\Big[\nabla_{\boldsymbol{\theta}_\lambda}\tilde{\boldsymbol{\lambda}}\mathbf{g}\Big] \tag{12}$$

$$\nabla_{\boldsymbol{\xi}}\mathcal{L} = -\nabla_{\boldsymbol{\xi}}\mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}[\mathbf{c}] \tag{13}$$

Different from the deterministic policy case, the gradients $\nabla_{\mathbf{x}}J$, $\nabla_{\mathbf{x}}\mathbf{g}$, and $\nabla_{\mathbf{x}}\mathbf{c}$ are no longer necessary when we update $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\xi}$ with stochastic gradient method. To compute a sample of the gradient in (11b)$\sim$(13), we only need to observe the value of J, $\mathbf{g}$, and $\mathbf{c}$ from the environment[1] when executing $\mathbf{x}$ at state $\mathbf{h}$. Therefore, $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_\lambda$, and $\boldsymbol{\xi}$ are updated by

$$\begin{aligned}
\boldsymbol{\theta}_\pi^{(t+1)} &\leftarrow \boldsymbol{\theta}_\pi^{(t)} + \delta_f(J - \tilde{\boldsymbol{\lambda}}^T\mathbf{g} - \boldsymbol{\xi}^T\mathbf{c})\nabla_{\boldsymbol{\theta}_\pi}\log(\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi))\\
\boldsymbol{\theta}_\lambda^{(t+1)} &\leftarrow \boldsymbol{\theta}_\lambda^{(t)} + \delta_\lambda\nabla_{\boldsymbol{\theta}_\lambda}\tilde{\boldsymbol{\lambda}}\mathbf{g}\\
\boldsymbol{\xi}^{(t+1)} &\leftarrow \left[\boldsymbol{\xi}^{(t)} + \delta_\xi\mathbf{c}\right]^+
\end{aligned} \tag{14}$$

Remark 2: Although (11b) is derived assuming discrete distribution of $\mathbf{x}$, it can also be derived from a continuous distribution of $\mathbf{x}$. Therefore, (14) (and the following updating rules in (16) and (17)) are also applicable for learning a continuous policy.

Remark 3: When there are no constraints in problem P4, (11b) reduces to

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}_f}\mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}\left[J(\mathbf{x},\mathbf{h})\right]\\
&\qquad = \mathbb{E}_{\mathbf{h},\mathbf{x}\sim\pi}\left[J(\mathbf{x},\mathbf{h})\nabla\log(\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi))\right]
\end{aligned} \tag{15}$$

which coincides with the policy gradient theorem [13] in reinforcement learning, and the update of $\boldsymbol{\theta}_\pi$ in (14) degenerates into the REINFORCE method [13].

The stochastic gradient update in (14) may exhibit large variance [13] and hence converge slowly. Inspired by the advantage actor-critic approach [14], we can subtract a term $\mathbb{E}_{\mathbf{x}\sim\pi}[J(\mathbf{x},\mathbf{h}) - \tilde{\boldsymbol{\lambda}}^T\mathbf{g}(\mathbf{x},\mathbf{h}) - \boldsymbol{\xi}^T\mathbf{c}(\mathbf{x},\mathbf{h})]$ into the parenthesis of (11a), which do not change the expectation of gradients but can reduce the variance. Then, the update for $\boldsymbol{\theta}_\pi$ becomes

$$\begin{aligned}
\boldsymbol{\theta}_\pi^{(t+1)} \leftarrow \boldsymbol{\theta}_\pi^{(t)} + \delta_\pi\Big[&(J - \mathbb{E}_{\mathbf{x}\sim\pi}[J]) - \tilde{\boldsymbol{\lambda}}^T(\mathbf{g} - \mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{g}])\\
&- \boldsymbol{\xi}^T(\mathbf{c} - \mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{c}])\Big]\nabla_{\boldsymbol{\theta}_\pi}\log(\pi(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}_\pi^{(t)})
\end{aligned} \tag{16}$$

Similarly, the update for $\boldsymbol{\theta}_\lambda$ and $\boldsymbol{\xi}$ can be derived as

$$\begin{aligned}
\boldsymbol{\theta}_\lambda^{(t+1)} &\leftarrow \boldsymbol{\theta}_\lambda^{(t)} + \delta_\lambda\nabla_{\boldsymbol{\theta}_\lambda}\tilde{\boldsymbol{\lambda}}(\mathbf{g} - \mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{g}])\\
\boldsymbol{\xi}^{(t+1)} &\leftarrow \left[\boldsymbol{\xi}^{(t)} + \delta_\xi(\mathbf{c} - \mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{c}])\right]^+
\end{aligned} \tag{17}$$

Again, the average terms can be approximated by neural networks as $\mathbb{E}_{\mathbf{x}\sim\pi}[J] \approx \bar{J}(\mathbf{h};\boldsymbol{\theta}_{\bar{J}})$, $\mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{g}] \approx \bar{\mathbf{g}}(\mathbf{h};\boldsymbol{\theta}_{\bar{g}})$ and $\mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{c}] \approx \bar{\mathbf{c}}(\mathbf{h};\boldsymbol{\theta}_{\bar{c}})$, which are updated by minimizing the L_2-norm loss with stochastic gradient descent as

$$\begin{aligned}
\boldsymbol{\theta}_{\bar{J}}^{(t+1)} &\leftarrow \boldsymbol{\theta}_{\bar{J}}^{(t)} - \delta_{\bar{J}}\nabla_{\boldsymbol{\theta}_{\bar{J}}}\left[J(\mathbf{x},\mathbf{h}) - \bar{J}(\mathbf{h};\boldsymbol{\theta}_{\bar{J}}^{(t)})\right]^2\\
\boldsymbol{\theta}_{\bar{g}}^{(t+1)} &\leftarrow \boldsymbol{\theta}_{\bar{g}}^{(t)} - \delta_{\bar{g}}\nabla_{\boldsymbol{\theta}_{\bar{g}}}\left\|\mathbf{g}(\mathbf{x},\mathbf{h}) - \bar{\mathbf{g}}(\mathbf{h};\boldsymbol{\theta}_{\bar{g}}^{(t)})\right\|^2\\
\boldsymbol{\theta}_{\bar{c}}^{(t+1)} &\leftarrow \boldsymbol{\theta}_{\bar{c}}^{(t)} - \delta_{\bar{c}}\nabla_{\boldsymbol{\theta}_{\bar{c}}}\left\|\mathbf{c}(\mathbf{x},\mathbf{h}) - \bar{\mathbf{c}}(\mathbf{h};\boldsymbol{\theta}_{\bar{c}}^{(t)})\right\|^2
\end{aligned} \tag{18}$$

The detailed learning procedure is provided in Algorithm 2.

[1]When model is available, we can compute the values J, $\mathbf{g}$, and $\mathbf{c}$ from their expressions.

Algorithm 2 Model-Free Unsupervised Learning (Stochastic)

```
1: Initialize neural networks $\bar{J}$, $\bar{\mathbf{g}}$, $\bar{\mathbf{c}}$, $\pi$, $\tilde{\boldsymbol{\lambda}}$ with random parameters
   $\boldsymbol{\theta}_{\bar{J}}$, $\boldsymbol{\theta}_{\bar{g}}$, $\boldsymbol{\theta}_c$, $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_\lambda$ and initialize multiplier $\boldsymbol{\xi}$.
2: for $t = 1, 2, \cdots$ do
3:     Observe $\mathbf{h}^{(t)}$ from the environment.
4:     Sample $\mathbf{x}^{(t)}$ from $\pi(\mathbf{x}^{(t)}|\mathbf{h}^{(t)};\boldsymbol{\theta}_\pi^{(t)})$ and execute $\mathbf{x}^{(t)}$.
5:     Observe values of $J^{(t)} = J(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$, $\mathbf{g}^{(t)} = \mathbf{g}(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$,
       and $\mathbf{c}^{(t)} = \mathbf{c}(\mathbf{x}^{(t)},\mathbf{h}^{(t)})$.
6:     Update $\boldsymbol{\theta}_{\bar{J}}$, $\boldsymbol{\theta}_{\bar{g}}$, $\boldsymbol{\theta}_{\bar{c}}$ by (18), and update $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_\lambda$, $\boldsymbol{\xi}$ by substi-
       tuting $\mathbb{E}_{\mathbf{x}\sim\pi}[J] \approx \bar{J}(\mathbf{h}^{(t)};\boldsymbol{\theta}_{\bar{J}}^{(t)})$, $\mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{g}] \approx \bar{\mathbf{g}}(\mathbf{h}^{(t)};\boldsymbol{\theta}_{\bar{g}}^{(t)})$ and
       $\mathbb{E}_{\mathbf{x}\sim\pi}[\mathbf{c}] \approx \bar{\mathbf{c}}(\mathbf{h}^{(t)};\boldsymbol{\theta}_{\bar{c}}^{(t)})$ into (16) and (17).
7: end for
```

III. Case Study: Power Control Problem

In this section, we illustrate how to apply the model-based and model-free unsupervised learning frameworks for solving optimization problems. For easy understanding, we consider a simple power control problem in point to point communications. To provide a baseline, we first derive the analytical solution of the problem. Then, we show how to employ the frameworks when the expression of the objective function are known and unknown.

In what follows we optimize the instantaneous transmit power to minimize the ergodic capacity under the constraints of average transmit power and maximum transmit power,

$$\mathsf{P5}: \max_{P(h)} \quad \mathbb{E}_h\left[R\left(P(h),h\right)\right] \tag{19}$$

$$\text{s.t. } \mathbb{E}_h\left[P(h)\right] \le \bar{P} \tag{19a}$$

$$0 \le P(h) \le P_{\max},\ \forall h \tag{19b}$$

where h is the small-scale channel gain, $P(h)$ is the transmit power adapted to h, $R(P(h),h)$ is the channel capacity, $\bar{P} > 0$ is the maximum average transmit power, and $P_{\max} > \bar{P}$ is the maximum instantaneous transmit power.

A. Analytical Solution

When the channel coding is sufficient long and the noise is Gaussian distributed, the channel capacity can be expressed as the Shannon's formula, i.e., $R\left(P(h),h\right) = \log_2(1 + \frac{hP(h)}{N})$, where $N > 0$ is the power of noise unified by the large-scale channel gain. Then the Karush-Kuhn-Tucker (KKT) conditions of problem (19) can be derived as [8],

$$\frac{1}{N/h + P(h)} + \lambda_1(h) - \lambda_2(h) - \xi = 0 \tag{20a}$$

$$\xi\left(\mathbb{E}_h\left[P(h)\right] - \bar{P}\right) = 0 \tag{20b}$$

$$\lambda_1(h)P(h) = 0,\ \forall h \tag{20c}$$

$$\lambda_2(h)\left(P(h) - P_{\max}\right) = 0,\ \forall h \tag{20d}$$

$$\text{(19a)},\quad \text{(19b)},\ \xi \ge 0,\ \lambda_1(h), \lambda_2(h) \ge 0,\ \forall h \tag{20e}$$

As proved in the Appendix, the solution of the problem is,

$$P^*(h) = \begin{cases} 0, & h \le \xi^* N\\ 1/\xi^* - N/h, & \xi^* N < h < \frac{N}{1/\xi^* - P_{\max}}\\ P_{\max}, & h \ge \frac{N}{1/\xi^* - P_{\max}} \end{cases} \tag{21}$$

where ξ^* satisfies $\mathbb{E}_h\left[P^*(h)\right] = \bar{P}$ and can be computed via bisection searching with known distribution of h. The solution in (21) differs from the water-filling structure [4] due to the additional constraint imposed by $P_{\max}$.

B. Model-Based Unsupervised Learning Method

Problem P5 may not have closed-form solution, say when the finite block-length channel coding is used such that $R(P(h),h)$ is with complex expression. In the sequel, we illustrate how to use model-based unsupervised learning to solve the problem.

The function to be optimized is approximated by a policy network $\tilde{P}(h;\boldsymbol{\theta}_P)$. The constraints in (19b) can be satisfied by setting the active function of the output layer in $\tilde{P}(h;\boldsymbol{\theta}_P)$ as `Sigmoid`, and multiplying the final output by $P_{\max}$. However, to validate the effectiveness of the multiplier network in handling the instantaneous constraints in functional optimization problems, we use `ReLU` as the active function of the output layer to only ensure $\tilde{P}(h;\boldsymbol{\theta}_P) \geq 0$, and introduce the multiplier network $\tilde{\lambda}(h;\boldsymbol{\theta}_\lambda)$ to ensure the constraint $\tilde{P}(h;\boldsymbol{\theta}_P) \leq P_{\max}$ with primal-dual stochastic gradient method given by (8). Then, the power control policy and the Lagrange multipliers can be updated by

$$\boldsymbol{\theta}_P^{(t+1)} \leftarrow \boldsymbol{\theta}_P^{(t)} + \frac{\delta_P}{|\mathcal{B}|}\sum_{h\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_P}\tilde{P}(\nabla_P R - \tilde{\lambda} - \xi) \quad (22a)$$

$$\boldsymbol{\theta}_\lambda^{(t+1)} \leftarrow \boldsymbol{\theta}_\lambda^{(t)} + \frac{\delta_\lambda}{|\mathcal{B}|}\sum_{h\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_\lambda}\tilde{\lambda}(\tilde{P} - P_{\max}) \quad (22b)$$

$$\xi^{(t+1)} \leftarrow \left[\xi^{(t)} - \frac{\delta_\xi}{|\mathcal{B}|}\sum_{h\in\mathcal{B}} \tilde{P}\right]^+ \quad (22c)$$

where δ_P, δ_λ, and δ_ξ are the learning rates, and $\mathcal{B}$ denotes a batch of training samples.

C. Model-Free Unsupervised Learning Method

When the channel coding is short or the noise is not Gaussian distributed, the Shannon's formula is not applicable and the expression of $R(P(h),h)$ is hard to obtain. In the following, we illustrate how to use the proposed model-free unsupervised learning to solve problem P5.

The objective function is approximated by introducing the value network $\tilde{R}(P,h;\boldsymbol{\theta}_R)$, which is then used to compute the approximated gradient $\nabla_P\tilde{R} \approx \nabla_P R$ for updating the policy network parameter $\boldsymbol{\theta}_P$ in (22a). The updates for $\boldsymbol{\theta}_\lambda$ and ξ are the same as in (22b) and (22c) since the expressions of constraints are known. After observing the actual data rate acheived by transmiting with power P at channel state h, the value network $\tilde{R}(P,h;\boldsymbol{\theta}_R)$ is trained based on the observed value of $R(P,h)$ according to (7) as

$$\boldsymbol{\theta}_R^{(t+1)} \leftarrow \boldsymbol{\theta}_R^{(t)} - \frac{\delta_R}{|\mathcal{B}|}\sum_{(P,h,R)\in\mathcal{B}} \nabla_{\boldsymbol{\theta}_R}\Big[R(P,h) - \tilde{R}(P,h;\boldsymbol{\theta}_R)\Big]^2 \quad (23)$$

where δ_R denotes the learning rate.

IV. Numerical and Simulation Results

In this section, we validate the proposed mode-free unsupervised learning frameworks by considering problem P5, where in simulation the channel coding is assumed long and the noise is assumed Gaussian.

The simulation setup is as follows. The maximal instantaneous and average transmit powers are $P_{\max} = 40$ W and $\bar{P} = 30$ W, respectively. The distance between the transmitter and the receiver is $d = 500$ m. The noise power spectral density is -174 dBm/Hz and the bandwidth is 20 MHz. We consider Rayleigh fading channels and the path loss is modeled by $35.3 + 37.6\log_{10}(d)$ in dB.

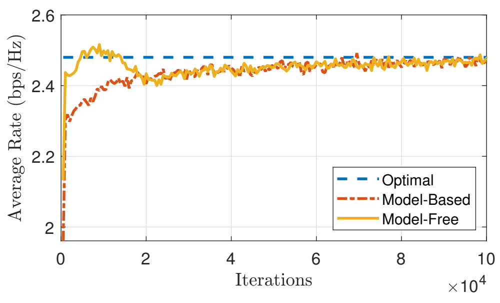
The hyper-parameters used for model-based and model-free frameworks are as follows. Both $\tilde{\lambda}$ and $\tilde{P}$ have three fully-connected hidden layers with 50, 40, and 30 nodes, respectively. $\tilde{R}$ has two hidden layers with 200 and 150 nodes, respectively. All the hidden layers and the output layers of $\tilde{\lambda}$ and $\tilde{P}$ use `ReLU` as the activation function. The output layer of $\tilde{R}$ has no activation function. We use Adam [15] for training all the neural networks with learning rate $\delta_P = \delta_\lambda = 10^{-3}$ for $\tilde{P}$ and $\tilde{\lambda}$, and $\delta_R = 5\times10^{-3}$ for $\tilde{R}$. The batch size is $|\mathcal{B}| = 32$. $\tilde{P}$ is initialized as 10. Both $\tilde{\lambda}$ and ξ are initialized as 0. The noise term for exploration in model-free learning is set as $n^{(t)} = \epsilon^{(t)}\mathcal{N}^{(t)}$ where $\mathcal{N}^{(t)}$ denotes Gaussian noise with zero mean and unit variance. The value of $\epsilon^{(t)}$ is set as 10 for the first 5×10^3 iterations and then decreases linearly to zero for the next 1.5×10^4 iterations. All the simulation results are averaged over 50 rounds of learning.

In Fig. 1, we compare the convergence of model-based and model-free unsupervised learning. Both model-based and model-free learning can converge to the average rate achieved by the optimal solution $P^*(h)$ numerically computed with (21) (with legend "Optimal"). The average rate achieved by model-free learning can be even higher than the optimal solution at the beginning due to violation of constraints. We show the violations of instantaneous and average constraints in Fig. 1(b) and Fig. 1(c), respectively. Since model-free learning needs the exploration to learn the expression of the objective function, the violations of constraints are more severe than model-based method at the beginning of learning due to insufficient training samples. With the increase of iterations, both model-based and model-free learning can satisfy all the constraints after convergence. Moreover, the number of iterations for converging to the optimal solution are close for model-based and model-free learning. This demonstrates the efficiency of proposed model-free unsupervised learning framework where the policy, multiplier, and value networks are trained simultaneously.

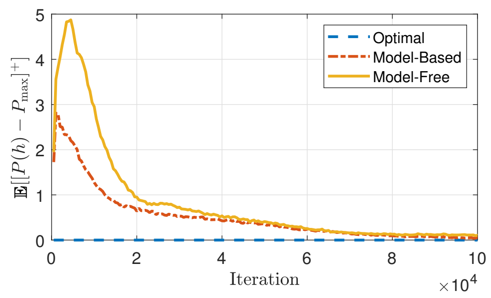
In Fig. 2, we compare the behavior of the policies learned by model-based and model-free frameworks with the optimal solution. We can see that the learned policies behave almost the same with optimal policy.

V. Conclusions

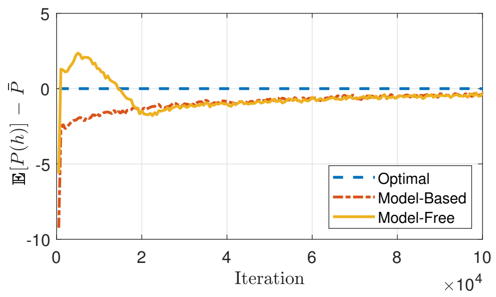
In this paper, we proposed an framework to solve optimization problems with constraints by model-free unsupervised learning, and revealed the connections with reinforcement learning. We illustrated how to apply the proposed framework by a power control problem. Numerical and simulation results



(a) Average rate.



(b) Instanstaneous constraint violation.



(c) Average constraint violation.

Fig. 1. Convergence comparison. The results are averaged over 500 successive iterations.

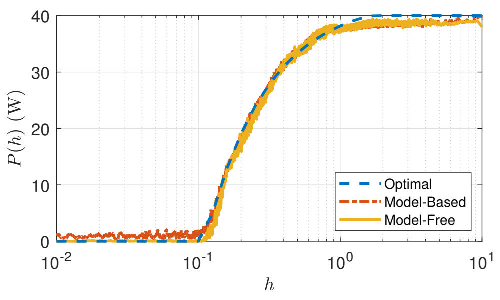


Fig. 2. Comparison of learned policy after 10^5 iterations.

validated our framework and showed that model-free unsupervised learning can converge to the optimal policy with similar speed as model-based unsupervised learning.

APPENDIX

To find the solution from the KKT conditions in (20), $P^*(h)$, $\lambda_1^*(h)$, $\lambda_2^*(h)$, and ξ^*, we first prove $\xi^* > 0$.

Assume $\xi^* = 0$. Since $\frac{1}{N/h+P^*(h)} > 0$ and $\lambda_1^*(h) \geq 0$, we have $\lambda_2^*(h) > 0$ according to (20a). Then, $P^*(h) = P_{\max}$ can be derived from (20d). In this case, $\mathbb{E}_h\left[P(h)\right] = P_{\max} > \bar{P}$, which violate the constraint in (19a). Therefore, $\xi^* > 0$. From (20b), we further have,

$$\mathbb{E}_h\left[P^*(h)\right] = \bar{P} \tag{24}$$

When $h < \xi^* N$, we have $\frac{1}{N/h+P^*(h)} - \xi^* < 0$. In this case, $\lambda_1^*(h) > 0$ according to (20a) and $P^*(h) = 0$ according to (20c). When $h = \xi^* N$, $\frac{1}{N/h+P^*(h)} - \xi^* < 0$ if $P^*(h) > 0$, which on the contrary results in $P^*(h) = 0$. Therefore, $P^*(\xi^* N) = 0$.

When $h > \frac{N}{1/\xi^* - P_{\max}}$, we have $\frac{1}{N/h+P^*(h)} - \xi^* > 0$. In this case, $\lambda_2^*(h) > 0$ according to (20a) and $P^*(h) = P_{\max}$ according to (20d). When $h = \frac{N}{1/\xi^* - P_{\max}}$, $\frac{1}{N/h+P^*(h)} - \xi^* > 0$ if $P^*(h) < P_{\max}$, which results in $P^*(h) = P_{\max}$, contradicting with $P^*(h) < P_{\max}$. Therefore, $P^*(\frac{N}{1/\xi^* - P_{\max}}) = P_{\max}$.

When $\xi^* N < h < \frac{N}{1/\xi^* - P_{\max}}$, we have $\frac{1}{N/h+P_{\max}} < \xi^* < \frac{1}{N/h+0}$. In this case, if $P^*(h) = 0$, then $\lambda_2^*(h) > 0$ according to (20a), which results in $P^*(h) = P_{\max}$, contradicting with $P^*(h) = 0$. Similarly, if $P^*(h) = P_{\max}$, then $\lambda_1^*(h) > 0$ according to (20a), which results in $P^*(h) = 0$, contradicting with $P^*(h) = P_{\max}$. Therefore, we have $0 < P^*(h) < P_{\max}$ and $\lambda_1^*(h), \lambda_2^*(h) = 0$. According to (20a) we further have $P^*(h) = 1/\xi^* - N/h$.

Finally, according to the solution of $P^*(h)$, ξ^* can be solved from (24).

REFERENCES

[1] L. Kong, S. Han, and C. Yang, "Hybrid precoding with rate and coverage constraints for wideband massive MIMO systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4634–4647, Jul. 2018.
[2] D. Liu and C. Yang, "Caching at base stations with heterogeneous user demands and spatial locality," *IEEE Trans. Commun.*, vol. 67, no. 2, pp. 1554–1569, Feb 2018.
[3] E. Zeidler, *Nonlinear functional analysis and its applications: III: variational methods and optimization*. Springer Science & Business Media, 2013.
[4] A. Goldsmith, *Wireless Communications*. Cambridge Univ. Press, 2005.
[5] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge Univ. Press, 2004.
[6] O. C. Zienkiewicz, R. L. Taylor, P. Nithiarasu, and J. Zhu, *The finite element method*. McGraw-hill London, 1977, vol. 3.
[7] C. Sun and C. Yang, "Unsupervised deep learning for ultra-reliable and low-latency communications," 2019. [Online]. Available: http://arxiv.org/abs/1905.13014
[8] J. Gregory, *Constrained optimization in the calculus of variations and optimal control theory*. Chapman and Hall/CRC, 2018.
[9] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
[10] C. Sun and C. Yang, "Learning to optimize with unsupervised learning: Training deep neural networks for URLLC," in *IEEE PIMRC, accepted*, 2019.
[11] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.
[12] 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," in *Proc. ICLR*, 2016.
[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Cambridge, 1998.
[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016.
[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2014.