A Reinforcement Learning Approach to Parameter Selection for Distributed Optimization in Power Systems

Sihan Zeng, Daniel K. Molzahn
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia USA
{szeng30, molzahn}@gatech.edu

Alyssa Kody*, Youngdae Kim†, Kibaek Kim†
Energy Systems*, Mathematics and Computer Science†
Argonne National Laboratory
Lemont, Illinois USA
{akody, youngdae, kimk}@anl.gov

***Abstract*—With the increasing penetration of distributed energy resources, distributed optimization algorithms have attracted significant attention for power systems applications due to their potential for superior scalability, privacy, and robustness to a single point-of-failure. The Alternating Direction Method of Multipliers (ADMM) is a popular distributed optimization algorithm; however, its convergence performance is highly dependent on the selection of penalty parameters, which are usually chosen heuristically. In this work, we use reinforcement learning (RL) to develop an adaptive penalty parameter selection policy for the AC optimal power flow (ACOPF) problem solved via ADMM with the goal of minimizing the number of iterations until convergence. We train our RL policy using deep Q-learning, and show that this policy can result in significantly accelerated convergence (up to a 59% reduction in the number of iterations compared to existing, curvature-informed penalty parameter selection methods). Furthermore, we show that our RL policy demonstrates promise for generalizability, performing well under unseen loading schemes as well as under unseen losses of lines and generators (up to a 50% reduction in iterations). This work thus provides a proof-of-concept for using RL for parameter selection in ADMM for power systems applications.**

***Index Terms*—alternating direction method of multipliers, alternating current optimal power flow, distributed optimization, reinforcement learning, deep Q-learning.**

I. Introduction

The rapid growth in distributed energy resources (DER) such as solar PV, batteries, and plug-in vehicles necessitates new computational methods for cooperatively controlling these devices to maximize the efficiency and reliability of power systems. Traditionally, set-points for controllable devices are determined by using centralized methods, meaning that all data are congregated in a central location (often an independent system operator), which solves a large-scale optimization problem. However, centralized methods may be unable to computationally manage the increase in problem size and complexity resulting from adding millions of DERs. Furthermore, centralized computing raises other issues such as data privacy [1], [2] and is also vulnerable via a single point of failure or attack. Consequently, there has been much interest from the power systems community in distributed computation methods, where large problems are partitioned into smaller problems that can be solved in parallel [3]. Distributed optimization can be used to either (1) physically spread computation across an electric network such that devices locally solve a small optimization problem and exchange solutions directly with neighboring devices until converging to the overall solution [4], or (2) partition large problems in the context of high-performance computing (HPC) [5].

Despite the promise of distributed methods for power systems applications, they have not yet been widely adopted in industry. A review by Wang et al. [4] finds that one reason for this lack of adoption is that distributed optimization algorithms "may require many iterations and in turn increase computational burden beyond the limit of practical interest for power industry." For example, commonly used distributed methods in power systems such as Alternating Direction Method of Multipliers (ADMM), Auxiliary Problem Principle (APP), and Analytical Target Cascading (ATC) may require hundreds or thousands of iterations to converge to a sufficiently high accuracy and only have convergence guarantees for a limited class of problems [3], [4], [6].

While the worst-case computational performance of optimization algorithms is characterized by complexity theory, in practice, user-selected algorithmic parameters can significantly reduce typical solution times. For example, it is widely known that the convergence performance of ADMM is sensitive to the choice of penalty parameters, which are heuristically defined [7]. Furthermore, for some classes of problems, like nonconvex problems, there are no ADMM convergence guarantees and poor parameter selection can lead to solution divergence. In [8], Mhanna et al. demonstrate that the nonconvex and NP-hard AC Optimal Power Flow (ACOPF) problem [9] solved via ADMM has widely varying convergence results based on the selection of penalty parameters, including divergence. Recent theoretical advancements [10] enable ADMM to guarantee convergence for the ACOPF problem; however, convergence performance still depends on parameter settings.

To speed up convergence and reduce the effort of penalty parameter tuning in ADMM, adaptive penalty parameter algorithms have been studied in order to update penalty parameters during the optimization using feedback from the previous iteration. Examples include residual balancing [11], which increase or decrease penalty parameters based on the relative magnitudes of the primal and dual residuals, and methods that use estimates of the local curvature of the dual function to inform updates [12]. Mhanna et al. in [13] demonstrate significantly improved convergence performance for the ACOPF problem using adaptive penalty parameter algorithms over vanilla ADMM with static penalty parameters. However, the techniques in [13] still rely on tuned parameters within the adaptive algorithm, and also require additional logic steps and the computation and storage of gradient information.

Ultimately, these existing adaptive penalty parameter algorithms rely on heuristics, presenting an opportunity for their replacement with machine learning techniques, which may be able to outperform. In this work, we develop a reinforcement learning (RL) [14] method to train a policy for selecting penalty parameters to accelerate the convergence of an ADMM algorithm for solving ACOPF problems. The ADMM parameter selection task has a sequential decision making structure, as penalty parameters are updated based on feedback from past iterations. RL, as a convenient framework for sequential decision making problems, is a natural fit for this task.

Machine learning techniques have been used to design optimization methods (e.g., [15], [16]). There are fewer works that develop embedded-ML methods specifically for distributed optimization algorithms. In [17], a recurrent neural network is trained to predict the converged values of variables in ADMM subproblems for DC-OPF. In [18], the authors replace ADMM subproblems with an RL policy that predicts solutions. In [19], the authors learn to solve ADMM subproblems by recasting them as deep neural networks. Recent contemporaneous work [20] trains an RL policy to tune parameters to accelerate ADMM convergence using policy gradient methods; however, they focus on convex QP problems with convergence guarantees and do not specifically consider power systems problems. Moreover, RL methods have shown promise in other power systems applications (e.g., [21], [22]).

In this work, we investigate the use of RL in the important task of learning ADMM penalty parameters. Transforming the penalty parameter selection problem into an RL problem, this work has three main contributions:

- Formulation of parameter selection for distributed ACOPF as a RL problem.
- Development of a novel deep Q-learning policy scheme for ADMM penalty parameter selection.
- Demonstration of trained policies on test networks with unseen loads and unseen line and generator contingencies.

The rest of the paper is organized as follows. In Section II, we introduce the ACOPF problem and its component-based ADMM formulation. We discuss the importance of the penalty parameter ρ for the convergence of the ADMM algorithm. In Section III, we briefly highlight the connection between the penalty parameter selection problem and RL and provide an overview of RL and deep Q-learning. In Section IV, we dive into our RL algorithm design, including the choice of the state space, action space, and reward function. We present numerical experiments in Section V, and conclude with future directions in Section VI.

II. Component-based Decomposition of ACOPF

We consider a component-based decomposition of ACOPF [8], [13] that can be efficiently solved by ADMM, where each component in the network (i.e., buses, lines, generators) form their own subproblems. Although region-based ADMM decompositions [10], [23] are also popular for power systems applications and result in fewer subproblems, the advantage of the component-based formulation is that each subproblem is small and can be solved efficiently, lending itself well to HPC implementations [5]. Furthermore, component-based decompositions do not require making partitioning decisions, which can impact performance.

A. ACOPF Formulation

We present the ACOPF problem formulation below in (1). This problem seeks the least costly operating points of the generators within their lower and upper limits while obeying physical laws. These physical laws are represented by power flow equations (1b)–(1c) and (1i)–(1l).

$$\underset{p_{g_i}, q_{g_i}, w_i, \theta_i, w_{ij}^R, w_{ij}^I}{\text{minimize}} \sum_{i \in \mathcal{B}} \sum_{g_i \in \mathcal{G}_i} f_{g_i}(p_{g_i}) \tag{1a}$$

subject to

$$\sum_{g_i \in \mathcal{G}_i} p_{g_i} - p_{d_i} = g_i^S w_i + \sum_{j \in \mathcal{B}_i} p_{ij}, \qquad \forall i \in \mathcal{B} \tag{1b}$$

$$\sum_{g_i \in \mathcal{G}_i} q_{g_i} - q_{d_i} = -b_i^S w_i + \sum_{j \in \mathcal{B}_i} q_{ij}, \qquad \forall i \in \mathcal{B} \tag{1c}$$

$$\sqrt{p_{ij}^2 + q_{ij}^2} \le \bar{r}_{ij}, \qquad \forall (i,j) \in \mathcal{L} \tag{1d}$$

$$\sqrt{p_{ji}^2 + q_{ji}^2} \le \bar{r}_{ij}, \qquad \forall (i,j) \in \mathcal{L} \tag{1e}$$

$$\underline{p}_{g_i} \le p_{g_i} \le \overline{p}_{g_i}, \qquad \forall g_i \in \mathcal{G}_i, \forall i \in \mathcal{B} \tag{1f}$$

$$\underline{q}_{g_i} \le q_{g_i} \le \overline{q}_{g_i}, \qquad \forall g_i \in \mathcal{G}_i, \forall i \in \mathcal{B} \tag{1g}$$

$$-2\pi \le \theta_i \le 2\pi, \qquad \forall i \in \mathcal{B} \tag{1h}$$

$$p_{ij} = g_{ii} w_i + g_{ij} w_{ij}^R + b_{ij} w_{ij}^I, \qquad \forall (i,j) \in \mathcal{L} \tag{1i}$$

$$q_{ij} = -b_{ii} w_i - b_{ij} w_{ij}^R + g_{ij} w_{ij}^I, \qquad \forall (i,j) \in \mathcal{L} \tag{1j}$$

$$p_{ji} = g_{jj} w_j + g_{ji} w_{ij}^R - b_{ji} w_{ij}^I, \qquad \forall (i,j) \in \mathcal{L} \tag{1k}$$

$$q_{ji} = -b_{jj} w_j - b_{ji} w_{ij}^R - g_{ji} w_{ij}^I, \qquad \forall (i,j) \in \mathcal{L} \tag{1l}$$

$$(w_{ij}^R)^2 + (w_{ij}^I)^2 = w_i w_j, \qquad \forall (i,j) \in \mathcal{L} \tag{1m}$$

$$\theta_i - \theta_j = \arctan(w_{ij}^I / w_{ij}^R), \qquad \forall (i,j) \in \mathcal{L} \tag{1n}$$

In this optimization problem, we use $\mathcal{B}, \mathcal{B}_i, \mathcal{G}_i$, and $\mathcal{L}$ to denote the set of buses, the set of buses connected to bus i, the set of generators at bus i, and the set of lines, respectively. The decision variables include p_{g_i} and q_{g_i}, which are the real and reactive power outputs of generator g_i at bus i, w_i and θ_i, which are the squared voltage magnitude $(= v_i^2)$ and angle at bus i, and w_{ij}^R and w_{ij}^I, which are defined to be $v_i v_j \cos\theta_{ij}$ and $v_i v_j \sin\theta_{ij}$, respectively, with v_i being the voltage magnitude at bus i and $\theta_{ij} := \theta_i - \theta_j$. $f_{g_i}(\cdot)$ is a quadratic function of the real power output that encodes the power generation cost. The other quantities in (1b)–(1m) are parameters that depend on the structure and physical properties of the power network (see [13] for more details).

B. ADMM Formulation for ACOPF

Consider the following optimization, which is the general problem form for ADMM:

$$\begin{aligned} \underset{x\in\mathbb{R}^{n_1},\bar{x}\in\mathbb{R}^{n_2}}{\text{minimize}} \quad & f(x) + g(\bar{x}) \\ \text{subject to} \quad & Ax + B\bar{x} = c, \end{aligned} \tag{2}$$

where $A \in \mathbb{R}^{n_3\times n_1}$, $B \in \mathbb{R}^{n_3\times n_2}$, and $c \in \mathbb{R}^{n_3}$, and where $f : \mathbb{R}^{n_1} \to \mathbb{R}$ and $g : \mathbb{R}^{n_2} \to \mathbb{R}$ are closed, convex functions. Let $y \in \mathbb{R}^{n_3}$ be the vector of Lagrange multipliers, used to enforce the linear equality constraint in (2). Then, we form the augmented Lagrangian as $L_\rho(x, \bar{x}, y) = f(x) + g(\bar{x}) + y^T(Ax+B\bar{x}-c)+(Ax+B\bar{x}-c)^\top\Omega(Ax+B\bar{x}-c)$, where the matrix $\Omega \in \mathbb{R}^{n_3\times n_3}$ is a diagonal matrix with the i-th diagonal entry defined as $\Omega_{ii} = \rho_i/2$. We define ρ_i as the i-th *penalty parameter*.

Let $k \in \mathbb{N}$ be the ADMM iteration counter, where iterates are marked via square brackets in superscript. Each iteration, we first update variable x according to (3a). Then, using this updated value of x, variable $\bar{x}$ is updated according to (3b). Last, the Lagrange multipliers are updated via (3c).

$$x^{[k+1]} = \underset{x}{\operatorname{argmin}}\, L_\rho(x, \bar{x}^{[k]}, y^{[k]}) \tag{3a}$$

$$\bar{x}^{[k+1]} = \underset{\bar{x}}{\operatorname{argmin}}\, L_\rho(x^{[k+1]}, \bar{x}, y^{[k]}) \tag{3b}$$

$$y^{[k+1]} = \underset{y}{\operatorname{argmin}}\, L_\rho(x^{[k+1]}, \bar{x}^{[k+1]}, y) \tag{3c}$$

This iterative process continues until the 2-norms of the primal and dual residuals, which represent the feasibility of the primal and dual problems, have have met their convergence thresholds, $\epsilon_p > 0$ and $\epsilon_d > 0$, respectively:

$$\left\|r_p^{[k]}\right\|_2 \le \epsilon_p \quad \text{and} \quad \left\|r_d^{[k]}\right\|_2 \le \epsilon_d, \tag{4}$$

where $r_p^{[k]}$ and $r_d^{[k]}$ are the primal and dual residuals:

$$r_p^{[k]} = Ax^{[k]} + B\bar{x}^{[k]} - c \tag{5}$$

$$r_d^{[k]} = 2\Omega A^T B\left(\bar{x}^{[k]} - \bar{x}^{[k-1]}\right). \tag{6}$$

Reference [13] proposes a method to decompose the ACOPF problem (1), based on the observation that components can be decoupled by duplicating variables connecting between them. Generators and buses are coupled through the p_{g_i} and q_{g_i} variables, and branches and buses are coupled through the $p_{ij}, q_{ij}, p_{ji}, q_{ji}, w_i, \theta_i, w_j,$ and θ_j variables for a given branch (i, j). By duplicating these variables and enforcing a consensus through coupling constraints, we can re-formulate the problem as the composition of small sub-problems. For example, we create new $p_{g_i(i)}$ and $q_{g_i(i)}$ variables, duplicates of p_{g_i} and q_{g_i}, respectively, and add consensus constraints $p_{g_i} - p_{g_i(i)} = 0$ and $q_{g_i} - q_{g_i(i)} = 0$. Similarly, we duplicate variables that connect between branches and buses, leading to new $p_{ij(i)}, q_{ij(i)}, p_{ji(i)}, q_{ji(i)}, w_{i(ij)}, \theta_{i(ij)}, w_{j(ij)}$, and $\theta_{j(ij)}$ variables and consensus constraints $p_{ij} - p_{ij(i)} = 0, q_{ij} - q_{ij(i)} = 0, p_{ji} - p_{ji(i)} = 0, q_{ji} - q_{ji(i)} = 0, w_{i(ij)} - w_i = 0, \theta_{i(ij)} - \theta_i = 0, w_{j(ij)} - w_j = 0$, and $\theta_{j(ij)} - \theta_j = 0$. By assigning $x = [(p_{g_i}, q_{g_i}, p_{ij}, q_{ij}, p_{ji}, q_{ji}, w_{i(ij)}, w_{j(ij)}, \theta_{i(ij)}, \theta_{j(ij)})]_{g_i\in\mathcal{G}_i}^{(i,j)\in\mathcal{L}}$ and $\bar{x} = [(p_{g_i(i)}, q_{g_i(i)}, p_{ij(i)}, q_{ij(i)}, p_{ji(i)}, q_{ji(i)}, w_i, \theta_i)]_{i\in\mathcal{B}}$ with proper A, B and $c = 0$, we have consensus constraints of the form in (2). Applying ADMM to the reformulation permits massively parallel computations that can be accelerated using GPUs [5].

Note that the i-th coupling constraint in the ADMM formulation is associated with penalty parameter ρ_i. In [8], improved convergence performance is observed for ACOPF when ρ_i values are assigned based on the type of coupling constraint they are penalizing. Therefore, we categorize the coupling constraints into two different types: constraints that correspond to the real (p) and reactive (q) power flows, and constraints that correspond to voltage magnitudes (v) and angles (θ). We use n_{pq} and $n_{v\theta}$ to denote the number of the two types of constraints, respectively. We use $\rho_{pq} \in \mathbb{R}^{n_{pq}}$ for the penalty parameters for the p or q coupling constraints and $\rho_{v\theta} \in \mathbb{R}^{n_{v\theta}}$ for the penalty parameters for the v or θ coupling constraints.

III. Reinforcement Learning Overview

From the perspective of accelerating convergence, we seek the optimal parameter ρ throughout the ADMM iterations to encourage the primal and dual residuals to reach the convergence thresholds in as few iterations as possible. The choice of ρ in the k-th ADMM iteration is based on the current iterates $x^{[k]}, \bar{x}^{[k]}, y^{[k]}$, and in turns affects $x^{[k+1]}, \bar{x}^{[k+1]}, y^{[k+1]}$, the iterates of the next iteration. This naturally makes the problem a sequential decision making problem, which motivates us to approach it using RL. In this section, we provide an overview of RL modeled as a Markov Decision Process (MDP) and discuss Q-learning, an effective class of RL algorithm that we will use in this work.

A. Reinforcement Learning & Markov Decision Process

Reinforcement learning is a framework for sequential decision making that involves an agent interacting with an environment. The agent observes the state and reward information from the environment and selects an action in response. The action makes the environment transition from the current state to the next state and reveal the next reward. The goal of the agent is to choose the optimal actions to maximize the discounted cumulative reward it receives from the environment.

Mathematically, we consider the Markov Decision Process (MDP), characterized by the 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ denote the state and action space. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition probability kernel that specifies the distribution of the next state given the current state and action. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $\gamma \in (0, 1)$ is the discount factor that discounts rewards received in the future. Due to the Markovian nature of the environment, selecting the optimal sequence of actions can be equivalently expressed as finding a policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$. The policy is a mapping from the state space to the probability simplex over the action space, and we use $\pi(a \mid s)$ to represent the probability of choosing action

a in state s. The RL agent seeks to maximize the discounted cumulative reward by solving the optimization problem

$$\max_{\pi} \ \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R(s^{[k]},a^{[k]})\right] \tag{7}$$
$$\text{s.t. } a^{[k]}\sim\pi(\cdot \mid s^{[k]}),\ \ s^{[k+1]}\sim\mathcal{P}(\cdot \mid s^{[k]},a^{[k]}),\ \forall k=0,1,\cdots$$

where $x\sim d$ denotes drawing a sample x uniformly from the distribution d.

Two main classes of methods to solve the RL problem (7) are the policy gradient algorithm and Q-learning. The method used in this work is a variant of Q-learning, which we briefly review in the following subsection.

B. Q-Learning

In RL, the "value" of a state-action pair under a policy π is measured by the discounted cumulative reward obtained by applying action a in state s and then following the policy π:

$$Q_\pi(s,a)=\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^k R(s^{[k]},a^{[k]}) \mid s^{[0]}=s,a^{[0]}=a\right].$$

This is commonly known as the Q function under policy π. Under mild assumptions on the reward function, there always exists a deterministic optimal policy π^* [24], which has a Q function obeying the Bellman equation for all $s\in\mathcal{S}$, $a\in\mathcal{A}$:

$$Q_{\pi^*}(s,a)=R(s,a)+\gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}[\max_{a'\in\mathcal{A}}Q_{\pi^*}(s',a')].$$

On the other hand, π^* can be determined from its Q function. Defining $a^*(s)=\operatorname{argmax}_{a\in\mathcal{A}}Q_{\pi^*}(s,a)$, we have

$$\pi^*(a\mid s)=\begin{cases}1, & \text{if } a=a^*(s),\\ 0, & \text{otherwise.}\end{cases}$$

In other words, the optimal policy π^* is to always take the action with the highest Q value. This suggests that to learn π^*, we can equivalently learn its Q function through stochastic approximation [25], where we maintain a table $Q^{[k]}\in\mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ to track Q_{π^*} and update it iteratively as

$$Q^{[k+1]}\left(s^{[k]},a^{[k]}\right)=Q^{[k]}\left(s^{[k]},a^{[k]}\right)+\alpha^{[k]}\Big(R\left(s^{[k]},a^{[k]}\right)$$
$$+\gamma\max_{a\in\mathcal{A}}Q^{[k]}\left(s^{[k+1]},a\right)-Q^{[k]}\left(s^{[k]},a^{[k]}\right)\Big),$$

where $s^{[k]},a^{[k]},s^{[k+1]}$ are samples collected when the agent interacts with the environment in the k-th iteration and $\alpha^{[k]}$ is the step size. As the dimension of the Q table grows linearly in the cardinality of the state and action space, function approximation is introduced to parametrize it in large-scale problems. In this work, we use a neural network to parametrize the Q function. We will use ψ to denote the parameters of the neural network and $Q_\psi:\mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}\to\mathbb{R}$ to denote the Q function parameterized by ψ. We also employ standard techniques such as double Q-learning [26] and prioritized experience replay [27] to stabilize and accelerate training.

IV. Algorithm Design

In this section, we use the RL framework in Section III to develop a method that learns the penalty parameter ρ in the ACOPF ADMM algorithm to accelerate its convergence.

While our objective is to reduce the number of ADMM iterations until convergence, the goal of an RL agent is to maximize the discounted cumulative reward it collects from the environment. To translate our objective to that of the RL agent, we have to model our ADMM parameter selection problem as a suitable RL problem, which includes identifying the environment and dynamics and making the proper choice of the state space, action space, and reward function.

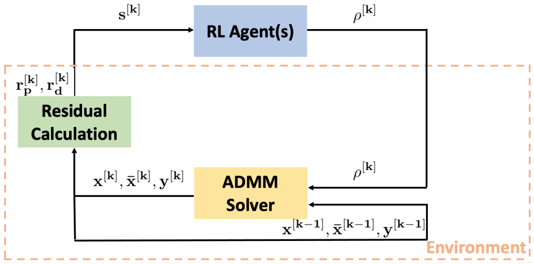

$\mathbf{s}^{[\mathbf{k}]}$
RL Agent(s)
$\rho^{[\mathbf{k}]}$
$\mathbf{r}_\mathbf{p}^{[\mathbf{k}]},\mathbf{r}_\mathbf{d}^{[\mathbf{k}]}$
Residual Calculation
$\mathbf{x}^{[\mathbf{k}]},\bar{\mathbf{x}}^{[\mathbf{k}]},\mathbf{y}^{[\mathbf{k}]}$
ADMM Solver
$\rho^{[\mathbf{k}]}$
$\mathbf{x}^{[\mathbf{k-1}]},\bar{\mathbf{x}}^{[\mathbf{k-1}]},\mathbf{y}^{[\mathbf{k-1}]}$
Environment

Fig. 1. ADMM Solver and RL Agent Interaction

A. RL Environment & Reward Function

We regard the ADMM solution process as the RL environment. Each iteration of the ADMM algorithm corresponds to one RL iteration. In iteration $k=0,1,\ldots$, the agent observes the current state of the ADMM solver $s^{[k]}$. Based on $s^{[k]}$, the agent selects an action $a^{[k]}$, which is simply a choice of $\rho^{[k]}$, the penalty parameter of the k-th iteration, and receives a reward $R(s^{[k]},a^{[k]})$, which we will design to reflect the value of the current state to the ADMM convergence. The parameter $\rho^{[k]}$ is then fed back to the ADMM solver for another ADMM iteration. This process is repeated until both the primal and dual residuals from the ADMM solve drop below the thresholds, i.e., (4). The interaction of the environment and the agent in ADMM solving process is shown in Figure 1.

State space: The state is an important source of information that should summarize the progress of the ADMM solve and include key factors necessary for the agent to make decisions about ρ. In this problem, we naturally expect the primal and dual residuals to contain information about the optimal choice of ρ. To ensure that $s^{[k]}$ sufficiently represents the state of the ADMM solving process, we include the past n-point history of the residuals in $s^{[k]}$.

$$s^{[k]}=[(r_p^{[k-n+1]},r_d^{[k-n+1]}),\cdots,(r_p^{[k]},r_d^{[k]})]\in\mathbb{R}^{2n\times(n_{pq}+n_{v\theta})}.$$

Action space: The algorithm used in this paper is a variant of Q-learning, which by design requires a discrete and finite action space. Since ρ is only restricted to being positive, ρ can be chosen from a continuous and infinitely large range of values. However, in the context of ACOPF problems, existing literature shows that ρ values picked from a much smaller range result in superior convergence speed. Specifically, [13] suggests using two different ρ for the two types of constraints:

for constraints related to real and reactive power, $\rho_{pq} = 400$ is used for IEEE 9-bus, 30-bus, and 118-bus systems; for constraints related to voltage, $\rho_{v\theta} = 40000$ is used for IEEE 9-bus and 30-bus systems and $\rho_{v\theta} = 4000$ is used for the 118-bus system. Though this particular choice of the parameters may not be optimal, it suggests a reasonable range for ρ to provide to the RL agent. We select $[100, 1000]$ as the range of ρ_{pq}, and $[500, 7000]$ for $\rho_{v\theta}$ in the 9-bus and 30-bus systems and $[200, 10000]$ in the 118-bus system, discretized to 10 possible actions for each constraint (see Table I).

TABLE I
RL ACTION SPACE (ρ) & INITIAL ρ VALUES

ρ Category	Initial Value	Action Space
ρ_{pq}	400	{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}
$\rho_{v\theta}$ (9-, 30-bus)	40000	{500, 2000, 5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000}
$\rho_{v\theta}$ (118-bus)	4000	{500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 5500, 7000}

Algorithm 1: Parameter Learning Through Q-Learning in ADMM ACOPF Solver

```
ADMM initialization: Initial parameters
    $x^{[0]} \in \mathbb{R}^{n_1}, \bar{x}^{[0]} \in \mathbb{R}^{n_1}, y^{[0]} \in \mathbb{R}^{n_3}, \rho^{[0]} \in \mathbb{R}^{n_3}$
RL initialization: Initial Q function parameter $\psi^{[0]}$,
    step size sequence $\alpha^{[k]}$, greedy policy parameter
    sequence $\epsilon^{[k]}$, length of state vector $n$, action space $\mathcal{A}$
for $k = 0, 1, 2, ...$ do
    if $k \geq n$ then
        Compute residuals $r_d^{[k]}$, $r_p^{[k]}$ from $x^{[k]}, \bar{x}^{[k]}$ and
        form state vector
        $s^{[k]} = [(r_p^{[k-n+1]}, r_d^{[k-n+1]}), \cdots, (r_p^{[k]}, r_d^{[k]})]$
        Select action $a^{[k]} \sim \widehat{\pi}^{[k]}(\cdot \,|\, s^{[k]})$ and translate to $\rho^{[k]}$
    else
        Use the initial $\rho$ value: $\rho^{[k]} = \rho^{[0]}$
    end if
    Perform an ADMM update (3) with the current
    penalty parameter $\rho^{[k]}$
    if $k \geq n$ then
        Receive reward $R(s^{[k]}, a^{[k]})$, observe the next
        state $s^{[k+1]}$, and compute the Q target
            $Q_{\text{target}} = R(s^{[k]}, a^{[k]}) + \max_a Q_{\psi^{[k]}}(s^{[k+1]}, a)$
        Update the Q function parameter
            $\psi^{[k+1]} = \psi^{[k]} - \alpha^{[k]} \nabla_\psi (Q_\psi(s^{[k]}, a^{[k]}) - Q_{\text{target}})^2 |_{\psi=\psi^{[k]}}$
        Update the $\epsilon$-greedy policy
            $\widehat{\pi}^{[k+1]}(a \mid s) = \begin{cases} 1 - \frac{(|\mathcal{A}|-1)\epsilon^{[k]}}{|\mathcal{A}|}, & \text{if } a = \hat{a}^{[k+1]}(s) \\ \frac{\epsilon^{[k]}}{|\mathcal{A}|}, & \text{otherwise} \end{cases}$
        where $\hat{a}^{[k+1]}(s) = \operatorname{argmax}_a Q_{\psi^{[k+1]}}(s, a)$.
    end if
    Terminate if ADMM has converged
end for
```

Reward function: The reward function is a crucial signal that affects the behavior of the agent. We have to carefully design the reward function to translate our objective, which is to accelerate ADMM convergence, correctly to the agent. The reward function R should be chosen such that $R(s, a)$ is large if taking action a while in state s leads to fast convergence and small if taking action a while in state s leads to slow convergence. With this in mind, a natural choice of the reward function is a large bonus given only to the convergence state; for instance,

$$R_{\text{conv}}(s^{[k]}, a^{[k]}) = \begin{cases} 200, & \text{if } \left\| r_p^{[k+1]} \right\|_2 \leq \epsilon_p \text{ and } \left\| r_d^{[k+1]} \right\|_2 \leq \epsilon_d, \\ 0, & \text{else.} \end{cases}$$

Due to the presence of the discount factor $\gamma \in (0, 1)$, the reward received further in the future becomes less valuable. Therefore, to maximize the discounted cumulative reward under this reward function, the agent will aim to reach the convergence state in as few iterations as possible.

Though this design of the reward function encodes our objective very well, it causes the agent to receive extremely sparse reward signals in the training process. Until the very last iteration, the agent will not receive any useful reward throughout the hundreds or thousands of iterations that are typically required for ADMM algorithms to converge for moderately sized ACOPF problems. Sparse rewards commonly cause exploration and credit assignment issues in RL [28] and significantly slow down the learning process.

To offer a denser signal to the RL agent, we add the residuals in the reward function. Specifically, the reward received by the agent in state $s^{[k]}$ is proportional to the reduction in $\|r_p^{[k+1]}\|_2$ and $\|r_d^{[k+1]}\|_2$ from $\|r_p^{[k]}\|_2$ and $\|r_d^{[k]}\|_2$:

$$\begin{aligned} &R_{\text{res}}(s^{[k]}, a^{[k]}) \\ &\quad = \frac{1}{Z_p}(\|r_p^{[k+1]}\|_2 - \|r_p^{[k]}\|_2) + \frac{1}{Z_d}(\|r_d^{[k+1]}\|_2 - \|r_d^{[k]}\|_2), \end{aligned}$$

where Z_p and Z_d are normalizing factors that balance the magnitude difference between the primal and dual residuals. This reward function makes sense, as achieving fast convergence is equivalent to quickly driving the residuals to the thresholds. This reward is non-zero in every ADMM iteration.

While we observe that the combination of R_{conv} and R_{res} works well in this problem, we further innovate the reward function design by taking advantage of the non-counterfactual nature of the environment. We note that in most RL problems, the environment transition is irreversible, that is, once an action $a^{[k]}$ is deployed in state $s^{[k]}$, the environment moves forward to the next state $s^{[k+1]}$, and the consequence of selecting a different action in $s^{[k]}$ is never observable. However, in this problem, the progress of every ADMM iteration can be saved and we can therefore try different actions in the same state and compare their outcomes. This feature of the environment affords more flexibility in the reward design.

In this work, we use a reward function computed with the help of a baseline policy $\tilde{\pi}$. In state $s^{[k]}$, we select the baseline action $\tilde{a}^{[k]} \sim \tilde{\pi}(\cdot \mid s^{[k]})$ and observe the resulting next state $\tilde{s}^{[k+1]}$ including primal and dual residuals $\tilde{r}_p^{[k+1]}$ and $\tilde{r}_d^{[k+1]}$. We note that this baseline action is only used to compute the residuals. We roll back to state $s^{[k]}$ once the residuals are collected. From state $s^{[k]}$, we then deploy the RL policy, making the environment transition to $s^{[k+1]}$ and reveal $r_p^{[k+1]}$ and $r_d^{[k+1]}$. The reward is defined as the relative advantage of the RL policy over the baseline:

$$R_b(s^{[k]}, a^{[k]}) = \frac{\|r_p^{[k+1]}\|_2 - \|\tilde{r}_p^{[k+1]}\|_2}{\|\tilde{r}_p^{[k+1]}\|_2} + \frac{\|r_d^{[k+1]}\|_2 - \|\tilde{r}_d^{[k+1]}\|_2}{\|\tilde{r}_d^{[k+1]}\|_2}.$$

This reward function essentially aims to achieve the same goal as R_{res}, but can have much smaller variance. To see this, note that $\|r_p^{[k+1]}\|_2 - \|r_p^{[k]}\|_2$ and $\|r_d^{[k+1]}\|_2 - \|r_d^{[k]}\|_2$ can fluctuate across several orders of magnitude through ADMM iterations regardless of the choice of ρ. The reward function R_b effectively removes the impact of the natural fluctuation of the residuals and makes the variance of R_b significantly smaller than that of R_{res}. It has been observed that reducing the variance of the reward is critical in accelerating learning and is also the motivation behind popular algorithms such as the advantage actor-critic (A2C) [29]. We emphasize that the sole purpose of the baseline policy is to offset the fluctuation in the norm of the residuals over iterations. Therefore, the baseline policy can be very simple. In the experiments of this work, the baseline policy is to always use $\rho_{pq} = 500$ and $\rho_{v\theta} = 500$. Accordingly, the reward function we choose in this work combines r_{conv} and r_b:

$$R(s^{[k]}, a^{[k]}) = R_{\text{conv}}(s^{[k]}, a^{[k]}) + R_b(s^{[k]}, a^{[k]}).$$

B. Factorized Entry-wise Policy

We have discussed the transformation of the ADMM parameter selection problem into a RL problem where the policy selects a vector ρ given the state vector. With the ten possible choices of ρ values for each constraint, the total cardinality of the action space is $10^{n_{pq}+n_{v\theta}}$, which grows exponentially in the number of constraints and quickly becomes computationally intractable. To address this issue, we reduce the action space by simplifying the policy using its structure.

We observe that the dimension of the ρ vector is equal to the number of constraints. Let π_i be the policy for updating parameter ρ_i with respect to the constraint i. We assume that each policy function (i.e., conditional probability distribution) is independent of the others. Then, if every entry of the state vector contains enough information to optimally determine the corresponding entry of ρ, the policy can be factorized as

$$\pi(a \mid s) = \prod_{i=1}^{n_{pq}+n_{v\theta}} \pi_i(a_i \mid s_i),$$

which means that we can equivalently train smaller policies π_i for each $i = 1, \ldots, (n_{pq} + n_{v\theta})$, whose effective action space has a cardinality of 10. Learning the set of small policies with its size scaling up linearly in the number of constraints, however, can still be computationally expensive. Therefore, we take one more step to simplify the policy by assuming that there exists two entry-wise policies π_{pq} and $\pi_{v\theta}$ that can optimally determine the mappings from the entries of state vector to the entries of ρ for all power and voltage constraints, respectively. This means that the policy can be further factorized as

$$\pi(a \mid s) = \prod_{i=1}^{n_{pq}} \pi_{pq}(a_{pq,i} \mid s_{pq,i}) \prod_{i=1}^{n_{v\theta}} \pi_{v\theta}(a_{v\theta,i} \mid s_{v\theta,i}).$$

As a result of this factorization, we only need to learn and maintain two small entry-wise policies. Since we use the Q-learning algorithm in this work with an action space of size 10 for each constraint, this amounts to learning two Q functions $Q_{pq}, Q_{v\theta} : \mathbb{R}^{2n} \times \mathbb{R}^{10} \to \mathbb{R}$.

In the ACOPF ADMM algorithm, we expect it to be generally impossible to determine the optimal ρ entry for a particular constraint without information from the other constraints. Moreover, there may not exist two unified policies π_{pq} and $\pi_{v\theta}$ that work optimally for all power and voltage constraints. However, simplifying the policy in this manner effectively reduces the learning complexity, and as we will show in Section V, the performance of the policy pair $(\rho_{pq}, \rho_{v\theta})$ achieves good empirical performance.

Along with advantages in computational tractability, another important benefit of the factorized entry-wise policy lies in its ability to be deployed to ACOPF ADMM problems with different numbers of constraints from the one seen by the RL agent in training. This means that the entry-wise policy pair trained under one power network can be flexibly applied to various other network structures. Later in Section V, we will discuss an important generalization of the learned policy to minor system modifications, where it is necessary for the policy to adapt to a change in the number of constraints.

V. Numerical Experiments

We demonstrate the performance of our RL model by training the parameter selection policy and testing its performance on the IEEE 9-bus, 30-bus, and 118-bus Matpower [30] test instances. Two additional evaluation tasks are carried out to validate the generalization of the learning performance to the practical scenarios in power system operations. In the first task, the RL policy is evaluated for its effectiveness in unseen load profiles in the original network. This is an important task as the loads of a power system change frequently, requiring the ACOPF problem to be solved repeatedly in an efficient way. The second task tests the RL policy on a slightly modified version of the system by removing generators and/or disconnecting transmission lines. This task is more challenging and also important in practice since we need to solve ACOPF problems under generator and line outages.

Two small-sized neural networks of identical structure (4 fully-connected layers with hidden dimension 256) are used to approximate Q_{pq} and $Q_{v\theta}$. The action space has dimension 10, and we choose the number of residual history points $n = 20$. This makes the input and output dimension of the neural network 40 and 10, respectively. We take the initial ρ_{pq} and ρ_{va} to be the values suggested by [13] (provided in Table I). Each test instance is solved from a cold-start in ADMM.

TABLE II
ADMM ITERATIONS OF RL POLICY UNDER TRAINING LOADS

	[Mhanna 2019]	RL policy	Iteration Reduction
9-bus	879	358	59.3%
30-bus	1400	738	47.3%
118-bus	525	343	34.7%

A. *Performance on Training Scheme*

The RL policy is trained under the default loading for 1000 RL episodes, where one episode is a complete ADMM solution process. Compared with the state-of-the-art ρ adjustment scheme in [13] that results in ADMM convergence in 879, 1400, and 525 iterations for 9-bus, 30-bus, and 118-bus systems, the RL policy reduces the number of ADMM iterations by at least 30% (Table II). To understand the mechanism behind the fast convergence under the RL policy, we show the primal and dual residuals over ADMM iterations under the RL policy and the scheme in [13] for the 9-bus system. While the scheme in [13] leads to frequent fluctuations of the residuals which prolong the ADMM solving process, the RL policy avoids these fluctuations. Although this trend is not as obvious in 30-bus and 118-bus systems, we still observe that the RL policy allows the residuals to drop more smoothly.

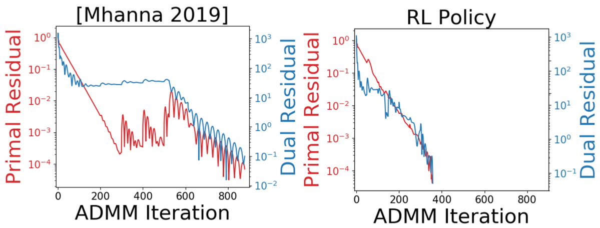


Fig. 2. Convergence of Residuals with RL Policy for the 9-bus System

B. *Generalization of RL Policy to Varying Loads*

We also test the generalization of the RL policy to varying loads. Note that the RL policy has only been trained on the default loads from MATPOWER, not on any other loading schemes. We create a dataset of 50 test instances by randomly perturbing the default loads in the range $[-10\%, 10\%]$ at each bus. We summarize the number of ADMM iterations to convergence in Table III. The RL policy reduces the ADMM iterations relative to the scheme in [13] by 28% to 50% across test cases.

TABLE III
ADMM ITERATIONS OF RL POLICY UNDER VARYING LOADS

	ρ selection method				Iteration Reduction
	[Mhanna 2019]		RL policy		
	mean	std	mean	std	
9-bus	813.4	20.4	407	9.9	50.0%
30-bus	1414.3	43.6	772.5	18.9	45.4%
118-bus	486.6	8	346	7.2	28.9%

C. *Generalization of RL Policy to Generator and Line Outages*

In practical situations, we may need to solve the ACOPF problem after generator and line outages. Thus, it is of interest to investigate the performance of the RL policy in a modified network. In this section, we evaluate the ADMM convergence speed when applied to systems with 1) one generator removed and 2) one line disconnected.[1] Again, note that the RL policies were trained on the original MATPOWER networks, without considering line or generator loses. Tables IV and V summarize the performance of the RL policy and its comparison with the state-of-the-art method in [13].

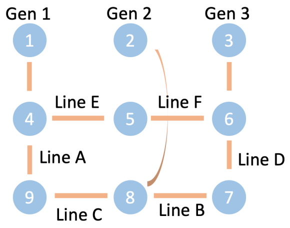
TABLE IV
ADMM ITERATIONS OF RL POLICY UNDER GENERATOR OUTAGES

	No. of instances	ρ selection method				Iteration Reduction
		[Mhanna 2019]		RL policy		
		mean	std	mean	std	
9-bus	3	856.0	221.4	654.0	119.9	23.6%
30-bus	6	1325.8	404.3	695.8	78.9	47.5%
118-bus	54	483.8	17.7	340.0	8.8	29.7%

TABLE V
ADMM ITERATIONS OF RL POLICY UNDER LINE OUTAGES

	No. of instances	ρ selection method				Iteration Reduction
		[Mhanna 2019]		RL policy		
		mean	std	mean	std	
9-bus	6	698.7	218.5	367.3	31.1	47.4%
30-bus	10	1455.5	225.6	800.4	93.2	45.0%
118-bus	50	486.5	6.0	346.1	6.1	28.9%

In the 9-bus system, there are 3 generator buses and 6 lines that can be disconnected while avoiding islands. In Figure 3, we detail the ADMM convergence under the RL policy for each outage scenario, and note that the proposed method always outperforms [13] by a large margin.



Outage	[Mhanna 2019]	RL Policy
Gen 1	555 iters	485 iters
Gen 2	1081 iters	727 iters
Gen 3	932 iters	750 iters
Line A	535 iters	335 iters
Line B	1178 iters	431 iters
Line C	591 iters	360 iters
Line D	604 iters	351 iters
Line E	680 iters	377 iters
Line F	604 iters	350 iters

Fig. 3. ADMM Convergence with RL Policy for the 9-bus System with Generator and Line Outages

D. *Generalization of RL Policy to Unseen Network Structures*

We also performed experiments on the generalization of the RL policy to networks it was not trained on. For example, one may be interested in training a RL policy for a 9-bus system and deploying it to a 30-bus system. Accordingly, we trained RL policies for several systems and tested them on several others. Though our policy factorization described in Section IV-B makes it possible to apply the RL policy to an ACOPF problem with a different number of constraints, experimentally, we found that policies trained in one network perform poorly in a completely different network. This observation strengthens our belief that there may not exist a universally optimal strategy that works for any ADMM problem. Hence, this observation supports the need for specialized approaches like the RL policies in this paper.

[1] We only consider line outages that do not island the network.

Remark 1: We note that the computational complexity of the RL policy for each constraint does not change with the dimension of system, as the dimension of the neural network is the same. However, the ADMM solver usually slows down as the system grows. Therefore, the computational time of deploying the RL policy should become increasingly negligible as the problem scales up. The training of the RL policy requires running 1000 episodes of complete ADMM solve, which is a substantial amount of time, but consists of offline computations that will not affect the online computational speed of deploying the RL policy.

VI. Conclusion & Future Work

The choice of penalty parameters is key for accelerating the convergence of the ACOPF ADMM algorithm. By recognizing this task as a sequential decision making problem, we propose a RL framework in which we properly design the state space, action space, and reward function. We demonstrate the superior performance of the learned RL policy over the state-of-the-art method in a range of scenarios. To the best of our knowledge, this is the first work to use machine learning for penalty parameter selection in distributed optimization for power systems applications. By reducing the number of ADMM iterations by up to 50%, this paper provides a successful proof of concept for using RL to enhance ADMM algorithms for power systems.

Future directions of the work include scaling up the method to larger systems and refining the RL training scheme to further improve ADMM convergence. We also plan to explore adapting this method to other ACOPF ADMM implementations where convergence can be theoretically guaranteed [10].

References

[1] B. Kroposki, A. Bernstein, J. King, D. Vaidhynathan, X. Zhou, C.-Y. Chang, and E. Dall'Anese, "Autonomous energy grids: Controlling the future grid with large amounts of distributed energy resources," *IEEE Power and Energy Magazine*, vol. 18, no. 6, pp. 37–46, 2020.

[2] M. Ryu and K. Kim, "A privacy-preserving distributed control of optimal power flow," *arXiv:2102.02276*, 2021.

[3] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, "A survey of distributed optimization and control algorithms for electric power systems," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017.

[4] Y. Wang, S. Wang, and L. Wu, "Distributed optimization approaches for emerging power systems operation: A review," *Electric Power Systems Research*, vol. 144, pp. 127–135, 2017.

[5] Y. Kim, F. Pacaud, K. Kim, and M. Anitescu, "Leveraging GPU batching for scalable nonlinear programming through massive Lagrangian decomposition," arXiv:2106.14995, Argonne National Laboratory, Tech. Rep., 2021.

[6] A. Kargarian, J. Mohammadi, J. Guo, S. Chakrabarti, M. Barati, G. Hug, S. Kar, and R. Baldick, "Toward distributed/decentralized DC optimal power flow implementation in future electric power systems," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2574–2594, 2016.

[7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.

[8] S. Mhanna, A. C. Chapman, and G. Verbič, "Component-based dual decomposition methods for the OPF problem," *Sustainable Energy, Grids and Networks*, vol. 16, pp. 91–110, 2018.

[9] D. Bienstock and A. Verma, "Strong NP-hardness of AC power flows feasibility," *Operations Research Letters*, vol. 47, no. 6, pp. 494–501, 2019.

[10] K. Sun and X. A. Sun, "A two-level ADMM algorithm for AC OPF with convergence guarantees," to appear in *IEEE Transactions on Power Systems*, 2021.

[11] B. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *Journal of Optimization Theory and Applications*, vol. 106, no. 2, pp. 337–356, 2000.

[12] Z. Xu, M. Figueiredo, and T. Goldstein, "Adaptive ADMM with spectral penalty parameter selection," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 718–727.

[13] S. Mhanna, G. Verbič, and A. C. Chapman, "Adaptive ADMM for distributed AC optimal power flow," *IEEE Transactions on Power Systems*, vol. 34, no. 3, pp. 2025–2035, 2019.

[14] X. Chen, G. Qu, Y. Tang, S. Low, and N. Li, "Reinforcement learning for decision-making and control in power systems: Tutorial, review, and vision," *arXiv:2102.01168*, 2021.

[15] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," *arXiv:2103.12828*, 2021.

[16] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 3981–3989.

[17] D. Biagioni, P. Graf, X. Zhang, A. S. Zamzam, K. Baker, and J. King, "Learning-accelerated ADMM for distributed DC optimal power flow," *IEEE Control Systems Letters*, vol. 6, pp. 1–6, 2022.

[18] P. Graf, J. Annoni, C. Bay, D. Biagioni, D. Sigler, M. Lunacek, and W. Jones, "Distributed reinforcement learning with ADMM-RL," in *American Control Conference (ACC)*. IEEE, 2019, pp. 4159–4166.

[19] X. Xie, J. Wu, G. Liu, Z. Zhong, and Z. Lin, "Differentiable linearized ADMM," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 6902–6911.

[20] J. Ichnowski, P. Jain, B. Stellato, G. Banjac, M. Luo, F. Borrelli, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Accelerating quadratic optimization with reinforcement learning," *arXiv:2107.10847*, 2021.

[21] F. Li and Y. Du, "From AlphaGo to power system AI: What engineers can learn from solving the most complex board game," *IEEE Power and Energy Magazine*, vol. 16, no. 2, pp. 76–84, 2018.

[22] L. Duchesne, E. Karangelos, and L. Wehenkel, "Recent developments in machine learning for energy systems reliability management," *Proceedings of the IEEE*, vol. 108, no. 9, pp. 1656–1676, 2020.

[23] T. Erseghe, "Distributed optimal power flow using ADMM," *IEEE Transactions on Power Systems*, vol. 29, no. 5, pp. 2370–2380, 2014.

[24] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[25] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[26] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv:1511.05952*, 2015.

[28] J. Hare, "Dealing with sparse rewards in reinforcement learning," *arXiv:1910.09281*, 2019.

[29] 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 *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1928–1937.

[30] R. Zimmerman, C. Murillo-Sánchez, and R. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 99, pp. 1–8, 2011.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).