Battery Control With Lookahead Constraints in Distribution Grids Using Reinforcement Learning

Joel da Silva André, Eleni Stai, Ognjen Stanojev, Gabriela Hug
EEH - Power Systems Laboratory, ETH Zürich, Physikstrasse 3, 8092 Zürich, Switzerland
dajoel@student.ethz.ch, {stai,stanojev,hug}@ee.ee.ethz.ch

Abstract—In this paper, a computationally efficient real-time control of a battery with lookahead state-of-energy constraints in active distribution grids with distributed energy sources is presented. The goal is to follow a previously computed dispatch plan or to optimize a monetary cost from buying and selling power at the point of common coupling. However, the lookahead constraints render the battery decisions non-trivial. The current practice in literature to solve this problem is Model Predictive Control (MPC), which does not scale for large grids. Instead, here, we propose a reinforcement learning approach based on the Deep Deterministic Policy Gradient (DDPG) algorithm. To satisfy the lookahead battery constraints we adapt the experience replay technique used in DDPG. To guarantee the satisfaction of the hard grid constraints, we introduce a safety layer that performs constrained optimization. The approach does not need forecasts contrary to MPC. We perform evaluations on a realistic grid and comparisons with Lyapunov optimization and MPC. We show that we can achieve costs close to MPC and Lyapunov, while reducing the computational time by multiple orders of magnitude.

Index Terms—Active distribution grids, battery energy storage, lookahead constraints, real-time control, deep reinforcement learning

I. INTRODUCTION

Energy storage in the form of batteries will likely play a crucial role in the operation of distribution grids that are characterized by a continuously growing integration of uncertain Distributed Energy Resources (DERs). Batteries, if appropriately controlled, can absorb power imbalances caused by the uncertain DERs and random load fluctuations, by discharging when demand is higher than supply and vice versa. In addition, nodal voltage magnitudes can be kept within safe operating limits through reactive power control. One of the main challenges in battery control lies in the lookahead state-of-energy (SoE) constraints, which need to be met to ensure proper energy management and dispatchability in future operation. More precisely, the lookahead constraints render the real-time battery control a multi-period problem with time-coupling constraints and thus require special attention.

The most commonly employed solution in the literature is Model Predictive Control (MPC) [1]–[3]. Nonetheless, MPC schemes do not scale for real-time applications due to computational limitations, and they also require forecast values. In addition, the formulated problems are inherently non-convex as a result of the non-linear power flow constraints.

A Lyapunov optimization-based approach, denoted as iLyPRC, has been recently proposed in [4] to replace MPC as it can achieve faster computation times while not relying on forecasts. However, its effectiveness is strongly conditioned on the choice of three parameter values that may require a sophisticated offline study. In this paper, we alleviate the aforementioned issue by employing a machine learning approach. We propose a battery control scheme based on reinforcement learning (RL) that (i) is fast and suitable for real-time applications, (ii) scales well and, (iii) inherently does not require forecasts due to the use of RL.

The application of RL to solve problems in the power grid domain was only recently put in focus. Earlier works [5]–[7] mostly focus on applying Q-learning methods. For instance, [5], [6] use Q-learning for battery control and demand response, and [7] for battery and voltage control. More recent literature focuses on the Deep Deterministic Policy Gradient (DDPG) RL algorithm [8], [9], which is an off-policy actor-critic method derived from Q-learning. Nevertheless, contrary to Q-learning, it is well suited to continuous control problems. Among the approaches most related to this work, [10]–[13] use DDPG to perform low complexity power systems control and [14], [15] use DDPG for low complexity battery control. These studies, however, do not consider lookahead constraints such as the lookahead battery SoE constraint considered in this paper. In general, the satisfaction of hard constraints in RL-based control schemes in power systems has hardly been considered. The authors in [16] introduce a safe DDPG RL algorithm to perform optimal voltage control in active distribution grids. Their RL algorithm incorporates a safety layer (first introduced in [17]) on top of the control action, so that the voltage constraints are not violated.

This paper presents an RL-based scheme for battery control in active distribution grids that computes the optimal battery charge/discharge power at each time interval to follow a previously computed dispatch plan or to optimize a monetary cost from buying and selling power at the PCC, while concurrently satisfying instantaneous hard grid constraints and lookahead battery SoE constraints. The problem is first formulated as a Finite Horizon Markov Decision Process (FHMDP). Subsequently, we develop an RL agent that employs the DDPG algorithm with appropriate enhancements, to control the battery. DDPG is chosen for two main reasons: (i) it can deal with continuous state and action spaces, and (ii) it is well-suited for developing online learning schemes. However, the lookahead constraints and the hard grid constraints require additional considerations in the application of DDPG. Specifically, for the satisfaction of the lookahead SoE constraints, our RL agent needs to learn a time-dependent policy. In addition, the rewards
associated with the lookahead SoE constraints are very sparse. Thus, when performing batch RL, one may not sample a sufficient number of such rewards from the replay buffer and in this case the estimation of the Q-function may be inaccurate. We solve this issue by combining two replay buffers when performing experience replay. Finally, to satisfy the hard grid constraints, we implement an external safety layer that corrects the RL agent’s action.

We perform numerical evaluations of the proposed scheme on a 4-bus toy grid and a 34-bus real Swiss grid. We numerically show that the lookahead SoE constraints are met. Moreover, we compare the performance with MPC and iLyapRC and show that our scheme (i) achieves a cost very close to iLyapRC and MPC, (ii) significantly reduces the run time reaching a speed up of a factor of 32 for iLyapRC and 184 for MPC, and (iii) contrary to iLyapRC and MPC, has excellent scaling properties. Thus, the proposed scheme allows for efficient control at very fast time scales and is scalable.

The rest of the paper is organized as follows. In Sec. II, we establish the problem formulation and in Sec. III, we give all the elements of the FHMTP. In Sec. IV, we provide the proposed RL-based solution. Lastly, Sec. V presents all the evaluations and comparisons of the proposed scheme. Finally, Sec. VI concludes the paper.

II. PROBLEM FORMULATION

A. Preliminaries

We consider a balanced distribution network represented by a connected graph \( G(N, E) \), with \( N := \{0, 1, \ldots, N\} \) denoting the set of nodes and \( E \) designating the set of lines. The substation node with ID 0 is the Point of Common Coupling (PCC) to the main grid and is selected to be the slack bus, while the remaining \( N \) nodes are PQ-buses with possible load and DER connections. A single battery connected at node \( b \in N / \{0\} \) is considered. The grid topology is assumed to be static, i.e., not to change during the real-time operation.

B. Constraints Formulation

The aim is to control the battery active and reactive power setpoints for \( K \) time intervals ahead, with duration \( \Delta t \) each. Specifically, at every control time \( k \in \mathcal{H} := \{0, \ldots, K-1\} \), the controller computes the battery active and reactive power values, \( B_P(k) \) and \( B_Q(k) \), respectively, which are applied over the time interval \( [k\Delta t, (k+1)\Delta t] \). Thus, the control vector is defined as \( u(k) := (B_P(k), B_Q(k))^T \). Moreover, we assume that the battery power output is constrained by lower and upper bounds, i.e., \( u(k) \) at every \( k \) belongs to the set

\[
\mathcal{U} := \{(B_P, B_Q)^T : B_P \in [B_{P,LO}, B_{P,UP}], B_Q \in [B_{Q,LO}, B_{Q,UP}]\},
\]

with \( B_{P,LO} \leq B_{P,UP} \) and \( B_{Q,LO} \leq B_{Q,UP} \).

Let SoE\((k)\) stand for the battery state-of-energy at time \( k \). The physical equation that describes the evolution of the battery SoE is given by:

\[
\text{SoE}(k+1) = \text{SoE}(k) + (B_P(k) - B_{loss}(k)) \Delta t, \quad \forall k \in \mathcal{H},
\]

where \( B_{loss}(k) \) stands for the internal battery losses. The initial SoE\((0)\) is assumed known.

As aforementioned in the introduction, the system is uncertain. The disturbance vector \( d(k) := (c(k)^T, L(k)^T)^T \) includes the uncertain energy prices \( c(k) \), and the non-battery power consumption at the PQ-buses denoted by \( L(k) \). The uncertainties are unknown a-priori and are only revealed after the control decision has been taken. Specifically, when taking the control decision at time interval \( k \), the disturbance \( d(k) \) is unknown and only \( d(k-1) \) is known.

The state of the system at the beginning of the time interval \( k \) (i.e., the time interval \([k\Delta t, (k+1)\Delta t]\)) is denoted as \( \chi(k) \) and consists of SoE\((k)\) and the complex bus voltage vector. However, the latter is computed based on \( L(k-1) \) and \( u(k-1) \), since \( L(k) \), \( u(k) \) are both unknown at the beginning of the time interval \( k \). In other words, \( \chi(k) := (v(k-1)^T, \text{SoE}(k))^T \).

Then, the evolution of the system can be represented by the following discrete-time state transition equation:

\[
\chi(k+1) = F(\chi(k), u(k), d(k)), \quad \forall k \in \mathcal{H},
\]

where the vector function \( F \) represents the physical equations of the system, i.e., the non-linear power flow equations and the battery SoE evolution equation (2). The controller observes \( \chi(k) \), decides \( u(k) \), and after the decision is applied and the uncertainty \( d(k) \) is revealed, the system transits to \( \chi(k+1) \).

Constraints on the state vector are imposed to keep the squared voltages \( v^M(k) := \text{diag}(v(k)v(k)^*) \) and the SoE within allowable limits. The vector of the squared voltage magnitudes is upper and lower bounded by the vectors \( v_{UP} \) and \( v_{LO} \), respectively. Furthermore, we impose upper and lower bounds on the battery SoE, i.e., \( \text{SoE}(k) \in [\text{SoE}_{E,LO}, \text{SoE}_{E,UP}], \forall k \in \mathcal{H} \), while the battery SoE at the end of the control period, \( \text{SoE}(K) \), should lie in the interval \([\text{SoE}_{E,K,LO}, \text{SoE}_{E,K,UP}] \), representing the lookahead battery constraint. Based on the above, the state constraint sets take the following forms:

\[
\mathcal{X} := \{ (v^M, \text{SoE}) : v^M \leq v_{UP} \leq v^M, \text{SoE} \in [\text{SoE}_{E,LO}, \text{SoE}_{E,UP}] \},
\]

\[
\mathcal{X}_k := \{ (v^M, \text{SoE}) : v^M \leq v_{UP} \leq v^M, \text{SoE} \in [\text{SoE}_{E,K,LO}, \text{SoE}_{E,K,UP}] \},
\]

where the state constraint set \( \mathcal{X} \) reflects voltage and SoE allowable limits, and \( \mathcal{X}_k \) in addition considers the terminal (or lookahead) SoE constraint.

C. Objective Function

Let us define \( w(k) = P_{pee}(k) - P_{disp}(k) \) as the difference between the realized, \( P_{pee}(k) \), and the scheduled, \( P_{disp}(k) \), active power at the PCC at time \( k \). The cost at each time interval is a weighted sum of the monetary cost \( C_m(\cdot, \cdot) \) from buying and selling energy at the PCC and the dispatch plan deviation cost \( C_d(\cdot) \):

\[
C(\chi(k), u(k), d(k)) = (1 - \omega_1)C_m(w(k), e(k)) + \omega_1 C_d(w(k)),
\]

where \( \omega_1 \in [0, 1] \) is the scaling weight, and the two cost functions are defined by:
the system state used by the RL agent is represented as:

\[ C_{m}(w(k), c(k)) = \left( \max(w(k), 0), \min(-w(k), 0) \right) c(k), \quad (7) \]
\[ C_{d}(w(k)) = Aw(k)^2. \quad (8) \]

We denote the vector of the uncertain buying and selling prices at time \( k \) as \( c(k) = (c_1(k), c_2(k))^T \). Finally, \( A > 0 \) is a scaling factor.

**D. The Optimal Control Problem**

Ideally, the optimal battery control decision is obtained by minimizing the expected time average of the cost function \( C(\chi(k), u(k), d(k)) \) over the whole control horizon:

\[ \min_{u(0), \ldots, u(K-1)} \mathbb{E} \left[ \frac{1}{K} \sum_{k=0}^{K-1} C(\chi(k), u(k), d(k)) \right], \quad (9) \]

subject to the state update equation (3) as well as to the state and control constraints, i.e., \( \chi(k) \in \mathcal{X}, \chi(K) \in \mathcal{X}_K \), and \( u(k) \in \mathcal{U} \). The solution to the problem is the optimal policy \( \pi^*(k) \) that at each time interval \( k \) maps the currently observed state \( \chi(k) \) into an optimal control action \( a^*(k) \), which in this paper we obtain by employing a deep RL algorithm.

**III. CONTROL PROBLEM AS MARKOV DECISION PROCESS**

In the finite horizon Markov decision process (FHMDP) framework, an agent interacts with a stochastic environment by making an observation of the current state \( s(k) \in \mathcal{S} \) and applying an action \( a(k) \in \mathcal{A} \) at each time interval \( k \). The action is selected by the agent based on an internal policy \( \pi : \mathcal{S} \rightarrow \mathcal{A} \). After applying the action, the environment converts into a new state \( s(k+1) \) depending also on the realization of the stochastic quantities, \( d(k) \). Additionally, the agent receives a reward signal \( r(k) \). The transition between the environment states is modeled by the transition function \( s(k+1) = p(s(k), a(k), d(k)) \), describing the dynamics of the environment. The agent’s objective is to find an optimal policy that maximizes the expected discounted long-term reward \( \sum_{t=k}^{K-1} \gamma^t \mathbb{E}[r(t)] \), with \( \gamma \in [0, 1] \) being the discount factor.

It is clear that the problem of Sec. II can be viewed as an FHMDP, with the battery controller being the agent, as we have defined the state, the action (i.e., the control variables), the time interval cost (6) (i.e., the opposite of the reward) and the state transition equation (3). Here, we adapt the FHMDP of Sec. II so that RL can be efficiently applied for deciding the control variables, and we clarify all of its elements.

**System State:** The system state \( \chi(k) \) is adapted as follows. First, it is augmented by adding a time element \( k \). This is required for keeping track of the time, so that the agent is able to satisfy the terminal constraints at \( k = K \) (in \( \mathcal{X}_K \)). In other words, the agent learns a time-dependent policy, which is essential for finite horizon settings [18]. Second, in order to render the learning process of the RL agent more efficient, instead of the electrical state \( v(k-1) \), we include in the state the squared bus voltage magnitudes, \( v^M(k-1) \), and the power injections at the buses, \( L(k-1) \). Third, we include the known price values \( c(k-1) \) and the known dispatch plan deviation at the PCC, \( w(k-1) \). Based on the aforementioned changes, the system state used by the RL agent is represented as:

\[ s(k) = (k, \text{SoE}(k), w(k-1), v^M(k-1), d(k-1)). \quad (10) \]

We observe that \( s(k) \) comprises observations that are either given (prices) or can be retrieved from the environment by means of real-time measurements. The constraints \( \mathcal{S} \) on the state can be inferred from (4)-(5).

**Action:** The RL agent decides the charging or discharging battery active power, as well as the reactive battery power injection. Thus, the action vector \( a(k) \) coincides with \( u(k) \) and is limited to the same bounds defined in \( \mathcal{U} \), i.e., \( A \equiv \mathcal{U} \).

**Reward Function:** The instantaneous reward at time \( k \) is the opposite of the weighted sum of the monetary cost and the square of the dispatch plan mismatch, as expressed on the right hand-side of (6). It is further augmented by two penalty terms to support the learning process in the satisfaction of the state constraints in \( \chi \) and \( \mathcal{X}_K \). As a result, we have

\[ r(k) = -\left( (1 - \omega_1)C_m(w(k), c(k)) + \omega_1C_d(w(k)) \right) \]
\[ + \omega_2 \max(\text{SoE}(K) - \text{SoE}_{UP}^L, \text{SoE}_{LO}^L - \text{SoE}(K), 0) \cdot 1_k \]
\[ + \omega_3 \max(v^M(k) - v_{UP}, v_{LO} - v^M(k), 0) \],

where \( \omega_2, \omega_3 \) are positive scaling weights. The second line penalizes the violation of the terminal SoE constraint in \( \mathcal{X}_K \) and thus applies only at time \( K \), which is considered using the flag \( 1_k = \max(k - K + 1, 0) \). The third line penalizes the violations of the voltage magnitude constraints in \( \chi \).

**State Transition:** When the action \( a(k) \) is applied and the uncertainty \( d(k) \) is revealed, the system transits to the next state, \( s(k+1) \). The state transition function includes the physical equations of the grid (i.e., the power flow equations) and of the battery SoE (i.e., (2)), similarly as \( F \) in (3). However, here we denote the state transition function as \( p \) due to the different state representation. In a real system, we do not need the formula of \( p \), as the agent directly learns it from interactions with the environment. An exception to that is if we consider the safety layer that requires a system model (Section IV-B). We also need a model for our simulation environment.

For representing the battery, we employ the model provided in [19] that is realistic in the context of energy management applications. It considers the internal losses of the grid-connected batteries by using equivalent virtual purely resistive lines integrated in the power flow model. Since charge/discharge losses are treated as grid losses, our model can be integrated into the load flow.

**IV. PROPOSED SOLUTION USING RL**

In finding the optimal policy, many RL algorithms rely on the action-value function (Q-function) defined by

\[ Q^\pi(s, a) = \mathbb{E}_\pi [G(k) \mid s(k) = s, a(k) = a], \quad (12) \]

where \( G(k) \) is the expected discounted long term reward at time \( k \) (i.e., \( G(k) = \sum_{t=k}^{K-1} \gamma^t \mathbb{E}[r(t)] \)) and \( Q^\pi(s, a) \) represents the expected discounted long term reward when taking action \( a \) in state \( s \) and following policy \( \pi. \)
A. DDPG-based Solution with Two Replay Buffers

To solve the FHMDP problem, we apply the off-policy, model-free DDPG RL algorithm [8]. Here, the Q-function (i.e., (12)) is approximated by a Neural Network (NN) called the critic network \( Q_\theta(s,a) \), with parameter \( \theta \). Similarly, the policy function is approximated by a NN called the actor network \( \mu_\phi(s) \), with parameter \( \phi \). In order to ensure a more stable learning process and operation of the algorithm, we apply and adapt the two main enhancements typically used by all DDPG-style algorithms, namely target networks and experience replay [20]. Target networks are used to stabilize learning, while the experience replay is used to alleviate the effects of correlation when learning only via sequential data. The basic idea is to store past experiences in a so-called replay buffer and use a random subset \( B \) of these experiences to update the Q-network, rather than the single most recent one.

We use two target networks: the target critic \( Q_{\theta_{\text{targ}}} \), with parameter \( \theta_{\text{targ}} \), and the target actor \( \mu_{\phi_{\text{targ}}} \), with parameter \( \phi_{\text{targ}} \). Both are defined as time-delayed soft copies of the original critic and actor networks and their parameters are slowly updated towards the parameters of the original networks:

\[
\theta_{\text{targ}} \leftarrow (1 - \tau) \theta_{\text{targ}} + \tau \theta, \\
\phi_{\text{targ}} \leftarrow (1 - \tau) \phi_{\text{targ}} + \tau \phi,
\]

where \( \tau \ll 1 \) is the soft update coefficient.

The replay buffer is updated at the end of the time interval \( k \), i.e., after the state has been observed, the agent’s action applied and the uncertainty revealed, by storing the experience tuple \((s(k), a(k), r(k), s(k+1))\).

However, the issue when using a single replay buffer is that the reward values associated with the terminal battery constraint, i.e., the second line in (11), are sparse as they are acquired only at time \( k = K \). To circumvent the sparsity issue, the approach of [21] is adopted\(^1\). Instead of one, two replay buffers are employed, namely, \( D_1 \) that stores all non-terminal experience tuples and \( D_2 \) that only stores terminal experience tuples. In this way, we can sample experience tuples from both replay buffers for the agent’s training batch and thus avoid including only non-terminal experiences, which would misguide the training procedure.

Now, the critic network is updated by a step of gradient descent using the following loss function that is expressed as an average over \(|B|\) samples drawn from both \( D_1 \) and \( D_2 \):

\[
L(\theta) = \frac{1}{|B|} \sum_{(s(\ell), a(\ell), r(\ell), s(\ell+1)) \in B} (Q_\theta(s(\ell), a(\ell)) - y(\ell))^2.
\]

The target \( y(\ell) \), with respect to which \( Q_\theta(s,a) \) is updated, is an approximation of the Bellman equation and is computed using the target networks as follows:

\[
y(\ell) = r(\ell) + \gamma Q_{\theta_{\text{targ}}}(s(\ell+1), \mu_{\phi_{\text{targ}}}(s(\ell+1))).
\]

The actor network is updated by a step of gradient ascent on the critic network with respect to the parameters \( \phi \) of the actor network. This is known as the policy gradient, or the gradient of the policy’s performance [9], and is given by:

\[
\nabla_\phi J(\mu_\phi) = \nabla_\phi \frac{1}{|B|} \sum_B Q_\theta(s(\ell), a(\ell) = \mu_\phi(s(\ell))),
\]

where \( J(\mu_\phi) \) is the policy performance function.

B. Safe Learning

Thus far, the way in which the proposed RL scheme is encouraged to satisfy the hard grid security and battery constraints in \( X \) and \( X_K \) is through the two penalty terms in the reward function (11). However, as learning provides no guarantees, it might occur that the constraints are violated. In this section, we propose a low-complexity optimization-based safety layer that guarantees the satisfaction of the battery SoE constraints in \( X \) and significantly assists in the satisfaction of the voltage magnitude constraints in \( X \). Considering that slight violations of the lookahead battery constraints in \( X_K \) do not affect the security of the grid operation, for them, we rely on the learning process with appropriately selected weights for rewards in (11) and the use of two replay buffers (Sec. IV-A).

The proposed safety layer consists of two functions that are applied sequentially. The first corrects the agent’s decision \( a(k) \) if voltage magnitude constraint violations are anticipated. The obtained action is further corrected by the second function such that the SoE bounds are met.

1) Enforcing Voltage Constraints: If the agent’s action \( a(k) \), violates the voltage magnitude constraints, we look for a feasible action \( x(k) \) as close as possible to the given \( a(k) \), as reflected in the following quadratic optimization problem:

\[
\min_{x(k) \in \mathcal{U}} \frac{1}{2} |x(k) - a(k)|^2 \tag{17}
\]

s.t. \( v^L \leq v^M(k-1) + S(k) \Delta x(k) \leq v^U \),

where \( \Delta x(k) = x(k) - a(k-1) \) represents a change in battery output in two consecutive time intervals, and \( S(k) = \partial v^M / \partial a \) is a voltage-action sensitivity matrix. The voltage resulting from \( x(k) \) is estimated by superimposing the known voltage magnitude \( v^M(k-1) \) and the predicted voltage change \( S(k) \Delta x(k) \). The sensitivity matrix \( S(k) \) is computed as the inverse of the power flow Jacobian matrix, which is found for every \( k \) by running an offline load flow computation for the estimated grid operating point using the uncorrected action \( a(k) \) and the known prosumption \( L(k-1) \).

Solving the optimization problem (17) increases the computational complexity associated with the agent training and execution. To avoid this drawback, we approximate the solution of (17) by decomposing it into a number of optimization problems that can run in parallel and for each of which a closed-form solution is obtainable. Namely, for every bus, \( i \in \mathcal{N} \), we solve the following quadratic convex problem:

\[
\min_{x_i(k) \in \mathcal{U}_i} \frac{1}{2} |x_i(k) - a_i(k)|^2 \tag{18}
\]

s.t. \( v_i^L \leq v_i^M(k-1) + S_i(k) \Delta x_i(k) \leq v_i^U \),

where \( v_i \) denotes the \( i \)-th entry of vector \( v \), and \( S_i \) represents the \( i \)-th row of matrix \( S \). Since only one of the constraints

\(^1\)Another potential approach would be to use a single replay buffer with prioritized experience replay, as in [22]. We will investigate this approach in our future work.
(the lower or the upper bound) can be active at a bus at a time, we can obtain a closed-form solution of its corresponding optimal Lagrange multiplier by applying the Karush-Kuhn-Tucker (KKT) conditions [17]. When the lower bound is active, the associated optimal Lagrange multiplier is given by:

\[ \lambda_i^* = \frac{\nu_i^{LO} - \nu_i^{M}(k-1) - S_i(k)a(k) + S_i(k)a(k-1)}{S_i(k)S_i(k)^T} \]

while the following holds if the upper bound is active:

\[ \lambda_i^* = -\frac{\nu_i^{UP} + \nu_i^{M}(k-1) - S_i(k)a(k) - S_i(k)a(k-1)}{S_i(k)S_i(k)^T} \]

The operator \([\cdot]^+\) stands for the operation \(\max\{\cdot, 0\}\). We estimate which constraint is anticipated to be active based on the value of \(\nu_i(k-1)\). Subsequently, we compute the bus index \(i^*\) such that \(\lambda_i^* = \max_{i \in \mathcal{N}} \lambda_i^*\). Finally, again by using the KKT conditions the corrected decision can be obtained as:

\[ x(k) = a(k) \pm \lambda_i^* \cdot S_i \cdot \text{sign} \]

where the minus sign holds if the lower bound is active for \(i^*\) and plus if the upper bound is active.

Note that the previously presented approach of enforcing voltage constraints is based on the safe learning methods of [16], [17], with appropriate enhancements for handling multiple active constraints.

2) Enforcing Battery Constraints: In order to prevent potential violation of the SoE constraints in \(\mathcal{X}\) when applying the obtained action \(x(k)\), a back-up controller is introduced to correct the active battery power \(B_P(k)\) such that

\[ B_P(k) = \begin{cases} \min \left( B_P(k), B_P^{UP}, \frac{\min(S_i(k)) - \min(S_i(k))}{\Delta t} \right), & B_P(k) \geq 0 \\ \max \left( B_P(k), B_P^{LO}, \frac{\min(S_i(k)) + \min(S_i(k))}{\Delta t} \right), & B_P(k) < 0 \end{cases} \]

Note that we do not include the SoE constraints in (18) as this would impede the existence of a closed form solution and would increase the solution algorithm’s run time. Moreover, note that the reactive power does not need correction because it is already selected from the admissible control set \(\mathcal{U}\) and it does not affect the battery SoE. Finally, it is important to clarify that for computing \(S(k)\), the agent runs a load flow and thus requires models of the grid and the battery, which are described in Section III. In our future work, we will investigate how to transport the safety layer to a model-free form.

C. Algorithm

Algorithm 1 is run by the battery RL agent either only for training or for simultaneous decision making and training. We run \(\mathcal{M}\) episodes, where each consists of \(K\) experience tuples (i.e., equal to the number of the control time intervals) that are derived from the agent-environment interaction (Section IV-A). To perform only training, we construct the training episodes using historical time-series of the prosumption at the nodes (\(L\)) and of the power prices (\(c\)). Otherwise, the episodes are based on observations.

At the beginning of each time interval, the agent decides \(a(k)\) by using the policy and a random noise sampled from a process \(Y\). The noise is added for exploration purposes. If any constraints are violated, the action is corrected with the safety layer. To check for violations of voltage magnitude constraints, it runs a load flow with the following inputs: (i) the bus power injections \(L(k-1)\) and (ii) the action \(a(k)\). Next, the corrected action is applied and the new state vector as well as the reward signal are obtained. In our simulation environment, for obtaining the new state vector, we run a load flow with inputs (i) the bus power injections at \(k\) (i.e., the revealed prosumption, \(L(k)\)) and (ii) the corrected action (battery power). For obtaining the reward, we use the new state vector as well as the prices at \(k\) (i.e., the revealed price values \(c(k)\)). The experience tuple created via this interaction is stored in the corresponding replay buffer (lines 14, 16). The actor and critic parameters are then updated (Sec. IV-A).

V. RESULTS

A. Setup

Algorithm 1 and the underlying actor and critic NNs were implemented in Python with Tensorflow 2.1 as the modeling framework and Keras as high-level interface to the Tensorflow library. Pandapower 2.4.0 [23] was used to simulate the distribution grid and the battery.

The monetary cost application is presented on a 4-bus toy grid, whereas the dispatch plan application is presented on a realistic Swiss grid with 34 buses. This choice of test grids allows for comparisons with [4] given that we have also chosen the same values for all common parameters. The DDPG agent is trained on \(\mathcal{M} = 20000\) episodes constructed based on different prosumption scenarios with \(K = 100\) time intervals and \(\Delta t = 9\) sec. For the exploration noise, we sample the Gaussian distribution with zero-mean and standard deviation \(\sigma = B_P^{UP}\). Furthermore, we apply an exponential decay (parameter \(\eta = 10^{-6}\) ) on \(\sigma\) to get high exploration early in training and exploit the agent’s policy more as the training progresses. Each replay buffer \(\mathcal{D}_{1,2}\) can store \(10^6\) experience tuples, the minibatch \(\mathcal{B}\) includes 1024 experience tuples sampled from the buffers, the actor and critic learning rates are respectively \(10^{-5}\) and \(10^{-4}\), the discount factor is \(\gamma = 1\), and the target update factor is \(\tau = 10^{-3}\). Finally, the actor network is a three-layer NN with 256, 256, 2 neurons for the three layers, respectively, and the critic network is a four-layer NN with 256, 256, 256, 1, neurons, respectively. The activation function of the hidden layers is RELU. Both NNs are obtained via experimentation.

B. Numerical Evaluations & Comparisons

We compare our agent’s performance to (i) iLypRC [4], (ii) MPC and (iii) the optimal solution given by an oracle, i.e., by a policy that knows the future realization of the uncertainty, on a set of testing scenarios for the prosumption. MPC and the oracle are exactly the same as presented in Sec. VII of [4]. Note that MPC requires forecasts and we apply the simple forecasting scheme described in [4] that is based on historical data. Also, for these comparisons, we consider our DDPG agent without the safety layer for fairness purposes. Subsequently, we also study the added value of the safety layer. The performance metrics are either the reward or the cost (i.e., the opposite of the reward) for each episode, as it is defined by (9) and the run time for the decision making.
Algorithm 1 DDPG Battery Control Agent
1: Initialize the critic $Q_{\phi}$ and the actor $\mu_{\phi}$ networks
2: Initialize the target critic $Q_{\theta_{\text{target}}}$ and the target actor $\mu_{\phi_{\text{target}}}$
3: Initialize the replay buffers $D_1, D_2$ as empty
4: for episode $= 1, 2, \ldots, M$ do
5: Obtain state $s(0)$ & Initialize a random process $\mathcal{Y}$
6: for $k = 0 \ldots K - 1$ do
7: Sample exploration noise $\epsilon$ from $\mathcal{Y}$
8: Select action $a(k) = \mu_{\phi}(s(k)) + \epsilon$
9: Project $a(k)$ to $U$
10: Correct $a(k)$ with the safety layer if necessary
11: Execute the corrected action to the grid
12: if $k = K$ then
13: Store $(s(k), a(k), r(k), s(k+1))$ in $D_2$.
14: else
15: Store $(s(k), a(k), r(k), s(k+1))$ in $D_1$.
16: end if
17: end for
18: Randomly sample a batch $B$ from $D_1$ and $D_2$
19: Compute targets with (15)
20: Update critic NN by a step of gradient ascent with (16)
21: Update actor NN by a step of gradient ascent with (14)
22: Update target critic and target actor NNs using (13)
23: end for

1) Monetary Cost Application on a 4-bus Toy System:
The considered system is a 4-bus line grid. The battery is connected at bus $b = 2$. The prosumption is placed at buses 1 and 4. The battery characteristics are $B_{UP}^P = -B_{LO}^P = 240$ kW, $\text{SoE}_{UP} = 167$ kWh, $\text{SoE}_{LO} = 0$ kWh. We study the monetary cost application with $\text{SoE}_{K,UP}^K = \text{SoE}_{UP}$, $\text{SoE}_{K,LO}^K = 0.93 \cdot \text{SoE}(0)$, $\text{SoE}(0) = 0.5 \cdot \text{SoE}_{UP}$. The real values of the buying and selling prices $(c_1(k),c_2(k))$ can vary among different times $k$, each one following a normal distribution with mean values equal to 1.4 and 0.75 €/MW, correspondingly. Moreover, they can vary up to 0.14 and 0.075 €/MW, respectively around the mean. Based on the prices, if not considering the lookahead SoE constraints, the optimal behavior is to export the maximum possible energy by discharging the battery. However, here, the battery decisions are non-trivial due to the terminal SoE constraint given by $\text{SoE}_{K,LO}^K$. Also, we set $w_1 = 0$, $w_2 = 21$, $w_3 = 10$.

Fig. 1(a) shows the learning process. The darker line represents the running-time average reward over the latest 50 episodes, which converges indicating a successful training. Fig. 1(b) and Table I summarize the comparative results. The episodes of the test set are based on 50 prosumption scenarios that are different from the training set. Our DDPG agent outperforms all other schemes in terms of run time, with the mean run time being 20 times lower than MPC and 10 times lower than iLyPC. The max run time is more than 27 times lower than the other algorithms. This improvement comes at an expense in terms of the cost, which is around 5.5% larger for the RL agent. Finally, Fig. 1(c) shows that the DDPG agent is able to comply with the terminal constraint of the battery’s SoE. In particular, the red line represents the $\text{SoE}_{K,LO}^K$ constraint and for all scenarios, the battery’s SoE at time $K$ is above it. Note that the SoE might fall below $\text{SoE}_{K,LO}^K$ for a time earlier than $K$, since the constraint is expressed only for the terminal time interval.

2) Dispatch Plan Application on a Real Swiss Grid: The considered grid is radial with the battery connected at bus 1 that directly connects to the PCC. The battery parameters are: $B_{UP}^P = -B_{LO}^P = 2$ MW, $\text{SoE}_{UP} = 2$ MWh, $\text{SoE}_{LO} = 0$ MWh. Other grid parameters can be found in [4]. We assume that only the bus 21, where the PV is located at, has uncertain prosumption [4], [19]. We study the dispatch plan application with $\text{SoE}_{K,UP}^K = 0.53 \cdot \text{SoE}_{UP}$, $\text{SoE}_{K,LO}^K = 0$, $\text{SoE}(0) = 0.5 \cdot \text{SoE}_{UP}$ and $P_{\text{PV}}(k) = 0$, $\forall k \in \mathcal{H}$. Due to the large PV generation, if not considering the lookahead SoE constraints, we should charge the battery at each $k \in \mathcal{H}$ for following the terminal SoE constraint.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>min/mean/max run time (s)</th>
<th>max/mean cost (std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>-</td>
<td>9.50/3.44 (3.39)</td>
</tr>
<tr>
<td>DDPG</td>
<td>0.003/0.008/0.004</td>
<td>9.70/3.64 (3.37)</td>
</tr>
<tr>
<td>iLyPC</td>
<td>0.018/0.220/0.039</td>
<td>9.51/3.45 (3.38)</td>
</tr>
<tr>
<td>MPC</td>
<td>0.045/0.265/0.079</td>
<td>9.52/3.45 (3.38)</td>
</tr>
</tbody>
</table>

Fig. 1. Monetary cost application on 4-bus grid
the dispatch plan. But, here the decisions should be optimized due to the terminal constraint given by $\text{SoE}^{K,\text{UP}}$. The reward weights are defined as $w_1 = 1, w_2 = 7000, w_3 = 10, A = 100$.

Fig. 2(a) shows that the learning process successfully converges. We compare the DDPG agent’s performance on 45 test episodes/scenarios of the uncertain PV generation. The results are summarized in Table II and Fig. 2(b). Again, the DDPG agent outperforms all other approaches in terms of run time. The mean run time is 32 times lower than iLypRC and 184 times lower than MPC. Importantly, contrary to iLypRC and MPC, our DDPG agent’s run time does not depend on the grid size, which can be seen by comparing the run time values for the 4-bus grid and the 34-bus grid in Tables I, II. This indicates the excellent scaling properties of the DDPG agent. With respect to the cost, DDPG achieves slightly larger costs than iLypRC, but significantly outperforms MPC, especially for the scenarios 7, 8, 9 (Fig. 2(b)). For these scenarios MPC does not perform well due to the inaccuracy of the forecasts, as also explained in [4]. Finally, from Fig. 2(c) we see that our DDPG agent was able to comply with the terminal constraint and keep the SoE$(K)$ below SoE$^{K,\text{UP}}$. The battery’s optimal charging and discharging behavior (given by the oracle) for most scenarios would have been such that SoE exactly reaches SoE$^{K,\text{UP}}$ at the terminal time $K$. iLypRC achieved this behavior for most scenarios and therefore its cost is always slightly lower than our agent’s cost at the expense of increased run time. Our DDPG agent was more conservative, which can be attributed to the performed exploration.

3) Safety Layer Performance: We evaluate the safety layer for the dispatch plan application on the 4-bus grid of Sec. V-B1. In particular, we compare the DDPG agent with and without safety layer in terms of the number of constraint violations, the run time and the cost. The battery parameters are $B^\text{UP} = -B^\text{LO} = 960$ kW, SoE$^\text{UP} = 668$ kWh, SoE$^\text{LO} = 0$ kWh. We set $P^\text{pcc}(k) = 0, \forall k \in H$, SoE$^{K,\text{UP}} = \text{SoE}^\text{UP}$, SoE$^{K,\text{LO}} = 0.67 \cdot \text{SoE}(0)$ and SoE$(0) = 0.5 \cdot \text{SoE}^\text{UP}$. Also, $w_1 = 1, w_2 = 50, w_3 = 100, A = 100$.

The results are shown in Fig. 3 and Table III. Clearly, the agent with the safety layer incurs much less voltage magnitude violations (16 in total) versus the agent without safety layer.

![Image](b) Comparison of cost values per scenario/testing episode

![Image](c) Battery SoE evolution

Fig. 2. Dispatch plan application on real Swiss grid

![Image](a) Episodic reward & its running average during training

![Image](b) Training

![Image](b) Testing

Fig. 3. Safety layer evaluation on 4-bus line grid
TABLE III

<table>
<thead>
<tr>
<th>Agent</th>
<th>w.o. SL</th>
<th>w. SL</th>
</tr>
</thead>
<tbody>
<tr>
<td>run/max mean run time (s)</td>
<td>0.0050/0.0080/0.004</td>
<td>0.011/0.19/0.013</td>
</tr>
<tr>
<td>max/mean cost (std)</td>
<td>1.92/0.79 (0.49)</td>
<td>1.91/0.76 (0.46)</td>
</tr>
</tbody>
</table>

(3781 in total). Note that there may still emerge some voltage violations after the application of the safety layer. This is due to multiple reasons (i) the safety layer solves a linearized and distributed problem, (ii) the realization of the uncertain assumption is not known at the time of the decision, (iii) the second-step correction of (22) might incur back some voltage violations. Nevertheless, due to the consideration of the penalty terms in (11), both agents are able to improve the average rewards as training progresses. Thus, with adequate training the safety layer is needed much less frequently.

The agent with safety layer has a higher run time, due to the added complexity of computing the sensitivity matrix and solving the optimization problem. However, due to the closed form solution, the added run time is only 0.009 sec on average. From Fig. 3(b), we see that both agents achieve similar costs, with the exception of scenarios 16 and 17 for which using the safety layer to satisfy the constraints in $X$ was at the expense of the satisfaction of the terminal SoE constraint leading to increased cost values (see (11)).

VI. CONCLUSIONS

In this paper, we designed a RL agent for the control of a battery with lookahead SoE constraints in a distribution grid with DERs. Our RL agent is based on the DDPG algorithm enhanced with a time-dependent policy, two replay buffers and a safety layer to satisfy the hard grid and battery constraints. The evaluations show the excellent scaling properties of our approach and that it significantly outperforms the state-of-the-art in run time at a small expense with respect to the cost. In our future work, we will extend this approach to consider multiple batteries in a distributed setting.

REFERENCES


