A Reinforcement Learning Approach for Branch Overload Relief in Power Systems

Mariana Kamel[a,b], Yawei Wang[a,c], Chen Yuan[a], Fangxing Li[b], Guangyi Liu[a], Renchang Dai[a], Xiuzhen Cheng[c]
[a] GEIRI North America, San Jose, CA, USA
[b] The University of Tennessee, Knoxville, TN, USA
[c] The George Washington University, Washington, D.C., USA

***Abstract*—This work presents a reinforcement learning (RL) approach to the problem of branch overload relief. An agent is trained to re-dispatch generators' real power output in order to adjust the power flow through the network so that none of its branches is overloaded. The generation re-dispatch agent is trained using the deep deterministic policy gradient (DDPG) algorithm. The proposed approach is tested on both the IEEE 14-bus and 39-bus systems. Once trained, the performance of the re-dispatch agents was compared against that of the classical optimal power flow (OPF) approach. The trained agents demonstrated better performance with both having: (a) close to a 100% success rate for cases which had an OPF solution; and (b) 75.7% and 57.6% success rates for cases which had no feasible OPF solution. Such results confirm the potential of the proposed DDPG-based RL re-dispatch approach as a reliable method for managing branch overloading.**

***Index Terms*—Branch overload, deep deterministic policy gradient, generation re-dispatch, reinforcement learning**

I. Introduction

Power systems are designed and operated with the main objective of having a stable, secure and continuous supply of electrical power to customers. However, the possibility exists for having an unpredictable disturbance that triggers a sequence of one or more dependent component outages (cascading outages) eventually leading to an interruption of the energy supply across major parts of the system i.e. a blackout. In [1], overloaded branches, i.e. transmission lines and transformers, are reported to be one of the main contributors to cascading outages and/or system blackout. Depending on the level of overload and/or the branch protection settings, the overloaded branch(s) might trip out. This, in turn, might cause additional branches to exceed their loading limits and also trip out. In order to prevent such cascading outages, the overload on system branches must be managed and relieved. One way to manage branch overloading is through generation re-dispatch. The problem of generation re-dispatch is typically formulated as an optimal power flow (OPF) problem in which a certain operational objective is to be minimized or maximized while satisfying a number of operational constraints [2-4]. In general, the OPF problem needs to be computationally affordable. This is true especially for OPF schemes proposed for real-time and/or emergency control. However, power systems are typically large and highly complex systems which makes the OPF problem one of a high computational burden. In general, the complexity of a power system is a critical issue, not only for the generation re-dispatch problem, but for all model-based control approaches as well. Such increasing complexity, in addition to the uncertainty in the operation of power systems, has started to draw researchers more towards adopting model-free control approaches. Within this context, the method of reinforcement learning (RL) enables control agents to learn an optimal control law from interaction with a system without the need of a system model. In the literature of power systems operation and control, reinforcement learning has been proposed and successfully implemented in a wide range of power system applications such as in [5-10]. This confirms the potential of reinforcement learning as a viable tool for solving power system control and operational problems. Accordingly, it is proposed in this work to implement a reinforcement learning based control whose objective is to relieve the overload on system branches through generation re-dispatch.

The remaining of this paper is organized as follows: Section II presents a brief overview of the reinforcement learning algorithm utilized in this work. Section III presents the design of the proposed RL based generation re-dispatch. In Section IV, simulation results are presented and discussed. Finally, Section V concludes the paper.

II. Reinforcement Learning and the Deep Deterministic Policy Gradient Algorithm

The overload mitigation scheme proposed here utilizes reinforcement learning to train a control agent to adjust the output power of the system generators in order to reduce the apparent power (MVA) flow of all overloaded network branches to be within acceptable limits. In reinforcement learning, the agent interacts with the environment in discrete time-steps. At each time-step t, the agent observes the system states s_t, decides on an action a_t and receives a reward r_t. The RL agent learns progressively as it accumulates more and more experience until it learns an optimal control law. Different learning algorithms have been

This work is supported by the State Grid Corporation technology project 5455HJ180020

proposed and utilized in the field of reinforcement learning. The one utilized in this work is known as the deep deterministic policy gradient (DDPG) algorithm proposed in [11]. This section presents a brief over-view of the DDPG algorithm. It starts with some key definitions, concepts and mathematical relations. Then, it proceeds to the description of the DDPG algorithm.

A. Definitions

1) Policy: A policy π, is a function that defines the behavior of the agent. A policy can either be deterministic or stochastic. A deterministic policy, $\pi: S \rightarrow A$, is one that maps state to specific actions. On the other hand, a stochastic policy outputs a probability distribution over a pre-defined set of actions $\pi: S \rightarrow \mathcal{P}(A)$.

2) Return: Return is defined as the sum of discounted future rewards $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i(s_i, a_i)$ where γ is a discounting factor, $\gamma \in [0, 1]$.

3) Value function: Value functions represent the expected value of a given state. There are two types of value functions: (1) state-value function, $V(s_t)$, which is defined as the expected return starting from state s_t and following policy π. (2) action-value function, $Q(s_t, a_t)$, which is defined as the expected return after taking action a_t in state s_t and following policy π thereafter.

4) Bellman's equation: In reinforcement learning, the goal is to learn a policy that maximizes the expected return in every state, i.e. its value function. The majority of learning algorithms utilize value functions (in particular the action-value function) to infer or learn the optimal policy. Those algorithms are all based on Bellman's principle of optimality [12]. This principle leads to the following necessary condition on the optimality of the action-value function:

$$Q^*(s_t, a_t) = \mathbb{E}\left[r_t(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\right] \quad (1)$$

where $\mathbb{E}[\cdot]$ indicates the expected value. The above equation, also known as the Bellman's equation, is a recursive relationship which allows for the use of iterative approaches to solve for the optimal action-value function $Q^*(s, \mathrm{a})$.

B. Deep Deterministic Policy Gradient (DDPG) Algorithm

The Deep Deterministic Policy Gradient (DDPG) is a learning algorithm designed for control problems where the agent policy is deterministic and the action space is continuous [11]. The DDPG utilizes an actor-critic frame of work which allows for concurrent learning of a policy function (the actor) and a Q-function (the critic). The DDPG uses neural networks as function approximators for both actor and critic functions. The actor network is parameterized by θ^{π} while the critic network is parameterized by θ^{Q}. The algorithm uses the Bellman equation to learn the Q-function, i.e. θ^{Q}, and uses the Q-function to learn the policy, i.e. θ^{π}.

1) Q-learning: As mentioned above, the Bellman's equation provides the basis for learning the optimal action-value function $Q^*(s, a)$. Now, given the parameterized critic function $Q(s, a|\theta^Q)$. Then, its parameters, θ^Q, are optimized by minimizing the mean-squared Bellman error (MSBE) given below in (3). This error indicates how close the value of Q comes to satisfying the optimality condition of (2):

$$L(\theta^Q) = E[(Q(s_t, a_t|\theta^Q) - y_t)^2] \quad (2)$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1})|\theta^Q) \quad (3)$$

y_t is known as the target. As can be seen from (3), the target value depends on the same parameters being optimized for, i.e. θ^Q. This fact is known to make learning through MSBE minimization unstable for neural networks. In order to stabilize the learning process, two techniques are adopted: a replay buffer, and a separate critic target network for calculating y_t. Readers are referred to [11] for more details on those two techniques.

2) Policy learning: The optimal policy is one which satisfies:

$$\theta^{\pi *} = arg\ \max_{\theta^{\pi}}(J = \mathbb{E}[Q(s, \pi(s|\theta^{\pi}))]) \quad (4)$$

Since the action space is continuous and the Q-function is differentiable with respect to action then, gradient ascent can be used to solve the optimization problem of (4). Accordingly, the parameters of the actor function are updated using the following gradient:

$$\nabla_{\theta^{\pi}} J = \frac{1}{N} \sum_{i} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})\,|_{s=s_i} \quad (5)$$

In [13], the authors proved that this is an equivalent to the policy gradient, hence the name. Since the actor network is also used in (3), then, in order to have a stable learning, a separate actor target network is used when calculating y_t.

Finally, the DDPG algorithm utilizes what is known as exploration noise which enables the agent to better explore the action space. A summary of the DDPG algorithm is given in Table 1.

TABLE 1
DDPG ALGORITHM

Randomly initialize critic Q and actor π with weights θ^Q and θ^{π}.
Initialize target network Q' and π' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^{\pi}$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \pi(s_t|\theta^{\pi}) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R

Sample a random mini-batch of N transitions $(s_i, a_i, r_i, s_{i+1})$from R
Calculate $y_i = r_i + \gamma Q'\left(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'}\right)$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
Update the actor policy using the sampled policy gradient:

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

Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$

end for
end for

III. Design of the DDPG-Based Generation Dispatch Control for Over-Load Relief of System Branches

A. States, Actions, and Reward Function Design

For the control problem in hand, the three sets of information interchanged between the environment and the agent i.e. system states, actions, and rewards are defined as follows: Given an N-bus system where $\mathcal{G}$ is the set of system generators (excluding the slack unit), $\mathcal{B}$ is the set of system branches, then; the system states are defined as:

$$S = \{s|s_t = \{I_{i,t}\ \forall\ i \in \mathcal{B}\} \cup \{P_{gj,t}\ \forall\ j \in \mathcal{G}\} \quad (7)$$

where $I_{i,t}$ is the current of the i[th] branch, $P_{gj,t}$ is the real output power of the j[th] generator.

As for the action space, it is defined as follows:

$$A = \{a|a_{j,t} = \Delta P_{gj,t}\ \forall\ j \in \mathcal{G}\} \quad (8)$$

where $\Delta P_{gj,t}$ is the amount of change in the real output power of the j[th] generator. As mentioned previously, the action space is continuous i.e. $\Delta P_{gj,t}$ is a number in the continuous range of $[\Delta P_{gj-min}, \Delta P_{gj-max}]$ where ΔP_{gj-min} and ΔP_{gj-max} represent the minimum and maximum change that generator j can accommodate during the control time step t. It should be noted that if the action decided on by the agent $\Delta P_{gj,t}$ results in violating the upper or lower limits of the generator output capability, then the generator output will be set at the violated limit.

As for the reward function, its formulation is given as follows:

$$r_t = -1 + r_{TML} \quad (9)$$

where

$$r_{TML} = \begin{cases} +20 & \text{If } I_{i,t} \le I_{i-max}\ \forall\ i \in \mathcal{B} \\ -20 & \text{If power flow diverges} \\ 0 & \text{Otherwise} \end{cases} \quad (10)$$

The above reward function is composed of two parts: (1) A penalty of -1 which means that the agent would be penalized for each taken step. This penalty is expected to lead the agent to arrive at an action policy which achieve the control objective in a minimum number of steps. (2) A terminal reward r_{TML}. This reward is given to the agent if the taken action terminates the given episode. An episode is terminated if either: (a) the taken action results in all lines secured. In that case, the agent is given a high reward of +20 and the episode is terminated. (b) The taken action results in an unsolved power flow case. In that case, the agent is given a high penalty of -20 and the episode is terminated. (c) The number of control steps reaches its maximum. If the maximum number of steps is reached without achieving the control objective, then the agent will not be rewarded.

B. Actor and Critic Functions Design

The actor network has an input layer, 2 hidden layers, and an output layer. The input layer has (B + G) neurons where B is the number of system branches and G is the number of generator units (excluding slack). The 2 hidden layer have 600 and 400 neurons respectively. The output layer has G neurons. A rectified non-linearity activation function is used for the 2 hidden layers while a tanh activation one is used for the output layer. The critic network has a similar structure: an input layer with (B + G) neurons, 2 hidden layers with 600 and 400 neurons each, and an output layer with only one neuron. Actions are included at the 2nd hidden layer.

C. DDPG Training Parameters Setting

For training the actor and critic networks, the following settings were used: learning rate of 0.0001 and 0.001 for the actor and critic networks respectively; a discount factor of $\gamma = 0.99$; a target update factor of $\tau = 0.001$; a mini-batch size of 64; and a replay buffer size of 10^4. As for the exploration noise, a normal (Gaussian) distribution with a standard deviation of 0.5 was utilized. Once the reply buffer is filled and the networks are starting to learn, the standard deviation is reduced at each training step by a factor of 0.9988.

D. DDPG Training Setup

In this work, an offline setup is used to train the RL agent, i.e. the RL agent interacts with a simulated rather than a real power system environment. In this setup, the power system environment is built in MATLAB. The RL agent is built in Tensorflow. Python is used to interchange information (system states, actions, and rewards) between the power system environment and the RL agent.

Matpower is used to generate the initial states. An initial state is an operating point where one (or more) of the system branches is overloaded. To generate these initial states: (1) the load at each bus was randomly set at a value between 80% and 120% of its base-case value. Similarly, the real power output of each generator was randomly set at a value within 80 to 120% of its base-case value. (2) a branch is randomly taken out and the system AC power flow is solved. If taking the branch out resulted in an unsolvable power flow case or in creating islands, then the created case is discarded. Otherwise, (3) the MVA flow through the system branches are calculated. If any of the calculated MVA flows exceed their corresponding limit, then, the case is used as an initial state for the RL algorithm.

IV. Simulation Results

The performance of the proposed RL-based control is investigated on both the IEEE 14-bus and 39-bus systems. The IEEE 14-bus system has 5 generators and 20 branches.

Accordingly, the total number of states and actions for its RL agent is 24 and 4 respectively. The IEEE 39-bus system has 10 generators and 46 branches and therefore, its RL agent will have 55 states and 9 actions. The obtained results are presented and discussed next.

A. Agent Training

The RL agent training results are presented in Figs. 1-4. These figures show the critic loss (given previously in (2)) and episode reward during agent training for each test system. As can be seen from these figures, the 14-bus and 39-bus agents were able to learn an optimal policy to mitigate the overload in less than 16000 episodes. In addition, as it was intended, the penalty on each steps led the RL agents to learn to achieve the control objective in a minimum number of control steps.

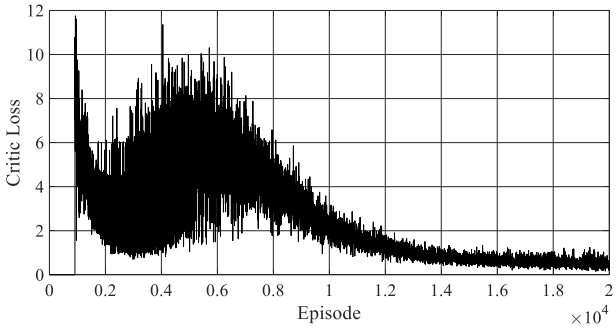

Fig. 1. Agent training results for the IEEE 14-bus system: critic loss.

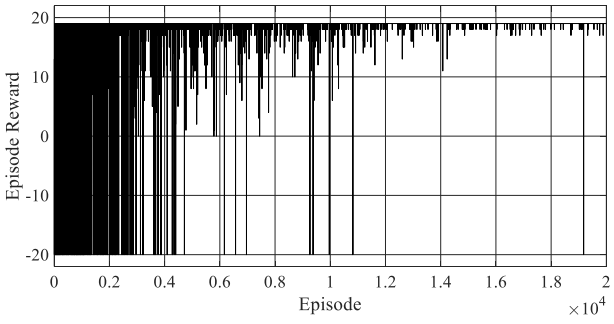

Fig. 2. Agent training results for the IEEE 14-bus system: episode reward.

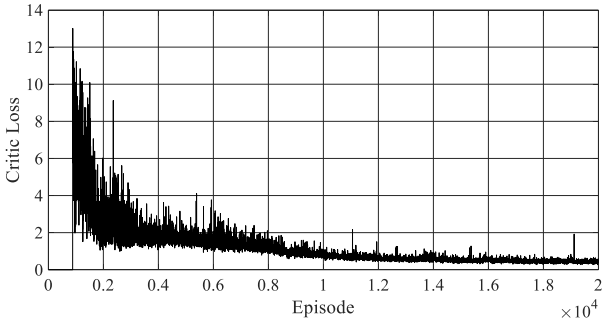

Fig. 3. Agent training results for the IEEE 39-bus system: critic loss.

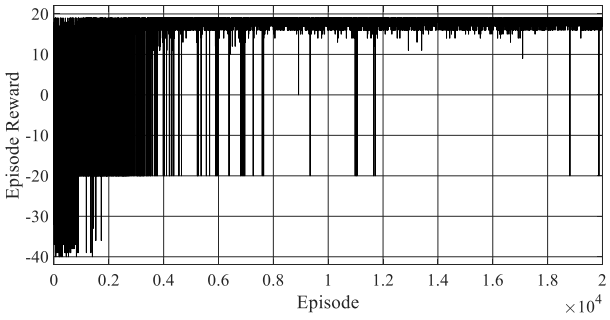

Fig. 4. Agent training results for the IEEE 39-bus system: episode reward.

B. Agent Performance Testing

The performance of the trained DDPG-based RL agents was tested and compared against the classical OPF approach. The OPF based generation re-dispatch is implemented using Matpower. Here, the objective of the re-dispatch problem is set to minimize the generation cost while satisfying: network constraints, limits on generation output power, and limits on branch MVA flow. Matpower utilizes a primal-dual interior-point method to solve for the optimal dispatch.

In order to compare the performance of RL agent to that of OPF, two different tests were carried out. In the first test, the cases used for testing are all cases which had an OPF solution. In the second test, the cases used for testing are ones which did not have an OPF solution.

1) Test-1: The number of cases used in this test is 5,000 cases, and all of them had feasible OPF solutions. The results of this test are shown in Figs. 5 and 6. Out of the 5,000 cases used for testing, the RL agent of the 14-bus system failed in 4 cases while that of the 39-bus system failed in 3. These results yield a success rate close to 100% for both RL agents.

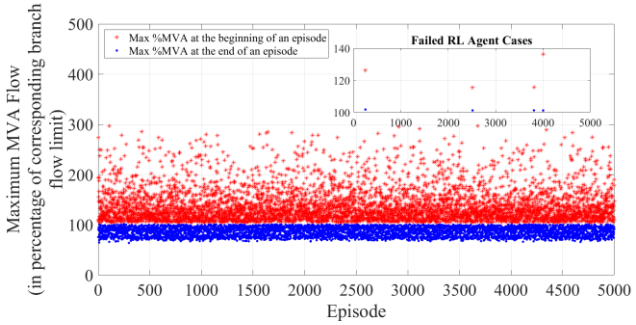

Fig. 5. Test-1 results for the IEEE 14-bus system: maximum percentage MVA flow at the beginning and end of each testing episode.

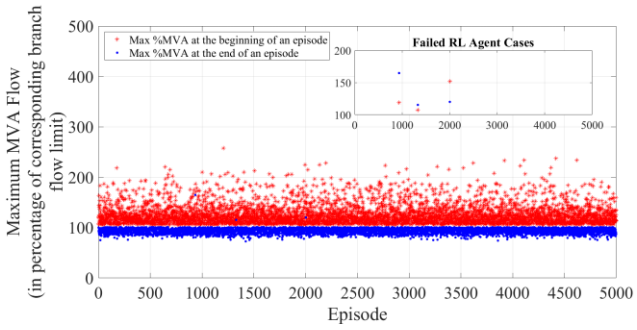

Fig. 6. Test-1 results for the IEEE 39-bus system: maximum percentage MVA flow at the beginning and end of each testing episode.

2) Test-2: In this test, a set of 5000 testing cases was used. All of the used cases had no OPF solution. The results of this test are shown in Figs. 7 and 8. As can be seen from this figure, the RL agent was successful in relieving branch overloads for most of the tested cases: 3786 and 2881 cases for the 14-bus and 39-bus systems respectively. These numbers correspond to 75.7% and 57.6% success rates. Further investigations into some of the cases were both OPF and the RL agent have failed revealed that generation re-dispatch was not sufficient as a stand-alone control action for mitigating branch overloads. Other control actions should have been considered such as network reconfiguration and/or load shedding.

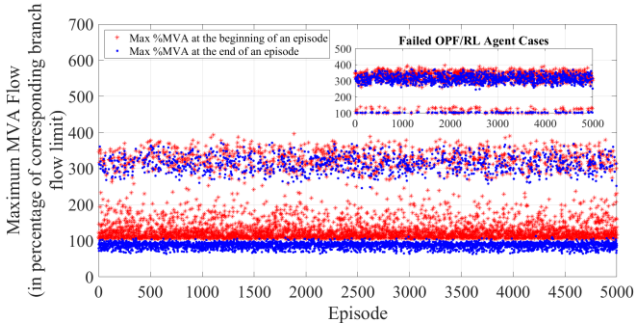

Fig. 7. Test-2 results for the IEEE 14-bus system: maximum percentage MVA flow at the beginning and end of each testing episode.

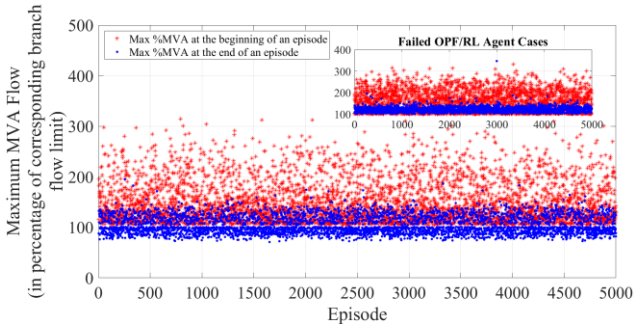

Fig. 8. Test-2 results for the IEEE 39-bus system: maximum percentage MVA flow at the beginning and end of each testing episode.

V. Conclusion

This work proposed a reinforcement learning based generation re-dispatch control whose objective is to relieve the overload on system branches. The training of the re-dispatch agent is proposed to be carried out using the deep deterministic policy gradient (DDPG). Both the IEEE 14-bus and 39-bus systems were utilized to test the performance of the proposed control. For both test systems, the re-dispatch agents were successful in learning a re-dispatch policy which relief branch overload in a minimum number of control steps. In addition, the DDPG based re-dispatch agents, once trained, were able to achieve a better performance than that of classical OPF based control. For cases were OPF had a solution, the RL agents of both systems had close to a 100% success rate. For cases were OPF had no solution, the RL agents had 75.7% and 57.6% success rates for the 14-bus and 39-bus systems respectively. This clearly shows that the proposed DDPG-based RL approach is of more reliability when it comes to managing branch overloading.

References

[1] H. Haes Alhelou, M. E. Hamedani-Golshan, T. C. Njenda, and P. Siano, "A survey on power system blackout and cascading events: Research motivations and challenges," *Energies,* vol. 12, no. 4, p. 682, 2019.

[2] J. Lavei, A. Rantzer, and S. Low, "Power flow optimization using positive quadratic programming," *IFAC Proceedings Volumes,* vol. 44, no. 1, pp. 10481-10486, 2011.

[3] X. Bai, H. Wei, K. Fujisawa, and Y. Wang, "Semidefinite programming for optimal power flow problems," *International Journal of Electrical Power & Energy Systems,* vol. 30, no. 6-7, pp. 383-392, 2008.

[4] B. Kocuk, S. S. Dey, and X. A. Sun, "Strong SOCP relaxations for the optimal power flow problem," *Operations Research,* vol. 64, no. 6, pp. 1177-1196, 2016.

[5] V. Nanduri and T. K. Das, "A reinforcement learning model to assess market power under auction-based energy pricing," *IEEE transactions on Power Systems,* vol. 22, no. 1, pp. 85-95, 2007.

[6] D. Ernst, M. Glavic, and L. Wehenkel, "Power systems stability control: reinforcement learning framework," *IEEE transactions on power systems,* vol. 19, no. 1, pp. 427-435, 2004.

[7] J. G. Vlachogiannis and N. D. Hatziargyriou, "Reinforcement learning for reactive power control," *IEEE transactions on power systems,* vol. 19, no. 3, pp. 1317-1325, 2004.

[8] T. Yu, B. Zhou, K. W. Chan, L. Chen, and B. Yang, "Stochastic Optimal Relaxed Automatic Generation Control in Non-Markov Environment Based on Multi-Step $ Q (\lambda) $ Learning," *IEEE Transactions on Power Systems,* vol. 26, no. 3, pp. 1272-1282, 2011.

[9] E. Jasmin, T. I. Ahamed, and V. J. Raj, "Reinforcement learning approaches to economic dispatch problem," *International Journal of Electrical Power & Energy Systems,* vol. 33, no. 4, pp. 836-845, 2011.

[10] D. Ye, M. Zhang, and D. Sutanto, "A hybrid multiagent framework with Q-learning for power grid systems restoration," *IEEE Transactions on Power Systems,* vol. 26, no. 4, pp. 2434-2441, 2011.

[11] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971,* 2015.

[12] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society,* vol. 60, no. 6, pp. 503-515, 1954.

[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.