Real-time Security Margin Control Using Deep Reinforcement Learning

Hannes Hagmar, *Student Member, IEEE,* Robert Eriksson, *Senior Member, IEEE,* and Le Anh Tuan, *Member, IEEE*

***Abstract*—This paper develops a real-time control method based on deep reinforcement learning (DRL) aimed to determine the optimal control actions to maintain a sufficient secure operating limit (SOL). The SOL refers to the limit to the most stressed pre-contingency operating point of an electric power system that can withstand a set of credible contingencies without violating stability criteria. The developed DRL method uses a hybrid control scheme that is capable of simultaneously adjusting both discrete and continuous action variables. The performance is evaluated on a modified version of the Nordic32 test system. The results show that the developed DRL method quickly learns an effective control policy to ensure a sufficient SOL for a range of different system scenarios. The impact of measurement errors and unseen system conditions are also evaluated. While the DRL method manages to achieve good performance on a majority of the defined test scenarios, including measurement errors during the training phase would improve the robustness of the control with respect to random errors in the state signal. The performance is also compared to a conventional look-up table control where the advantages of the DRL method are highlighted.**

***Index Terms*—Deep reinforcement learning, preventive control, proximal policy optimization, secure operating limit**

I. Introduction

An electric power system that can withstand the loss of any single system component (N-1) without losing stability is generally referred to as being operated *securely*. To ensure that a power system is secure, system operators continuously estimate security margins and take preventive actions as soon as the security margins are deemed insufficient. This paper is devoted to the determination of the optimal control actions to ensure a sufficient *secure operating limit* (SOL). The SOL is the margin to the most stressed pre-contingency operating point that can withstand a set of credible contingencies without violating defined stability criteria [1]. Thus, the SOL provides a measure of how far a system can move from its current operating point and still be able to maintain N-1 security. The estimation of the SOL is computationally demanding and requires time-domain simulations (or a quasi-steady-state approximation) to account for the dynamic response after a disturbance. However, it generally provides a better measure of the security margin than conventional methods that are based on static assessments.

Typical preventive control actions used to ensure sufficient security margins include (1) management of reactive power resources and voltage control through tap changes and capacitor/reactor switching, and (2) generation rescheduling and load curtailment intended to relieve loading of stressed transmission lines and buses [2]. Traditionally, power system control schemes have been based on look-up tables that are pre-defined through offline simulations based on various typical scenarios. However, as control actions and their level of activation are correlated with different costs, such non-optimal control schemes can significantly decrease the operational efficiency of the system.

Optimal control methods such as model predictive control have in several studies been proposed to deal with various types of electric power system control [2], [3]. However, power systems exhibit complex characteristics with a large number of states, nonlinear dynamics, and system uncertainties [3], [4]. To be able to compute the optimal control actions in a time frame required by system operators, significant simplifications of the system model are then generally required. Data-based control schemes, such as in [5], [6], are an alternative approach based on an (offline) assessment of different states and actions. The optimal actions for each state are stored in a database and supervised learning algorithms are trained to map a state to a set of optimized actions. However, the combinatorial complexity in assessing all the state-action pairs, especially if the number of available control actions is high, may result in slow learning and makes the method not suited to handle changes (e.g. in topology) of the underlying power system.

In recent years, significant progress has been made in solving complex control problems by using reinforcement learning (RL). RL is a data-driven approach where a control agent learns an optimal policy through interactions with a real power system or its simulation model [7], [8]. Its combination with deep learning, called deep reinforcement learning (DRL), has proven effective in solving complex control problems in environments such as games [9], [10], autonomous driving [11], and robotics [12]. DRL enables automatic high-dimensional feature extraction, making the control agent capable of handling a large number of states and actions that are involved in electric power system control.

Previous implementations of DRL in electric power system control have so far mainly been focused on emergency control, which has the role of controlling the system back into a stable state after a disturbance has already occurred [13]. Implementations include methods adapted for automatic voltage control [14]–[17], optimal load shedding [4], [18], [19], generator dynamic breaking [4], and oscillation damping [20]. DRL-based implementations adapted for preventive control found in the literature are few and in [13], the authors argue

The work presented in this paper has been financially supported by Energiforsk and Svenska kraftnät (Swedish National Grid) with project numbers EVU10140 and EVU10450.

that the reason may be because these control problems have traditionally been formulated as static optimization problems. One example of an RL-based implementation for preventive control is presented in [21], which aimed to determine the optimal control of active power generation for preventing cascading failures and blackouts. However, the method was based on a tabular form of Q-learning, which in general is not suited to handle large state and action spaces.

Although previous DRL implementations in electric power system control have achieved good performance on the presented test systems, some limitations in terms of practical control remains. To achieve efficient control, system operators are generally required to simultaneously control both discrete (e.g. switching of a shunt capacitor) and continuous (e.g. the level of active power generation rescheduling and load curtailment) action variables. However, state-of-the-art DRL algorithms such as deep Q-networks, or deep deterministic policy gradients are generally designed to only control *either* discrete *or* continuous action variables, and all previously mentioned implementations of DRL in electric power system control have been reduced to rather simple control schemes where only a single type of action space has been considered.

In this paper, we address the lack of preventive control methods and introduce a new method based on DRL aimed at determining the optimal control actions to maintain a sufficient SOL. We also introduce a hybrid control scheme, which is capable of simultaneously adjusting both discrete and continuous action variables. To the authors' best knowledge, this is the first paper to develop a DRL method to address the advanced control problem of optimally maintaining a sufficient SOL, as well as providing a preventive control method that can handle multiple discrete and continuous actions simultaneously.

The main contributions of this study can be summarized as:

- Development of a preventive control tool based on DRL aimed to optimally control and ensure a sufficient SOL. The DRL-based tool monitors the current state through measurements, and if the system moves too close to the security boundary, it suggests optimal control actions to steer the system back into a secure operation again.
- A new DRL architecture based on the proximal policy optimization (PPO) algorithm is developed that can handle a hybrid of continuous and discrete action spaces simultaneously. The hybrid DRL agent can simultaneously control reactive power injections through switching of shunt reactive power devices, as well as controlling power flows by load and generation rescheduling in stressed system areas.
- An investigation into several aspects of the hybrid DRL agent, including robustness to different simulation scenarios and noise in the inputs.

The rest of the paper is organized as follows. In Section II, the theory regarding RL, the PPO algorithm, and adaptations for hybrid control are presented. In Section III, the proposed method is presented along with the steps for developing the training data and the training of the developed hybrid DRL agent. In Section IV, the results and the discussion are presented. Concluding remarks are presented in Section V.

II. Reinforcement learning: Theory

Reinforcement learning is a data-driven approach used to solve complex and sequential optimal control problems. RL problems are generally modeled as discrete-time Markov decision processes (MDPs), where an agent uses its policy to interact with the MDP to give a trajectory of states, actions, and rewards. The received reward - also referred to as the reinforcement signal - is used to determine whether the taken actions were effective. The most common objective of the agent is to maximize the expected sum of future rewards over time. Through continuous interactions with the environment, the agent is then trained to achieve this goal [22].

In this paper, the considered MPD is comprised of: a state space $\mathcal{S}$; a hybrid action space with both discrete and continuous actions $\mathcal{A}$; an initial state distribution with density $p_1(s_1)$; a stationary transition dynamics distribution $p(s_{t+1}|s_t, a_t)$ which satisfies the Markov property $(s_{t+1}|s_1, a_1, ...s_t, a_t) = (s_{t+1}|s_t, a_t)$; and a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote a_t, s_t, and R_t the action, the state, and the reward, respectively, taken at time t [22]. A parametrized policy is used to select actions in the MPD. In the formulation used in this paper, the policy is assumed to be stochastic and can be formulated by $\pi_\theta(a_t|s_t) : \mathcal{S} \longrightarrow \mathcal{P}(\mathcal{A})$ where $\mathcal{P}(\mathcal{A})$ is a set of probability measures on $\mathcal{A}$, $\theta \in \mathbb{R}^n$ is a vector of n parameters, and $\pi_\theta(a_t|s_t)$ is a conditional probability density of taking action a_t in state s_t associated with the policy. The return G_t^γ is the total discounted reward from time step t and onward, defined as:

$$G_t^\gamma = \sum_{k=t}^{T} \gamma^{k-t} R(s_k, a_k) \tag{1}$$

where $0 < \gamma < 1$ is a discounting factor. The value function is defined as the expected total discounted reward in state s when following the policy: $V^\pi(s) = \mathbb{E}\left[G_t^\gamma | s_t = s; \pi\right]$. The action-value function is defined as the expected total discounted reward in state s when taking action a and *then* following the policy: $Q^\pi(s, a) = \mathbb{E}\left[G_t^\gamma | s_t = s, a_t = a; \pi\right]$.

A. Policy Gradients and Actor-Critic Methods

Policy gradient methods are a class of model-free RL algorithms that learns a parameterized policy that can select actions without requiring a value function [23]. These methods seek to maximize a defined objective function $J(\theta)$ parametrized by θ, and their updates commonly approximate gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)} \tag{2}$$

where $\widehat{\nabla J(\theta_t)}$ is a stochastic estimate of the gradient of the objective function with respect to θ_t [23] and α is the learning rate used in the optimization. One of the most commonly used gradient estimator in RL has the following form:

$$\widehat{\nabla J(\theta_t)} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right] \tag{3}$$

where the expectation $\hat{\mathbb{E}}_t [...]$ indicates an empirical average over a batch of samples drawn from the MDP, $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the gradient of the parametrized policy, and $\hat{A}_t$ is an

estimator of the advantage function at time step t, which can be formulated as:

$$\hat{A}_t = \hat{Q}^{\pi}(s_t, a_t) - \hat{V}^{\pi}_{\phi}(s_t) \tag{4}$$

where $\hat{V}_{\phi}$ and $\hat{Q}^{\pi}$ is an estimate of the value function and the action-value function, respectively. In the problem formulation used in this paper, relatively short episodic tasks are considered. This allows us to use the sample return G^{γ}_t from (1) to estimate the value of $\hat{Q}^{\pi}$, same as the approach used in the REINFORCE algorithm [24]. The value function is generally unknown and a function approximator is instead used, parametrized by a weight vector ϕ. The value function is learned simultaneously as the policy, commonly by minimizing a new cost function $L(\phi)$, based on the mean-squared error (or some other loss function) between the true value function $V^{\pi}(s_t)$ and its approximation $\hat{V}^{\pi}_{\phi}(s_t)$:

$$L(\phi) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s_t) - \hat{V}^{\pi}_{\phi}(s_t)\right)^2\right] \tag{5}$$

By computing the gradient of $L(\phi)$ and taking stochastic gradient-descent (or more efficient algorithms based on stochastic gradient-descent such as RMSprop [25]) on batches of data, the parameter vector ϕ that minimizes the loss can be found. By definition, G^{γ}_t is an unbiased estimate of the value function and can thus be used as a target value in the training [23]. The presented approach can be viewed as an actor-critic architecture where the policy π_{θ} is estimated by the *actor* and the value function $\hat{V}^{\pi}_{\phi}$ is estimated by the *critic*. The parameters used in forming the policy (θ) and the value function (ϕ) are in this paper representing the node weights of two separate neural networks, and the goal of training the networks is to find the optimal weights for these networks.

B. Proximal Policy Optimization

Policy gradient methods are generally trained on-policy, which means that they need to sample transitions following the *current* policy. Once the policy has been updated by training on the current batch of transitions, the samples need to be discarded and new sampled. A tempting solution is to either run multiple steps of optimization on the same batch of data or to use a large learning rate. However, doing so is not well-justified, and empirically it often leads to destructively large policy updates [26]. The PPO algorithm, first presented in [26], presents a solution to these problems. In this paper, we use the "clipped" version of the PPO algorithm, with the objective function:

$$J^{clip}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{6}$$

where r_t is a probability ratio given by:

$$r_t = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{7}$$

and ϵ governs the clipping range of the objective function, and θ_{old} refers to the vector of policy parameters used in sampling the transitions and thus before any update of the policy parameters. The clipped objective function ensures that we do not move *too* far away from the current policy, which allows us to run multiple epochs of gradient ascent on the samples without causing destructively large policy updates. The r_t-ratio is always equal to 1 for the first epoch, when current policy $\pi_{\theta}(a_t|s_t)$ is the same as was used to sample the transitions $\pi_{\theta_{\text{old}}}(a_t|s_t)$. For each epoch, the policy is trained to *increase* the probability ratio r_t above 1.0 when the advantage function is *positive*, thus making advantageous actions more probable to be chosen by the policy. Similarly, the policy is trained to decrease the probability ratio r_t *below* 1.0 when the advantage function is *negative*, thus making disadvantageous actions less probable to be chosen by the policy in the future.

C. Adaptations for Continuous-Discrete Control

To handle a hybrid actions space of continuous and discrete actions, we follow the implementation introduced in [27]. The hybrid policy $\pi_{\theta}(\mathbf{a}|s)$ is defined as a state-dependent distribution that jointly models discrete and continuous random variables. Independence between action dimensions, denoted by a^i, is assumed and the hybrid policy can be written as:

$$\begin{aligned}\pi_{\theta}(\mathbf{a}|s) &= \pi^c_{\theta}(\mathbf{a}^{\mathcal{C}}|s)\pi^d_{\theta}(\mathbf{a}^{\mathcal{D}}|s) = \\ &= \prod_{a^i \in \mathbf{a}^{\mathcal{C}}} \pi^c_{\theta}(a^i|s) \prod_{a^i \in \mathbf{a}^{\mathcal{D}}} \pi^d_{\theta}(a^i|s)\end{aligned} \tag{8}$$

where $\mathbf{a}^{\mathcal{C}}$ and $\mathbf{a}^{\mathcal{D}}$ are subsets of action dimensions with continuous and discrete values respectively (where $\mathcal{C}$ and $\mathcal{D}$ represents continuous respectively discrete action spaces), and $\mathbf{a}$ is a vector of both discrete and continuous actions. We represent each component of the continuous policy π^c_{θ} as a normal distribution:

$$\pi^c_{\theta}(a^i|s) = \mathcal{N}\left(\mu_{i,\theta}(s), \sigma^2_{i,\theta}(s)\right) \tag{9}$$

We also represent each component of the discrete policy π^d_{θ} as a Bernoulli distribution parameterized by state-dependent probabilities $P_{\theta}(s)$:

$$\pi^d_{\theta}(a^i|s) = Bernoulli^i\left(\alpha^i_{\theta}(s)\right), \quad \forall i, s\sum_{j=1}^{2} P^i_{j,\theta}(s) = 1 \tag{10}$$

where θ are the parameters of the policy components that we want to optimize. In this paper, we will represent the outputs $\mu_{i,\theta}(s), \sigma^2_{i,\theta}(s)), \pi^d_{\theta}(a|s)$ as outputs from a single branched neural network.

III. Proposed framework for Real-time Control of Secure Operating Limits

In this section, the framework used in training the real-time SOL control algorithm is presented. The algorithm works by continuously monitoring the current state of the system through a set of measurements. If the hybrid DRL agent assesses that the system is not secure, or has a too small margin towards the security boundary, actions are initiated to steer the system to a more secure state. Other types of control, such as maintaining system voltages within acceptable limits, is not included in the developed control, but could theoretically be added as additional control goals.

The hybrid DRL agent is trained on different loading scenarios where the security state of the system is varied to reflect

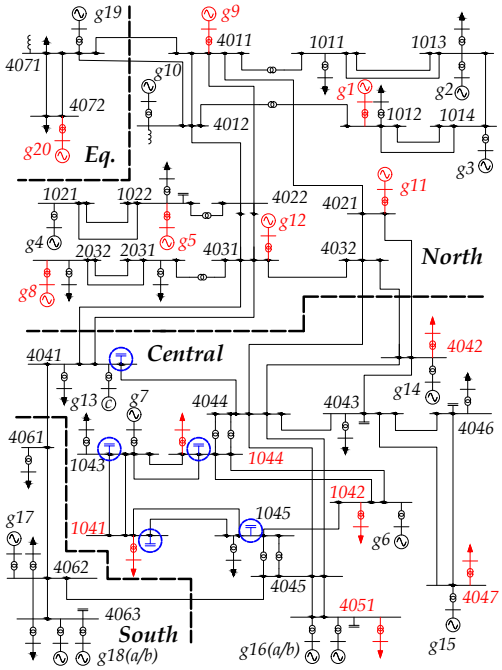

Fig. 1. One-line diagram of the modified Nordic32 system [28]. Loads and generators included in the continuous action space are marked in red, while shunt capacitors participating in the discrete action space are marked in blue.

the different operating conditions that may occur in a real power system. All simulations have been tested on the slightly modified version of the Nordic32 test system, detailed in [28]. The main characteristic of the system is sensitivity towards long-term voltage instability, and we will limit the scope of the stability criteria mainly to this instability phenomenon. A one-line diagram of the test system is presented in Fig. 1. The training data were generated using PSS®E 35.0.0 with its in-built dynamic models [29].

An overview of the interaction between the hybrid DRL agent and the environment is illustrated in Fig. 2. The hybrid DRL agent is based on the PPO algorithm and consists of an *actor* used to form the policy and to choose actions, and a *critic* used to evaluate the taken actions and to estimate the value function. The actor and the critic are developed using two separate neural networks, further discussed in Section III-B. All the steps in the data generation and the training of the networks are detailed in the following sections.

The control problem is defined as an episodic task that ends whenever the SOL in the system is restored above a certain threshold, or until a maximum number of time steps T is reached, whichever comes first. For each training iteration, batches of transitions consisting of $N = 64$ episodes, with a total episode length of a maximum $T = 8$, are generated. Thus, the training is performed on batches consisting of a (maximum) of $NT = 512$ samples, but the samples may be less if the episodes are ended earlier than at $t = 8$. To improve the performance of the training, the learning rate used in optimizing the actor and the critic network varies during training, further specified in Section III-E. To speed up training, we also parallelized the data generation and used multiple CPU cores to generate data. Each parameter used in the training and the generation of data is presented in Table I.

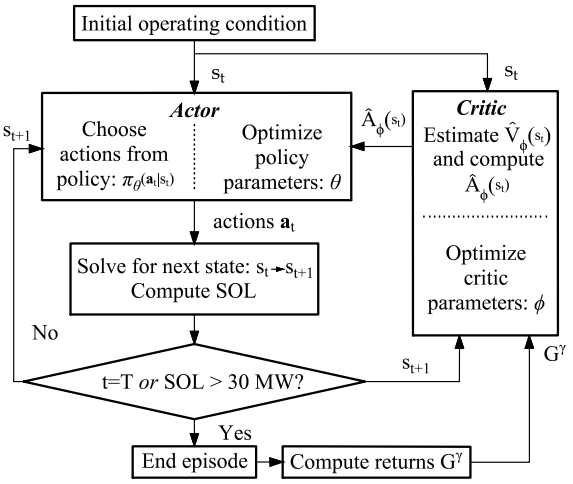

Fig. 2. The generation of training data and the interaction between the actor and critic network and the environment (the power system).

A. Generation of new initial states

A large range of different initial OCs was generated to serve as training data for the algorithm. The initial OCs were generated around the operating point of the simulated modified Nordic32 system denoted as "operating point B" in [28]. All loads in the system were randomly and individually varied by multiplying the active load value with a random variable generated from a uniform distribution (95 % of the original load as lower limit, 105 % of the original load as upper limit). The range was chosen to reflect the different loading levels that may occur in a power system. In real applications, training data could be sampled both from historic OCs and by efficient database generation schemes [30]. The power factors of all loads were kept constant. The total change in loading was *partly* randomly distributed among all the generators in the system and was based on a combination of a uniform distribution and the initial active power produced by each generator. Thus, a generator with a higher rated capacity had a higher probability of compensating for a larger share of the increased load.

To further stress the initial OCs and to provide a wider range of different operating states, a single random topology change could also be introduced. With the same probability for each scenario, either a line was disconnected between buses (i) *4032-4044*, (ii) *4032-4042*, (iii) *4031-4041*, or (iv) no change was introduced. The disconnected lines were introduced to model the impact that possible topology changes might have on the training of the hybrid DRL model. In actual implementations, all possible topology changes and components (e.g. generators) that could be taken out of service and be imperative for the system stability should be included in the generation of the training data.

The generated initial OCs were first solved by a full Newton-Raphson load flow but were re-initialized in case the load flow did not converge. After a solved load flow solution, a state vector s_t consisting of measurements of:

1) Bus voltage magnitudes of all buses in the system.
2) Active and reactive power flows on all transmission lines and transformers.

TABLE I
DESIGN AND HYPERPARAMETERS USED IN TRAINING

		Parameter	Values
Architecture	*Critic*	Number of inputs	499
		Neurons in first layer	128
		Neurons in second hidden layer	64
		Final activation function	Linear
		Hidden layer activation	RelU
	Actor	Number of inputs	499
		Neurons in common hidden layer	128
		Neurons in each separate hidden layer	64
		Final activation for μ_{cont}	Linear
		Final activation for σ_{cont}	Softplus
		Final activation for $P(D_x\|s)$	Sigmoid
		Hidden layer activation	RelU
	Training	Maximum episode length (T)	8
		Max Epochs (K)	10
		PPO clip parameter (ϵ)	0.2
		Discount factor (γ)	0.99
		Optimizer	RMSprop [25]

The current time step of the episode was also added to the state vector. The state vector was then normalized by subtracting the mean value of each state value and then dividing by its standard deviation. The mean and standard deviation of each state value was computed from previously sampled states and a double-ended list with a maximum of 10 000 sampled states was stored. As more and more iterations were performed, the double-ended list was filled up and old sampled states were discarded.

B. Sampling of actions and estimation of value function

The sampled state s_t was then passed to the actor network, illustrated in Fig. 3 and further detailed in Table I. The actor network shares a common hidden layer, then a separate hidden layer is used for each type of activation function. The network outputs parameters used in defining the Normal distribution (9) and the Bernoulli distributions (10) that are used for the continuous and discrete action variables, respectively. The Normal distribution is parametrized by a mean value μ_{cont} and a standard deviation σ_{cont}, while the Bernoulli distribution is parametrized by a single probability parameter ranging from 0-1. The mean value μ_{cont} is computed using a linear activation function in the final layer, the standard deviation σ_{cont} is computed using a softplus activation function that ensures that the value never becomes negative, while the Bernoulli distribution is achieved using a sigmoid activation function in the final layer that ensures that the output is bounded between 0 and 1. Actions were then sampled from the defined distributions.

The sampled continuous action value was used to reschedule power generation and perform load curtailment to reduce the transfer of power through the system. This was achieved by reducing the consumption in the "Central" area at certain participating load buses: *4042*, *4047*, *4051*, *1041*, *1042*, *1044*, while simultaneously increasing the generation in the "North" area at certain participating generators: *G1*, *G5*, *G8*, *G9*, *G11*, *G12*, *G20*. The sampled continuous action value controlled how much in total the *active* power should be reduced in the system for all

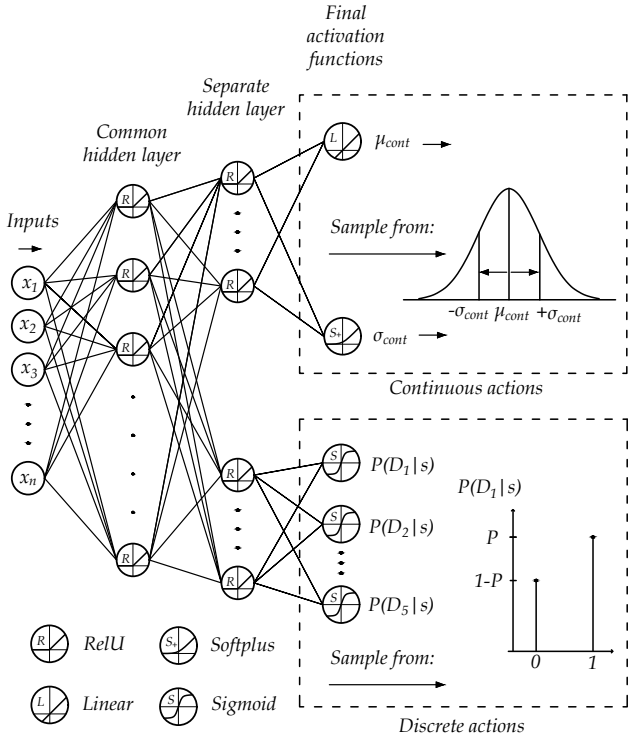

Fig. 3. Architecture of the actor network.

participating load buses. The load decrease was distributed on each of the participating load buses based on their initial load *before* the change, while the power factors of all loads were kept constant. The discrete action variables included the switching of an additional 100 MVAr of reactive power support from any, or several, shunt capacitors. The discrete actions (D_1-D_5) controlled the shunt capacitors at the following buses: D_1: *1041*, D_2: *1043*, D_3: *1044*, D_4: *1045*, D_5: *4041*. The participating buses and equipment for the continuous and the discrete actions are all marked in red respectively blue, in the line diagram in Fig. 1.

The sampled state s_t was also passed to the critic network which estimated the value function $\hat{V}^{\pi}_{\phi}(s_t)$ of the current state. The critic network is separate from the actor network and consists of a NN with two hidden layers and a linear final activation function, further detailed in Table I.

C. Taking steps and computing the SOL

The sampled actions were then taken in the system and a new load flow solution was computed, resulting in a new state s_{t+1}. The SOL for the s_{t+1} state was then computed. Estimating the SOL is relatively complex and involves checking the system's capacity of maintaining stability after different contingencies and for various levels of system stress. To reduce the number of required simulations, the system's stability was only assessed with respect to a single dimensioning fault, namely the disconnection of generator *G14*. In real applications and if the developed method is to be used for larger systems, all possible contingencies that can be dimensioning for the security margin of the system should be evaluated.

The SOL for the next state s_{t+1} was computed using a dual-binary search similar to the one presented in [31]. The process

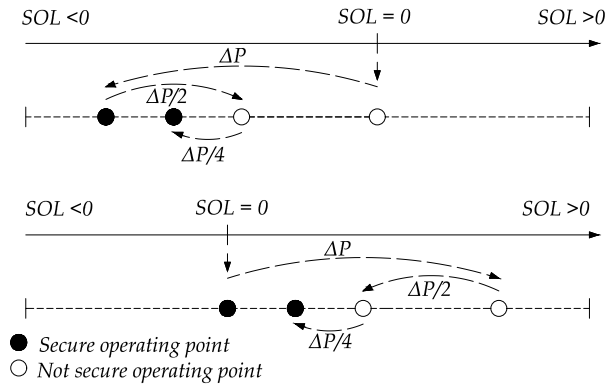

Fig. 4. Illustration of the dual binary search for SOL for a secure and a non-secure initial operating point.

is illustrated in Fig. 4 and is based on searching through a narrowing interval by iteratively testing the system security with respect to a dimensioning fault and different levels of system stress. A dynamic simulation was first initialized followed by a disturbance to test whether the s_{t+1} state was secure or not. The upper part in Fig. 4 illustrates the search process when the state was not secure, while the lower part in Fig. 4 illustrates the search process when the state was secure. Black dots indicate a secure state, while white dots indicate a state that is not secure.

We exemplify the search process for a secure starting state. If the starting state was found secure, the system stress for state s_{t+1} was increased by increasing the loads in the "Central" area with a total of ΔP=128 MW, while simultaneously adjusting the generation in the "North" area with the same amount. The power factors of all loads were kept at their initial values and the distribution of the added load and generation were again based on the initial load or the rated capacity of each generator. After the loads and the generation were updated, the load flow was reiterated which then served as a starting point for the new dynamic simulations. If the new state was found to be secure, the system stress was again increased with ΔP. In case it was not found to be not secure, the system stress was reduced by $\Delta P/2$. The dual binary search then continued until a secure operating point was found and when the step size in system stress change was equal to 1 MW.

Each dynamic simulation ran for a maximum of 800 seconds but was stopped in advance if the case either collapsed (any bus voltage below 0.7 pu) or if the system stabilized early. The system was considered secure if, at the end of each simulation, *all* transmission bus voltages were above 0.90 pu. If the SOL was above 30 MW, the episode was ended. If not, the new state (s_{t+1}) then served to sample new actions and estimate new values of the value function.

D. Computing rewards and advantage estimates

Once all episodes had been generated, the returns and the advantage estimates were computed. The reward R_t for the taken actions was computed by a combination of the resulting SOL and the costs for the continuous (C_{cont}) and the discrete (C_{disc}) actions, respectively. In this study, the reward is unitless, but should in real applications reflect the actual monetary cost of different actions and the corresponding rewards when the control goal is either achieved or missed. *Any* activation of the discrete actions contributed to a negative reward of -3, representing the cost of the mechanical wear that is involved in switching the shunt capacitors. The cost for the continuous actions contributed to a negative reward of -0.1 per adjusted MW in the power transfer. Changing a total of $\pm$ 200 MW would thus result in a negative contribution to the reward of -20. This negative reward reflects the system cost of market adjustments or load/generation curtailment. The control goal of the hybrid DRL agent is to always restore the SOL to a value equal or above 30 MW, which would ensure that a sufficient security margin is achieved and that the N-1 contingency criterion always would be satisfied with some margin. If the SOL was below 30 MW, a negative reward of -50 was added to the total reward for that time step. The final reward when accounting for the resulting SOL was then computed as:

$$R_t = \begin{cases} C_{cont} + C_{disc} + \text{SOL} - 50, & \text{if SOL} \leq 30 \text{ MW} \\ C_{cont} + C_{disc}, & \text{otherwise} \end{cases} \tag{11}$$

The returns G_t^γ were estimated using (1) and the advantage estimates $\hat{A}_t$ were estimated using (4) and the estimated value of the state $\hat{V}_\phi^\pi$ given by the critic network. A discounting factor γ of 0.99 was used to compute the returns. All sampled data and transitions were then stored in a buffer which is later used in training the actor and the critic networks.

E. Training actor and critic networks

Once the full N episodes were sampled, the actor and the critic networks were trained. The training was performed using the software Tensorflow in Python which automatically computes the gradients on the defined cost functions. The actor objective J^{clip} from (6) was augmented by adding an entropy bonus to incentivize exploration of the policy [26]. The entropy $S(\pi_\theta)$ for each Normal and Bernoulli distribution defined by the hybrid policy $\pi_\theta(\mathbf{a}|s)$ was computed and added as an entropy bonus J^{ent} to the final actor objective function. A coefficient β^{ent} was multiplied with the entropy bonus term to adjust the weights in relation to the J^{clip} losses. The final objective function for the actor network was computed by taking the mean value of all samples for all N episodes on the sum $J^{clip} + \beta^{ent} J^{ent}$. The critic objective function was defined as the mean squared error (MSE) of the value function error, computed from (5). The final critic objective function was formed by taking the mean value of all samples for all N episodes on the MSE of the value function error.

Once the actor and critic objectives were formed, they were optimized using the RMSprop algorithm, which is an adaptable algorithm suitable for gradient-based optimization of stochastic objective functions. The actor objective function was maximized with respect to θ, while the critic network objective function was minimized with respect to ϕ. The training was performed for $K = 10$ epochs on the whole batch of N episodes simultaneously. It should be mentioned that although a search of suitable hyperparameters was conducted, the performance could have been improved even more by

TABLE II
LEARNING RATES AND BATCH SIZES OVER ITERATIONS

Iteration	α_{actor}	α_{critic}	β^{ent}
≤ 200	$1 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	0.01
≤ 400	$3 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	0.0
> 400	$1 \cdot 10^{-5}$	$3 \cdot 10^{-5}$	0.0

further optimizing training parameters such as the learning rate or the number of hidden neurons in each layer. To speed up training and to stabilize it during later stages, the learning rate and the β^{ent} coefficient were adjusted as the number of training iterations increased. The values of the learning rates and the used β^{ent} coefficient over training iterations are specified in Table II.

IV. RESULTS AND DISCUSSION

A. Training results

The hybrid DRL agent was trained for a total of 1 000 iterations, corresponding to a total of 64 000 episodes, and approximately 94 000 individual samples. The training performance over episodes is presented in Fig. 5. The total episode reward is presented in sub-figure (i). In sub-figure (ii), the episode reward zoomed in at the final performance is presented. The final SOL and the number of episode time steps (length) are presented in sub-figures (iii) and (iv), respectively. The red line shows a centered moving average computed over the mean value over 5 000 episodes. To better visualize the results, only every *100th* value during the training is illustrated in the figure. The results show that the performance improved rapidly until around 1 000 episodes, after which the policy managed to achieve a SOL above the threshold value of 30 MW using only a single time step for a majority of the episodes. In sub-figure (ii), it can be observed that the performance continued to improve over time after 1 000 episodes, but at a much slower rate. The main increase in performance after 1 000 episodes is mainly a result of the decreasing exploration rate of the policy.

In Fig. 6, the development of the different policy parameters is presented. The policy parameter governing the standard deviation σ_{cont} increased at first, which was then followed by the mean value μ_{cont} being adjusted. After the model was trained on approximately 40 000 samples, the mean value μ_{cont} stabilized. After that, the model improved mainly by reducing its exploration rate (the standard deviation σ_{cont} and the randomness of the discrete actions). In sub-figure (iii) of Fig. 6, the probability of the taken action D_3 is illustrated, showing that the policy became more and more certain of whether the discrete action should be activated or kept inactivated. Similar training development for the other discrete actions was observed as well.

B. Testing the hybrid DRL agent

During training, the actor used a stochastic policy which allowed it to automatically explore the available action space. While the exploration rate (the standard deviation of the continuous action space and the randomness of the discrete action spaces) of the agent decreased during the final part of the training, it would require a significantly longer training time for it to reach a point where it essentially converged towards a *fully* deterministic policy. When implementing the policy online it is generally more suitable to transform the control policy into a deterministic one and always pick the actions that with the *highest probability* are optimal. When testing the algorithm, the continuous action was thus controlled directly by the mean value μ_{cont}. Each of the discrete actions was activated whenever any of the defined Bernoulli probabilities satisfied $P(D_i|s) \geq 0.5$. The hybrid DRL agent was tested on three different test sets of data, each detailed below.

1) **Test set 1**: Data generated in the same way as for the training data, but using a deterministic policy instead.
2) **Test set 2**: Introducing new unseen OCs by increasing the variation of the generation and load configurations.

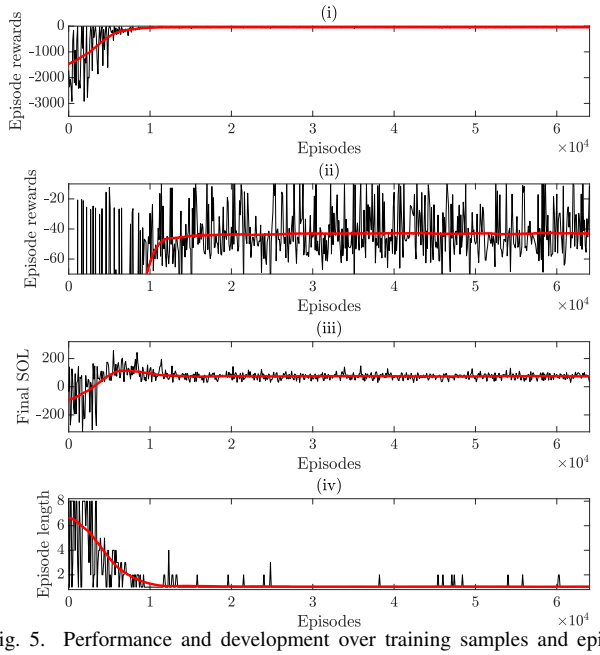

Fig. 5. Performance and development over training samples and episodes: Sub-figures showing (i) episode rewards during all episodes; (ii) episode rewards zoomed in at final performance; (iii) final SOL; (iv) episode length. The red line indicates a moving average computed over the mean of 64 data points. For better visualization, every *100th* value is illustrated.

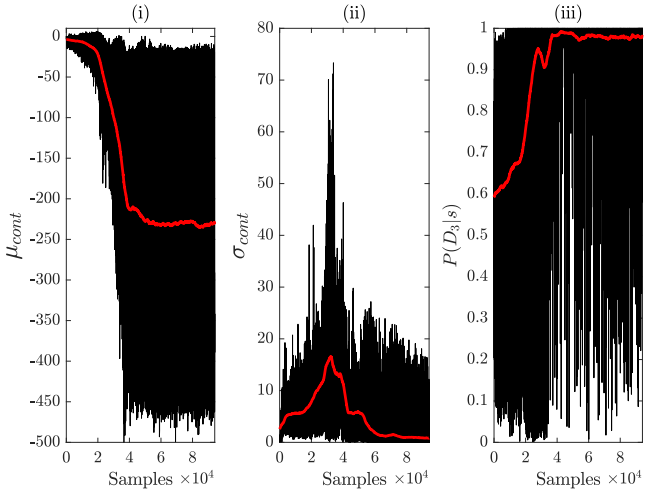

Fig. 6. Development of policy parameters over episodes. Sub-figures showing (i) the mean value output μ_{cont}; (ii) the standard deviation output σ_{cont}; (iii) the discrete probability of taking $P(D_3|s)$.

TABLE III
AVERAGE PERFORMANCE OF THE HYBRID DRL AGENT

	Reward	Final SOL [MW]	Episode length [steps]
Test set 1	-42.0	72.4	1.04
Test set 2	-40.0	78.7	1.08
Test set 3	-67.0	59.4	1.65

Instead of randomly adjusting each load between 95 % to 105 % as specified in Section III-A, the OCs were adjusted randomly between 90 % to 110 %. The deterministic policy was used to choose actions.

3) **Test set 3**: Introducing random measurement errors by multiplying each of the state values with a random number with a mean of 1 and a standard deviation of 0.005. The deterministic policy was used to choose actions.

The performance of the hybrid DRL agent when tested on the different test sets is presented in Fig. 7 and Table III, each consisting of 100 episodes. For test set 1, the average episode length was 1.04 steps, indicating that the hybrid DRL agent managed to ensure a sufficient security margin in a single time step for almost all different scenarios. The average final SOL was 74.2 MW, which is relatively close to the control goal of ensuring a final SOL above 30 MW, indicating that a close to minimal amount of actions have been activated to achieve the control goal. It should further be noted that this value included scenarios that were secure from the start as well. The average episode reward was -42.0.

For test set 2, where new unseen OCs were introduced by increasing the variation by which the loads and generation were initialized, showed good performance as well. The average episode length increased slightly to 1.08 steps, indicating that some of the unseen OCs forced the hybrid DRL agent to require a few more steps before the SOL was restored. The average final SOL is 78.7 MW, which again is relatively close to the control goal of restoring the SOL above 30 MW. The average episode reward ended up at -40. It should be noted that this value cannot directly be compared to the one for test set 1, as the system now is tested on scenarios with a larger variation of the load and generation configurations.

For test set 3, where the impact of random measurement errors on the state values was evaluated, the performance of the DRL agent dropped significantly. The average reward reduced to -67.0, while the average episode length increased to 1.65 steps. Thus, for several of the scenarios in the test set, the hybrid DRL agent required multiple control steps before the SOL was restored above the threshold. This can also be observed in Fig. 7. The results show the importance of incorporating random errors that exist in real power systems, but which are generally not present when training the method on purely simulated data. Thus, it is important to include such random errors during the training phase, which would force the DRL agent to become more robust with respect to random smaller perturbations in the state signal.

C. Comparison to a fixed look-up table

Conventional methods for preventive control typically rely on system operators to choose suitable control actions by matching the current system state with the nearest system state defined in a preventive control look-up table. The difficulty of assessing raw measurements (commonly filtered through a state estimator) from the system, require system operators to pre-process the system state, generally by first computing the security margin and then, in case the security margin is below a defined threshold, take measures to restore it. Thus, the system operator has to (i) first estimate the security margin, (ii) possibly activate actions to restore it, and finally (iii) re-evaluate the security margin to ensure that it is above the defined threshold. In comparison, the developed hybrid DRL agent can take actions by *directly* monitoring the system state, without the need for the additional first step (i) with time-consuming data pre-processing and computation of the SOL.

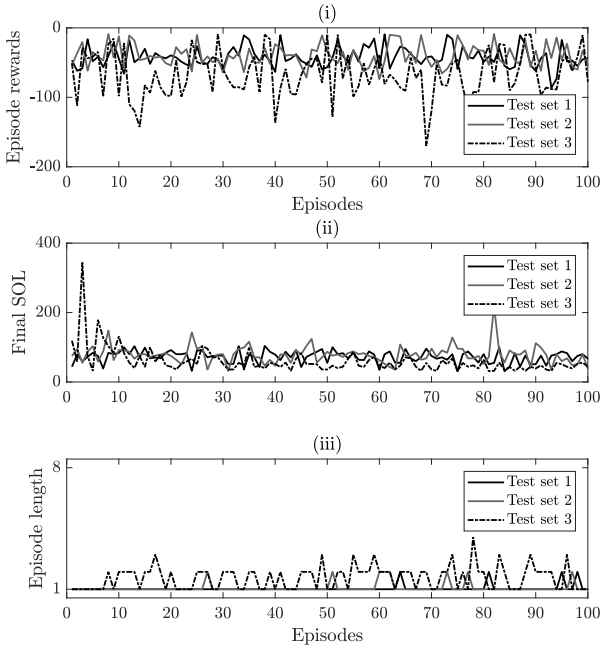

Fig. 7. Performance when assessing the hybrid DRL agent on different test sets and when transforming into a deterministic policy. Sub-figures showing (i) episode rewards; (ii) final SOL; (iii) episode length. Mean values of the performance indices are presented in Table III.

In this section, we compare the performance of the developed hybrid DRL agent to that of a fixed look-up table with different actions for different levels of pre-computed SOLs. Since the fixed look-up table require an initial computation of the SOL before any actions can be initiated, a penalty is added to be able to compare its performance to that of the hybrid DRL agent. The pre-processing of measurement data and the initial computation of the SOL corresponds to an additional episode step for the hybrid DRL agent. Thus, a penalty of -50 (the penalty added for the DRL agent at every time step it does not achieve its control goal) and an added time step for each episode, are added to the performance of the fixed look-up table. The actions available for the fixed look-up table are based on the range of actions that were found suitable after training the hybrid DRL agent, see Fig. 6. The same costs for each of the actions as for the hybrid DRL agent were used. The actions were designed to *always* ensure that the SOL was restored above the threshold of 30 MW using a single set of actions. The different actions of the fixed look-up table that

TABLE IV
ACTIVATION LEVELS FOR THE FIXED LOOK-UP TABLE CONTROL SCHEME

Initially estimated SOL	Actions activated Continuous	Discrete
SOL > 30 MW	0 MW	None
-50 MW< SOL $\leq$ 30 MW	-50 MW	D_1
-100 MW< SOL ≤ -50 MW	-100 MW	$D_1 + D_2$
-50 MW< SOL ≤ -100 MW	-150 MW	$D_1 + D_2$
-200 MW< SOL ≤ -100 MW	-200 MW	$D_1 + D_2 + D_3$
-300 MW< SOL ≤ -200 MW	-350 MW	$D_1 + D_2 + D_3$
SOL < -300 MW	-450 MW	All

TABLE V
AVERAGE PERFORMANCE WHEN USING A FIXED LOOK-UP TABLE CONTROL SCHEME. VALUES IN PARENTHESIS PRESENTS THE PERCENTAGE INCREASE TO THE AVERAGE PERFORMANCE OF THE HYBRID DRL AGENT

	Reward	Final SOL [MW]	Episode length [steps]
Test set 1	-105.2 (150 %)	198.7 (174 %)	2.0 (92 %)
Test set 2	-99.8 (149.5 %)	197.0 (150 %)	2.0 (85 %)
Test set 3	-105.2 (57 %)	198.7 (235 %)	2.0 (21 %)

were taken for the different levels of the initially estimated SOLs are presented in Table IV.

In Table V, the performance when using the defined fixed look-up table control on the three different test sets that were defined in Section IV-B, is presented. For each metric and test set, the percentage difference in performance compared to the hybrid DRL agent is also presented in parenthesis after each value. The results show that the proposed hybrid DRL agent performed significantly better on all of the defined test sets. The (negative) average reward increased from 57 % for test set 3, up to 150 % for test set 1. Similarly, the average episode length increased with 21 % for test set 3, up to 92 % for test set 1. While it should be noted that the steps defined in the look-up table control scheme were chosen somewhat arbitrarily, it also reflects the typical difficulty that system operators face when defining such control rules in actual operation.

D. Practical aspects and requirements

The developed hybrid DRL agent is proposed to be used as an online tool for system operators to control and ensure a sufficient SOL in real-time. While the DRL agent can be applied online to automatically activate actions, it can also be used to suggest actions that system operators after evaluation can manually choose to activate. Theoretically, it could also be used as a method that could complement how system operators today develop their preventive control look-up tables. Instead of manually assessing the effectiveness of different actions on a few typical system scenarios, the DRL framework could be used to more efficiently evaluate what range of actions that will be efficient for a larger range of different scenarios and automate the evaluation process.

The inability to anticipate the behavior of neural networks and in extension DRL methods is a major barrier in their application in safety-critical systems, such as electric power systems [32]. While neural networks might have high prediction accuracy on unseen test data, they can be highly vulnerable to so-called adversarial examples, where small input perturbations may lead to poor performance. For instance, the results of the hybrid DRL agent in test set 3 illustrated the impact that relatively small changes in the input data can have on the performance. A possibility to strengthen the belief of system operators in DRL control is to use verification methods of the neural network behavior, such as the methods proposed in [32], or by adopting methods that can provide estimates of the uncertainty of the agent's decisions [33]. By using such verification methods and/or methods to estimate the actions' uncertainty, system operators could have guarantees on when the DRL agent is reliable to use and when conventional control systems should be used instead. It should be stressed that preventive control is fundamentally different from emergency control in that the system will generally be stable even if the taken actions do not make the system secure instantly. Thus, although it is desirable to as fast as possible achieve a N-1 secure state, it will not have the same severe impact on the system as a wrong set of actions might have during an emergency event.

Finally, the time consumption during data generation and the high requirement of computational power should be addressed. The training of the hybrid DRL agent is time-consuming, both due to the large requirement of training data and the fact that training scenarios were generated using full dynamic simulations. The parallelization of the data generation on different CPU cores *significantly* increased the efficiency by which the data was generated. Furthermore, the use of the PPO algorithm, which allowed multiple epochs of training on the same batch data, further increased the efficiency of the training. A possibility to further speed up the generation of training data is to compute the SOL based on quasi-steady-state (QSS) simulations or combinations of QSS and dynamic simulations, as suggested in [1] or [34]. While QSS methods, or combinations of QSS and dynamic simulations, do not provide as accurate estimations of the SOL as if it had been computed using a full dynamic simulation, it is still significantly more accurate than most methods today that are based on static estimations of the security margin.

V. CONCLUSION

This paper introduces a new method for optimal control to maintain a sufficient SOL in real-time. The SOL is a security margin that is based on evaluating the dynamic performance of the system in a stressed operating point and provides a more accurate estimate of the security margin than conventional methods that are based on static assessments. The optimal control method is based on DRL and introduces a hybrid control scheme that can simultaneously control both discrete (e.g. switching of a shunt capacitor) and continuous (e.g. the level of active power generation rescheduling and load curtailment) action variables to ensure that the SOL is above defined threshold values.

The method showed good performance on the developed test sets and managed to control the SOL above the defined threshold values for a single time step for a majority of the scenarios. When tested on disturbance scenarios and OCs that were not included in the training, the method's performance dropped, which highlights the need for an efficient database

generation and the need for methods to verify the accuracy of the DRL agents policy. The method was further compared to the performance of a fixed control scheme based on look-up tables and proved to provide a more efficient control for all of the defined test sets.

REFERENCES

[1] T. Van Cutsem, C. Moisse, and R. Mailhot, "Determination of secure operating limits with respect to voltage collapse," *IEEE Transactions on Power Systems*, vol. 14, no. 1, pp. 327–335, Feb 1999.

[2] F. Capitanescu and T. Van Cutsem, "Preventive control of voltage security margins: a multicontingency sensitivity-based approach," *IEEE Trans. Power Syst.*, vol. 17, no. 2, pp. 358–364, 2002.

[3] M. Zima and G. Andersson, "Model predictive real-time control of electric power systems under emergency conditions," in *Real-Time Stability in Power Systems*, S. Savulescu, Ed. Springer, 2004, pp. 367–385.

[4] Q. Huang *et al.*, "Adaptive power system emergency control using deep reinforcement learning," *IEEE Trans. Smart Grid*, vol. 11, no. 2, pp. 1171–1182, 2020.

[5] H. Cai, H. Ma, and D. J. Hill, "A data-based learning and control method for long-term voltage stability," *IEEE Trans. Power Syst.*, vol. 35, no. 4, pp. 3203–3212, 2020.

[6] Q. Li, Y. Xu, and C. Ren, "A hierarchical data-driven method for event-based load shedding against fault-induced delayed voltage recovery in power systems," *IEEE Trans. Industrial Informatics*, vol. 17, no. 1, pp. 699–709, 2021.

[7] D. Ernst, M. Glavic, and L. Wehenkel, "Power systems stability control: reinforcement learning framework," *IEEE Trans. Power Syst.*, vol. 19, no. 1, pp. 427–435, 2004.

[8] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020.

[9] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529–533, 2015.

[10] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[11] A. E. Sallab *et al.*, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[12] S. Gu *et al.*, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3389–3396.

[13] M. Glavic, "(deep) reinforcement learning for electric power system control and related problems: A short review and perspectives," *Annual Reviews in Control*, vol. 48, pp. 22–35, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578819301014

[14] R. Diao *et al.*, "Autonomous voltage control for grid operation using deep reinforcement learning," in *2019 IEEE Power Energy Society General Meeting (PESGM)*, 2019, pp. 1–5.

[15] S. Wang *et al.*, "A data-driven multi-agent autonomous voltage control framework using deep reinforcement learning," *IEEE Trans. Power Syst.*, vol. 35, no. 6, pp. 4644–4654, 2020.

[16] B. L. Thayer and T. J. Overbye, "Deep reinforcement learning for electric transmission voltage control," in *2020 IEEE Electric Power and Energy Conference (EPEC)*, 2020, pp. 1–8.

[17] J.-F. Toubeau *et al.*, "Deep reinforcement learning-based voltage control to deal with model uncertainties in distribution networks," *Energies*, vol. 13, no. 15, p. 3928, 2020.

[18] J. Zhang *et al.*, "Deep reinforcement learning for short-term voltage control by dynamic load shedding in China southem power grid," in *2018 International Joint Conference on Neural Networks*, 2018.

[19] C. X. Jiang *et al.*, "Power system emergency control to improve short-term voltage stability using deep reinforcement learning algorithm," in *2019 IEEE 3rd International Electrical and Energy Conference (CIEEC)*, 2019, pp. 1872–1877.

[20] Y. Hashmy *et al.*, "Wide-area measurement system-based low frequency oscillation damping control through reinforcement learning," *IEEE Trans. Smart Grid*, vol. 11, no. 6, pp. 5072–5083, 2020.

[21] S. Zarrabian, R. Belkacemi, and A. A. Babalola, "Reinforcement learning approach for congestion management and cascading failure prevention with experimental application," *Electric Power Systems Research*, vol. 141, pp. 179–190, 2016.

[22] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[25] Tensorflow Keras Optimizers: RMSprop. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop

[26] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] M. Neunert *et al.*, "Continuous-discrete reinforcement learning for hybrid control in robotics," in *Conference on Robot Learning*. PMLR, 2020, pp. 735–751.

[28] T. Van Cutsem *et al.*, "Test systems for voltage stability studies," *IEEE Trans. Power Syst.*, vol. 35, no. 5, pp. 4078–4087, 2020.

[29] *PSS®E 35.0.0 Model Library*, Siemens Power Technologies International, Schenectady, NY, Apr. 2019.

[30] F. Thams *et al.*, "Efficient database generation for data-driven security assessment of power systems," *IEEE Transactions on Power Systems*, pp. 1–1, 2019.

[31] H. Hagmar, R. Eriksson, and L. Tuan, "Fast dynamic voltage security margin estimation: Concept and development," *IET Smart Grid*, vol. 3, 04 2020.

[32] A. Venzke and S. Chatzivasileiadis, "Verification of neural network behaviour: Formal guarantees for power system applications," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 383–397, 2021.

[33] C.-J. Hoel, K. Wolff, and L. Laine, "Ensemble quantile networks: Uncertainty-aware reinforcement learning with applications in autonomous driving," *arXiv preprint arXiv:2105.10266*, 2021.

[34] T. Van Cutsem, M.-E. Grenier, and D. Lefebvre, "Combined detailed and quasi steady-state time simulations for large-disturbance analysis," *International Journal of Electrical Power & Energy Systems*, vol. 28, no. 9, pp. 634 – 642, 2006.

Hannes Hagmar (Student member, IEEE) received the M.Sc. degree in electric power engineering from Chalmers University of Technology, Gothenburg, Sweden in 2016. Between 2016 to 2017, he worked at RISE Research Institutes of Sweden with research in electric transmission systems and measurement technology. He is currently pursuing a Ph.D. degree at Chalmers University of Technology. His research interest includes power system dynamics and stability, integration of renewables, and machine learning.

Robert Eriksson (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2005 and 2011, respectively. From 2013 to 2015, he held a position as an Associate Professor with Center for Electric Power and Energy, Technical University of Denmark - DTU, Kongens Lyngby, Denmark. He is currently with the Swedish National Grid, Department of Power Systems. Since 2020, he has been an Adjunct Professor with the KTH Royal Institute of Technology. His current research interests include power system dynamics and stability, automatic control, HVDC systems, and DC grids.

Le Anh Tuan (Member, IEEE) received the BSc degree in power systems from Hanoi University of Technology, Vietnam in 1995, the MSc degree in energy economics from the Asian Institute of Technology, Thailand in 1997, and the PhD degree in power systems from the Chalmers University of Technology, Sweden in 2004. He is currently an Associate Professor at the Department of Electrical Engineering, Chalmers University of Technology. His current research interests include power system operation and planning, power market and deregulation issues, grid integration of renewable energy and electric vehicles.