POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Scalable Power Network Control with Reinforcement Learning

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Daniele Paletti, 10572260

Advisor:
Prof. Marcello Restelli

Co-advisors:
Alberto Maria Metelli

Academic year:
2021-2022

Abstract: Climate is undergoing a severe crisis. Ecosystems are collapsing, global temperature is rising, and extreme atmospheric phenomena are more frequent than ever. Renewable and low-carbon forms of energy represent the only way out of the climate crisis. Solar and wind generators have unpredictable energy generation patterns as their production depends on the weather. Integrating such generators with the power grid poses significant challenges. Artificial intelligence (AI) may help grid operators tackle these new challenges.
To foster new AI solutions, RTE (Réseau de Transport d'Electricité) has been organizing the "Learning To Run a Power Network" (L2RPN) challenge. The power network control problem is cast in the Reinforcement Learning (RL) framework. An RL agent observes the state of the power network and takes action based on the current observation and previous experience to maximize the cumulative sum of a scalar reward signal.
Our solution is a novel hierarchical multi-agent RL model. A set of agents deals with local neighborhoods of the network, a small number of managers filters their decision, and a head manager selects one of the managers' decisions. The hierarchy is dynamically generated given the grid topology and updated while learning.
We test our solution on two increasingly complex networks and show performance superior to a challenging expert system. Our results demonstrate the feasibility of multi-agent power network control bridging the general power network literature and the L2RPN challenge community.

Key-words: Power Networks, Reinforcement Learning, Multi-Agent Systems, Hierarchical Decision-Making, Community Detection, Graph Neural Networks

1. Introduction

1.1. Motivation

Power grids enable energy transportation from production plants to consumers through thousands of kilometers of transmission lines and the work of numerous people. In 2020, almost 85% of the energy produced came from the combustion of fossil fuels that emit greenhouse gases [2]. Those emissions have been consistently growing. To prevent the irreversible destruction of many ecosystems reducing greenhouse gas emissions has become urgent. Alternative energy conversion devices that exploit renewable and low-carbon forms of energy are becoming ubiquitous. Such devices have drastically improved over the past two decades. A growing amount of research investigates the feasibility of a 100% renewable energy system [8].
However, solar and wind power have some drawbacks regarding their integration into power networks. Their production highly depends on the weather which is hard to predict accurately [10]. Energy storage facilities allow us to tackle such unpredictability, but our overall storage capacity is low.
As of today, increasingly uncertain power injection patterns and the inherent complexity of power networks make the control problem harder than ever. Highly trained engineers called dispatchers ensure the system's security by performing several actions:

- **switching**: managing power overflows and preventing cascading failures by changing interconnection patterns of transmission lines to redirect power flows;
- **redispatching**: asking producers or consumers to change what they inject into the power network;
- **storing**: modifying the amount of power produced or absorbed by storage units;
- **curtailing**: limiting the amount of energy injected by renewable generators in case of overproduction or local issues.

Dispatchers rely on their understanding of the system because current optimization methods are lagging, and satisfying heuristics are still under test. In the future, we hope artificial intelligence (AI) can assist dispatchers in making better decisions to control the power network and keep all equipment secure.

1.2. Goals

Since 2019, RTE (Réeseau de Transport d'Electricité) has been organizing the "Learning to Run a Power Network" (L2RPN) Challenge [45] to foster AI applications to power network control. RTE provides the physical simulation of a power grid, and participants build algorithms to safely and efficiently control such grid. Power network control in the L2RPN challenges is framed as a sequential decision-making problem and cast in the Reinforcement Learning (RL) framework. RL is a machine learning (ML) paradigm that deals with learning an optimal behavior (i.e., policy) in a given environment by maximizing the cumulated sum of a scalar reward signal. An RL agent observes the current state of the network and decides what action to take, such action changes the environment's state, and the loop repeats. The actions allowed to the autonomous agent are akin to those available to a human dispatcher: switching, redispatching, and curtailing. The control problem ends when the total demand is not met anymore, i.e., a blackout is triggered.
The first edition of the L2RPN challenge took place in 2019. The provided network was composed of 20 powerlines, 11 loads, and 5 generators. In 2020, the power network contained 59 powerlines, 37 loads, and 22 generators. In 2021, organizers added an alarm feature: the agent alarms the operator if an emergency is predicted up to 30 minutes in advance. Overall, the complexity of the control task keeps rising each year, and the winning solution becomes more articulated.
However, scientific challenges foster competition rather than collaboration. Challengers often build instance-optimized throw-away models to maximize their score and win the challenge. Such behavior hurts scientific progress. Building a brute-force model whose sole purpose is achieving the top score in a specific environment is certainly not a positive route to solve real-world problems.
Conversely, challenges offer standardized and reproducible benchmarking platforms. Practitioners can evaluate the performance of their models on publicly available benchmarks fostering more open and inclusive research. Reproducibility is a crucial issue in AI research [18], results are often difficult to replicate, and models are hardly comparable. Scientific challenges are a step in the right direction in alleviating the AI reproducibility crisis.
Therefore, we decided to take the best of both worlds by evaluating our model on the challenge platform without taking part to the competition. This way, we stress the relevance of the task proposed in the challenge and give us sufficient time to devise a solution free of the competition peculiarities.

1.3. Contributions

We created a hierarchical multi-agent RL system. Power networks contain numerous nodes, and the challenges model far smaller environments than real-world grids. We deployed one RL agent per node. On top of this, we decided not to have a purely horizontal decision structure as the complexity of consensus scales with the number of agents trying to reach an agreement [1]. We built a hierarchical decision structure. Managers handle communities of agents and select the best decision among the ones proposed by each agent. Finally, a head manager collects managers' proposals and decides on the one to enact. Both managers and the head manager are RL agents.
In the results section, we evaluate our model on two environments of increasing size and complexity. We show that the model is capable of performing substantially better than the baseline expert system proposed by challenge organizers in both settings.
We argue that multi-agent power network control is feasible and effective. Multi-agent systems allow controlling huge environments by naturally factorizing them, thus promising real-world scalability.
Our main contributions are:

- a novel hierarchical multi-agent RL architecture for power network control;
- an empirical evaluation of the proposed model on two challenge environments.

In the following sections, we detail the prerequisite knowledge, the problem setting, the literature on RL applications to power networks, previous challenge winners, the proposed model, the empirical model evaluation, and some possible extensions to this work.

2. Background

The nature of learning is interacting with our environment. Reinforcement learning (RL) [64] consists in learning what to do to maximize a numerical reward signal cumulated over time . The learner must discover which actions yield the most reward by trying them. Actions affect the immediate reward, the following situation, and the subsequent rewards. The idea is that an agent must be able to sense the state of its environment and must be able to take actions to affect the state and reach a specific goal.
In the following sections, we discuss the place of RL among machine learning paradigms, the formalization of the RL problem, a specific solution method, and how a RL agent senses a graph environment. Refer to fig. 1 for a high-level overview of an RL agent interacting with a graph environment.

2.1. Machine Learning Paradigms

The three main ML paradigms are supervised, unsupervised, and reinforcement learning. Table 1 outlines the main aspects of each paradigm.

Supervised Learning
▷ Model learns from labeled examples
▷ Aims at generalizing context
▷ Relies on an external supervisor

Unsupervised Learning
▷ Model learns from unlabeled examples
▷ Aims at uncovering structure in data
▷ Does not rely on external supervision

Reinforcement Learning
▷ Agent learns from experience
▷ Aims at learning an online control task
▷ Relies on an a scalar signal called reward

Table 1: ML paradigms.

2.1.1 Supervised Learning

Supervised learning [6] deals with learning from an external supervisor's labeled examples. Labels are the ground truth in a dataset. For example, classifying images by subjects is a supervised task. In this case, labels would specify the subjects for each photo.
Each sample is a description of a situation together with a specification: the label of the correct action the system should take in that situation. This paradigm aims at learning a generalized representation of a context from a limited amount of examples.
Supervised learning is not adequate for learning from interaction. Interactive problems often make it impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. An agent must be able to learn from its own experience.

2.1.2 Unsupervised Learning

Unsupervised learning [6] is about finding structure hidden in collections of unlabelled data. Unsupervised learning comprehends many tasks, among which:

- **clustering**: a data mining technique that groups unlabeled data based on their similarities or differences
- **association rules**: a rule-based method for finding the relationship between variables in a given dataset
- **dimensionality reduction**: a technique used when the number of features, or dimensions, in a given dataset is too high to reduce the number of data inputs to a manageable size

The main difference with RL is that the latter tries to maximize a reward signal instead of finding a hidden structure. Uncovering structure in an agent's experience can be helpful in RL but by itself does not address the RL problem. Therefore, we consider RL to belong to a different ML paradigm.

2.1.3 Reinforcement Learning

The main challenge in RL and not present in other kinds of learning is the tradeoff between exploration and exploitation. On the one hand, the agent has to *exploit* what it has already experienced to obtain a reward. On the other hand, the agent has to *explore* to take better actions in the future. The problem is that the agent can pursue neither exploration nor exploitation exclusively without failing the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tested many times to gain a reliable estimate of its expected reward.

2.2. Reinforcement Learning Formalization

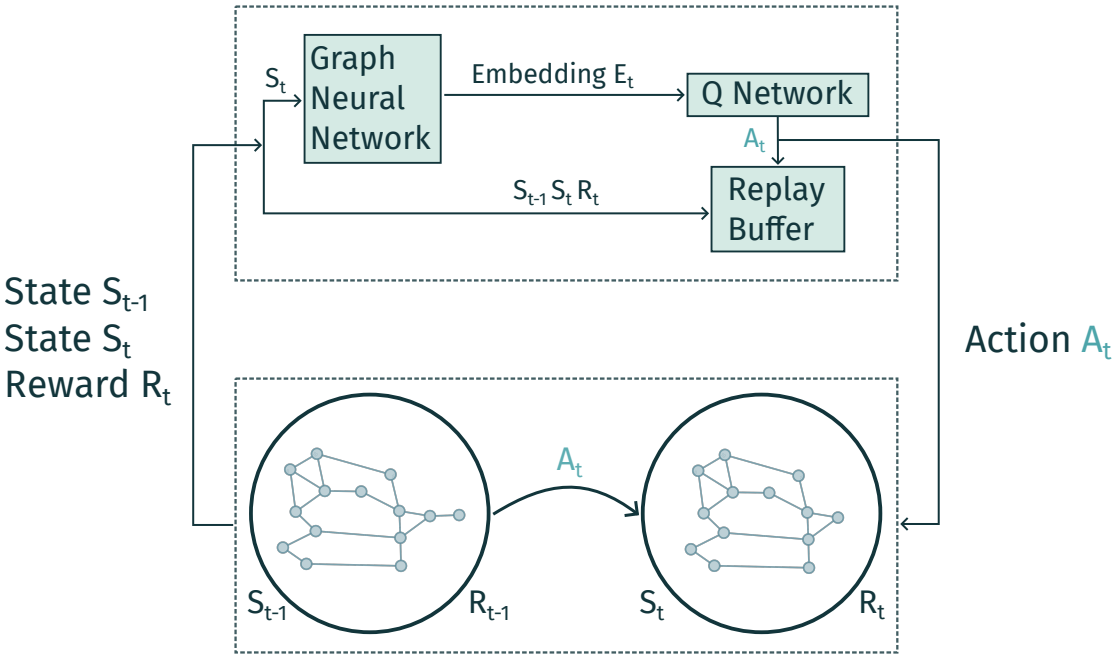


Figure 1: RL agent interacting with a graph environment.

In the RL framework, the decision maker is called *agent*, and the recipient of agent actions is called *environment*. We model the environment as a Markov Decision Process (MDP) that is a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$ where:

- $\mathcal{S}$ is a set of states called *state space*
- $\mathcal{A}(s)$ is a set of actions available in state $s \in \mathcal{S}$ called the *action space*
- $p(s', r|s, a)$ is the probability that action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ will lead to state $s_{t+1} \in \mathcal{S}$ yielding reward $r(s, s', a)$ and is called *state transition function*

The states of an MDP must satisfy the Markov Property , which intuitively means the current state and action include all aspects of the past agent-environment interaction that impacts the future.

Definition 2.1 (Markov Property)**.** Given $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ the next state $s' \in \mathcal{S}$ is conditionally independent of all other previous states and actions.

A finite MDP has finite state and action spaces.
At each timestep t the agent receives some representation of the environment's state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}(s_t)$. As a consequence of action a_t, the agent finds itself in a new state $s_{t+1} \in \mathcal{S}$ and receives a reward $r_t \in \mathcal{R}(s_t, s_{t+1}, a)$. The MDP and agent together thereby give rise to a sequence or trajectory

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2 \dots \tag{1}$$

See fig. 1 for a depiction of an agent-environment interaction. For a more in-depth treatment of the RL problem formalization refer to [64].

2.2.1 Expected Return

The agent's reward $r_t \in \mathbb{R}$ models the agent's goal. The *reward hypothesis* [64] implies that such a reward expressed through a scalar signal may describe any goal.

Definition 2.2 (Reward Hypothesis)**.** Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)

More formally, we seek to maximize the expected return where the return G_t is defined as the discounted sum of the rewards:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = r_{t+1} + \gamma G_{t+1} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

where γ is a parameter $0 \leq \gamma \leq 1$ called the discount factor.
The discount factor determines the present value of future rewards. A reward received k timesteps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. If $\gamma < 1$, the expected return has a finite value, while if $\gamma = 0$, the agent maximizes only immediate rewards.

2.2.2 Value Functions

Value functions are functions of states or state-action pairs that estimate how good it is for the agent to be in a given state or to perform a given action.
A *policy* is a mapping from each state to a probability distribution over the action space. Suppose the agent is following policy π at time t then $\pi(a|s)$ is the probability of playing action $a \in \mathcal{A}(s)$ when in state $s \in \mathcal{S}$. RL methods specify how the agent's policy changes due to its experience.
The *state value function* of a state $s \in \mathcal{S}$ under a policy π denoted $v_\pi(s)$ is the expected return when starting in $s \in \mathcal{S}$ and following π after that:

$$V_\pi = \mathbb{E}_\pi[G_t | S_t = s] \tag{3}$$

Similarly, we define the *action-value function* of a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$ under a policy π denoted $q_\pi(s, a)$ as the expected return starting from $s \in \mathcal{S}$ when taking action $a \in \mathcal{A}$ and then following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \tag{4}$$

2.2.3 Bellman Equations

The goal of an RL agent is to find the optimal policy for a given MDP. Value functions define partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states:

$$\pi \geq \pi' \iff V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in \mathcal{S} \tag{5}$$

Thus the optimal policy is defined as:

$$\pi^*(s) \in argmax_\pi V_\pi(s) \quad \forall s \in \mathcal{S} \tag{6}$$

A fundamental property of the value functions used throughout RL is that they can be defined recursively through their Bellman equations:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, s)[r + \gamma V_\pi(s')] \quad \forall s \in \mathcal{S} \tag{7}$$

$$Q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \tag{8}$$

We can now define the Bellman optimality equations, which are the definitions of the value functions associated with the optimal policy. Therefore, we can define both value functions without referencing any specific policy:

$$V^*(s) = max_{a \in \mathcal{A}(s)} Q_{\pi^*}(s, a) = max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')] \tag{9}$$

$$Q^*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma max_{a' \in \mathcal{A}(s')} Q^*(s', a')] \tag{10}$$

For finite MDPs, the Bellman optimality equations have a unique solution. If the transition probabilities are fully known, one could solve the equations and get Q^*. Once Q^* is known, the optimal policy π^* follows by choosing the action $a \in \mathcal{A}(s)$ that maximizes Q^* for the current state s.
Solving the Bellman optimality equations provides one route to finding an optimal policy, thus solving the RL problem. However, at least three factors make solving Belmann equations impossible or infeasible:

- **environment dynamics**: transition probabilities in non-trivial environments are often unknown and very hard or even impossible to find;
- **computational resources**: environments such as a chessboard or a large network yield huge state spaces, which make the Bellman equations unfeasible to solve;
- **Markov property**: real-world environments rarely respect the Markov property.

Many RL methods can be understood as approximating the Bellman optimality equation using experienced transitions instead of knowledge of the expected transitions.

2.3. Temporal-Difference Methods and Deep RL

In the following sections we detail several Temporal-Difference (TD) methods to build a Double Dueling Deep Q Network Agent with Prioritized Experience Replay. This is the architecture found in all the RL agents of our multi-agent system. Refer to algorithm 1 for an overview of the training process of such agents.

Algorithm 1 Double Dueling DQN with Prioritized Experience Replay

Input: minibatch k, step-size ν, replay period K, size N, exponents α, β, budget T
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe s_0 and choose $a_0 \sim \pi_\theta(s_0)$
for $t = 1 \dots T$ **do**
  Observe s_t, r_t, γ_t
  Store transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in $\mathcal{H}$ with maximal priority $p_t = max_{i<t} p_i$
  if $t = 0 \mod K$ **then**
    for $j = 1 \dots k$ **do**
      Sample transition $j \sim P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$
      Compute importance-sampling weight $w_j = \frac{(N \cdot P(j))^{-\beta}}{max_i w_i}$
      Compute TD-error $\delta_j = r_j + \gamma_j Q_{target}(s_j, argmax_a Q(s_j, a)) - Q(s_{j-1}, a_{j-1})$ with q-values computed as $Q(s,a) = V(s) + (A(s,a) - max_{a' \in \mathcal{A}} A(s,a'))$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(s_{j-1}, a_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \nu \cdot \Delta$ and reset $\Delta = 0$
    Every t_{target} steps copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $a_t \sim \pi_\theta(s_t)$
end for

2.3.1 Q Learning

One of the early breakthroughs in RL is Q-learning [74], which is an off-policy TD control algorithm. TD methods can be seen as a combination between Monte Carlo (MC) ideas and Dynamic Programming (DP). Like MC, TD methods can learn directly from raw experience without a model of the environment's dynamic. Like DP, TD methods update estimates based in part on other learned estimates without waiting for an outcome. On top of this, Q-learning is off-policy because it learns the state-action value function by following a policy different from the policy it is estimating, called behavior policy.
The Q-learning update is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma max_{a \in \mathcal{A}(s_t)} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{11}$$

where action a_t is sampled from a policy derived from Q. We call the greedy policy $max_{a \in \mathcal{A}(s_t)} Q(s_{t+1}, a)$ the *target policy* as it directly estimates the optimal policy π^*

2.3.2 Deep Q Networks

A Deep Q-Network (DQN) [46] is an agent which combines Q-learning with deep neural networks. The general idea is to use a deep neural network to approximate the optimal action-value function, defined in eq. (10). Non-linear approximators have long caused instabilities to RL agents when used to approximate the action-value function due to three key issues:

- **correlation between observations**: the sequence of observations yield naturaly correlated observations
- **action-value dynamics**: a small update to q may significantly change the policy
- **action values and target values correlation**: action values and target values are directly correlated given $r + max_{a'}\gamma q(s', a')$

Mnih et al. [46] implemented *experience replay* to randomize the data and remove the sources of correlation. Experience replay means storing the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each timestep t in a dataset $D = \{e_1, \dots, e_t\}$ and applying Q-learning updates on minibatches of experiences sampled uniformly from the dataset. The q-learning update at iteration i minimizes the following loss:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{12}$$

Where $U(D)$ is the uniform distribution over the agent's past experiences. Then, the action-value function is parametrized for the weights of the deep neural networks approximating it and is written $Q(s, a; \theta_i)$ where θ_i are the parameters of the network at timestep i. We also use a target network used to compute the target with parameters θ_i^- at timestep i; such parameters are updated with θ_i every C steps and held fixed between individual updates. The loss in eq. (12) represents the mean squared error of the Bellman equation discussed in section 2.2.3. The target values are replaced with estimated ones and defined $r + \gamma max_{a'} Q(s', a'; \theta_i^-)$. This loss is then minimized through stochastic gradient descent.

The DQN algorithm is both model-free and off-policy. Firstly, it solves the RL task directly using samples from the environment without explicitly estimating the reward or the transition function. Then, it is off-policy because it learns about the greedy policy $a = argmax_{a'} Q(s, a'; \theta)$ while following a behavior distribution that ensures adequate exploration of the state space. In practice, the behavior distribution is often selected by an epsilon-greedy policy that follows the greedy policy with probability $1 - \epsilon$ and selects a random action with probability ϵ.

Algorithm 2 DQN with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1 \dots M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1 \dots T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_t observe reward r_t and next state s_{t+1}
    Preprocess the observed state $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

2.3.3 Double Deep Q Networks

Van Hasselt et al.[68] showed that the Q-learning algorithm, as discussed in section 2.3.1, suffers from substantial overestimation in some games belonging to the Atari 2600 domain [75] which is a popular benchmark in the RL community. They tracked down this overestimation issue to the max operator used both in standard Q-learning and DQN, which use the same values both to select and evaluate an action, refer to eqs. (11) and (12). This

formulation makes it more likely to select overestimated values, resulting in over-optimistic value estimates. They thus decided to implement double Q-learning [25] for DQNs to decouple selection and evaluation.
For each update, one set of weights is used to determine the greedy policy and the other to determine its value. We can rewrite the target in eq. (12) to untangle the selection and evaluation:

$$Y_t^{DQN} = r_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta_t); \theta_t) \tag{13}$$

Finally, we can rewrite the q-learning target in the double q-learning formulation by replacing the second set of weight vectors with θ^-, the weights of the target network in the DQN formulation.

$$Y_t^{DoubleDQN} = r_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta_t), \theta_t^-) \tag{14}$$

2.3.4 Dueling Networks

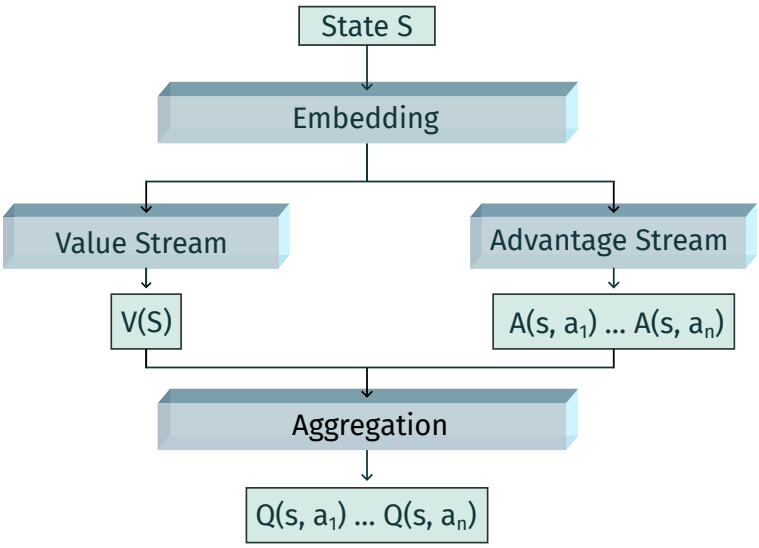


Figure 2: Dueling network architecture.

Many works focus on designing improved control and RL algorithms or incorporating existing neural network architectures. Conversely, Wang et al. [73] concentrate primarily on creating a neural network architecture that is better suited for model-free RL called *dueling architecture* (see fig. 2). The proposed architecture explicitly separates the representation of state values and action advantages.
The advantage function is a value function defined as:

$$A(s, a) = Q(s, a) - V(s) \tag{15}$$

The advantage function encodes a measure of the relative importance of each action by subtracting from the action-value function the state-value function.
The main idea is to have a common embedding that separates into two streams that encode state values and action advantages separately. The two streams are combined via a special aggregating layer to produce an estimate of the action-value function Q. The naive solution would be to build Q as follows

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{16}$$

where V is the output of the state-value stream parametrized to the weights of the embedding θ and the weights of the state-value stream β. On the same line, A is the output of the advantage stream parametrized to the embedding weights and the advantage stream weights. The issue is that eq. (16) is unidentifiable because given Q we cannot recover V and A uniquely, which in practice means poor performance. To solve this issue, the authors modified the action-value specification as follows:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)) \tag{17}$$

Now, for the optimal action a^* :

$$a^* = argmax_{a' \in \mathcal{A}} Q(s, a'; \theta, \alpha, \beta) = argmax_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha) \tag{18}$$

we obtain

$$Q(s, a^*; \theta, \alpha, \beta) = V(s, \theta, \beta) \tag{19}$$

In conclusion, the dueling network automatically produces separate estimates of the state value function and advantage function without extra supervision. Therefore, the network can learn which states are valuable without learning the effect of each action for each state. Such behavior is beneficial in states where actions do not affect the environment in any relevant way.

2.3.5 Prioritized Experience Replay

Experience replay has been devised together with Deep Q Networks [46] to break the temporal correlations by mixing more and less recent experiences at each update and reusing rare experiences in multiple updates.
Schaul et al. [57] investigated how prioritizing which transitions are replayed can make experience replay more efficient and effective than if all transitions are replayed uniformly. The key idea is that an RL agent can learn more effectively from some transitions than others. Transitions may be more or less surprising, redundant, or task-relevant. Some transitions may not be immediately valuable to the agent but might become so when the agent's competence increases. Experience Replay frees online learning agents from processing transitions in the exact order they are experienced. Prioritized replay further frees agents from considering transitions with the same frequency they are experienced.
The central component of prioritized replay is the criterion by which the importance of each transition is measured. One idealized criterion is the amount the RL agent can learn from a transition in its current state, which we call expected learning progress. While this measure is not directly accessible, a reasonable proxy is the magnitude of transition's Temporal Difference (TD) error δ_t defined as:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{20}$$

The TD error measures the difference between the estimated value of s_t and the better estimate $r_{t+1} + \gamma V(s_{t+1})$, thus indicating how unexpected the transitions are. This is particularly suitable for incremental online RL algorithms such as Q-learning that already compute the TD error and update the parameters in proportion to δ. However, greedy TD-error prioritization has several issues:

- **local updates**: to avoid expensive sweeps over the entire replay memory, TD errors are only updated for the transitions that are replayed, which means that transitions with a low TD error on the first visit may not be replayed;
- **noise spikes**: stochastic rewards may induce large approximation errors;
- **small subset of experience**: errors shrink slowly, meaning that initially, huge error transitions get replayed frequently.

All the cited issues make the system prone to overfitting.
Thus, stochastic sampling is introduced by interpolating between pure greedy optimization and uniform random sampling. Schaul et al. define the probability of sampling transition i as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{21}$$

where $p_i > 0$ is the priority of transition i. The exponent $\alpha \in [0, 1]$ determines how much prioritization is used, and $\alpha = 0$ corresponds to the uniform case. The variant we will consider is the direct proportional prioritization where $p_i = |\delta_i| + \epsilon$ and ϵ is an arbitrarily small constant.
In deep Q networks, we minimize an expectation with regard to a uniform sampling distribution as in eq. (12). The sampling from the replay buffer is still expected to be uniform. Prioritization modifies this distribution in an uncontrolled manner by oversampling transitions with high priority. Therefore, they introduced importance sampling to down-weight the importance of oversampled transitions. We now correct the bias with the following weighting

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta \tag{22}$$

where N is the size of the replay buffer. The weighting in eq. (22) fully compensate for the non-uniform probabilities $P(i)$ if $\beta = 1$. Finally, these weights are added into the Q-learning update by using $w_i \delta_i$ instead of δ_i. Refer to algorithm 1 for a more detailed view of the training process of an RL agent employing prioritized experience replay.

2.4. Graph Neural Networks

Concluding the background, we discuss how RL agents in our system perceive a graph environment through Graph Neural Networks (GNNs) [21, 56]. The GNN formalism is a general framework for defining deep neural networks on graph data. The key idea is that we want to generate representations of nodes that depend on the graph's topology and any feature information we have. We will now give an overview of GNNs' inner functioning and discuss the main layers used in our work. For a more detailed discussion, refer to [24].

2.4.1 Neural Message Passing

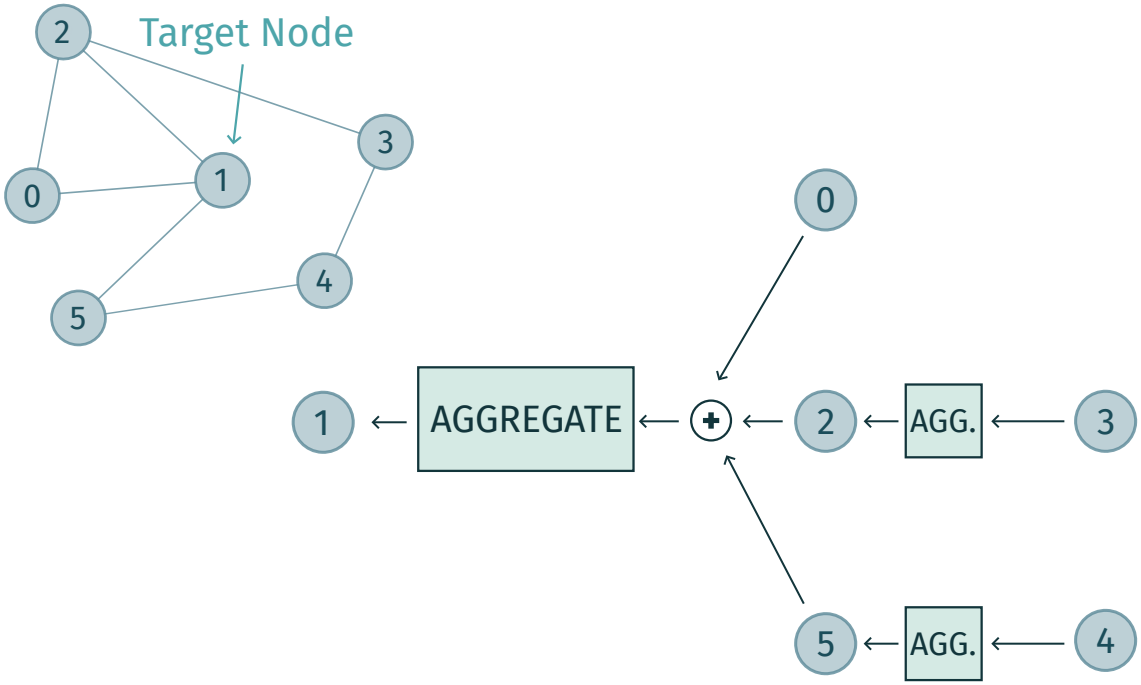


Figure 3: High-level view of neural message passing.

The defining feature of a GNN is that it uses a form of neural message passing in which vector messages are exchanged between nodes and updated using neural networks [19].
During each message-passing iteration in a GNN, a hidden embedding $\mathbf{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to the information aggregated from u's graph neighborhood $\mathcal{N}(u)$. This message passing update can be expressed as:

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^j \quad \forall v \in \mathcal{N}(u)\})) = \text{UPDATE}^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}) \tag{23}$$

where UPDATE and AGGREGATE are arbitrarily differentiable functions and $\mathbf{m}_{\mathcal{N}(u)}$ is the message that is aggregated from u's graph neighborhood $\mathcal{N}(u)$. Superscripts distinguish the embeddings and functions at different iterations of message passing.
At each iteration k, the AGGREGATE function takes as input the set of embeddings of the nodes in u's graph neighborhood $\mathcal{N}(u)$ and generates a message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ based on this aggregated neighborhood information, see fig. 3. The UPDATE function then combines the message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ with the previous embedding $\mathbf{h}_u^{(k-1)}$ of node u to generate updated embedding $\mathbf{h}_u^{(k)}$. The initial embeddings at $k = 0$ are set to the input features of all the nodes $\mathbf{h}_u^{(0)} = \mathbf{x}_u \quad \forall u \in \mathcal{V}$. After running K iterations of the GNN message passing, we can use the output of the final layer to define the embeddings for each node:

$$\mathbf{z}_u = \mathbf{h}_u^{(K)} \quad \forall u \in \mathcal{V} \tag{24}$$

2.4.2 The Basic GNN

Until now, we have described an abstract GNN, as in eq. (23). We will now discuss the basic implementation of UPDATE and AGGREGATE. The basic GNN message passing is defined as:

$$\mathbf{h}_u^{(k)} = \sigma(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}) \tag{25}$$

where $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ are trainable parameter matrices and σ denotes an elementwise non-linearity. The bias term $\mathbf{b}^{(k)} \in \mathbb{R}^{d^{(k)}}$ is often omitted for notational simplicity, but including the bias term can be important to achieve strong performance.

Message passing in the basic GNN framework is analogous to a standard multi-layer perceptron (MLP) as it relies on linear operations followed by a single elementwise non-linearity. Firstly, we sum the messages incoming from the neighbors. Then, we combine the neighborhood information with the node's previous embedding using a linear combination. Finally, we apply an elementwise non-linearity.

We can now define the basic GNN through the UPDATE and AGGREGATE functions:

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v \tag{26}$$

$$\text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \sigma(\mathbf{W}_{\text{self}}\mathbf{h}_u + \mathbf{W}_{\text{neigh}}\mathbf{m}_{\mathcal{N}(u)}) \tag{27}$$

3. Problem Description

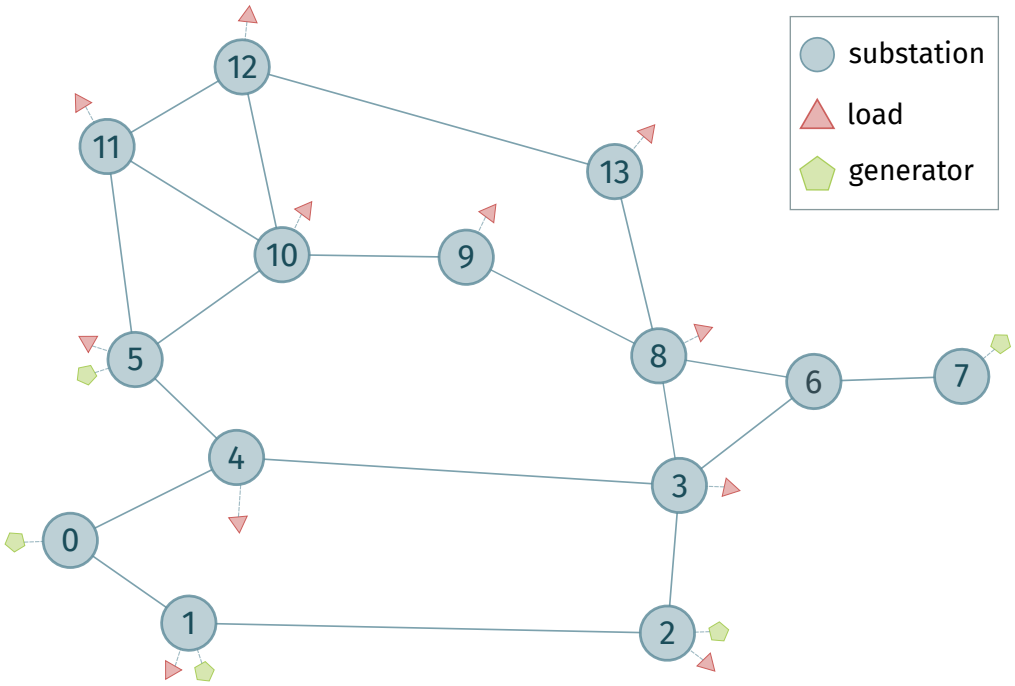
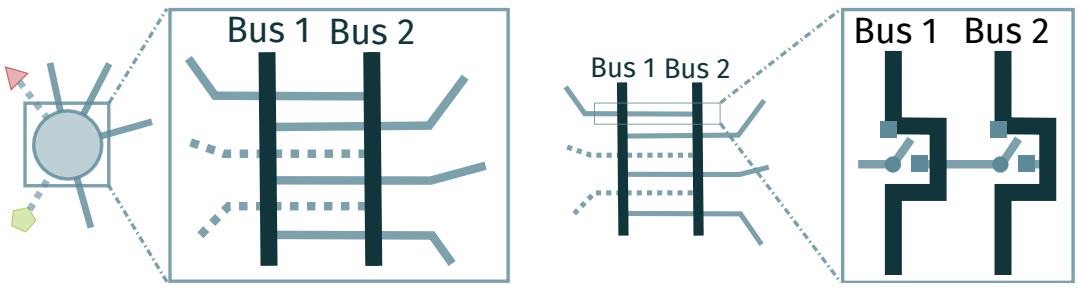


Figure 4: L2RPN case 14 environment.

The "Learning to run a power network" (L2RPN) challenge is a series of competitions that model the sequential decision-making environments of real-time power network operation. The participants' algorithms must control a simulated power network within a Reinforcement Learning (RL) framework [45].
The French electricity network management company RTE (Reseau de Transport d'Electricite) has been organizing the L2RPN challenges since 2019 because power networks have faced systemic changes that bring current technology to the edge. The advent of intermittent renewable energies on the production side and prosumers on the demand side, coupled with the globalization of energy markets over a more interconnected European grid, poses significant challenges to grid operators [44].
Currently, operators must analyze massive amounts of data to make complex and coordinated decisions over time. For this reason, recent advances in Deep Reinforcement Learning [59] have drawn the attention of the power system community for their capacity to learn representations and their parallelizable architectures.
RTE has therefore developed Grid2Op [13], a Python module that casts the power grid operational decision process into a Markov Decision Process (MDP).
Grid2Op represents a power grid as a set of objects: powerlines, loads, generators, and batteries with substations linking everything together. Powerlines connect substations and allow power to flow from one place to another. A graph akin to the one in fig. 4 naturally models the power grid as we have described it. See table 3 for an overview of the node and edge features in the Grid2Op's power network model.
However, our power network model needs to take into account also the internal structure of substations, where



(a) Buses (b) Switches

Figure 5: Substation inner working.

	substations	powerlines	topologies	action space size
rte case 5	5	8	31.320	585
rte case 14	14	20	1.397.519.564	3580
l2rpn wcci 2020	36	59	1.88 e+21	3.941.849
IEEE case 118 [31]	118	186	3.88 e+76	13.419.900

Table 2: Complexity of different power networks available in Grid2Op.

we find two busbars to which every grid object connects, as in fig. 5a. Substations connect elements through switches, as depicted in fig. 5b, which allow the operator to electrically separate elements from each other. Any switching action which isolates a load or a generator from the rest of the network yields an invalid topology and causes a blackout.

On top of validity considerations, action space size has the utmost importance when dealing with power networks. Given a substation with k elements there are 2^{k-1} different configurations while N powerlines imply 2^N actions. Each generator can be redispatched, which means the network operator may request a power plant to change the production set point to deal with a higher or lower power request. To deal with such a combinatorial explosion, Grid2Op introduces several operational constraints that limit the agent's actions at each timestep.

First, the number of concurrent powerlines modifications is limited to one. Powerline changes have severely nonlinear effects on the grid, fostering human operators to avoid concurrent powerline modifications. Second, substations mutations may affect no more than two substations together. Human operators can only focus on a few simultaneous operations, so the number of substations affected by their intervention rarely exceeds two. Finally, you can act on the same grid object with limited frequency. The more you reconfigure a real-world switch, the faster you can expect it to age, and the more fragile it will eventually become. In table 2, we summarize the actions space size of several environments subject to the operational constraints discussed so far.

The model we just discussed significantly simplifies an actual power grid. Above all, real power networks accommodate a large variety of heterogeneous objects with much more complex properties than loads, generators, substations, and powerlines. Moreover, substations are significantly simplified both in the number of buses they contain and in the switching structure, which is usually more articulated to ease frequent switching operations.

In conclusion, we frame the network operation problem as an MDP. The agent is the network operator, capable of performing actions on the environment, which is the power network itself.

The main MDP components are:

- **states**: description of the physical network, including load, generators, and line connections;
- **actions**: topological changes or generator redispatching;
- **reward**: some measure of economic, environmental, or security cost or performance;
- **transition function**: a function determining the new state of the network given the previous state and the action taken by the agent.

The RL task structure is episodic, with episodes running from midnight until the agent disconnects a load from a generator or the episode ends, the following midnight; timesteps are recorded every five minutes. Loads and generators each day follow a specific behavior mimicking that of an actual power grid.

feature	node or edge	description
active power flow	node and edge	The power consumed or utilised in the grid
reactive power flow	node and edge	the part of complex power that corresponds to storage and retrieval of energy rather than consumption
voltage	node and edge	difference in electric potential
phase angle	node and edge	phase difference between voltage and current in the circuit
powerline capacity	edge only	observed current flow divided by the thermal limit
cooldown	edge only	the number of time step the powerline is unavailable

Table 3: Node and edge features available in the Grid2Op power network model.

4. Related Works

The increasing complexity of today's large interconnected power systems requires advanced control techniques. Better communications infrastructure, more robust computational capabilities, and new control devices open up the possibilities to implement advanced control schemes which can process the observations realized on the power system and control it appropriately [85]. Embedding the learning methods into the control schemes is an effective way to endow the controllers with the capability to learn and update their decision-making [20]. RL offers a panel of methods that allows controllers to learn a goal-oriented control law from interactions with a system or its simulation model [9, 64]. In this section, we will first investigate the state of the art in applying reinforcement learning to power networks' operational control, and then we will discuss the solutions of the previous L2RPN winners.

4.1. RL for Power Network Operational Control

Reference	Approach	Notes
[76]	Nash Q-Learning	Eligibility traces for multi-temporal decisions
[78]	Coordinated Q learning Agents	One agent per power network area
[79]	Deep Forest	Agents handle different timescales
[83]	Decentralized Deep Forest	Leader-Follower Approach
[29]	Deep Q Learning	Framework akin to Grid2Op used
[53]	Deep Q Learning	Comparison with tabular Q-learning
[77]	Deep Q Learning	DQL applied to multiple timescales
[12]	Deep Q Learning	Voltage and set-point control together

Table 4: Main papers discussing RL applications to power networks.

Many decision, control, and optimization problems arise in power networks, including energy management, demand response, electricity market and operational control, to mention a few. The focus of our work is *active control*: maintaining a continuous supply of power to all the consumers in the system. Operational management is becoming increasingly challenging as solar and wind power become more prevalent. Renewable generator production depends on the weather, and typically the energy storage capacity is low. Therefore, DRL is widely used to estimate control strategies and optimization policies under large-scale scenarios with limited information [20]. See table 4 for an overview of the main works in RL applied to power network operational control and table 5 for a summary of the main tradeoffs induced by the adoption of RL in power network control. There are many lines of work in this area, we now investigate the most prominent ones.

4.1.1 Renewable Energies

Xi et al. [76] propose a smart generation control scheme to tackle the penetration of renewable energy generators in the smart grid. The model chosen by the authors is a multi-agent reinforcement learning approach based on Nash Q-learning [35] and eligibility traces [61]. Nash Q-learning is a training algorithm for multi agent systems based on finding the Nash equilibrium in a game among the agents to be trained. Yin et al. [78] follow a similar approach by deploying one independent deep Q learning agent for each area of the power network but training them independently.
The penetration of renewable energy in the smart grid poses demanding challenges to power grid operation. In Grid2Op latest challenges, the energy mix includes an ever-increasing amount of renewable energy. Grid2op models renewable sources as non-controllable generators; we can only curtail their production, thus significantly limiting the reaction surface of the agent to network emergencies. The work on operational control is far more extensive than the work framed in the Grid2Op framework but pursues the same targets.

4.1.2 The Curse of Dimensionality

Zhou et al. employ Deep forest [87] to tackle the curse of dimensionality. The Deep forest model is a tree-ensemble model which builds a deep decision tree in a data dependent way without resorting to backpropagation

for parameter update. The deep forest component is applied to predict the next systemic state of a power system, including both emergency and routine states. Yin et al. [79] extend the work on deep forests for power network control outlining a preventive strategy for reducing emergencies on large inter-connected power grids with continuous disturbances. On top of deep forest, Zhang et al. [83] employ a decentralized consensus algorithm based on virtual generation tribes [84]. The idea behind virtual generation tribes is that a different agent manages each power network area. Agents and leader share Q-matrices, and the leader chooses the global action to take. Multiple agents form the RL controller, each of which predicts one movement per next timestep. Reliability, fault tolerance, and scalability are mandatory requirements for any production-ready power network control system. Multi-agent approaches are common in operational control. To our knowledge, nobody proposed multi-agent methods among the Grid2Op previous winners due to the challenge format requiring more instance-optimized models. On the other hand, we decided to leverage the maturity of the Grid2Op ecosystem while taking a step back from the challenge format by building a multi-agent controller.

4.1.3 Power Network Simulation Frameworks

Huang et al. [29] apply RL to build an adaptive emergency control scheme. On top of this, they developed a power grid control benchmarking tool akin to Grid2Op called Reinforcement Learning for Grid Control. It exposes smaller power systems but with more extensive details concerning the electrical components of the grid. Another reinforcement learning framework for optimal operational control and maintenance is developed by Rocchetta et al. [53]. Similarly to previous proposals, renewable generators are the focus of the investigation, together with the capability of RL to handle a larger state-action space with respect to tabular reinforcement learning.
Deep function estimators allow RL agents to scale in large state-action spaces by decoupling the policy from a large table that must host all state-action pairs. While no Grid2Op participant proposed tabular approaches, enumeration techniques have been used in some solutions to filter the action space. Enumeration approaches while challenge-specific may give valuable insights for building expert systems which often go hand in hand with RL agents in the power network context.

4.1.4 Voltage Control

On top of reducing emergencies, another common target in literature is voltage control. In this case, we aim to control bus voltage within the desired region when large consumers such as electric vehicles and intermittent generators enter the network. Yang et al. apply RL on two timescales [77] . On a faster timescale, the agent configures the optimal set point of smart inverters, while on a slower timescale, it configures capacitors to minimize the long-term discounted voltage deviations. In the same line of work, Diao et al. [12] develop GridMind, a RL agent which learns its policy from offline simulations and can adapt not only to load or generations changes but also to topological changes.
Voltage control is a complex problem of its own and it is not modeled in Grid2Op. Voltage control needs a much deeper understanding of the inner working of network devices, while Grid2Op main objective is to give an entry point to reinforcement learning practitioners in the power network world.

RL Advantages	**RL Disadvantages**
▷ Leverage incomplete information	▷ No consideration for devices' physical structure
▷ Learn continuous control under continuous state-action spaces	▷ Lacking of multi-timescale decisions
▷ Deal with unpredictable emergencies	▷ Setpoint control is not coupled with voltage and frequency control

RL Directions
▷ Combination of RL and classical control methods
▷ Hierarchical strategies layering control and optimization
▷ Merging of grid data features with device model features

Table 5: RL in the power network literature.

4.2. L2RPN Past Challenges and Previous Winners

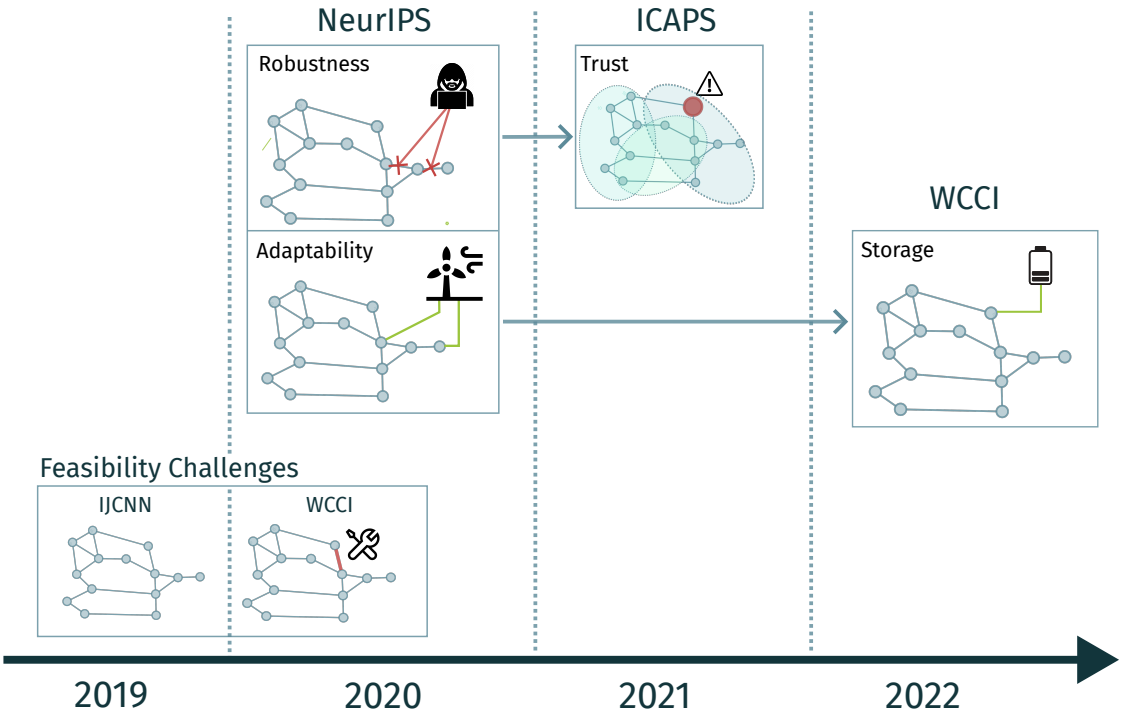


Figure 6: L2RPN challenges timeline.

4.2.1 2019 Feasibility Challenge

In the 2019 edition [44], the organizers provided a slightly adapted version of the IEEE 14-bus network [32] with 20 powerlines, 11 loads, and 5 generators. The competition's goal was to operate the power grid in real time over several days at a 5-minute resolution. More precisely, the objective was to manage the power flows at every timestep given production, loads, and the grid topology. The score function gave incentives to optimize the margins through all lines to maximize the overall residual grid capacity given powerline capacities (i.e., thermal limits). An agent could only use topological actions on lines and substations to manage the grid while meeting further operational constraints on actions and overloads. In 2019, the organizers set the following operational constraints:

- **reaction time**: time to react to an overload before the line gets disconnected by protections, set to 1 timestep (i.e., 5 real-time minutes);
- **activation time**: there is a maximum number of actions performed in a given period, one action per substation in this case;
- **recovery time (cooldown)**: due to the physical properties of the assets, there is a time interval after activation before flexibility can be reactivated, set to 3 timesteps in this challenge. Flexibilities are akin to budget. When you use one, you consume part of your budget that you will recover some time after.

First Place The winners [39] used the Double Deep Q-Learning (DDQN) algorithm [68] along with imitation learning [30] to initialize the policy. Imitation learning is a supervised learning method used to pre-train RL agents by providing good initial policies in the form of neural network weights. A power grid simulator generates massive datasets, which they further process before being used to train the agent. On top of imitation learning, a guided exploration training method is used instead of the more traditional epsilon-greedy way. At every timestep, the agent selects a fixed number of actions with the highest Q-values, and then their performance is simulated on the fly. Then the action with the highest reward is chosen and added to memory. With the help of the action simulation function, the training process is more stable, and a better experience is stored and used to update the agent. Finally, an early warning mechanism enhances system robustness. The idea of such a mechanism is to simulate the effect of doing nothing before all the other actions. If the loading level of any of

the power lines is higher than a predetermined threshold, then the guided exploration process starts otherwise the agent does nothing.

Discussion In this work, we immediately see the relevance of expert rules in automatic power grid control. Ideally, the agent should only act against emergencies because controlling an entire power network is infeasibly complex to learn and would yield a naïve optimal policy for most of the non-emergency situations. Indeed,any situations do not require any intervention, while others require obvious interventions such as reconnecting disconnected powerlines. In our design, we kept our expert system as lean as possible to emphasize agent behavior framing the model as a support system, not a standalone grid controller. Conversely, many challengers took the opposite direction because expert systems yield significant score improvements.

4.2.2 2020 NeurIPS and WCCI Challenges

In the 2020 edition, the power network was much more complex. It comprised 59 powerlines, 37 loads, and 22 generators [43]. The overall goal was to avoid blackouts while optimizing the cost of operations under safety constraints. Environments were still episodic, running at a 5-minute resolution over a week. The game-over condition is triggered if total demand exceeds total production.

WCCI Winner The winner of the WCCI edition [80] performed an actions space reduction to 1000 elements using an expert system and initialized a policy parametrized by an extensive feed-forward neural network. Then, they trained a policy using evolutionary black-box optimization. Yoon et al. assumed reconnecting was always the best action to take, as for the winners of the 2019 edition [39]. They let the agent intervene only in hazardous situations. The existence of a line in which the power flow is larger than the threshold hyperparameter determines a dangerous condition, a setting naturally modeled as a semi-MDP [64]. The main challenge of the Grid2Op environment is the size of the state and action spaces. To address this problem, Yoon et al. adopted an actor-critic architecture where function approximators trained through Soft Actor-Critic (SAC) [23] represent the policy and the value function. In addition, they used the afterstate representation to capture many state-action pairs being led to an identical after state by leveraging the transition structure. The idea of the after-state representation is that the agent does not learn state-action pairs but learns the value, called afterstate value, of the state reached by playing a given action in a given state called afterstate value. The main advantage of using such an approach in this context is that a graph has many symmetries, so the space of afterstates is practically smaller than that of state-action pairs. Generally, it is tough to take exploratory actions in the Grid2Op environment. The power grid will fail in a few steps if the agent takes random actions. For example, the agent with the random policy performs vastly worse than the "do nothing" policy. The random exploration policy would often get stuck at bad local optima that executes only one or two actions. A more structured exploration is key to successful training. To this end, Yoon et al. extend the actor-critic algorithm to a two-level hierarchical decision model by defining the goal topology configuration as the high-level action. This way, they can take full advantage of the afterstate representation since the goal topology captures the equivalence of many sequences of primitive actions that lead to identical topologies. In addition, exploration with goal topology is more effective than with primitive actions since the policy only needs to focus on where to go, that is, the desirable topology under the current situation, without needing to care about how to get there. The high-level policy proposes a goal topology, while the low-level policy needs to find the action sequence that changes the current topology into the goal topology. Thus, they consider a rule-based approach for the low-level policy where the rule determines the order of substations to execute the bus assignment actions.

NeurIPS Winner At NeurIPS, the challenge comprised two tracks, one with an adversary randomly cutting some wires and the other on a more extensive network based on the IEEE 118 network [31], which is a standardized approximation of the American electric power system in the midwest as of December 1962. The winning team [86] of both tracks used a neural network policy to select the top-k actions and applied an optimization algorithm to choose the best one . Zhou et al. introduced a novel search-based planning algorithm that performs Search with the Action Set (SAS). The goal of such an algorithm is to maximize the average long-term reward. The policy network outputs a probability distribution over the actions at each simulation step, and the top-k activities with the highest probabilities form the action set. They then leverage the simulation function for action selection to ensure that the action meets the constraint by simulating the outcome for each action and filtering out actions that violate the restrictions. Finally, the algorithm selects an action from the set based on the value function with the future state predicted by the simulation function. Prior work uses supervised learning to approximate the actual value function with the trajectory data. In power grid management, they found an alternative estimate function that does not rely on approximation. The idea is that the unsolved overloaded power line can induce more overloaded power lines and even lead to large-scale blackouts. Thus, they define the max overload among all the powerlines as a risk function and replace the value

function with this function when choosing the action to play. The main issue to overcome when using the risk function instead of the value function is that the risk function depends on the simulation function, which is often not differentiable. To overcome this issue, they apply black-box optimization of evolution strategies (ES) [55] to update the policy, which does not require backpropagating the gradients for optimization. The idea is that a population of parameter vectors is derived from current parameters and evaluated in the environment. After that, the parameter vectors with the highest scores will be recombined and form the next generation. In SAS, they repeatedly inject gaussian noise into the parameter vector of the original policy and obtain a bunch of policies for exploration. Overall, during the optimization process, they generate the exploration policy, collect the sampled noise parameters and the related reward for the computation of the combined gradient, and update the policy parameters.

Discussion WCCI winners stress two of the most fundamental challenges of the power network control problem, action space size and long-term exploration. Regarding action space filtering, they use a greedy agent to filter out irrelevant actions, which in general, introduces bias in the policy and does not scale to larger topologies while performing exceptionally well in the challenging context. On the other hand, the hierarchical structure of their decision-making system greatly informs our approach.
NeurIPS winners employ the action set approach to deal with action space size by learning to filter the action space as part of the control task. Such an approach allows for avoiding bias through action filtering while leveraging the advantages of action filtering. We decided to couple a hierarchical system with action selection to reduce the action space along the hierarchy while keeping the system differentiable. Our approach allowed us to take advantage of gradient descent while resorting to action selection for control stability and efficiency.

4.2.3 2021 L2RPN with Trust Challenge

In 2021 the L2RPN with trust challenge was organized to explore the role of artificial agents as support systems for human power grid operators [42]. Existing AI technology lacks the robustness and trustworthiness required for high-consequence, high-impact decision-making in real-time network operations. Experienced operators in power networks deploy extensive domain or expert knowledge that current machine learning models need to represent more adequately. The main idea behind this challenge is that the agent can actively send warning signals to the human when the agent's confidence about their actions is low. Sending these warning signals improves reliability and credibility by providing selective enough details. The warning signals can be discrete or continuous information about the confidence or aiming at explaining the warnings. In this challenge, regional signals improve agents' credibility. The attention budget develops over time, similar to intimacy. The attention budget decreases when the agent warns the human. Intimacy can increase if the warning is relevant, or otherwise it will reduce. In case of an unwarned failure, intimacy decreases substantially while the operator could have paid attention. The attention budget is a balance for operators to decide when they can trust the agent or their experience. A more accurate and transparent agent will build trust and result in greater public attention and reduced supervision requirements. The competition ran on one-third subgrid of the IEEE 118 bus system. The renewable share made up 20% of the overall energy mix, which is a proxy for high variability in network operation parameters.

Third Place Enlite AI won third place in 2021 by extending the RL algorithm AlphaZero [60] to apply it to the topology planning problem in large power grids. As in the original formulation of AlphaZero, they implemented a distributed Monte Carlo tree search (MCTS) sampling [66], which generates an asynchronous stream of MCTS sampled trajectories inserted in a rolling trajectory buffer. Once the buffer contains enough transitions, they are filtered for dead-end pruning and then fed to the policy and value network. MCTS is costly with such a sizeable state-action space as it explicitly explores state-action pairs. Therefore, they developed a heuristic value function with early stopping to avoid frequently visited states. Enlite AI implemented a critical state observer in line with the past winners, which bypasses the agent if no line overload exceeds a given value and executes no action. They converted the action space to a sub-step action space allowing them to take multiple discrete steps at a time, thus skipping explicit action space reduction while still keeping the model efficient. While the most crucial bit of the implementation, more details have yet to be given on the actual transformation to the action space. The reward was the exponential decay of the weighted sum of the maximum load on any powerline and the number of offline powerlines. Feature engineering was kept to the minimum by adding load-generation deltas to substations and powerlines and employing stacking and normalization. Regarding alarm-specific mechanisms, they used a multi-step simulation rollout to detect simulation states in which the agent must act. An alarm is sent to the human operator when the system detects a future state in which the agent must act.

Second Place H. Martinez won the second prize by extending the approach used by the team, ranking second in the 2020 edition at NeurIPS. The idea is that a teacher model finds a greedy action to minimize the maximum load rate of all lines by enumerating all possible steps and saves it in an action library. After filtering out the less frequent actions, they obtained an action library of 208 actions. A tutor model plays a greedy strategy but with an action space of 208 actions achieving a score of 54.69 in the competition. Once fixed the operational plan, they devised the alarm strategy. We can pre-simulate grid evolution to have faster alarm scoring, as alarms do not affect the grid state but the scoring function. After storing all the pre-simulated episodes, they train a network for raising the alarm together with some expert rules that decide for alarms based on powerlines overload.

First Place Finally, Elixir Technologies won first place. As per the other solutions, the general pipeline runs the grid state through an expert module before eventually sending it to the agent module. The expert module employs the usual expert rules. Wang et al. implemented a multi-step topological action combo agent through a policy network with planning. After the agent chooses the action combo, an emergency module checks whether the action still overflows the network. In that case, it takes into account dispatching actions. If even dispatching actions fail the model resorts to emergency actions. Finally, when the simulation indicates that the grid will fail at the following step, the model checks if some disconnections may save the network. Regarding the alarm mechanism, the agent raises the alarm every time it cannot solve the overflow by itself but needs to resort to the emergency module.

Discussion Simulations occupy a significant place in raising a timely alarm. In 2021 more than ever, expert systems were of utmost importance. This time, the alarm mechanism called for more complex rules and more extensive use of planning.
Enlite AI stressed the relevance of simulations by fitting Alpha Zero to the model. Alpha Zero relies heavily on planning, extensive action filtering, and a heuristic value function that allows them to apply it to power grid control. Action filtering is also central in H. Martinez's solution. In this case, a system of greedy agents takes care of running through many simulations and role out useless or inherently harmful actions. Combining actions is far from being trivial, as many combinations yield invalid grid states or induce abrupt changes, which cause cascading failures resulting in blackouts. Finally, action combos allowed Wang et al. to win first place.

Challenge	Approach	Notes
IJCNN 2019 [39]	Double Deep Q Learning	Imitation learning for initialization
WCCI 2020 [80]	Actor Critic	Semi-MDP setting with afterstates
NeurIPS 2020 [86]	Planning	Gradient-free optimization
ICAPS 2021 (3rd place) [15]	Alpha Zero	State value heuristic
ICAPS 2021 (2nd place) [15]	Teacher-Tutor	Greedy action filtering
ICAPS 2021 (1st place) [15]	Planning	Dynamic action composition

Table 6: L2RPN winners

5. Proposed Solution

5.1. Motivation

Renewable energies are posing severe challenges to power grid operational control. Solar panels and wind turbines are weather-dependent generators that have non-stationary power output. However, Europe and many other national and international organizations aim at carbon neutrality by 2050 [28, 67] to alleviate climate change.
In this context, Réseau de Transport d'Électricité (RTE) organized the L2RPN challenges [54]. All participants compete on controlling a power grid simulated through the Grid2Op [13] framework. Challenges started in 2019, and the last edition took place at WCCI 2022 (see fig. 6 for a timeline of the challenges until today). Three years later, tens of teams have used the Grid2Op module, and the power grids have grown in size and complexity.
Grid2Op shadows many power network complexities behind the RL environment interface, thus, allowing state-of-the-art models to be applied to this domain [15]. However, challenges force participants willing to achieve top scores to build instance-optimized models that are often too specific to the problem at hand to generalize to unseen topologies or more realistic scenarios. Conversely, these challenges have the merit of bridging the gap between the power system community and the RL community by lowering the barrier of entry into power network control.
Many works have been developed before and beside the L2RPN effort. In particular, a promising line of work is the application of Multi-Agent RL (MARL) to power network control [17, 78]. Such efforts have been carried out on custom simulators posing significant challenges to reproducibility and benchmarking. Therefore, we decided to bridge the more general power network control literature to the challenge format to get the best of both worlds.
We built a hierarchical, multi-agent RL system and benchmarked it on the Grid2Op environments. We devised our system with real-world scalability in mind without considering the peculiarities of any of the Grid2Op environments. This approach allows us to avoid the pitfalls of the challenge format while still getting the stability and reproducibility advantages of the L2RPN environments.

5.2. Architectural Description

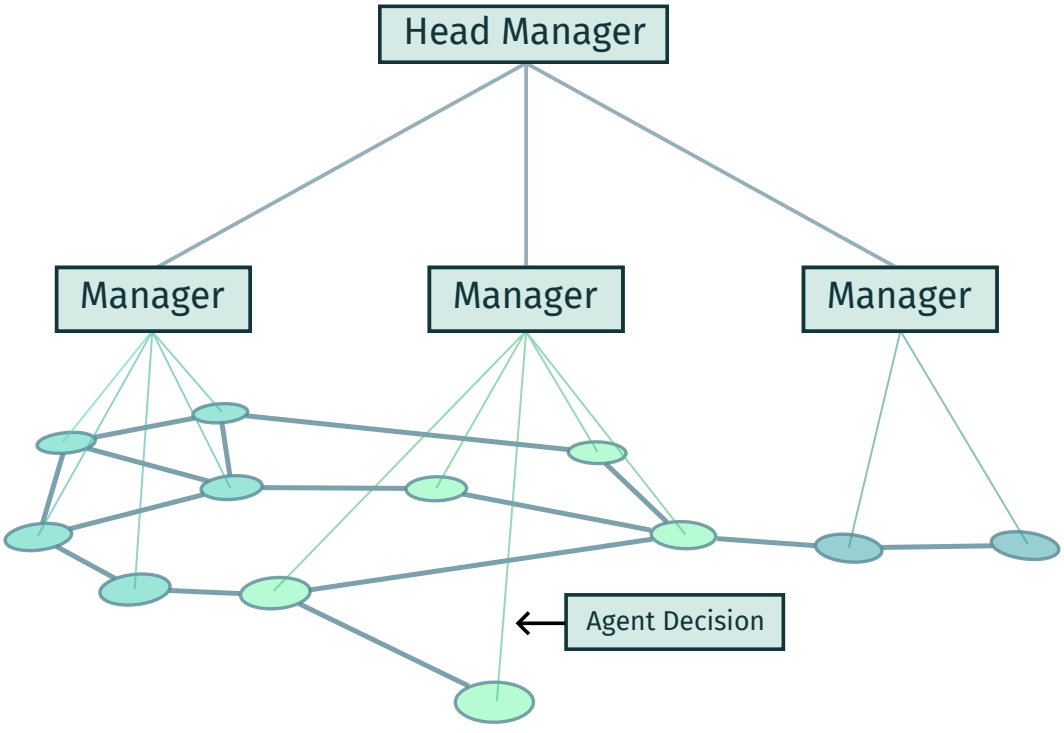


Figure 7: High-level system architecture.

We developed a hierarchical multi-agent RL system:

- **hierarchical**: different decision levels;
- **multi-agent**: multiple agents coordinate to take a decision.

The system has three main actors: substation agents, community managers, and a head manager. All system actors are RL agents.

Our design assumes that every substation in the power network can house a substation agent. A substation is a physical enclosure in which high-tension cables connect to buses, so housing a machine which runs a lightweight agent should be of no particular difficulty. Each of these agents perceive their immediate neighborhood, the measurements taken on the directly connected cables and substations. We assume it is feasible to send information from one substation to the directly connected ones as the maximum distance between the two enclosures is upper-bounded by the maximum length of high-tension cables with respect to the working voltage and the maximum tolerated electricity loss. In general, we assume it is always feasible to communicate between all actors in the system. On top of this, every substation houses a variable number of buses; in the Grid2Op framework, every substation houses two buses. Substation agents are designed to deal with any number of buses.

The graph in fig. 4 shows an example of a high-level power network model in Grid2Op. System actors see a slightly more detailed representation. In the lower-level representation, each node represents a bus, and each edge represents a cable. In principle, each substation agent deals with an arbitrary number of neighborhoods because a substation may have any number of buses inside. Therefore, the perception of a single substation agent is a sequence of graphs of arbitrary length. Each of these graphs contains one of the buses inside the substation, and the directly connected cables and buses. Generally, each of these graphs has an arbitrary number of nodes and edges depending on the network's static and dynamic topology. The static topology of the network describes the fixed number of substations in the network, the number of buses they contain, and the cables deployed. The dynamic topology describes which cables are connected to which bus in any substation at any given moment.

Each community manager handles some community of agents. Communities are detected online as the graph changes, and each community is assigned to a manager based on the previous communities it has handled. Thus, a manager perceives a subsection of the graph, called community, in which every substation agent takes an action. After substation agents decide, each manager must choose which of these decisions to submit to the head manager. Community managers may handle more than one community or none at all.

Finally, the head manager receives the proposals from the community managers and must choose one to execute. A proposal contains the substation agent chosen by a given community manager which means the action selected by such substation agent. Therefore, executing a proposal means executing the action selected by the chosen substation agent. The head manager decides given a summarized version of the graph in which every node represents a community to which the related manager attaches a decision. The only edges represented are inter-community edges.

See fig. 7 for an overview of the system architecture and algorithm 3 for an high-level description of the whole system's training loop.

In the following sections, we discuss how all system's actors embed their observations, how communities are detected on dynamic graphs and how managers are dynamically assigned to communities.

5.2.1 Graph Embedding

All actors in the system must deal with arbitrary graph topologies. Feed Forward Neural Networks [65] can only deal with fixed input sizes, thus they cannot be employed in this context. Graph Neural Networks [56] (GNNs) were introduced as a generalization of recursive neural networks that can directly deal with any graph class. Therefore, every embedding in the system has as a top section a GNN layer. In particular, we used three main layers: Graph Attention Layers (GAT) [70], Edge-GAT (EGAT) [36], and Graph Convolutional Layers [38].

Agents have an embedding section comprised of one EGAT Layer followed by a dueling network. We tried to keep agents' networks as small as possible to reduce computational complexity and to account for the fact that each agent sees a small section of the network. Managers have the same network structure but with two EGAT layers on the embedding section as in the transductive learning architecture [70]. Finally, the head manager employs two GAT layers with larger embedding sizes as more information needs to be condensed at the top level.

Substation agents and community managers perceive a graph environment with the features discussed in table 3. On the other hand, the head manager sees a summarized graph without edge features, therefore it does not use EGAT layers which embed node and edge features together. The summarized network has as node features the mean of the node embeddings over all community nodes and the managers' choice. Node embeddings are computed by the GNN layers in each manager embedding and then the mean is taken over each community. Edge features in this representation do not have a clear meaning anymore. Inter-community edges cannot be

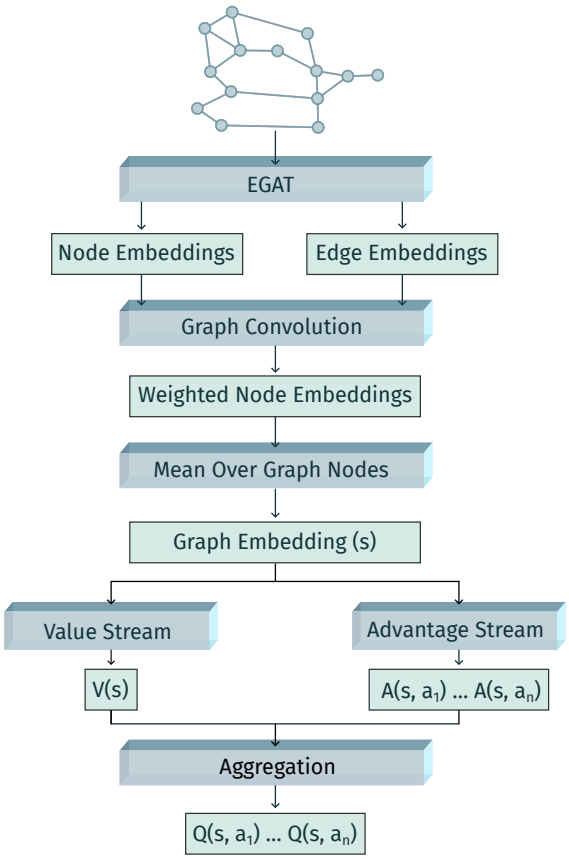


Figure 8: Substation agent architecture.

regarded as powerlines, so we removed edge features from the head manager's observations. Thus, GAT layers are used in place of EGAT layers.
After the first section of the embedding, we need to map all the node embeddings to a single graph embedding as we need to go deeper down to the dueling section, which expects a tensor as input. The main idea is that a set of node features may be mapped to a graph embedding by taking their mean over the whole graph [24]. Many other strategies are possible [24], but we opted for a more straightforward approach not to add additional complexities to the architecture. Agents and managers deal with node and edge features. Thus, we need to map node and edge features to a single tensor before taking the mean. This mapping is accomplished through a graph convolution on node features weighted with respect to the edge features. Once the node features are obtained, we are back to the head manager case, and we can extract the graph embedding from the mean of the node embeddings. For an overview of a substation agent's architecture refer to fig. 8. Managers have a the same architecture but with two EGAT layers instead of one in the graph embedding section. The hyperparameters chosen for all the actors' embeddings are discussed in appendix A.
The following sections discuss the main characteristics of the embedding layers used throughout all the system actors.

Graph Attention Layer (GAT) Attention mechanisms [4] have become almost a de facto standard in many sequence-based tasks. One of the benefits of attention mechanisms is that they allow for dealing with a variable-sized input focusing on the most relevant parts of the information to make decisions. Vaswani et al. [69] showed that attention is sufficient to build a robust model obtaining state-of-the-art performance on machine translation tasks. Inspired by this line of work, Velickovic et al. [70] introduced GATs.
The input to the layer is a set of node features $\mathbf{h} = \{h_1, \ldots, h_N\}$ with $h_i \in \mathbb{R}^F$ where N is the number of nodes and F the number of features. The layer produces a new set of node features $\mathbf{h}' = \{h'_1 \ldots h'_N\}$ with $h'_i \in R^{F'}$. The transformation is comprised of a learned linear transformation $\mathbf{W} \in \mathbb{R}^{F' \times F}$ applied to every node $h'_i = \mathbf{W}h'_i$ and then attention coefficients are computed for each pair of directly connected nodes:

$$e_{ij} = a(h'_i, h'_j) \tag{28}$$

to indicate the importance of node j's features to node i. The attention mechanism a is a single-layer feed forward neural network parametrized by a weight vector $\mathbf{a} \in \mathbb{R}^{2F'}$ and applying LeakyReLU nonlinearity. This

mechanism is applied to the concatenation h'_i and h'_j denoted $h'_i \parallel h'_j$:

$$e_{ij} = \text{LeakyReLU}(\mathbf{a} \cdot [h'_i \parallel h'_j]) \tag{29}$$

Edge-GAT (EGAT) GAT layers cannot handle edge features, but edge features are of utmost importance in power networks. Edge features encode all the information about a given powerline, without which we cannot represent the power network (table 3 discusses relevant edge features). Therefore, we chose EGAT layers [36] to leverage the attention mechanism while still using edge features.
The idea of EGAT layers is adding edge features to GAT computations by changing eq. (29) to:

$$e_{ij} = Ff'_{ij} \tag{30}$$

where F is a learnable matrix and f'_{ij} is:

$$f'_{ij} = LeakReLU(\mathbf{a} \cdot [h'_i \parallel f_{ij} \parallel h_j]), \tag{31}$$

where f_{ij} are edge features between node i and node j.

Graph Convolutional Layers A Graph convolutional layer applies a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ where F' is the output feature space, and F is the input feature space. We define the graph convolution as follows:

$$h' = \sigma\left(b + \sum_{j \in \mathcal{N}(i)} \frac{e_{ji}}{c_{ji}} h_j \mathbf{W}\right), \tag{32}$$

where σ is a non linearity (e.g. ReLU), $\mathcal{N}(i)$ is the neighbourhood of i, e_{ji} is the edge features between node j and i, $c_{ji} = \sqrt{|\mathcal{N}(j)|}\sqrt{|\mathcal{N}(i)|}$ and σ is an activation function.

5.2.2 Dynamic Community Detection

As the capacity of the power grid increases, grid structure becomes more complicated. Community detection allows us to factor the power grid in subnetworks of more manageable size. In particular, we employ it to factor in the observation space of managers. Power grids are dynamic networks which can be represented as a sequence of network snapshots changing over time $G = \{G^{(0)}, \ldots, G^{(t)}\}$ where $G_{t+1} = G^{(t)} \cup \Delta G^{(t)}$ and $\Delta G^{(t)} = (\Delta V^{(t)}, \Delta E^{(t)})$ respectively the sets of nodes and edges being changed during the time period $(t, t+1]$.
Modularity [48] is a widely used criterion to evaluate the quality of a given community structure. Community structures with high modularity have denser connections among nodes in the same communities but sparser connections among nodes from different communities. Given network $G = (V, E)$, with $n = |V|$ nodes, $m = |E|$ edges, A_{ij} the sum of the weights of all the edges between nodes i, j and k_i the sum of the weights of all edges linked to node i, its modularity is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[A_{ij} - \frac{k_i k_j}{2m}\right] \delta_{ij} = \frac{1}{2m} \sum_{c \in C} \left(\alpha_C - \frac{\beta_c^2}{2m}\right) \tag{33}$$

where $\alpha_c = \sum_{i,j \in c} A_{ij}$, $\beta_c = \sum_{i \in c} k_i$, δ_{ij} equals to 1 if i, j are in the same community else equals to 0.
Since the modularity optimization problem is known to be NP-hard, various heuristic approaches have been proposed [11, 14, 47]. Most of these algorithms have been superseded by the Louvain algorithm [7].
However, our final aim is to detect communities in a dynamic graph, the ever-changing observations of the power grid that the system receives at each timestep. In particular, the power network is represented as the set of buses connected to at least one other bus. This representation implies that the number of nodes changes together with the number of edges. This kind of representation is a constraint imposed by the Grid2Op framework. We could have preprocessed the observations to avoid node variability, but we preferred to keep preprocessing to the bare minimum to avoid additional bias as other challengers have already discussed [15]. We employ the dynamic community detection algorithm DynaMo [88] to handle dynamic graphs.
As communities change, managers need to be reassigned to new communities. In this section, we devise a mechanism to assign managers to communities similar to the ones they have handled in previous network states.
In the following sections, we describe the Louvain algorithm [7], the DynaMo algorithm [88], and the community assignment strategy.

Louvain Algorithm Louvain algorithm [7] maximizes the modularity using a greedy optimization approach composed of three steps:

- **initialization**: each vertex forms a singleton community;
- **local modularity optimization**: each vertex moves from its community to its neighbor's community to maximize the local modularity gain;
- **network compression**: nodes belonging to the same community are aggregated as super nodes, and a new network is built with the super nodes.

Louvain algorithm repeats the last two steps until the modularity improvement is negligible.

Dynamo Given a dynamic network $G = \{G^{(0)}, \ldots, G^{(t)}\}$ where $G^{(0)}$ is the initial network snapshot, let $C = \{C^{(0)}, \ldots, C^{(t)}\}$ denote the list of community structures of the corresponding network snapshots. The aim is to detect $C^{(t+1)}$ given $G^{(t)}, C^{(t)}$ and $\Delta G^{(t)}$. DynaMo[88] has three components:

- **initialization**: use Louvain algorithm [7] to compute $C^{(0)}$ which generates comparatively accurate community structure of $G^{(0)}$
- **adaptive modularity optimization (DynaMo)**: given $G^{(t)}, C^{(t)}, \Delta G^{(t)}$ update the community structure of $G^{(t+1)}$ from $C^{(t)}$ to $C^{(t+1)}$ while maximizing the modularity gain using predesigned strategies that fully depend on $\Delta G^{(t)}$ and $C^{(t)}$. This is the core component of the DynaMo framework aiming to maximize the modularity gain while maintaining efficiency.
- **refinement**: once the obtained modularity of $C^{(t+\lambda)}$ is less than a predefined threshold, use $G^{(t+\lambda)}$ as the new initial network snapshot to restart our algorithm from the initialization step. This component prevents DynaMo from being trapped in suboptimal solutions [88].

Community Assignment Dynamic community detection yields an updated community structure at every timestep. Managers must be reassigned to the new communities when the community structure gets updated. Managers may handle more than one community because graph neural networks allow managers to embed graphs with different topologies seamlessly.

We cast the manager assignment problem as a community tracking problem [22]. Community tracking deals with the evolution and structure of communities over multiple time steps in a dynamic network, where the life cycle of each community is characterized by a series of significant events such as splitting or merging. A key question concerns how to map communities at time $t+1$ to the existing communities at time t.

We decided to adapt an approach devised by Greene et al. [22] for social-network mining to our use-case. Our main concern was computational efficiency. Such a procedure is part of the training loop and is executed at each step. Therefore, we needed a simple and memory-less procedure.

Given a community structure $\mathbb{C}_t$, the next community structure detected is denoted $\mathbb{C}_{t+1}$. We want to match the communities in $\mathbb{C}_{t+1}$ with $\mathbb{C}_t$ so that managers from $\mathbb{C}_t$ handle similar communities in $\mathbb{C}_{t+1}$. The similarity between two communities c_a, c_b is measured through the Jaccard distance [33]:

$$\text{sim}(c_a, c_b) = \frac{|c_a \cap c_b|}{|c_a \cup c_b|}. \tag{34}$$

Given c to be assigned and a manager M_j that handles communities $c_1, \ldots, c_n$ we compute:

$$M_j^c = max_{c_i \in \{c_1, \ldots, c_n\}}(\text{sim}(c, c_i)), \tag{35}$$

then we assign c to the manager M maximizing M_j^c among all system's managers. Through this formulation, a community is handled by the manager assigned to the most similar community at the previous step.

5.3. Model training

In algorithm 3 we have summarized the most relevant steps in training our system. The main idea is that higher up the hierarchy the information is more condensed but more general while at the lower levels is more specific and nearer to the unfiltered environment observation. Agents see a small unfiltered portion of the graph while managers see a larger portion with the added information of the action agents would have taken at the lower level. The head manager finally sees a summarized version of the graph in which nodes become supernodes and edges no longer represent powerlines, such representation condenses all the network.

Algorithm 3 Training Algorithm

Input: Training steps T, environment $\mathcal{E}$, batch size B, learning frequency f
Deploy one agent for each substation $\{A_1, ..., A_m\}$ given m substations in $\mathcal{E}$
Deploy one head manager H
Observe the initial observation s_0 from $\mathcal{E}$
Detect community structure in s_0 and assign one manager per community $\{M_1, \ldots, M_c\}$ given c communities in s_0
for $i = 1 \ldots t$ **do**
  Each agent A_j observes its current neighborhood $\mathcal{N}_{A_j}(s_i)$ and picks an action a_{A_j}
  Each manager M_k observes assigned communities $\mathcal{C}_{M_k}(s_i|\mathbf{a_k})$ with $\mathbf{a_k}$ being the actions community members have picked, and chooses agents A_{M_k}
  The head manager observes the summarized graph $\mathcal{S}(s_i|A_{M_1}, \ldots, A_{M_c})$ and chooses one community C handled by manager M
  Execute action $a = a_{A_M}$, receive next observation s_{i+1} and reward r
  Detect community structure in s_{i+1} and assign managers to new communities (refer to section 5.2.2)
  Agent A_M registers the transition $(\mathcal{N}_{A_M}(s_i), a, \mathcal{N}_{A_M}(s_{i+1}), r)$ in the replay buffer
  Each agent A_j observes its neighborhood $\mathcal{N}_{A_j}(s_{i+1})$ and picks an action a'_{A_j}
  Manager M registers the transition $(\mathcal{C}_{M_k}(s_i|\mathbf{a_k}), A_M, \mathcal{C}_{M_k}(s_{i+1}|\mathbf{a'_k}), r)$ in the replay buffer
  Each manager M_k observes assigned communities $C_{M_k}(s_{i+1}|\mathbf{a'_k})$ and chooses an agent A'_{M_k}
  Head manager H registers the transition $(\mathcal{S}(s_i|A_{M_1}, \ldots, A_{M_c}), M, \mathcal{S}(s_i|A'_{M_1}, \ldots, A'_{M_c}), r)$ in the replay buffer
  if $i \mod f = 0$ **then**
    Each actor in the system updates its weights by sampling B transitions from the replay buffer
  end if
end for

6. Experimental Results

6.1. Experimental Setup

Training a Multi-Agent Deep RL system is a notoriously challenging task [16, 49] requiring careful system tuning. In the following sections, we discuss the most relevant training choices adopted throughout all the experiments. Refer to appendix A for further discussion on hyperparameter values.

6.1.1 Expert System

We paired our RL system with a rule-based expert system with the following rules:

- **line reconnection**: reconnect a disconnected powerline every time there is a chance to do so;
- **emergency handling**: if no line exceeds the overflow threshold, do nothing; otherwise query the RL system.

The use of this system is dictated by the complexity of learning the whole operational control task; this way, the agent only learns to deal with emergencies. The emergency handling rule depends on an overflow threshold. The overflow threshold is a limit value for the maximum load on any of the powerlines and goes from 0, representing no load, to 1, representing overload. We set it to 0.99 [58]. The higher the threshold, the fewer times the RL system is queried. We decided to stick to the expert system developed by Serre et al. [58] for the baseline of the WCCI 2022 competition. Using a simple expert system emphasizes the behavior of the learning agent. Many challengers employed far more complex expert systems [80]. Since the expert system handles reconnection actions, we restricted the RL system to operating on generators. In particular, the RL system may redispatch non-renewable generators or curtail renewable generators (see section 1.1 for further discussion on redispatching and curtailing).

6.1.2 Optimization Procedure

Each agent in the system has its optimizer to learn network parameters. All the optimizers in the system are Adam optimizers [37] as it is a widely adopted solution in practice [3, 27, 46]. Adam optimizers have two main parameters of interest for RL, the learning rate and the Adam-ϵ. The learning rate determines how significant weight updates are, thus influencing the system's sensibility to local optima and the overall training stability. The Adam-ϵ is used to avoid zero division during the optimization procedure. While usually kept very low, such hyperparameter brings significant regularization benefits in the RL domain [27] so we decided to set it at 10^{-3}. Such value is far higher than the usual value for supervised learning 10^{-8} [52]. While a popular choice in practice, Adam is subject to exploding gradients as many others optimization procedures [51]. To tackle such a problem, we clip the gradients computed by Adam at 40 before backpropagating them. This procedure has been popularized by Hessel et al. [27] and has been widely adopted since then.

6.1.3 Reward and Scoring

The agent receives 1 for every timestep in which no blackout happens otherwise it receives -1 and the episode ends. The interval has been chosen because it has been found to have stabilizing effects on DQNs [46]. Then, the reward structure has been kept as simple as possible to avoid introducing bias in the learning process.
One of the advantages of the challenge format is agent scoring. Practitioners can use any reward during training, but all agents are evaluated with the same scoring function in the evaluation phase. As a scoring function, we used the WCCI 2022 scoring function. The score is computed over episodes: scenarios over which the system took action. The score function represents the cost of operations of a power grid, including the cost of any blackout that could occur. The cost of operations is defined as:

$$c_{\text{operations}}(t) = c_{\text{losses}}(t) + c_{\text{redispatching}}(t) + c_{\text{curtailment}}(t), \tag{36}$$

and the cost of a blackout is defined as:

$$c_{\text{blackout}}(t) = \text{load}(t) \times p(t), \tag{37}$$

where $p(t)$ is the marginal price of electricity and $\text{load}(t)$ is the overall grid load at time t. This score penalizes agents that redispatch generators too often. Then, it also penalizes blackouts when the electricity demand (i.e., the load) is high. Finally, the score is scaled between 0 and 100 where 0 is the score achieved by the Do-Nothing Agent (DNA). The DNA is an agent which does nothing at each timestep.

6.1.4 Exploration Strategy

We employ the ϵ-greedy strategy with an exponentially decaying ϵ [46]. The idea is that at the beginning of the learning process the agent should explore more while exploiting the acquired knowledge later (refer to section 2.3.2 for further discussion on ϵ-greedy policies).
On top of this, we also use an increasing prioritization β [57] from 0.5 to 1 . The idea is to employ a more uniform sampling in the first exploration phases while activating prioritization in later stages when the TD errors have stabilized (refer to section 2.3.5 for further discussion on β).

6.1.5 Benchmarks

Challenge organizers propose the DNA as a strong benchmark [58]. Power networks are highly non-linear systems. Therefore, many actions induce cascading effects to the point of causing undesired blackouts. The DNA does not do anything, thus avoiding any destructive action.
Another strong benchmark is the expert system described in section 6.1.1. Such a system can be used as a benchmark by pairing it with the DNA. The idea is to follow the reconnection policy and do nothing in emergencies.
The expert system and the DNA achieve the same score on the L2RPN Case 14 Environment. They both score 0 as discussed in section 6.1.3. They achieve such a score because the environment has neither powerline maintenance nor an attacker. Therefore, the line reconnection policy of the expert system has no effect and we employ the DNA as a benchmark in the L2RPN Case 14 Environment.
On the NeurIPS 2020 Environment, the expert system achieves a higher score than the DNA. The line reconnection policy correctly deals with some of the challenges introduced by maintenance and attackers. We will use the expert system as a benchmark to correctly estimate the performance improvement introduced by our RL system.

6.2. Results on the L2RPN Case 14 Environment

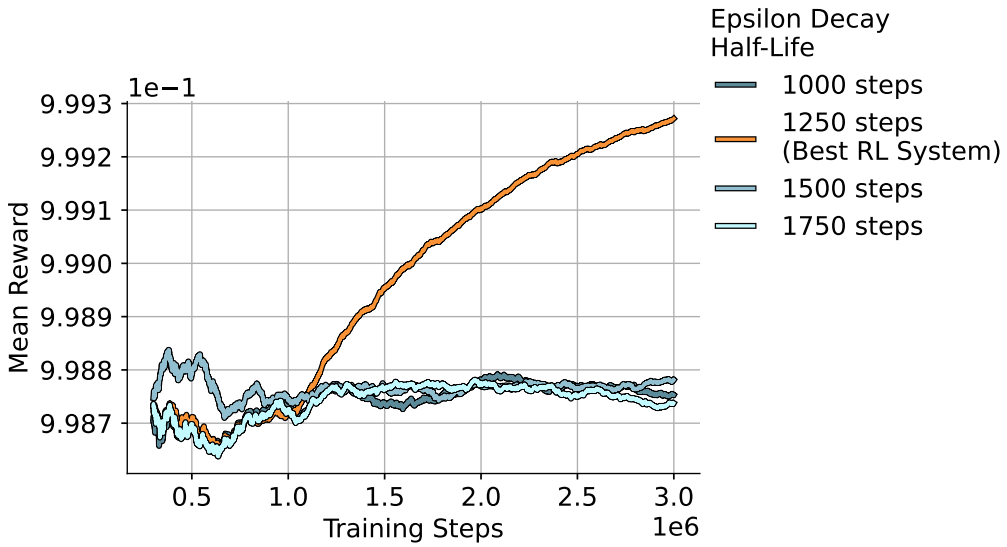


Figure 9: Multiple training runs with different values for the exponential ϵ-decay half-life in the L2RPN case 14 Environment. The ϵ-decay half-life is set equal for agents, managers and head manager.

We conducted the first set of tests on the L2RPN Sandbox Case 14 environment. This environment uses the IEEE case14 power grid [32] with two added generators. It counts 14 substations, 20 powerlines, 6 generators, and 11 loads. Refer to fig. 4 for an overview of the grid.
In fig. 9, we show the rewards achieved by the system when changing the ϵ-decay half-life. We can see that the system is remarkably susceptible to such a hyperparameter. There is a sensibly different result for one specific value. This behavior is to be attributed to the existence of two local maxima in the environment at hand, one far higher than the other. By changing the decay rate of ϵ, we expose the agents to different degrees of exploration, which means that they explore different portions of the state-action space with varying levels of extensiveness. If the exploration phase is too short, the agent does not evaluate a sufficient portion of the state-action space, thus often finding the most prominent local maxima.

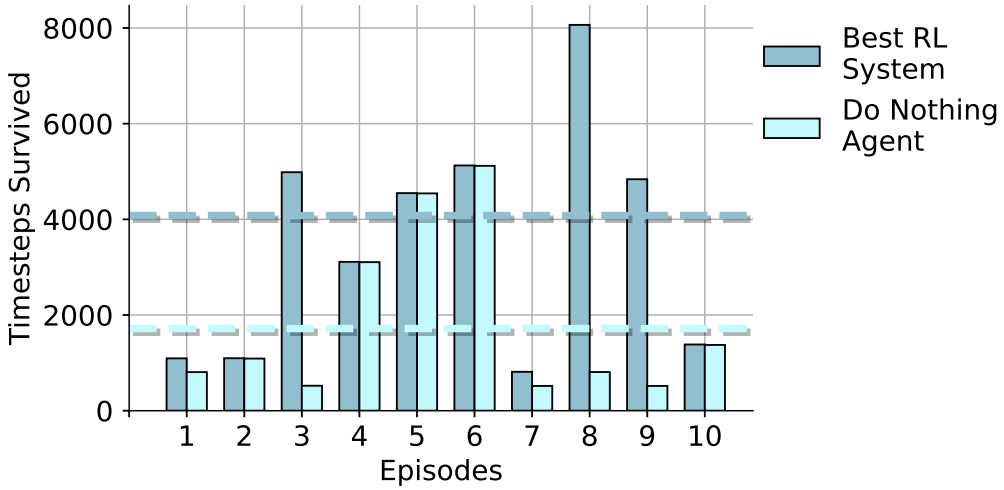


Figure 10: Timesteps survived by the best trained RL agent and the Do Nothing agent on 10 test episodes in the L2RPN case 14 Environment.

On the other hand, in case of excessive exploration, the agent may encounter a local maximum by randomly trying actions and, while the exploration phase declines, miss sufficient incentives to deviate to other policies. Both these two phenomena are visible in fig. 9, mainly due to the limited size of the environment. Nonetheless, this environment allows us to show the correct functioning of our system in optimizing the long-term cumulated reward. In this context, such consideration is not trivial given that all actors except for the head manager have partial observations of the MDP [40] and the head manager sees a summary of the whole observation. On top of this, the only actors taking direct action on the system are the substation agents, while all other actors can only choose among those actions.
The best agent, the one with an ϵ-decay half-life of 1250, achieves a mean score of 35 over 100 on a 30-episode test set. Such a score is very promising for a system without optimization for the specific instance. The challenge organizers on a larger environment with a single-agent model achieve a score of 22 over 100 with almost twice the training steps [58]. While their environment is far larger, this sets a reference to interpret the score.
In fig. 10, we show the timesteps the best RL system survived with respect to those of the DNA. We show the survived timesteps because the scores are compound metrics that can be hard to interpret relative to the simpler concept of steps survived throughout an episode without causing a blackout. On top of this, challenge organizers scale the score between 0 and 100 by setting at 0 the score of the DNA. In this instance, survived timesteps give a more precise performance outline. Our agent defeats or equals the DNA in every episode, thus confirming the positive performance announced by the mean score. It is essential to notice that our agent, at worst, equals the DNA. Such behavior is not always expected. In larger environments, the action space is larger and thus contains many more destructive actions. Learning to avoid destructive actions can be challenging when dealing with large action spaces.

6.3. Results on the NeurIPS 2020 Environment

This set of tests is conducted on the NeurIPS 2020 robustness track environment. This environment is part of the IEEE 118 grid [31], where some generators have been added. It counts 36 substations, 59 powerlines, 22 generators, and 37 loads. Some loads represent interconnections with other grid sections, meaning there can be negative loads. On top of this, an opponent will randomly attack some lines of the grid every day. Finally, challenge designers introduced planned maintenance that disconnects powerlines for a given time. The expert system handles maintenance and attacks; every time a line gets disconnected, the expert system reconnects it, as per section 6.1.1. Therefore, the expert system, which selected as benchmark in section 6.1.5 scores 7.39 points more than the DNA.
In fig. 11, we repeated the tests we conducted on the case 14 environment. Firstly, the impact of ϵ is far less dramatic than in section 6.2. This behavior is to be referred to the larger size of the environment, which makes it less prone to local maxima. The most striking behavior is that of the model with 1500 steps as ϵ-decay half-life. We notice the training process converging to a local maximum with a mean reward of 0.994 and being unable to further optimize the mean reward. Power networks yield many of these local maxima due to their highly non-linear dynamics. All the other tests do not get trapped in such maximum. The agent with the largest half-life is the best performing among our experiments. As expected, in a larger and more complex environment

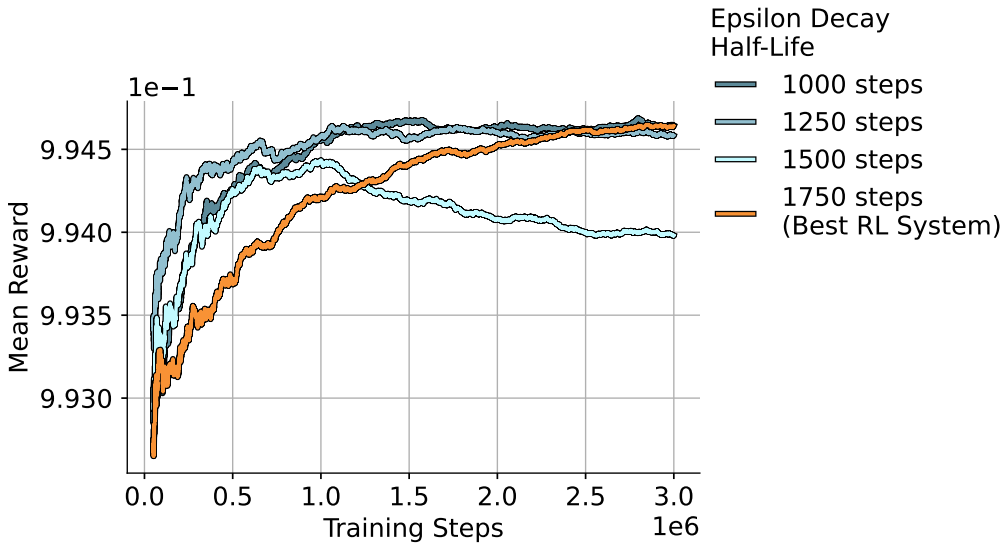


Figure 11: Multiple training runs with different values for the exponential ϵ decay half-life in the L2RPN NeurIPS 2020 Environment. The ϵ-decay half-life is set equal for agents, managers and the head manager.

thorough exploration is crucial.
Figure 12 shows the best agent's performance. As discussed in section 6.1.3, the score is computed by summing the operational cost of managing the power network and the costs induced by the possible blackout. In this case, we decided to show the scores and not the timesteps survived due to the greater weight the operational costs have in this network with respect to the smaller one. A larger network has far higher and more variable operational costs, thus inducing greater variability in the score with respect to the timesteps survived. As depicted in fig. 12, the expert agent performs worse than the Do Nothing Agent in episode 7. The expert system survives longer but the blackout happens in a moment with higher loads in the networks, thus inducing a cost that exceeds the benefits of having survived more. In the same episode, our system can find a policy that makes the system perform better than the DNA. In general, our agent achieves a mean score of 10.09 over 30 episodes, which is 2.70 points larger than the expert system alone and significantly higher than the DNA.

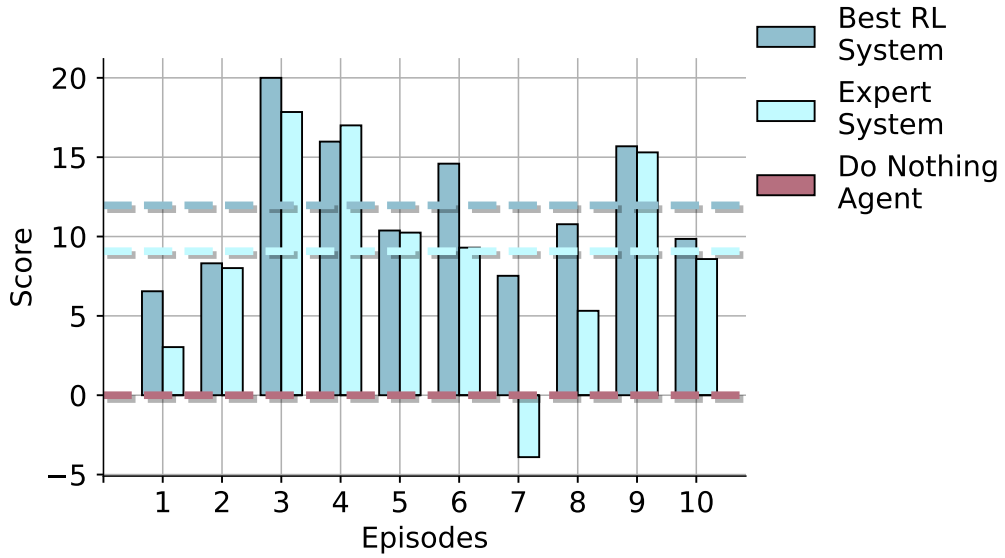


Figure 12: Score of the best trained RL system with respect to the expert system and the DNA on 10 test episodes in the L2RPN NeurIPS 2020 Environment

7. Conclusions and Future Work

7.1. Conclusions

The advent of renewable energy on the production side poses significant challenges to grid operators. For this reason, RTE instituted the L2RPN challenge [45, 54]: a series of competitions that model the sequential decision-making environments of real-time power network operation. The power systems community has been lately focusing on Deep RL [59] solutions for their capacity to learn representations and their parallelizable architectures.
Consequently, much literature focuses on applying RL models to power grid operational control. On the one hand, several authors [76, 78, 79, 87] have focused on multi-agent RL systems to deal with the size of real-world power networks. On the other hand, the challenge format asks participants to build ad-hoc solutions incentivizing single-agent models [15, 39, 80, 86].
All this considered, we decided to take advantage of the maturity of the L2RPN ecosystem without taking part in any challenge. The L2RPN setting provides standardized environments and evaluation procedures, which are vital for reproducibility and benchmarking. Conversely, the challenge setup forces participants to build throw-away models optimized for the task. We thus decided to build a model and evaluate it on the L2RPN environments without taking part to any competition.
We developed a novel hierarchical multi-agent RL system. The system has three main actors: substation agents, community managers, and a head manager. Every substation houses an agent which, at every timestep, must select an action given its immediate neighborhood. Every community of agents is handled by a manager which, at every timestep, must choose an agent given the actions they have selected. Finally, the head manager must choose one manager given the agent they have selected.
The hierarchical structure we described allows us to decrease the action space complexity with respect to the grid size. Thus, we say it is scalable. Substation agents see an action space that depends only on their immediate neighborhood. Managers have an action space size upper-bounded by the number of community in the networks, which is usually much smaller than the number of substations.
We significantly reduce the action space handled by single agents by dynamically factorizing it over the graph topology. The factorization is dynamic because the communities are updated at each timestep. Thus, the number of choices each community manager evaluates changes with network topology.
At training time, the RL system has been paired with an expert system introduced in the 2022 baseline by challenge organizers [58]. Participants have widely used expert systems far more complex than ours [80]. Our expert system greedily reconnects the lines when it can and does nothing except for emergencies when it queries the RL system. As a result, we significantly reduce the scope of the task from complete operational control to emergency handling while still giving the utmost relevance to the RL system.
We scored the system on the L2RPN case 14 environment and the NeurIPS 2020 environment using the same scoring as the WCCI 2022 challenge. On the L2RPN case 14 environment, we scored 35 out of 100 points above the do nothing agent. This result is encouraging with respect to the 22 points obtained by challenge organizers with an ad-hoc single-agent approach with more training time on a larger environment [58]. On the L2RPN NeurIPS 2020 environment, the expert system alone scores 7.3, and our whole system scores 10.09. Scoring above the expert system in such a large environment demonstrates the feasibility of multi-agent power network control. In particular, we validated our approach's soundness, showing how the system correctly optimizes the cumulated reward against 36 substations and 22 controllable generators.
Concluding, in our work, we developed a novel hierarchical multi-agent RL architecture and tested it on the L2RPN environments. We showed the feasibility of our approach and tested different exploration setups.
We sincerely hope scientific challenges will become meaningful collaboration opportunities rather than mere competitions.

7.2. Future Work

7.2.1 Intrinsically Motivated RL

Throughout all our experiments, we used the same exploration strategy for all agents, managers, and the head manager. However, every agent and manager sees a different section of the grid and thus needs a specific degree of exploration dependent on the actual complexity of this task. Devising such an exploration strategy for each agent is infeasible and hard to scale to large topologies.
Intrinsic motivation plays a prominent role in human development and learning, and researchers in many areas of cognitive science have emphasized that intrinsically motivated behavior is vital for intellectual growth. Intrinsically motivated RL [5, 82] deals with applying intrinsic rewards to drive an agent's policy optimization

process. Barto et al. [5] used the term intrinsic reward to refer to rewards that produce analogs of intrinsic motivation in RL agents and extrinsic reward to refer to rewards that define a specific task or rewarding outcome as in standard RL applications.
One of the most prominent exploration strategies based on intrinsic motivation is curiosity-driven exploration [50]. Pathak et al. define curiosity as the error in an agent's ability to predict the consequence of its actions and use that as an intrinsic reward signal.
Our system would greatly benefit from implicit motivation approaches. Each actor perceives different sections of the environment. Hence, implicitly motivated exploration allows diverse exploration strategies adapted to each agent's task. Specific exploration strategies for each agent [34] would directly follow from the difference in their perception, thus pushing our system towards a more generalizable form.

7.2.2 Pointer Networks

Managers deal with variable inputs because they need to choose among a varying number of agents. We can model their task as a ranking among the input agents. A community manager must choose the best agent from a community, where the best one is the one that maximizes the cumulated reward when chosen by the head manager. In our solution, we decided to use managers that output a vector of preferences for each agent and take as the choice the preferred one. This approach has one major drawback. The manager needs to output a vector with a length equal to the number of substations. Such architecture is dictated by the fact that, in the worst case, a manager may handle a community covering the whole network. In our implementation, we mask all the agents which do not participate in the community at hand and choose the preferred one by taking the highest entry among the non-masked values.
Our approach to community managers hinders real-world scalability as the output vector size of each manager is tied to the number of substations of the network. We propose to implement the decision layer of managers with Pointer Networks [71]. A pointer network learns the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. Such a mechanism allows the selection of the best element from a variable-sized input sequence. On top of this, Graph Pointer Networks [41] and Hybrid Pointer Networks [63] show successful pointer networks applications to graphs.
On top of restricting managers' output layer, pointer networks would also allow generalizing the whole system further. Through a pointer network, managers may choose the best-k agents from a community, thus exposing more choices to the head manager.
Concluding, pointer networks and their graph applications give further flexibility to managers and head managers while breaking the explicit dependency by the number of substations and the decision surface of each manager.

7.2.3 Power Supply Modularity

Louvain Modularity (see eq. (33)) is a purely topological measure. Therefore, our community detection algorithm ignores the electrical characteristics of the network. To solve this issue, we propose to adopt Power Supply Modularity [72] as an alternative measure to replace the Louvain modularity in DynaMo [88]. Wang et al. propose Power Supply Modularity to consider the complex electrical properties and the functionality of power networks. The EFS matrix is proposed to reflect the characteristics of buses and lines regarding power transmission efficiency. The EFS matrix is the sum between the two buses' electrical coupling strength and the power supply strength. The electrical coupling strength between two buses is the sum between transmission capacity and admittance. The power supply strength quantifies the association between generators and loads.
Such an extension to the community detection procedure would allow the system to stress the power network's electrical characteristics further and reflect them in the community structure.

7.2.4 Recurrent DQNs

One of the main issues in training our RL system is partial observability [40]. Agents and managers see just a portion of the whole state, thus perceiving stochasticity in environment dynamics. Hausknecht et al. introduced deep recurrent Q-learning [26] to deal with Partially Observable MDPs (POMDPs) [62]. The idea is to add a Long-Short Term Memory Module (LSTM) [81] to the agent embedding to filter out the stochasticity by integrating over a sequence of observations. Adding LSTMs to each agent subject to partial observability would significantly increase the system's computational cost. On the other hand, we expect LSTMs to greatly enhance each actor's performance, potentially giving considerable system benefits.

References

[1] Rawad Abdulghafor, Shahrum Shah Abdullah, Sherzod Turaev, and Mohamed Othman. An overview of the consensus problem in the control of multi-agent systems. *Automatika*, 59(2):143–157, April 2018.

[2] International Energy Agency. Greenhouse Gas Emissions from Energy Data Explorer – Data Tools. https://www.iea.org/data-and-statistics/data-tools/greenhouse-gas-emissions-from-energy-data-explorer, 2022.

[3] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, and Charles Blundell. Never Give Up: Learning Directed Exploration Strategies, February 2020.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, May 2016.

[5] Andrew G Barto, Satinder Singh, and Nuttapong Chentanez. Intrinsically Motivated Learning of Hierarchical Collections of Skills. *Proc. Int. Conf. Develop. Learn.*, page 8, 2004.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

[7] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.*, 2008(10):P10008, October 2008.

[8] Dmitrii Bogdanov, Ashish Gulagi, Mahdi Fasihi, and Christian Breyer. Full energy sector transition towards 100% renewable energy supply: Integrating power, heat, transport and industry sectors including desalination. *Applied Energy*, 283:116273, February 2021.

[9] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, zeroth edition, July 2017.

[10] Xiuhua Cai, Hongxing Cao, Xiaoyi Fang, Jingli Sun, and Ying Yu. A View for Atmospheric Unpredictability. *Frontiers in Earth Science*, 9, 2021.

[11] aaron clauset, m. e. j. newman, and cristopher moore. finding community structure in very large networks. *phys. rev. e*, 70(6):066111, December 2004.

[12] Ruisheng Diao, Zhiwei Wang, Di Shi, Qianyun Chang, Jiajun Duan, and Xiaohu Zhang. Autonomous Voltage Control for Grid Operation Using Deep Reinforcement Learning. In *2019 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, Atlanta, GA, USA, August 2019. IEEE.

[13] Benjamin Donnot. Grid2Op. Réseau de transport d'électricité, October 2022.

[14] J. Duch and A. Arenas. Community detection in complex networks using Extremal Optimization. *Phys. Rev. E*, 72(2):027104, August 2005.

[15] EPRI. RTE and EPRI Present Learning to Run a Power Network L2RPN 2021 – Towards an Machine Learning Based, February 2022.

[16] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning, May 2018.

[17] Yuanqi Gao, Wei Wang, and Nanpeng Yu. Consensus Multi-Agent Reinforcement Learning for Volt-VAR Control in Power Distribution Networks. *arXiv:2007.02991 [cs, eess]*, July 2020.

[18] Elizabeth Gibney. Could machine learning fuel a reproducibility crisis in science? https://www.nature.com/articles/d41586-022-02035-w, 2022.

[19] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry, June 2017.

[20] Mevludin Glavic, Raphaël Fonteneau, and Damien Ernst. Reinforcement Learning for Electric Power System Decision and Control: Past Considerations and Perspectives. *IFAC-PapersOnLine*, 50(1):6918–6927, July 2017.

[21] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, July 2005.

[22] Derek Greene, Dónal Doyle, and Padraig Cunningham. *Tracking the Evolution of Communities in Dynamic Social Networks*, volume 2010. IEEE, August 2010.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, August 2018.

[24] William L Hamilton. Graph Representation Learning. *self-published*, page 141, 2020.

[25] Hado Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[26] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs, January 2017.

[27] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning, October 2017.

[28] White House. ICYMI: President Biden Signs Executive Order Catalyzing America's Clean Energy Economy Through Federal Sustainability | CEQ. https://www.whitehouse.gov/ceq/news-updates/2021/12/13/icymi-president-biden-signs-executive-order-catalyzing-americas-clean-energy-economy-through-federal-sustainability/.

[29] Qiuhua Huang, Renke Huang, Weituo Hao, Jie Tan, Rui Fan, and Zhenyu Huang. Adaptive Power System Emergency Control Using Deep Reinforcement Learning. *IEEE Transactions on Smart Grid*, 11(2):1171–1182, March 2020.

[30] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, April 2017.

[31] ICSEG. IEEE 118-Bus System - Illinois Center for a Smarter Electric Grid (ICSEG).

[32] ICSEG. IEEE 14-Bus System - Illinois Center for a Smarter Electric Grid (ICSEG).

[33] Paul Jaccard. The Distribution of the Flora in the Alpine Zone. *The New Phytologist*, 11(2):37–50, 1912.

[34] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A Ortega, DJ Strouse, and Joel Z Leibo. Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning. *J. Compar. Physiol. Psychol.*, page 10, 1950.

[35] Michael P. Wellman Junling Hu. Nash q-learning for general-sum stochastic games | The Journal of Machine Learning Research. https://dl.acm.org/doi/abs/10.5555/945365.964288.

[36] Kamil Kamiński, Jan Ludwiczak, Maciej Jasiński, Adriana Bukala, Rafal Madaj, Krzysztof Szczepaniak, and Stanisław Dunin-Horkawicz. Rossmann-toolbox: A deep learning-based protocol for the prediction and design of cofactor specificity in Rossmann fold proteins. *Brief Bioinform*, 23(1):bbab371, September 2021.

[37] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[38] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017.

[39] Tu Lan, Jiajun Duan, Bei Zhang, Di Shi, Zhiwei Wang, Ruisheng Diao, and Xiaohu Zhang. AI-Based Autonomous Line Flow Control via Topology Adjustment for Maximizing Time-Series ATCs, November 2019.

[40] Qinghua Liu, Alan Chung, Csaba Szepesvári, and Chi Jin. When Is Partially Observable Reinforcement Learning Not Scary?, May 2022.

[41] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning, November 2019.

[42] Antoine Marot, Benjamin Donnot, Karim Chaouache, Adrian Kelly, Qiuhua Huang, Ramij-Raja Hossain, and Jochen L. Cremer. Learning to run a power network with trust, April 2022.

[43] Antoine Marot, Benjamin Donnot, Gabriel Dulac-Arnold, Adrian Kelly, Aïdan O'Sullivan, Jan Viebahn, Mariette Awad, Isabelle Guyon, Patrick Panciatici, and Camilo Romero. Learning to run a Power Network Challenge: A Retrospective Analysis. *arXiv:2103.03104 [cs, eess]*, October 2021.

[44] Antoine Marot, Benjamin Donnot, Camilo Romero, Luca Veyrin-Forrer, Marvin Lerousseau, Balthazar Donon, and Isabelle Guyon. Learning to run a power network challenge for training topology controllers. *arXiv:1912.04211 [cs, eess, stat]*, December 2019.

[45] Antoine Marot, Isabelle Guyon, Benjamin Donnot, Gabriel Dulac-Arnold, Patrick Panciatici, Mariette Awad, Aidan O'Sullivan, Adrian Kelly, and Zigfried Hampel-Arias. L2RPN: Learning to Run a Power Network in a Sustainable World NeurIPS2020 challenge design. *arxiv*, page 26, 2021.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[47] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, September 2006.

[48] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, June 2006.

[49] Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving Stability in Deep Reinforcement Learning with Weight Averaging. *arXiv*, page 5, 2017.

[50] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction, May 2017.

[51] George Philipp, Dawn Song, and Jaime G. Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions, April 2018.

[52] PyTorch. Adam — PyTorch 1.13 documentation. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html, 2022.

[53] R. Rocchetta, L. Bellani, M. Compare, E. Zio, and E. Patelli. A reinforcement learning framework for optimal operation and maintenance of power grids. *Applied Energy*, 241:291–301, May 2019.

[54] EPRI RTE. L2RPN challenge. https://l2rpn.chalearn.org/.

[55] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning, September 2017.

[56] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009.

[57] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay, February 2016.

[58] Gaëtan Serré, Eva Boguslawski, Benjamin Donnot, Adrien Pavão, Isabelle Guyon, and Antoine Marot. Reinforcement learning for Energies of the future and carbon neutrality: A Challenge Design, July 2022.

[59] Joohyun Shin, Thomas A. Badgwell, Kuang-Hung Liu, and Jay H. Lee. Reinforcement Learning – Overview of recent progress and implications for process control. *Computers & Chemical Engineering*, 127:282–294, August 2019.

[60] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[61] Satinder P. Singh and Richard S. Sutton. Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning*, 22(1):123–158, January 1996.

[62] Matthijs T J Spaan. Partially Observable Markov Decision Processes. *Reinforcement Learning*, page 27, 2012.

[63] Ahmed Stohy, Heba-Tullah Abdelhakam, Sayed Ali, Mohammed Elhenawy, Abdallah A. Hassan, Mahmoud Masoud, Sebastien Glaser, and Andry Rakotonirainy. Hybrid Pointer Networks for Traveling Salesman Problems Optimization. *PLoS ONE*, 16(12):e0260995, December 2021.

[64] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction*. MIT Press, November 2018.

[65] Daniel Svozil, Vladimír Kvasnicka, and Jiří Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, 39(1):43–62, November 1997.

[66] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte Carlo Tree Search: A review of recent modifications and applications. *Artif Intell Rev*, July 2022.

[67] European Union. 2050 long-term strategy. https://climate.ec.europa.eu/eu-action/climate-strategies-targets/2050-long-term-strategy_en.

[68] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning, December 2015.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017.

[70] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks, February 2018.

[71] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. *arXiv:1506.03134 [cs, stat]*, January 2017.

[72] Xiaoliang Wang, Fei Xue, Shaofeng Lu, Lin Jiang, Ettore Bompard, and Marcelo Masera. Understanding Communities From a New Functional Perspective in Power Grids. *IEEE Systems Journal*, 16(2):3072–3083, June 2022.

[73] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv*, page 9, 2020.

[74] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Mach Learn*, 8(3):279–292, May 1992.

[75] Papers with Code. Papers with Code - Atari 2600 Venture Benchmark (Atari Games). https://paperswithcode.com/sota/atari-games-on-atari-2600-venture.

[76] Lei Xi, Jianfeng Chen, Yuehua Huang, Yanchun Xu, Lang Liu, Yimin Zhou, and Yudan Li. Smart generation control based on multi-agent reinforcement learning with the idea of the time tunnel. *Energy*, 153:977–987, June 2018.

[77] Qiuling Yang, Gang Wang, Alireza Sadeghi, Georgios B. Giannakis, and Jian Sun. Two-Timescale Voltage Control in Distribution Grids Using Deep Reinforcement Learning. *IEEE Transactions on Smart Grid*, 11(3):2313–2323, May 2020.

[78] Linfei Yin, Tao Yu, and Lv Zhou. Design of a Novel Smart Generation Controller Based on Deep Q Learning for Large-Scale Interconnected Power System. *Journal of Energy Engineering*, 144(3):04018033, June 2018.

[79] Linfei Yin, Lulin Zhao, Tao Yu, and Xiaoshun Zhang. Deep Forest Reinforcement Learning for Preventive Strategy Considering Automatic Generation Control in Large-Scale Interconnected Power Systems. *Applied Sciences*, 8(11):2185, November 2018.

[80] Deunsol Yoon, Sunghoon Hong, Byung-Jun Lee, and Kee-Eung Kim. Winning the L2RPN Challenge: Power Grid Management via Semi-Markov Afterstate Actor-Critic. In *International Conference on Learning Representations*, September 2020.

[81] Yong Yu. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures | Neural Computation | MIT Press. https://direct.mit.edu/neco/article/31/7/1235/8500/A-Review-of-Recurrent-Neural-Networks-LSTM-Cells.

[82] Mingqi Yuan. Intrinsically-Motivated Reinforcement Learning: A Brief Introduction, June 2022.

[83] Xiao Shun Zhang, Qing Li, Tao Yu, and Bo Yang. Consensus Transfer Q -Learning for Decentralized Generation Command Dispatch Based on Virtual Generation Tribe. *IEEE Transactions on Smart Grid*, 9(3):2152–2165, May 2018.

[84] Xiaoshun Zhang. Virtual generation tribe based robust collaborative consensus algorithm for dynamic generation command dispatch optimization of smart grid | Elsevier Enhanced Reader.

[85] Zidong Zhang, Dongxia Zhang, and Robert C. Qiu. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems*, 6(1):213–225, March 2020.

[86] Bo Zhou, Hongsheng Zeng, Yuecheng Liu, Kejiao Li, Fan Wang, and Hao Tian. Action Set Based Policy Optimization for Safe Power Grid Management, June 2021.

[87] Zhi-Hua Zhou and Ji Feng. Deep Forest: Towards An Alternative to Deep Neural Networks. *Electronic proceedings of IJCAI 2017*, pages 3553–3559, 2017.

[88] Di Zhuang, J. Morris Chang, and Mingchen Li. DynaMo: Dynamic Community Detection by Incrementally Maximizing Modularity. *IEEE Transactions on Knowledge and Data Engineering*, 33(5):1934–1945, May 2021.

A. Training Hyper-Parameters

	Value	Description
Safe max-ρ	0.99	Expert system threshold over which the RL system handles an emergency
Learning rate	1e-5	Optimizers' learning rate for gradient descent
Learning frequency	4	Number of training steps between weight updates
Gradient Clipping	40	Gradients clip value before backpropagation
Loss Function	Huber Loss	Loss minimized by network optimizers
Huber-δ	2	huber loss clips gradients to δ for residual absolute values larger than δ
Replacement steps	100	Training steps to wait before copying parameters from the Q network to the target network
Batch Size	288	Number of observations sampled from the replay buffer on each weight update
Adam-ϵ	1.5e-4	Adam parameter for numerical stability, avoids division by zero during optimization
Exploration-ϵ half-life	Case 14: 1250 NeurIPS 2020: 1750	ϵ exponential decay half-life, governs the length of the exploration phase
Exploration-ϵ Range	[0, 1]	ϵ exponential decay limits starts from 1 and decays to 0
Prioritization-α	0.7	Determines the prioritization degree when sampling from the replay buffer, goes from 0 to 1
Prioritization-β Range	[0.5, 1]	Determines the bias correction when computing priorities in the replay buffer
Prioritization-β Grow Rate	half of ϵ half-life	Determines how bias correction changes in the replay buffer during learning
Replay Buffer Capacity	Agents: 1e3 Managers: 1e4 Head Manager: 1e5	Old transitions are replaced by new ones when capacity is over
Graph Embedding Depth	Agents: 2 Managers: 3 Head Manager: 3	Number of layers in the embedding section
Graph Embedding Size	Agents: 128 Managers: 512 Head Manager: 512	Output feature size of the embedding section
Advantage Stream Size	Agents: 256 Managers: 512 Head Manager: 512	Output feature size of the linear layer in the advantage stream
Value Stream Size	Agents: 256 Managers: 512 Head Manager: 512	Output feature size of the linear layer in the value stream

Table 7: Most relevant training hyperparameters. When not specified the hyperparameter value is the same across all system's agents.

Abstract in lingua italiana

Stiamo attraversando una crisi climatica senza precedenti. Gli ecosistemi stanno collassando, la temperatura sta aumentando e fenomeni atmosferici estremi sono sempre più frequenti. Le energie rinnovabili e a basso impatto ambientale rappresentano l'unica via d'uscita. I generatori solari e quelli eolici hanno andamenti imprevedibili data la loro dipendenza dalle condizioni meteorologiche. L'intelligenza artificiale (IA) può aiutare gli operatori di rete ad affrontare queste nuove sfide.
Per incentivare nuove soluzioni basate sull'IA, l'RTE (Réseau de Transport d'Electricité) organizza dal 2019 delle competizioni, denominate "Learning to Run a Power Network" (L2RPN). Il problema di controllo della rete viene specificato in queste competizioni come un problema di Apprendimento per Rinforzo (AR). Un agente di AR osserva lo stato della rete elettrica e compie un'azione basata sullo stato corrente della rete e sulle sue passate esperienze, in modo da massimizzare la somma cumulata di un segnale di rinforzo.
La nostra soluzione è un nuovo modello gerarchico multi-agente di AR. Un insieme di agenti gestisce il grafo localmente, un piccolo numero di gestori filtra le loro decisioni e, infine, un direttore seleziona una delle decisioni proposte dai gestori. La gerarchia viene generata dinamicamente sulla base della topologia di rete e viene aggiornata durante l'apprendimento.
La bontà del modello viene valutata in due ambienti di dimensione e complessità crescenti, dimostrando performance superiori a un sistema esperto particolarmente sfidante. I nostri risultati dimostrano l'applicabilità dei sistemi multi-agente alla gestione di reti elettriche. Col nostro lavoro vogliamo a creare un ponte tra i precedenti lavori sullla gestione automatica di reti elettriche e la comunità scientifica cresciuta intorno a L2RPN.

Parole chiave: Reti Elettriche, Apprendimento per Rinforzo, Sistemi Multi-Agente, Sistemi di Decisione Gerarchici, Riconoscimento di Comunità, Reti Neurali per Grafi