Understanding Reinforcement Learning Control in Cyber-Physical Energy Systems

Maurizio Repetto
Energy Department
Polytechnic University of Turin
10129 Turin, Italy
maurizio.repetto@polito.it

Francesco Moraglio
Energy Department
Polytechnic University of Turin
10129 Turin, Italy
francesco.moraglio@polito.it

Gianmarco Lorenti
Energy Department
Polytechnic University of Turin
10129 Turin, Italy
gianmarco.lorenti@polito.it

***Abstract*—The possibility of modeling a renewable energy system as a Cyber-Physical Energy System (CPES) offers new possibilities in terms of control. More precisely, this document discusses the applicability of Reinforcement Learning (RL) techniques to CPES. By considering a benchmark algorithm, we focus on conceptual and implementation details and on how such details relate to the problem of interest. In this case, we simulate how a RL model can optimize the energy storage control in order to reduce energy costs. The work also discusses the issues that arise in RL models and the possible approaches to these difficulties. Specifically, we propose investigating a better exploitation of the memory mechanism.**

***Index Terms*—Reinforcement Learning, Memory, Cyber-Physical Energy System, Control, Simulation, Energy Storage.**

I. Introduction

Optimal control of power systems is a well-established area of research, both in academia and industry. Today, the chance of formally treating a complex Energy System (ES) as Cyber-Physical Energy System (CPES) brings new opportunities in terms of resolutive approaches and testing accuracy.

This modeling possibility arises from the growing interest and recent developments in the areas of Smart Metering (SM) and Digital Twins (DT). On the one hand, SM provides us access to large amounts of data, for example those collected by sensors located in plants. On the other hand, DT models reached satisfying leves of accuracy, customizability and computational efficiency, thus becoming an ideal testbed for ES research. More precisely, we consider the application of Reinforcement Learning (RL) techniques to the domain of ES control. The high penetration of renewables into the energy mix of real-world systems requires the control strategy to exihibit higher degrees of adaptability and autonomy. In fact, solar and wind power generation are intermittent by nature, while contemporary commodity markets show strongly bullish trends and, consistently with the *inverse leverage effect* [10], high levels of price volatility. The exposure of the system to such environmental conditions makes the problem of control considerably non-linear in the space of observable variables, whose dynamics typically shows significant stochasticity and unpredictability.

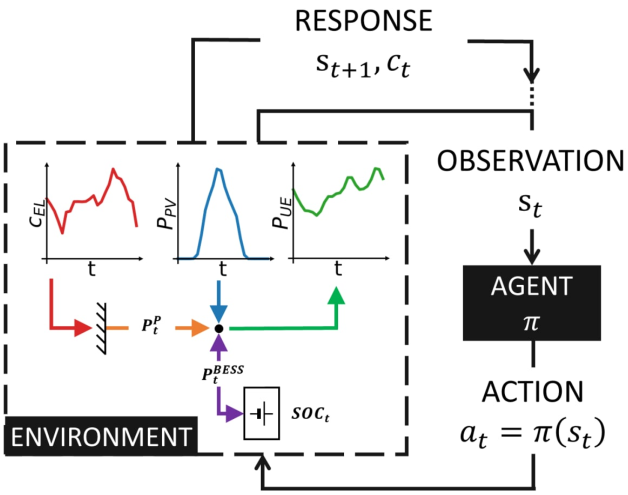

Fig. 1. Agent-Environment interaction in a CPES.

From another point of view, effective management of the Energy Storage System (ESS) could lead to remarkable economic and environmental benefit. In fact, storage is the enabling technology to catch arbitrage oppurtunities in such an high-volatility scenario. For example, excess production in off-peak hours can be stored and sold during peak hours. This becomes particulary interesting in the current period, as markets continue seeing high values of the peak-base spread. In other words, good exploitation of production, price and consumption patterns which appear to the control entity is an highly desirable property for any operational strategy. Hence, the metaphor of RL as way to perform behavioural learning seems particularly suited to this context. The idea is modeling the control unit as a learning entity which adapts the CPES behaviour consistently with the feedback signals it receives from the surrounding environment. Fig. 1 gives a schematic representation of this hypothetical situation. Still, RL models are intrinsecally abstract and moving from concepts to implementations is often challenging.

Some authors [5] applied RL to CPES control problems, often employing the most advanced available algorithms. Nevertheless, to the best of our knowledge, few studies analyze algorithmic details, especially when dealing with deep RL, whose introduction is very recent [7]. That means, these papers usually give little attention to topics such as applicability issues and model interpretability. This can become particulary problematic, as there are several factors in the implementation of this method that are crucial to its success [6]. We wrote this work with the intention of stimulating new reaserch directions in this field.

The paper is structured as follows. Section II contains theoretical remarks on RL, while Section III describes the examined algorithm in detail. In Section IV we define a benchmark problem and in following Section V we sum up implementation details for the problem under consideration. Section VI contains experimental results, while final Section VII discusses a theoretical and applicative issue, proposing possible solutions.

II. Theoretical recall

RL is an area of Machine Learning concerned with the problem of learning from interaction. In general, we think of a learning and decision making entity, named Agent, that continually interacts with its surrounding Environment.

Consider for example Fig. 1. Here the control unit of the CPES represents the Agent, while we can describe the Environment as a number of observable variables: electric load, photovoltaic production, ESS state of charge and market prices.

At each step t of the learning process, the Environment presents a state s_t to the Agent, which chooses and executes an action a_t. The Environment in turn responds to the Agent by presenting a new state s_{t+1}; this state transition also has a cost for the Agent, that is c_t. Roughly speaking, the ultimate goal of the learner is minimizing the long-run expenditure.

We typically deal with RL problems using the Markov Decision Process (MDP) formulation. More precisely, an MDP is a tuple (S, A, T, C, γ):

- S is the state space.
- A is the (Agent's) action space.
- T is the transition function, which determines the dynamics of the model. For instance, one could have

$$\begin{aligned} T : S \times A \times S &\longrightarrow [0, 1] \\ (s, a, s') &\longmapsto \mathbb{P}(s'|s, a), \end{aligned} \tag{1}$$

  that is, the probability of transitioning to state s' after executing action a in state s.
- C is a cost function,

$$\begin{aligned} C : S \times A \times S &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto c = C(s, a, s'). \end{aligned} \tag{2}$$

  It returns the cost of transitioning to state s' after choosing action a in state s.
- $\gamma \in (0, 1)$ is a discount factor that determines how significant short-term costs are compared to long-term costs.

Solving an MDP [1], i.e. solving the problem of control in an MDP, means finding a policy

$$\pi : S \longrightarrow A, \tag{3}$$

which is a function that, for each observed state $s \in S$, returns the optimal action to be taken, $a = \pi(s) \in A$. The objective of the optimization is minimizing the expected cumulative discounted cost:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t c_t \right], \tag{4}$$

where $\mathbb{E}_\pi$ denotes the expectation w.r.t. policy π (i.e. when the Agent chooses actions according to π) and

$$c_t = C(s_t, a_t, s_{t+1})$$

is the cost of the $(t + 1)$-th state transition; t is the timestep. A policy that minimizes (4) is denoted $\pi^\star$.

In this context, algorithms for MDPs need a way to link the optimality criterion (which refers to long-run costs) to policy functions (which are "instantaneous"). A very common approach consists in exploiting the so-called Value Functions (VFs), which measures how convenient it is to be in a certain state (or choosing a certain action). In particular, the state-action VF, also known as Q-function, is

$$\begin{aligned} Q^\pi : S \times A \longrightarrow& \mathbb{R} \\ (s, a) \longmapsto Q^\pi(s, a) =& \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k c_{t+k} | s_t = s, a_t = a \right]. \end{aligned} \tag{5}$$

Denote by $Q^\star$ the Q-function for the optimal policy $\pi^\star$. One can show [2] $Q^\star$ satisfies a recurrence relation named Bellman Optimality Equation (BOE). If state space S is discrete, BOE reads

$$Q^\star(s, a) = \sum_{s' \in S} T(s, a, s') \left[C(s, a, s') + \gamma \min_{a'} Q^\star(s', a') \right], \qquad s \in S, a \in A. \tag{6}$$

For each $s \in S$, we can determine the optimal action greedily from $Q^\star$ by selecting

$$\pi^\star(s) = \underset{a \in A}{\operatorname{argmin}} Q^\star(s, a), \tag{7}$$

thus explicitely defining an optimal policy $\pi^\star$.

A widespread algorithm named Q-Learning [2] [4] relies upon these (6) (7) properties to estimate $Q^\star$ and to compute $\pi^\star$ from it. This algorithm initializes the Q-values at random and iteratively improves its estimates by sampling state transitions and rewards. More precisely, Q-Learning improves the Q-function estimate by the following update rule:

$$\begin{aligned} &Q_{t+1}(s_t, a_t) = \\ &Q_t(s_t, a_t) + \alpha \left(c_t + \gamma \min_{a \in A} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right), \end{aligned} \tag{8}$$

where t denotes the timestep and $\alpha \in (0, 1]$ is the step-size parameter.

Traditional variants of Q-Learning work by storing and updating a tabular representation of the Q-function (e.g. using an hash table with keys (s, a), $s \in S$, $a \in A$). Still, if the state space S is inherently large or continuos, the $Q^\star$ estimate is typically implemented via Function Approximation (FA) [1]. The essential idea is fitting a regression model to the dataset of observed system transitions, which allows generalization to new experience.

III. Algorithm

The model we are considering in this study is a simplified version of Deep Q-Network (DQN), first appeared in [7]. For convenience, we named our model Simple Q-Network (SQN). More precisely, we are considering a threefold extension to the classical Q-learning that includes the following improvements.

1) The model approximates the Q-function with a Multi-Layer Perceptron (MLP), which is trained in batch mode. This idea was firstly introduced in [8] and developed further in [7]. In particular, the latter work proposes a deep convolutional architecture for FA and this kind of neural net is becoming increasingly popular also in RL [5]. Despite these networks offer significant advantages in terms of accuracy and generalization capability, we preferred sticking to simpler models, which are computationally much more permissive. More details on FA can be found in Section V.
2) SQN, just like DQN, implements a mechanism known in literature as *Experience Replay* (ER). That is, the algorithm relies upon storing and maintaining a large memory buffer of previous experiences, i.e. state transistions and rewards. When the model updates the network parameters, mini-batches of experience of size B are drawn uniformly at random from the buffer.
   Notice the training procedure is run only every $K \in \mathbb{N}$ steps.
3) SQN also inherits from DQN an improvement named *Target Network* (TN). Everytime the algorithm runs a training void, it transforms the sampled batch of experience to a training dataset. where also the response values are predicted using a neural net. This is typically the same net used in determining optimal actions (i.e. the Q-Network). Introducing a TN means using a separate net to predict target values. The TN shares the same architecture of the Q-Network, but its parameters are updated only every $K_T \geq K$ iterations. This enhancement results in increased training stability.

Algorithm 1 gives a schematic representation of SQN. This model includes a few other technicalities we briefly enumerate below.

- According to the classic literature about Q-learning [2], we make use of the ϵ-greedy exploratory policy. That is, in order to balance the exploration-exploitation tradeoff, random actions can be drawn with probability ϵ. In our case, this probability is initialized to 1 and then linearly annealed to its final value ϵ.
- Memory Buffer M is typically implemented with a (fixed) maximum size m; this way the Agent only stores the last m experienced state transitions.
- Gradient Descent (GD) updates are performed using the classic method with learning rate η, inlcuding a momentum β.

This model is simpler than deep-learning-based methods that tend to appear in recent publications on the application of RL to ES [5]. In fact, we remark that our goal is understanding the possibilities and drawbacks of applying RL methods to CPES, rather than devoloping a performing model for a specific problem.

Last but not least, a remarkable difference between DQN and SQN is that the former is tailored for episodic (that is, finite-horizion) tasks, while the latter is, at least in principle, thought for optimizing control in an infinite-horizon setting: recall the objective function in (4). The choice was made since we consider this setting more natural to the ES domain; nevertheless a finite-horizon setting could increase overall model performace.

Algorithm 1 Simple Q-Network (SQN)

```
Require: $\epsilon$, $\gamma$, $B$, $K$, $K_T$
  Initialize action-value net $Q$ with weights $\theta$
  Initialize target net $Q^T$ with weights $\theta^T = \theta$
  Initialize memory buffer $M$
  for all timestep $t \in \{0, \dots, t_{\max} - 1\}$ do
    Observe state $s_t$
    $\epsilon_t \leftarrow \epsilon + (1 - \epsilon) * \left(1 - \frac{t}{t_{\max}}\right)$
    Draw $u \leftarrow$ random(0,1)
    if $u < \epsilon_t$ then
      $a_t \leftarrow$ random(A) {Select random action}
    else
      $a_t \leftarrow \underset{a \in A}{\operatorname{argmin}} Q^\star(s_t, a; \theta)$
    end if
    Execute action $a_t$
    Observe new state $s_{t+1}$ and cost $c_t$
    Save transition $(s_t, a_t, s_{t+1}, c_t)$ in $M$
    if $t \mod K = 0$ then
      Sample $B$ transitions $(s_k, a_k, s_{k+1}, c_k)$ from $M$
      for all transitions $k$ do
        $y_k \leftarrow c_k + \gamma \min_{a \in A} Q^T(s_{k+1}, a; \theta^T)$
      end for
      Perform gradient descent update on $\theta$ {Run training procedure}
    end if
    if $t \mod K_T = 0$ then
      $\theta^T \leftarrow \theta$ {Update target parameters}
    end if
  end for
```

IV. Energy System Case Study

Below we describe the simple ES model we used to assess the behaviour of the proposed RL algorithm.

Consider an MDP describing the control of a storage-integrated photovoltaic (PV) system that also has to satisfy a load; also suppose this CPES can purchase energy from the grid. By contrast, market sales operations are not allowed, that is any excess PV power which exceeds load, if the ESS is full, is injected to the grid without any economic reward. This assumption is based on two considerations.

1) This model is a benchmarking system: excluding the possibility of market operations results in a simpler policy to be learned. For example, this decision implies smaller action space size.
2) In the domain of smartgrids, current European regulations give more relevance to self-consumption rather than to market activity.

State space S is continuous and each $s \in S$ is a real-valued vector of the observable variates, namely

$$s_t = \left[t, \mathrm{SOC}_t, P_t^{PV}, P_t^{UE}, c_t^{EL}\right], \tag{9}$$

where for each time step t and considering an hourly discretization of time

- SOC_t is battery State-Of-Charge;
- P_t^{PV} is photovoltaic power production;
- P_t^{UE} denotes (electric) load and
- c_t^{EL} is the current electricity price.

The battery control unit represents the Agent in this problem, that has to understand the dynamics of the system in order to better handle the ESS. We consider for simplicity binary BESS operations (all-or-nothing). We consequently assume that SOC_t takes discrete values. Hence, action space A is finite and, for each $s \in S$, feasible actions are a subset $A(s) \subseteq A$:

$$A = \{\texttt{"DISCHARGE"}, \texttt{"IDLE"}, \texttt{"CHARGE"}\}$$

The cost function in this case is the (hourly) expense for power purchasing, namely

$$c_t = c_t^{EL} * P_t^P, \tag{10}$$

where P_t^P, which denotes the purchased power at each timestep t, can be easily derived from the dynamics of the system.

The goal of the *Agent* is thus learning to minimize the long-run energy-purchasing expenditure (while it is given only instantaneous observations).

V. Implementation Details and Considerations

A. Simple Q-Network

The model described here as SQN has been implemented using `Python` programming language. We have tried to translate the (relative) conceptual simplicity of the algorithm to the implementation level. In other words, only few external libraries have been used:

- `Numpy`;
- `Scikit-Learn` (MLP implementation);
- `Pandas` (dataset handling).

Many hyperparameters concurrently determine the behaviour of the algorithm, with the possibility of observing opposite effects. Repeated testing and intuitive reasoning lead us to the values summarized in Table I.

The model is very sensitive to the choice of the discount factor γ. Even slightly lower values, such as $\gamma = 0.95$, resulted in an impossibility of learning the mid-run behaviour of the state variables.

In the setting of CPES, also the ER mechanism is of crucial importance. In fact, commodity-related time series data (e.g. prices, load etc.) typically exihibit evident temporal patterns such as strong seasonalities. Statistically speaking, this means highly correlated data is input to the model, thus resulting in high estimation variance [3] even if training is run in batch mode. By contrast, sparse sampling from a (large) memory buffer significantly reduces data covariance.

TABLE I
Hyperparameter settings in SQN

PARAMETER	DESCRIPTION	VALUE
γ	Discount factor	0.99
ϵ	Final exploration	0
B	Batch size	8
m	Maximum size of memory buffer	1000
K	Update action net every K steps	4
K_T	Update target net every K_T steps	24

B. Function Approximation

Consider the MLPs used in FA and target computation. The topology consists two hidden layers of ten neurons each and it relies on the *ReLU* activation function. Both specifications were determined as the result of a trial-and-error procedure.

At each step t, the model process the state vector as defined in (9): the Agent only observes the instantaneous values that each enivronmental variable takes. Including past experience would obviously increase model perfomance, but we preferred sticking to this simpler represantion for better understandability.

State data were rescaled to $[-1, 1]$ according to heuristically determined ranges, while battery actions were mapped to the set $\{-1, 0, 1\}$.

The learning rate of the net is fixed and set to $\eta = 10^{-3}$, while the (fixed) momentum is $\beta = 0.9$. Our model also applies a slight weight regularization with coefficient $\alpha = 10^{-6}$ (w.r.t. the L^2-norm of parameter vector θ).

During the early development stages of this model, we were considering the possibility of employing an even simpler method for FA, that is linear regression. Neverthless, after obtaining poor results, we moved to such simple MLPs. We think the results obtained with linear learners were so bad because the Q-function could be highly nonlinear even for the simplest CPES control problems. In fact, literature [1] explains that linear FA performs poorly without careful design of the regression features, that typically includes the use of basis functions. This process, which requires an high level of domain-specific knowledge, was not intuitive (even) for the

problem under discussion.
The implementation of a TN also appears useful in the control of CPES. In fact, it can be seen as a periodic weight reset mechanism for the network which determines the ”direction” of learning for the action network. This becomes particularly useful if the reset period K_T corresponds to the intrinsic seasonality of the data under observation.

VI. Data and Experimental Results

In order to understand the working of SQN, we repeatedly tested it over simple, synthetic data. All data have hourly resolution and daily period, that is $T = 24$ hours.

- The **PV** production curve was obtained by averaging a month of daily profiles retrieved from PVGIS [14]. The PV plant is a set of panels whose nominal maximum power is 15 kWp.
- We derived the **load** profile by scaling the aggregate load of a multiutility company. Fig. 2 is a representation of the curve.
- We computed synthetic market **prices** with a sinusoidal curve, namely

$$c_t^{EL} = \frac{c_{\max} - c_{\min}}{2} \sin\left(\frac{2\pi t}{\tau}\right) + \frac{c_{\max} + c_{\min}}{2}, \quad (11)$$

where $c_{\max} = 80$ and $c_{\min} = 10$ denote maximum and minimum electricity prices expressed in euros, respectively, while $\tau = \frac{2}{3}T = 16$ is a heuristic value.

Total dataset length is $t_{\mathtt{max}} = 17520$, that is two years of hourly time series data.
Maximum energy capacity of the ESS is 10 kWh. The discretization of charge and discharge actions has a step of 1 kWh. By convention, negative values correspond to charge actions, while positive values to discharge operations.
With slight abuse of term, we used the first 90 % of the data as ”training” set. That is, exploration stops after $90\% * t_{\mathtt{max}}$ steps and ϵ, the probability of choosing random actions, is set to zero.
The model gradually learns how to reduce the daily expenditure, as shown in Fig. 3. Here, as a benchmark we also reported the minimum cost obtained with the true optimal policy $\pi^\star$. We are able to do so since the setting is completely determined and Mixed-Integer Linear Programming (MILP) techniques can solve this problem. Oscillations in the curve indicate the presence of instability and, possibly, forgetting phoenomena (see next Section VII). Fig. 4 shows the final policy (i.e. the one obtained after the end of the training period) with pink triangles. What the model obtains is both heuristically and quantitatively good. We can infer this from a comparison with $\pi^\star$, in green boxes. Plot also shows the net load $P_t^N = P_t^{UE} - P_t^{PV}$ and the synthetic prices.

VII. Open Issues

Several difficulties arise when applying RL methods to solve control problems in CPES. The devoloper should take diverse, non-intuitive decisions, especially when dealing with the implementation phase. Many examples can be found in previous Sections III and V.

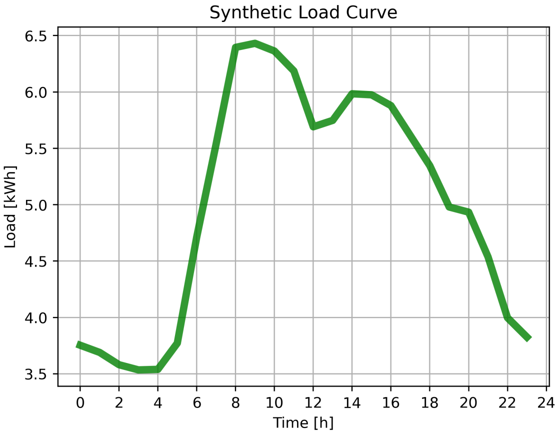

Fig. 2. The sampled and scaled load curve.

From a more general viewpoint, however, it is worth mentioning an intrinsic problem of neural-fitted Q-Learning, which is particularly evident in CPES control (see below why). This issue is known in neural network literature as *catastrophic forgetting* [9].
Recall that the network of the Agent, in SQN, does not learn a policy directly, but it learns Q-values, which can be interpreted as a proxy for the policy[1]. It might happen, during network training, than successive improvements of the Q-approximation imply the choice of suboptimal actions [6].
Such suboptimal actions drive the system towards unconvenient areas in the state space, thus potentially causing the beginning of a vicious cycle:

- Agent chooses suboptimal actions;
- Agent observes ”bad” states as a result of suboptimal actions;

[1] There exists a class of RL algorithms, namely Policy Gradient (PG) methods, which directly optimize a policy. Despite PG models do not suffer from value-related biases, they tend to be much more unstable [1].

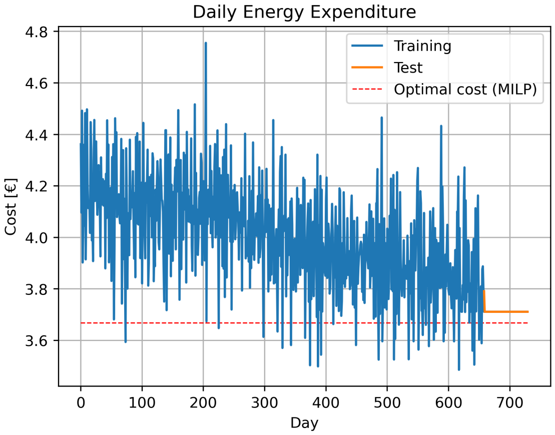

Fig. 3. The model optimizes its behaviour in order to decrease daily cost.

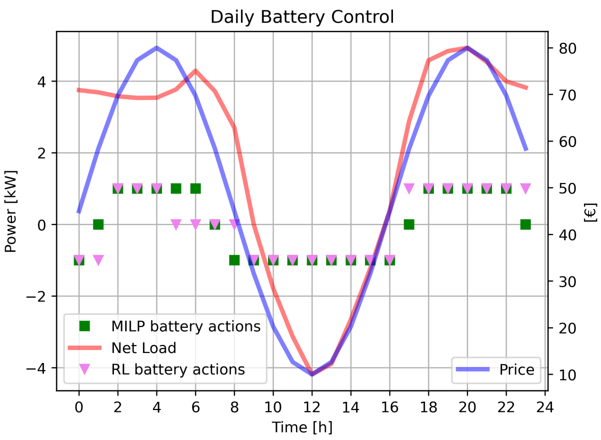

Fig. 4. Battery operation during a day.

- Agent pays high costs regardless of the action it chooses;
- Agent tries adapting its policy to this unstable condition, causing its net to "forget" previous experience.

The mechanism of ER partially reduces this problem: presenting past situations to the learning Agent prevents it from optimizing its policy on unfair state space areas. Still, the algorithm may run into catastrophic forgetting due to the intrinsic limit of Q-function approximation.

We think the main drawback of this class of algorithms is the inefficient handling of the memory buffer. DQN was originally developed to learn old videogames [7], whose mechanics are usually repetitive and this repetitiveness makes the choice of a strongly randomized memory more suitable. By contrast, controlling a CPES requires an higher degree of adaptability, together with a better exploitation of past information, which exihibits more variability. Consider for example the yearly seasonalities of energy demand: the state transitions the model has observed in past winters should be better exploited in the following winter.

Take for instance Michigan-Style Learning Classifier Systems (LCS) [12]. These are genetic-based ML models designed to solve a RL problem. Optimization operates at the level of individuals, but the solution is given by the entire rule population. Each rule, or *detector*, is evolved to recognize certain patterns in input data and to respond properly. We remark that this pattern-recognition capability is also a fundamental feature of antibodies in Artificial Immune Systems (AIS) [11]. Recent research [13] shows there are several other conceptual similarities between AIS and RL: we can, at least theoretically, reformulate adaptive immunity as a network-based RL problem. We think these ideas from the metaheuristics area could be really helpful in developing more suitable memory management strategy. Nevertheless, neither LCS nor AIS can be intuitively applied to this context.

In fact, LCS have been developed for domains where we can naturally apply binary representation of features and conventional genetic operators (mutation, crossover). This is not the case in the inherently continuous domain of CPES. Moreover, the population of solutions in Michigan LCS directly represents the policy, while in SQN the memory buffer is a technicality to improve the estimate of the Q-function, from which the policy is derived.

As for AIS, their principal charachteristic is the possibility of performing pattern recognition, rather than function optimization (possibly, a behavioural function). More precisely, it is the population of antibodies (the *immune memory*) that recognizes patterns in new observations, typically with similarity-based techniques. We think such behaviour could be really helpful if hybridized with RL, but designing a reasonable solution is not trivial. In particular, it seems diffcult to properly define affinities in the state-action space.

VIII. Conclusion

This paper takes under examination the possibility of applying RL methods to the control of CPES. Researchers should give particular attention to algorithmic details, since these often result in highly unpredictable behaviour.

Certain instabilities can be tackled on the hyperparametric level, e.g. by properly searching adequate values. By contrast, other issues come from intrinsic limits in the design of common RL algorithms. For this reason we hope this paper will encourage insiders to dive deeper into the understanding of the fundamental logics of RL.

For our part, we are already reasoning about possible ways to better define the ER and memory mechanism, as we think RL could be fundamental in the proper control of future energy systems.

References

[1] Wiering, M. and Otterlo, M. (2012). *Reinforcement Learning: State-Of-The-Art*. Springer.

[2] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* Second Edition. The MIT Press.

[3] Hastie, T.J., Tibshirani, R. and Friedman, J.H. (2001). *The Elements of Statistical Learning.* Springer.

[4] C. J. C. H. Watkins (1989). *Learning from delayed rewards* Ph.D. dissertation, King's College, Cambridge, England, 1989.

[5] Perera, A.T.D. and Kamalaruba, P. (2020) *Applications of reinforcement learning in energy systems.* Renewable and Sustainable Energy Reviews. 137 (2021).

[6] Roderick, M., MacGlashan, J. and Tellex, S. (2017). *Implementing the Deep Q-Network.* ArXiv. https://arxiv.org/abs/1711.07478

[7] Mnih, V. et al. (2015). *Human-level control through deep reinforcement learning.* Nature 518, 529–533 (2015).

[8] Riedmiller, M. (2005). *Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method.* ECML'05: Proceedings of the 16th European conference on Machine Learning. 317-328. Springer

[9] Ratcliff, R. (1990). *Connectionist models of recognition memory: constraints imposed by learning and forgetting functions.* Psychological review, 97 2, 285-308.

[10] Geman, H. (2005). *Commodities and Commodity Derivatives.* Wiley.

[11] Read, M., Andrews, P.S. and Timmis, J. (2012). *An Introduction to Artificial Immune Systems.* Originally published in Rozenberg, G. et al. *Handbook of Natural Computing.* (2012). Springer.

[12] Michalewicz, Z. (1999). *Genetic Algorithms + Data Structures = Evolution Programs* . Springer.

[13] Kato, T. and Kobayashi, T.J. (2021). *Understanding adaptive immune system as reinforcement learning* Phys. Rev. Research 3, 013222

[14] Photovoltaic Geographical Information System (PVGIS). Solar Radiation Tool. https://ec.europa.eu/jrc/en/pvgis