POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF8250AE - Reinforcement Learning

Project report

Reinforcement learning for power grid management

Henri Paquette - 1961065
Marianne Lado-Roy - 1847092
Mathieu La Brie - 2403325

Polytechnique Montréal
December 20th 2024

Introduction

Power grids are the essential structures responsible for transmitting and distributing electricity from power plants to our homes, businesses, and industries. However, the management of power grid can face numerous challenges. Equipment failures, often due to aging infrastructure and more recently, to the intensification of severe weather events such as storms and heatwaves, can lead to outages. Furthermore, the growing demands for energy and the integration of weather-based renewable energies has pushed the grid infrastructure to become more and more complex and unpredictable. All of theses factors have highlighted the importance of improving and automating grid management systems to ensure both reliability and sustainability in unpredictable climatic conditions. Traditional grid management approaches often rely on rule-based systems and the experience of operators [1], which lack flexibility in non-stationary and unpredictable environments. Reinforcement Learning has been proposed [1, 2] as an alternative by enabling autonomous agents to learn and adapt optimal control strategies directly from interaction with the grid environment. In this report, we will frame the application as a reinforcement learning problem, discuss the methodology for training the algorithms and report on our baselines and results.

Application: Power grid management

A power grid is a complex network designed to generate, transmit, distribute, and manage electricity from power sources to consumers. In this section, we will present the components of the power grid and the principles of grid management.

Components of a power grid

The first component of the power grid is the generation network, which converts energy from various sources into electrical power. The sources of energy in one grid can be diverse, from thermal, nuclear or hydroelectric power plants, to weather-based sources such as wind or solar power. Once electricity is generated, it is transported over long distances through the transmission network, which consists mainly of high-voltage power lines and transmission substations, which serve the purpose of connecting high-voltage power lines together. The final stage of the power grid is consumption, where electricity is used by the customers. Demand and generation vary based on time of day, weather conditions, and economic activity.

Grid management

The operation of a power grid relies on maintaining three primary constraints across all areas of the network and at all times [1]. First, the thermal limits of transmission lines cannot be exceeded for too long. If that is the case, security mechanism automatically disconnect the line, provoking an outage. Second, the voltage across the system must be kept within a specified range. Lastly, power generation must match consumption in real-time, as electricity cannot be stored on a large scale [3]. If these constraints are not met, the consequences can be severe, ranging from localized equipment failures to widespread blackouts. To avoid theses unwanted consequences, grid management is performed in control centers to maintain the network in normal operating conditions in real-time. The state of the grid, state of generators production and load consumption is monitored in real-time by sensors and the operators can act remotely on the network from control centers. Looking at the state of the network, automated programs and operators take decisions to maintain the network operation under the normal-operation constraints. Typical actions that can be taken include (1) modifications of the network topology to re-direct power flows, (2) modification of productions and (3) load management.

Reinforcement learning problem

In the context of reinforcement learning, the grid management problem has been modelled as a Markov Decision Process by [4, 5]. In this framework, the agent is the network operator and the environment is the power grid. The objective of the agent is to maintain the network under normal operating conditions at the lowest cost. The `Grid2Op` [6] module has been developed to model sequential decision making on a powergrid. In the next subsections, we will present the `Grid2Op` module and its formalization of the problem as an MDP.

The `Grid2Op` module

Grid2Op is an open-source Python framework designed to simulate the operation of electricity grids and evaluate the performance of decision-making algorithms at them. It provides a realistic environment for training and testing reinforcement learning agents, allowing them to take actions similar to what grid operators would to manage a grid. The platform includes a variety of scenarios, grid topologies, and evaluation metrics to benchmark the effectiveness of different strategies. In Grid2Op, the the grid is simplified as graph representing the power grid components and how they are interconnected (topology). Finally, we note that the environment in Grid2Op is episodic, consisting of pre-defined, finite scenarios that micking how production, consumption and the state of the powergrid vary in real life.

Formalization of the problem as an MDP

In Grid2Op, each time step begins with the agent observing the grid's current state. The agent uses this data to decide on an action. After the action is taken, the possibility of transition to the next state is evaluated by verifying the action against operational constraints. Then, a power-flow simulator is used to calculate the outcome, updating the grid's state, and the agent receives a reward based on the resulting performance.

Action space: (1) Discrete actions (topology change): An agent can change the topology by connecting/disconnecting power lines or by node splitting at substantiation. The discrete action space if 2^k dimensional, k being the number of elements connected on the grid. (2) Continuous actions: An agent can choose to increase or decrease the power production of a generator (dispatching action), decrease the power production of a renewable generator (curtailing action) or change the setpoint of storage units. The continuous action space has is at least an n-dimensional continuous space, n being the number of generators and storage.

Observation space: The state of the environment is partially perceived by the agent through the real-time monitoring of the power grid components. The observation space is the part of the state space that is shown to the agent. Theses observations can be grouped as (1) Load, generators and powerline status, (2) Topology configuration, (3) Dispatching, curtailment and storage status, (4) Attributes related to the time and (5) Operational constraints status, which is a mixe of discrete and continue observation.

Transitions: In this framework, the transition from the current state s_t to the next state s_{t+1}, after the agent has selected action a_t is composed of two aspects. First, the validity of the action is verified against operational constraints. Second, if the action is valid, a power-flow simulator calculated the influence of the action a_t on the grid and the resulting next state of the grid s_{t+1}.

Reward function: In this framework, there are many reward functions that are available and that depend on the state of the environment. The different rewards mostly incorporate one or many of theses aspects: a positive reward for each time steps that the agent survived, a positive reward if the state of the grid power is in the safe range and negative if not, and finally, a penalty relative to the economic cost of the grid.

Methodology

The grid2op environment include the use of chronics, which are different possible scenarios that could happens in real life. Since the agent cannot interact with the real world and is trained off-line, it is essential to divide the chronics into a train, validation and test environment to make sure our agent is not overfitting on the training environment. The chronics' data is split with a 80%, 10%, 10% proportion for train, validation and test. The validation environment is used to validate each agent trained on the training data and the test set is only used to compare the agents to baselines. To begin the study, different agents were trained in the simplest environment available: *l2rpn_case14_sandbox*. This environment counts 14 substations, 20 lines, 6 generators and 11 loads, all generators being non-renewable with a controllable power capacity, which simplify further the policy. To simplify the action space, the agent will not be directly in control of the generators output power, instead some planing will be programmed into the agent to always try to satisfy the power demands on the grid. The agent will have to balance this demands by changing the substation busbars's configurations. Thus, the action space will be completely discrete, which guides our choice of algorithm.

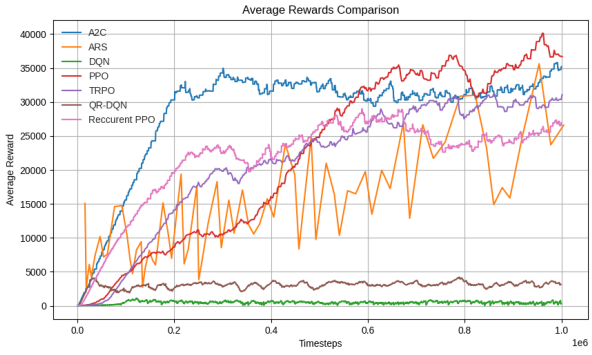
The chosen reward function is the *L2RPNReward*, the official reward for the L2RPN competition. The reward

is defined as the sum of the squared margins for all powerlines. The margin for a powerline is (Thermal limit − Flow in amps)/Thermal limit if the flow is within the thermal limit, and 0 otherwise. Thus, the reward is $\sum_{\text{Powerlines}}(\text{Margin})^2$ for each time step. This reward will is chosen because it quantifies well the stability of the grid, which is the ultimate goal and implicitly encapsulates the goal of running the grid the longer possible without shutdown and avoiding overflow in power line that could caused serious damage and cost. The first metrics to evaluate will be the total reward on validation environment during and after training, the sampling efficiency and the time to train the agent. Those first metrics will guide us for the choice of algorithm that will most likely suit well our action and state space. Once we selected the best algorithm, our goal will be to improve it experimenting how the size of the action and observation space affect the agent's capacity to learn. The performance of our agent will be judged based on how well they perform on the test set relative to two baseline agents: an agent that is taking a random action every step and an other agent that will never take any action.

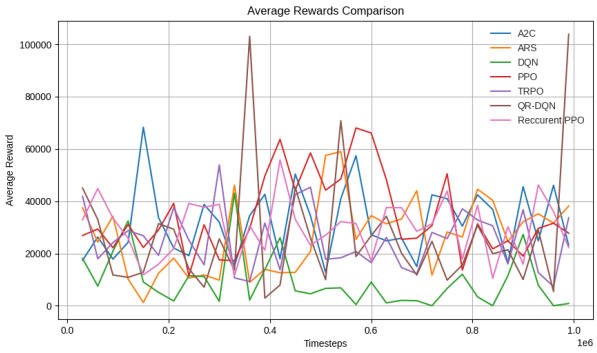
Baselines and Benchmarks

Comparison of algorithms during training

Since our action space is discrete, we chose some algorithms that are well suited for discrete action space. We started by training our agent using deep Q-network (DQN), the Quantile Regression DQN (QR-DQN), Synchronous Advantage Actor Critic (A2C), Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO) and Augmented Random Search (ARS) [7] only a small action and observation space containing only the most critical element. This "default" action space is made of set_line_status and set_bus and the "default" observation space is made out of rho, p_or, gen_p and load_p (see the documentation for more information [8]). Looking at figure 1, a substantial difference in results can be observed between the rewards during training and rewards during validation. During training, some models seem to outperform other. For instance, A2C shows good learning at the beginning and PPO gradually catches up. However, when comparing the models evaluated on the validation set, these differences aren't as apparent partially because some models keep exploring during the training which limits their possible reward. Figure 1b shows little disparities in the models, the only outstanding point being the poor performance of the DQN algorithm, mirroring its training performance. As for the QR-DQN, it's performance seems good, but we note that the training took significant time. Considering the low computing resources and time available for the team, this might restrict the utilization of this algorithm.



(a) The average reward during training for multiples agents on default parameters.



(b) The average reward during evaluation with the greedy policy for multiples agents on default parameters.

Figure 1

Comparison with baselines algorithms

To truly assess the performance of the different algorithms, we use the validation set to compare them to the two baseline agents, a Random policy Agent and a Do Nothing policy Agent. The test set was only use to test our final agent. The Random Agent takes actions without any strategy (chooses the action from a uniform distribution), while the Do Nothing Agent simply remains idle throughout the task. Figure 3a shows the cumulative reward

per episode of the trained agents and baseline agents on 50 scenarios taken from the validation set. Looking at this figure, it is surprising how many agent don't do any better than the Do-Nothing agent, represented with the thick grey line. Table 1 shows the average reward of each algorithm over the 50 scenarios. Only the QR-DQN and the A2C agents succeed in beating significantly the Do Nothing agent, as highlighted in bold in the table. Beating the Do Nothing agent is crucial, as this mean the agent is actually learning to manage the grid, not just avoiding making any actions to survive the longest time possible.

Table 1: Average cumulative reward over 50 scenarios of the test set.

DoNothingAgent	RandomAgent	RecurrentPPO	QRDQN	TRPO	PPO	DQN	ARS	A2C
28764	30	28764	**82252**	28764	28974	7839	28158	**34169**

Even though the QR-DQN agent performs significantly better than the other agents, we have chosen to discard it and continue with the second best performing agent, the A2C algorithm. The major issue with the QR-DQN agent is the time and resources needed to train it, which would prohibit us to further explore how modifying the structure of the observation space can enhance the performance of our agent, which we do in the next section.

Increasing the observation space

The grid2op environment offers a vast amount of possible observations and a significant challenge is to choose the right observation to well optimized the trade-off of adding valuable information and increasing dimensionality and complexity. For this, we started by having a basic understanding of the most important information to manage power grip. The most important being the terminal limit of each grid line and the active power circulating in each line, the active power produced by each generator and the active power consumes by each load (which are the default observation used). Other observations as the reactive power, the voltage, the phase angle on lines, generators, loads and electrical information on line extremities could also add significant information. Among those, we tried combinations of those observations with different complexity of the function approximation used for the policy and the state value. The function approximator being an MLP, we also experimented with different architectures for the number of hidden layers and their sizes. Our findings shows that adding information about reactive power, voltage or phase angle tends to add too much complexity and our agent wasn't able to extract valuable features from all those observations. A key finding was to pass as an observation the full status of connection of every power line to our agent while keeping the rest of the observations to only the active power of each component and the thermal limit. For the MLP, we found that keeping the number of hidden layers to only 2 with 64 and 32 weight was the best way to extract the information from those observation. The result of some of those experiment are shown in figure 2, which highlight the performance gained by adding the line_status and the lost of performance by adding more observation. This configuration of observation and MLP architecture allowed our agent to learn useful action instead of learning to only avoid taking any action at all.

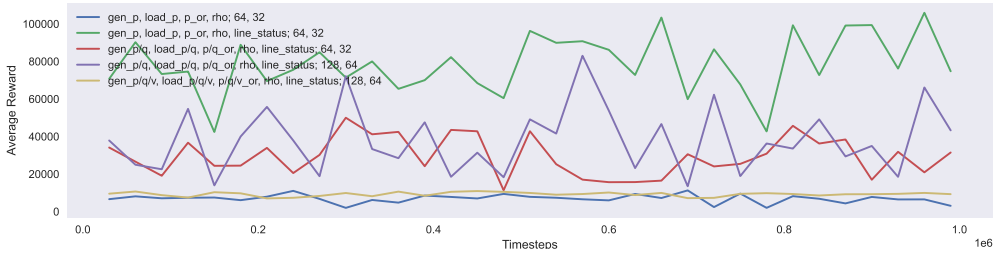
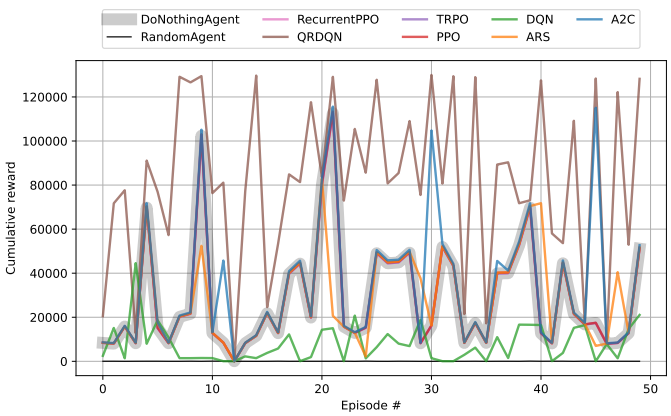


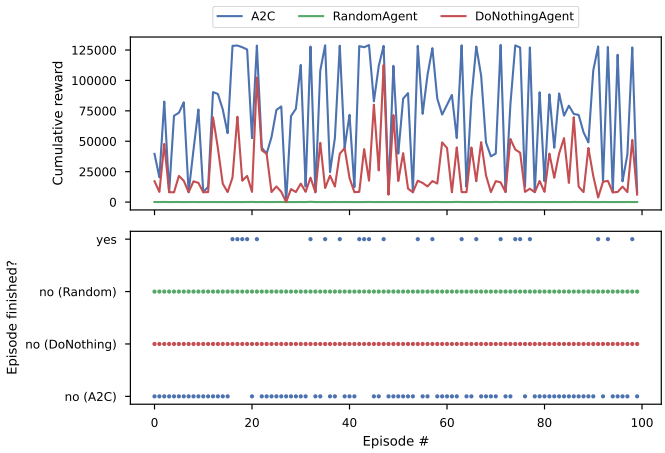
Figure 2: Comparison of the evaluation score on the validation environment for different observation spaces and MLP architectures during the training.

By experimenting with different observation space and hyperparameters, we found that the best performing agent was the A2C using only the gen_p, load_p, p_or, rho and line_status with an MLP with two hidden layers of 64 and 32. Figure 3b shows two metrics comparing this best agent to the baseline agents on 100 scenarios from the test set. The first graph shows again the cumulative reward per scenario, the second graph

shows if the scenario was completed by the agent (yes=completed). Looking at theses results, we see that our agent surpasses both baselines, and is the only one able to finish scenarios.



(a) Performance of all preliminary algorithms and baseline agents on 50 scenarios of the test set using the default parameters, action and observation.



(b) Performance of the final tuned agent (A2C) and baseline agents on 100 scenarios of the test set.

Figure 3

Discussion

The best algorithms for this grid management problem were the QRDQN and the A2C. Those two algorithms are well suited for our stochastic environment, QRDQN capturing the uncertainty of future reward by predicting a distribution for each action and A2C being well suited for stochastic environment by being able to produce stochastic policy. Those two algorithm were able to provide a good exploration and exploitation trade-off to learn a good policy during training and both being sample efficient thanks to the replay buffer of QRDQN and the parallel processing of A2C. We also found that adding the observation of line_status in addition to the default observation was adding significant performance, which means that the agent does not naturally keep a good tracking of how it's action changed which powerline is connected or disconnected.

For further improvement, we recommend to implement a model based reinforcement learning strategy. Since the environment state and action space is very large and the environment dynamic being very complex, the agent could gain a lot performance by learning the transition probability and reward distribution of the environment. This learned model of the environment could be used at every step in a real time planning fashion to simulate the future accumulated reward for some proposed action by the agent. Since the agent's performance is not based on fast decision-making, but less frequent good decisions, a monte carlo tree search algorithm would be well suited for this, considering that the agent would have the time to plan ahead for multiple action possibilities for each step. Some additional planning could have been directly programmed into the agent for simple, but hard to learn action. For example, to force the agent to do nothing in the case where the line is stable or to reconfigure the line to its original configuration when possible [2].

Link to the project notebook and code

The notebook and code for this project can be found on github at https://github.com/Toaster545/RL_Project.git. The notebook can be run on google colab.

References

[1] Adrian Kelly et al. *Reinforcement Learning for Electricity Network Operation*. Mar. 2020. DOI: 10.48550/arXiv.2003.07339. arXiv: 2003.07339 [eess]. (Visited on 12/16/2024).

[2] Antoine Marot et al. "Learning to Run a Power Network Challenge: A Retrospective Analysis". In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. PMLR, Aug. 2021, pp. 112–132. (Visited on 12/16/2024).

[3] von Meier, Alexandra. *Electric Power Systems: A Conceptual Introduction*. Wiley, June 2006. ISBN: 978-0-470-03642-6.

[4] Benjamin Donnot et al. *Introducing Machine Learning for Power System Operation Support*. Sept. 2017. DOI: 10.48550/arXiv.1709.09527. arXiv: 1709.09527 [stat]. (Visited on 12/16/2024).

[5] Antoine Marot et al. *Learning to Run a Power Network Challenge for Training Topology Controllers*. Dec. 2019. DOI: 10.48550/arXiv.1912.04211. arXiv: 1912.04211 [eess]. (Visited on 12/17/2024).

[6] B. Donnot. *Grid2op- A testbed platform to model sequential decision making in power systems*. 2020. URL: %5Curl%7Bhttps://GitHub.com/Grid2Op/grid2op%7D.

[7] Horia Mania, Aurelia Guy, and Benjamin Recht. *Simple random search provides a competitive approach to reinforcement learning*. 2018. arXiv: 1803.07055 [cs.LG]. URL: https://arxiv.org/abs/1803.07055.

[8] *Welcome to Grid2Op's Documentation — Grid2Op 1.10.4 Documentation*. URL: https://grid2op.readthedocs.io/en/latest/index.html (visited on 12/20/2024).