Report for the "Electricity power network" Mini-project

Team name/project: L2RPN
Group name: Electricity
Team members:

ARBACHE Jean <jean.arbache@u-psud.fr> group 1
GARDILLE Arnaud <arnaud.gardille@u-psud.fr> group 1 bis
NAJAR Farid <farid.najar@u-psud.fr> group 1
PATTE Sebastien <sebastien.patte@u-psud.fr> group 3
RANAIVOHARISOA Andrianiaina <andrianiaina.ranaivohraisoa@u-psud.fr> group 3
SCHOECH Nathan <nathan.schoech@u-psud.fr> group 1

Challenge URL: https://codalab.lri.fr/competitions/398
GitHub repo of the project: https://github.com/Mini-projet-Electricity/Mini-projet-Electricity.git
YouTube video link :
Last submission id : 8206
Presentation slides link: https://prezi.com/p/6pstzlw6dbru/electricity/?utm_content=2001&refcode=email00selligent000v0&utm_medium=email&utm_source=prezi&utm_campaign=16803456

Context and Introduction :

Issues

- What's the topic of the project ?

The L2RPN project we chose to work consists of managing a power grid using machine learning. The objective is to assist human dispatchers in making the right decision in real life. This project addresses technical aspects related to the transmission of electricity in high voltage power networks (63kV and above), such as those managed by the company RTE (Réseau de Transport d'Electricité), the French TSO (Transmission System Operator). Numerous improvements of the efficiency of energy infrastructure are anticipated in the next decade from the deployment of smart grid technology in power distribution networks to more accurate consumptions predictive tools.

- What do we manage inside our power grid system ?

We distinguish three Main parts inside a power grid : production, transmission and distribution. It is clearly shown in illustration 1. We are only interested in controlling the transmission, that is to say the high voltage line and the transmission substation.

•

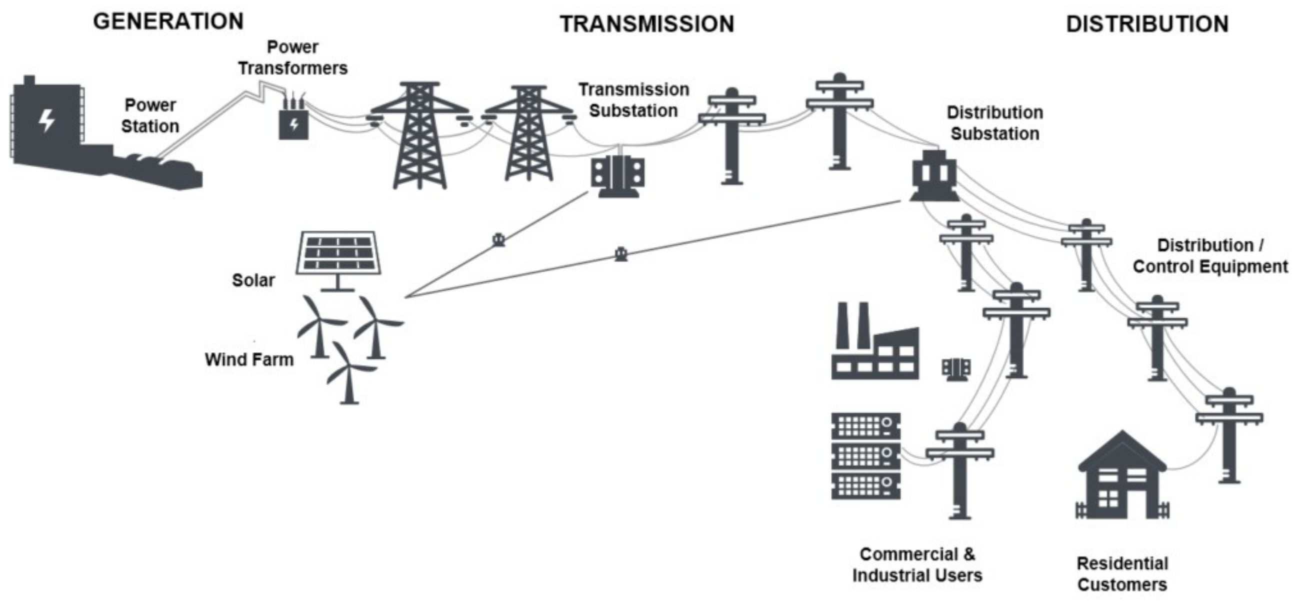


Illustration 1: A power grid system

What are the constraints to respect ?

Firstly, a line mustn't be overloaded, or it will melt. And one a line is destroyed, it can creates a chain reaction and damage the whole circuit. Secondly, A central station must always get rid of the energy it produces. Otherwise it could be destroyed, as happened in Fukushima. Thirdly, the consumption needs must be addressed, since an lake of energy destabilize the distribution, and is generally responsible of an electrical blackout.
In Illustration 2 we can see the consequences when a line get overloaded. The line on top has been overloaded at first (threshold =100) then it will be melt down because of heat. As consequences the load of this line will be distributed on other lines. This distribution will led to the overload of the three line. Then the destruction of the whole grid.

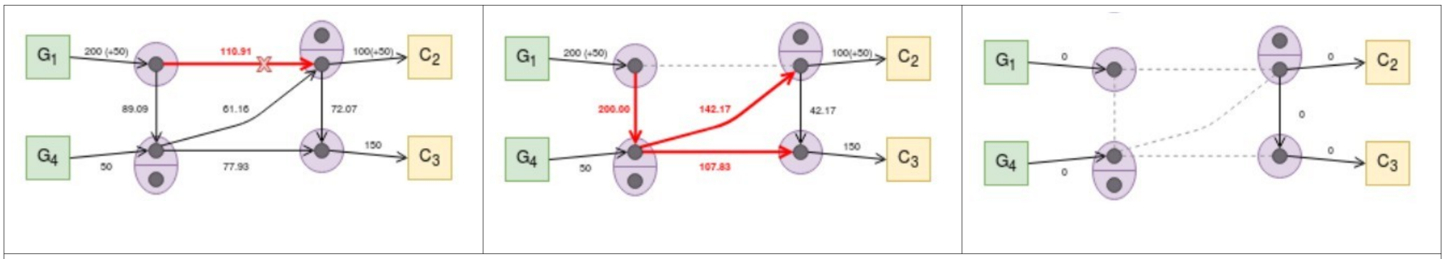


Illustration 2 : Dominoes effect in the grid
Yellow squares represents the consumption. Green squares represent power generator.
Lines in black are line loaded with a tension < 100 (The threshold). Lines in red are overloaded lines (tension>threshold). Pointed lines are broken lines.

- Why does the energy transition makes thing complicated for dispatchers ?

The difficulty of the task arises from the complexity of the network architecture, also called grid topology, which frequently changes due to events such as hardware failures, planned maintenance or preventive actions. On top of that, rising renewable energies are less predictable than conventional productions systems (e.g. nuclear plants), bringing more uncertainty to the productions schemes. Also, electric vehicle should increase the power consumption. What's more, electricity can hardly be store, the only reasonable mean being to dam electricity inside a hydroelectricity power generator. It remains complex, and as a consequence, unconsumed power is generally lost.

In this context, we are interested in developing tools that will assist dispatchers to maintain a power grid safe and face the increasing complexity of their task.

- What is reinforcement learning ?

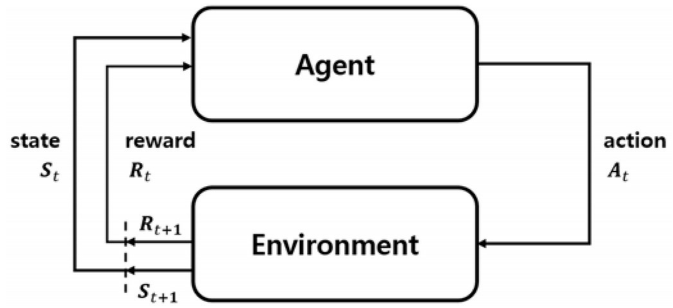


Illustration 3: principle of reinforcememnt learning

Reinforcement learning is one of the three main category of machine learning. It’s principle is presented on figure 3. Considering the current state of the environment, the agent takes an action. The environment returns its new state and a reward, which can be negative. The objective of the agent it to maximize it’s total reward.

It is generally considered as being quite difficult to use, and to implement, but contains very powerful methods. With very few information, the agent can adapt itself to an evolving environment, and the most important is that he is capable to consider the interactions between states on the long term thank to rewards. But the computation needed increases with the number of possible states.

- It is possible to use reinforcement learning ?

The power grid can be accurately simulated with a simulator implementing the laws of physics (ordinary differential equations). Therefore our problem could be solved by reinforcement learning, since we can train an agent using an Environment simulator. However implementing reinforcement learning algorithm on such complicated environment (infinite continuous states) need advanced skills in both mathematics and machine learning.

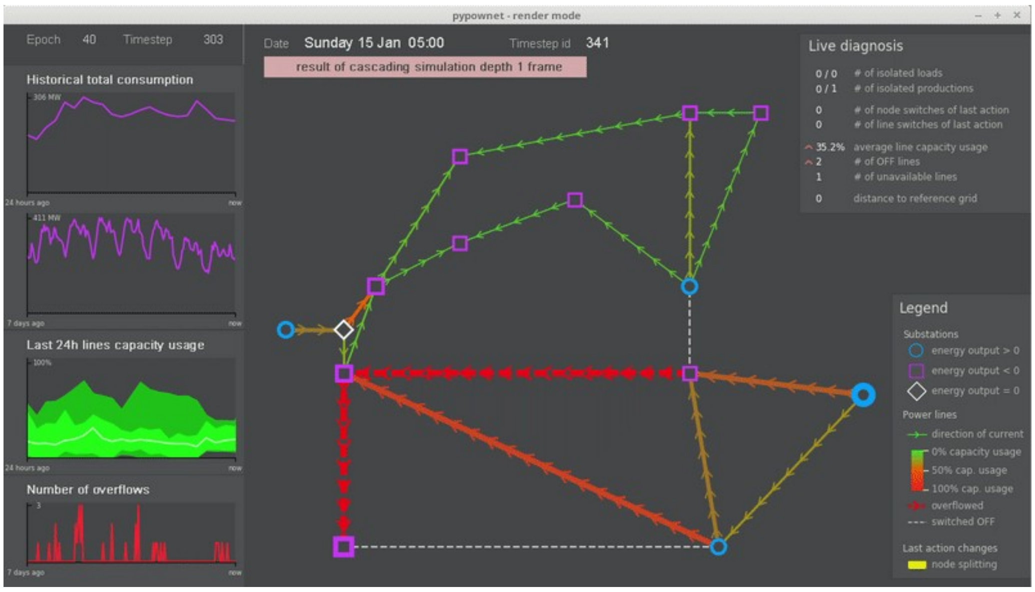
- How did you proceed anyway ?

After presenting the simulator we used, we will explain how we used the baseline agents and supervised learning to bypass the difficulty of using reinforcement learning. We wanted to improve the rapidity of the best pre-existing agent, while preserving it’s performance.

The simulator

The environment simulator that we use is *PyPowNet, which stands for* PYthon POWer NETwork.
It simulate a power grid, it is to say a graph with current fluctuation from power plants to consumption centre, passing through connected lines and substations.

We show on figure 4 the render of pypownet. In reality, we didn’t used it for technical and efficiency issues. But it gives a good idea of the simulated environment. On this picture, two lines are dangerously overloaded, and two other lines are switched off. The production is represented by blue circles, and the consumption by pink squares. The white diamond is a substation.



Live diagnosis

```
0 / 0    # of isolated loads
0 / 1    # of isolated productions
0        # of node switches of last action
0        # of line switches of last action
^ 35.2%  average line capacity usage
^ 2      # of OFF lines
1        # of unavailable lines
0        distance to reference grid
```

Illustration 4: the renderer of pypownet

Here on figure 5 stands an examples of production, with different power sources. Each of them has it's own behaviour, which is generally unpredictable. We note the fluctuation of renewable energies.

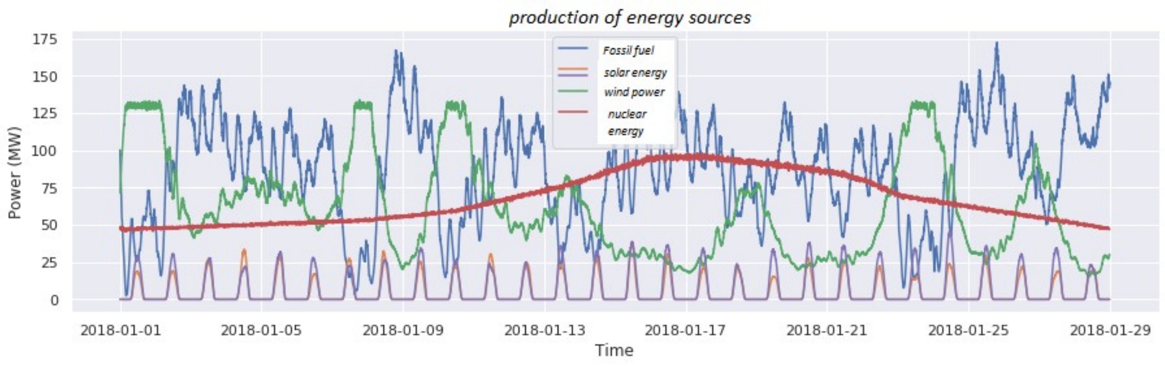


Illustration 5: Production of energy sources

On figure 6, the consumption is presented. It is periodical by day and by week, and therefore quite easy to predict.

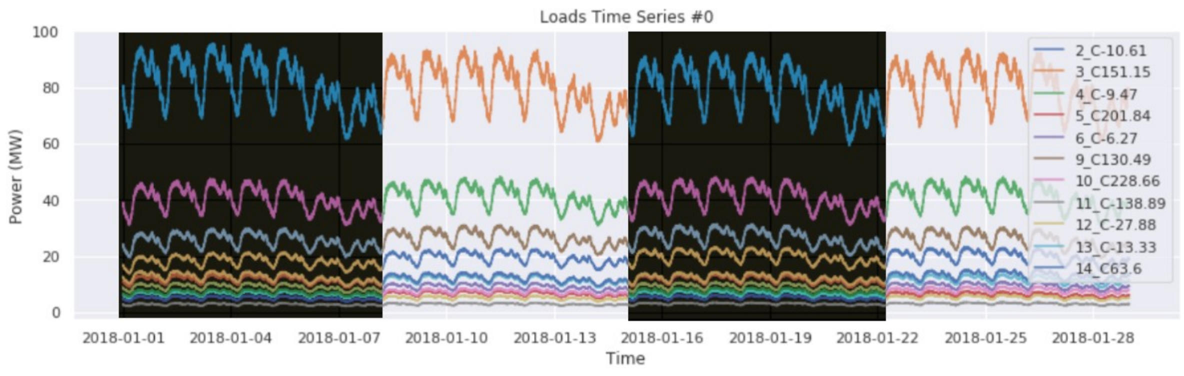


Illustration 6: The temporal periodicity of the electrical energy consumption

- How should an agent interact with this environment ?

An agent is an object, which has a constructor, and has a mandatory method : *act(State) → action. Giving a state, the agent has to return what it thinks to be the best action. The other method, feed_reward(…), allows a reinforcement learning agent to take the reward into account concerning it's policy. Since we do not use reinforcement learning, this method isn't used inside of our agents.*

Pypownet comes with 4 baseline agents, with self explanatory names : doNothing, greedySearch, randomLineSwitch and RandomNodeSplitting. Greedy search is a tree search agent : it tries as possibilities on its own, and select the ones that should return the best reward. It is quite slow, and only watches one step forward, but makes good enough decisions.

- What does the reward correspond to ?

The reward decrease when a line is overloaded. There is a big penalty facing a game over. The reward is better when the grid is "close" to the original grid. That explains the good score of the doNothing agent.
As the grid being small, taking an action generally gives bas a negative reward. That's illustrated by the scores of the random agents. But a "do nothing" action cannot give a positive reward. This is well illustrated at figures X and X, at the end of the report.

Algorithms description :

The environment is made so that we can use reinforcement learning algorithms inside of our agents. This approach is quite complicated, since there in an infinite number of possible state. We read in some articles (*reference*) about deep reinforcement learning. This methods seems very interesting, but way to complicated to implements for undergraduate students. Also, it would have been pretty slow to run.

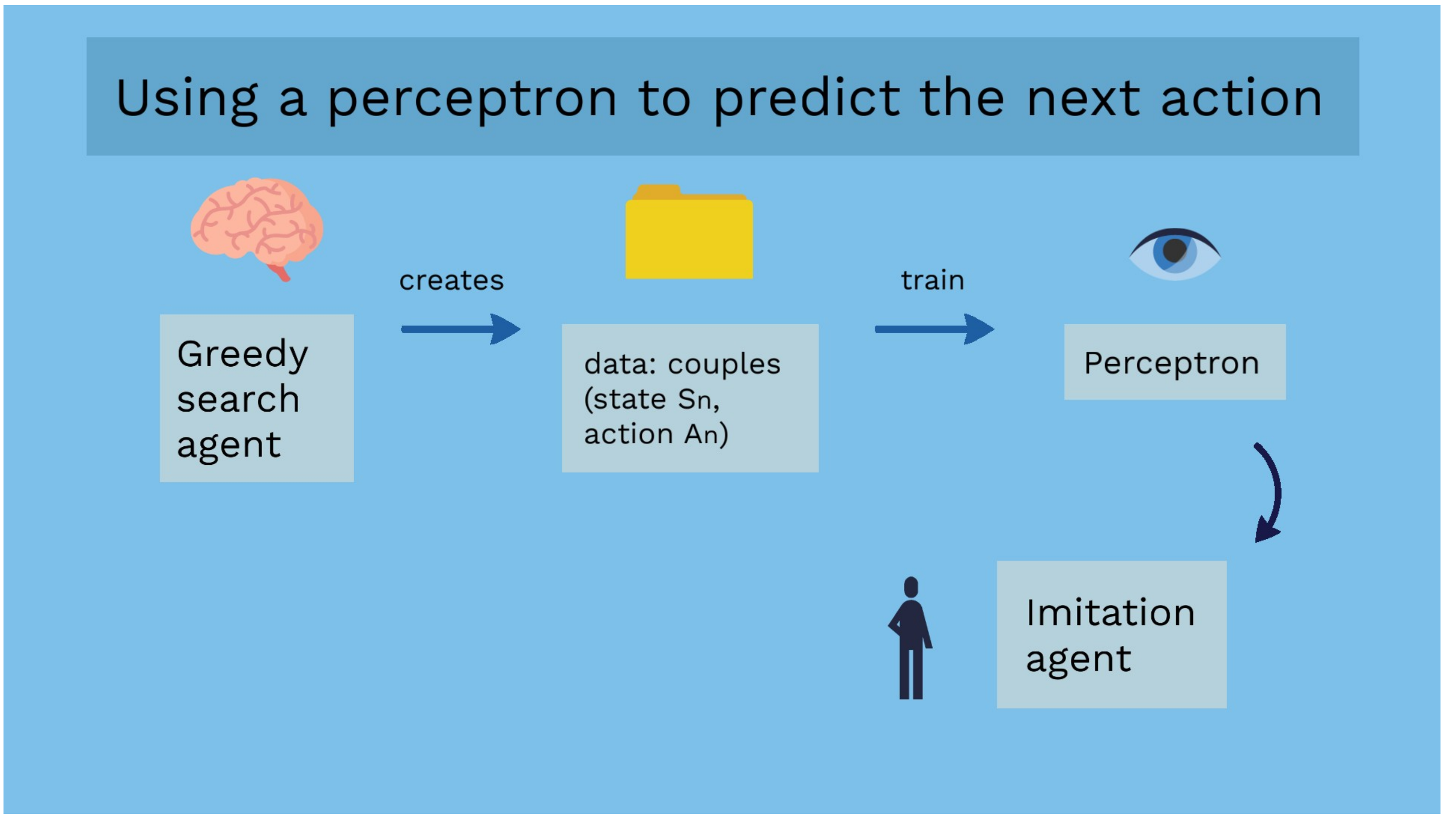
As a goal, we tried to imitate the greedySearch agent, which is quite good but very slow : It takes minutes to run, despite our grid being simple. We knew supervised learning algorithms gives an answer very fast.
So the idea was to imitate greedySearch, in order to approach it's performances, but being much faster to answer.

Imitation agent

Fist of all, you got to know som stuff about perceptrons.
A "perceptron" is a supervised learning method, which usually reveal itself very good at classification. And that's exactly what we want! With a sufficient amount of labelled data, it can predict the label of a data it has never seen before, provider that it has already seen some similar ones. Please read the Wikipedia (ref) page for more informations.

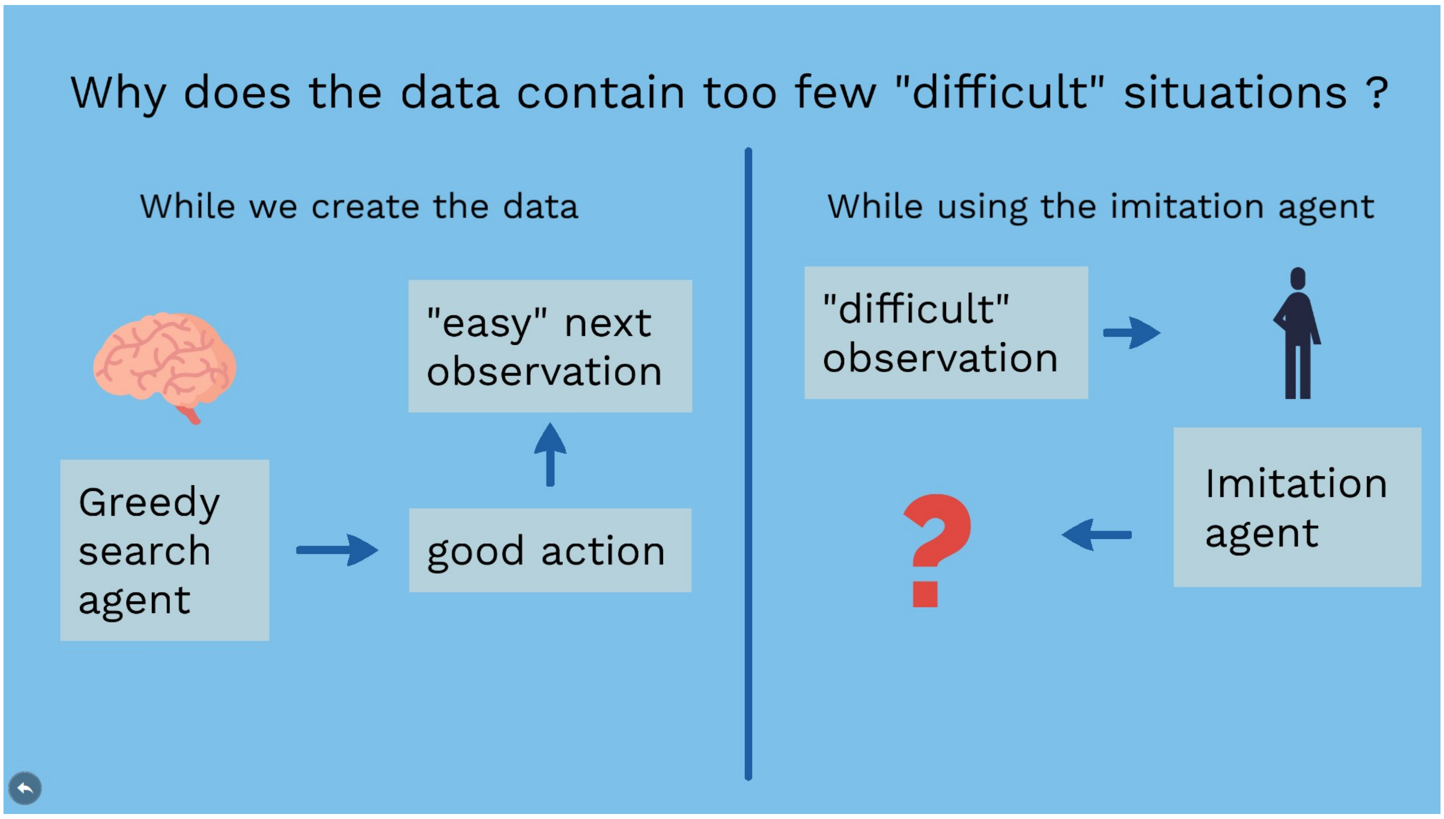
Training a perceptron

The first step in creating an imitation agent was obviously to produce the data from greedySearch. This method is shown on scheme 1, and is implemented in "Creating an imitation agent.ipynb".
To begin with, we let greedy run for thousands of iterations, and save both the action it faces, and the action it takes. With that dataset, we train a Then we export it's parameter inside our imitation agent, so that the imitation agent can use the perceptron to determine the action to take when the environment send him a state.


Using a perceptron to predict the next action
Greedy search agent
creates
data: couples (state Sn, action An)
train
Perceptron
Imitation agent

Scheme 1: The training of our imitation agent

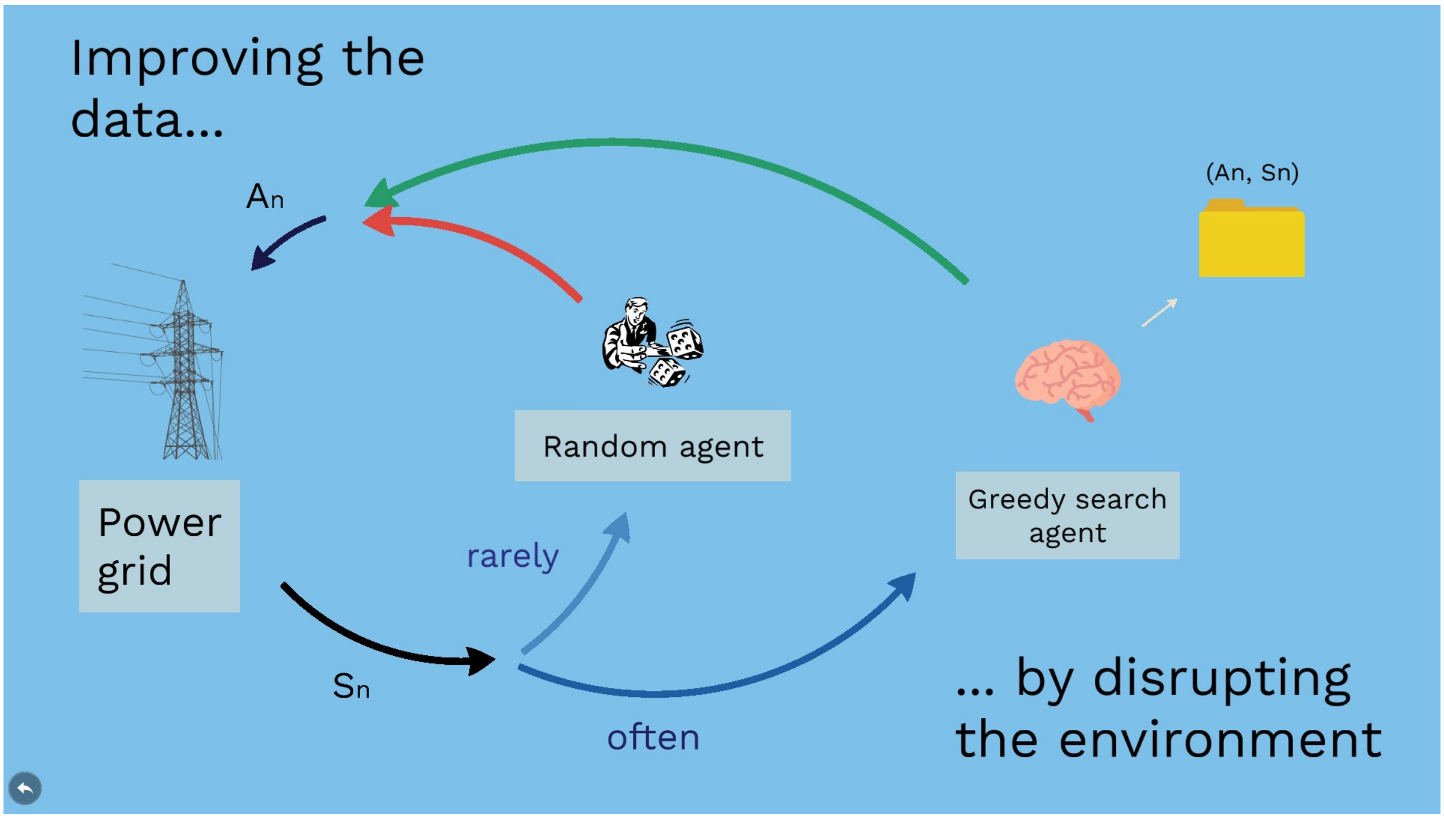
However, unlike greedySearch, our agent is very bad facing a difficult situation. Why is that ?
Because greedySearch always take good decisions, so that the environment is very stable, and it rarely faces difficult situation. But when it appends, he knows what to do, unlike our poorly trained agent. It is so like training someone on driving in a straight road and then test him in the city of Paris.
This problem is illustrated of scheme 2.



Scheme 2: Training issue

Improving the data

We must force greedySearch to face some difficult situations during the training and then learn from him what to do in this case. In order to do so, we sometimes (with a certain probability distribution) call randomLineSwitch or randomNodeSplitting instead of greedySearch, without saving those iterations. It disrupt the environment, and by consequence helps improving the data.

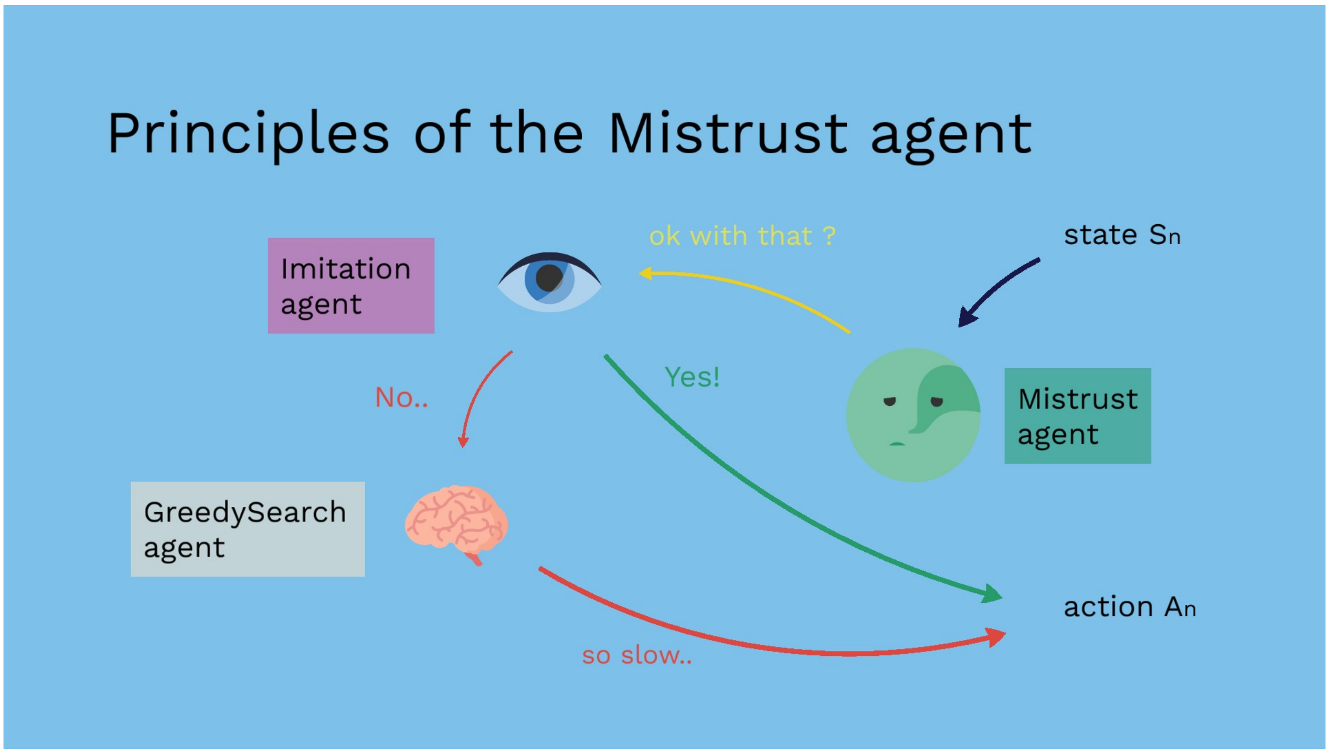


Scheme 3: Disrupting the environment

As it is usually the case for supervised learning, the performance increases with the size of the dataset. It will be clearly shown on the graph 7, a the end of the rapport.
At this stage, our agent works fine. However, we would like him to keep working fine, even when it reasonably recedes from it's training conditions.

Mistrust agent

We created an evolution of the imitation agent, called the Mistrust agent.
As the imitation agent, it still has an trained perceptron. But it also has an instance of greedySearch.
The perceptron of scikitLearn, the python module we used for supervised learning, can give the probability it this it action to be the right one. We call this mesure “certainty”.
When the certainty is lower than a predefined threshold, the agent as it's greedySearch instance to choose the action. Otherwise, it returns the action given by the perceptron.
This decision system is illustrated by the scheme 4.



Scheme 4: The mistrust agent

Credit system

The mistrust agent sometimes calls greedySearch too often. In the submission of the Mistrust agent with a big dataset, greedySearch was called nearly 45% of the time. Despite it's efficiency, it remains far too slow to be of a real interest. With the same certainty threshold, the same agent with as small dataset only called greedySearch 10% of the time. We deduce that a constant threshold doesn't allow us to control the frequency of the calls.

As an answer to this problem, we've set in place a credit system. When the Mistrust agents calls greedySearch, the threshold decrease. Otherwise, it increases as little. We choosed the parameters, so that greedySearch is called around 10 percent of the time.

Results :

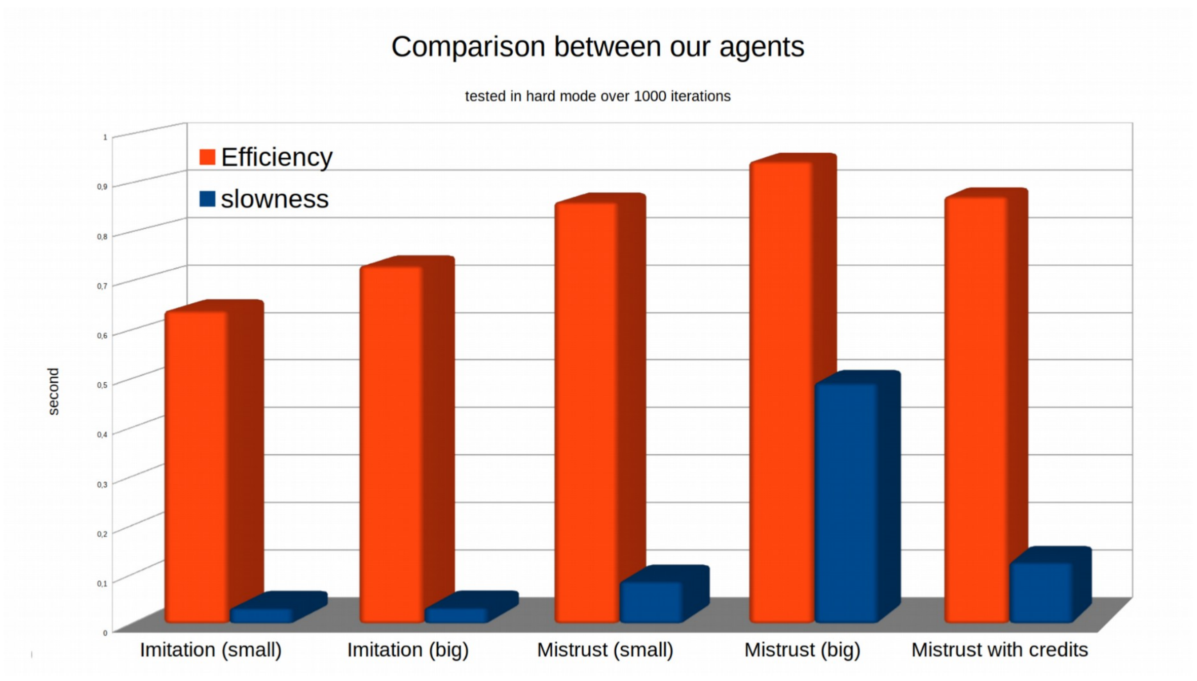


Illustration 7: Comparison between our agents

Illustration 7 shows that agents trained with a bigger dataset are more efficient. Obviously, that doesn't affect the rapidity in the case of the imitation agent, because the perceptron answer at constant speed. The problem described in the previous paragraph with the mistrust agent appears clearly here. The credit system seems to be a really good compromise.
Now let’s compare him with the baseline agents.

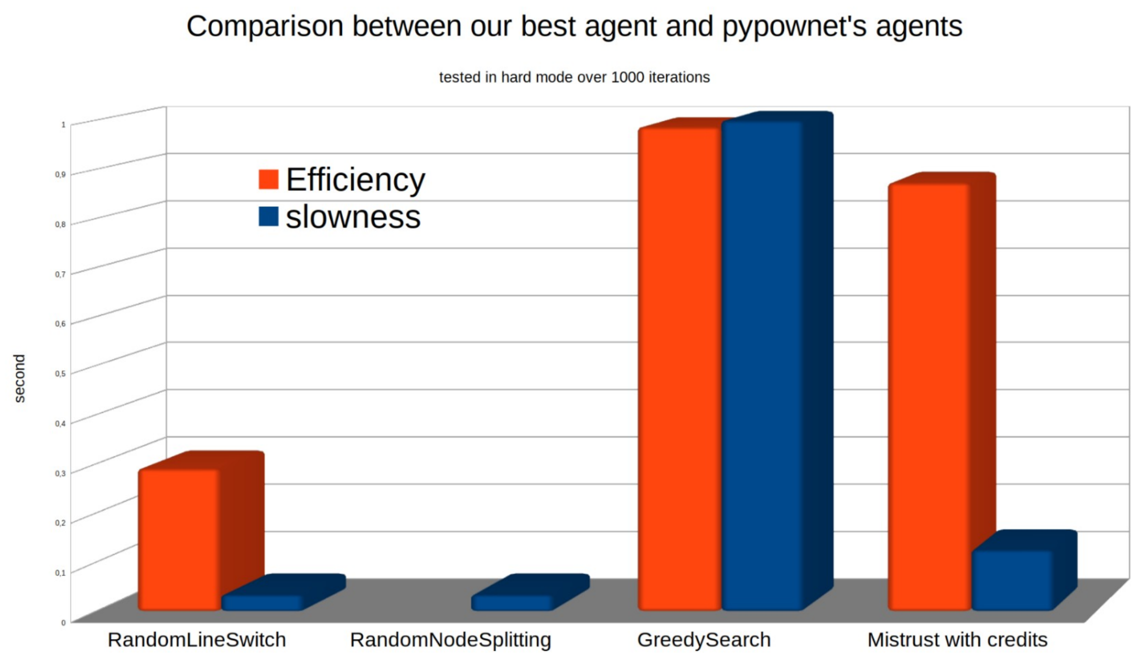


Illustration 8: Comparison between our best agent and pypownet’s agents

On the previous illutation, the 7th one, we can see that the “random agents” are very fast, but with terrible scores, for obvious reasons. Mistrust with credits approaches the performance of GreedySearch, while being ten times faster.

Further analysis

Now let’s look deeper into the interactions between the agents and the environment.

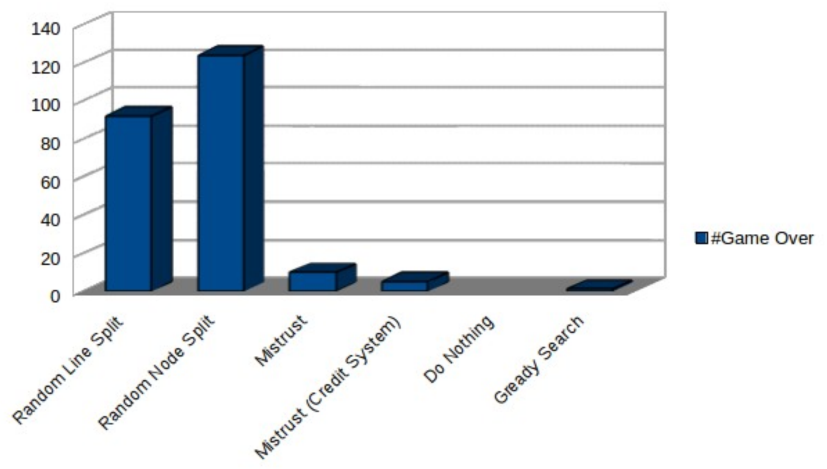


Illustration 9: number of Game over per agent (over 1000 iterations; in hard mode)

We see that the doNothing agent doesn't face any game over. One of our supervisor explained us the reason why: the simulated grid is very small, so that taking an action modifies the grid a lot, and has big consequences. Also, the initial grid already has a good configuration. By consequence, doing something is rarely a good choice. That would big much different with a bigger grid. Unfortunately, that would be too heavy to simulate.

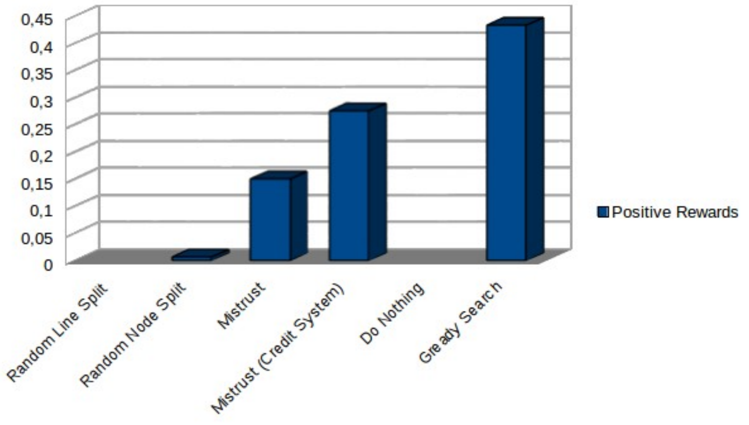


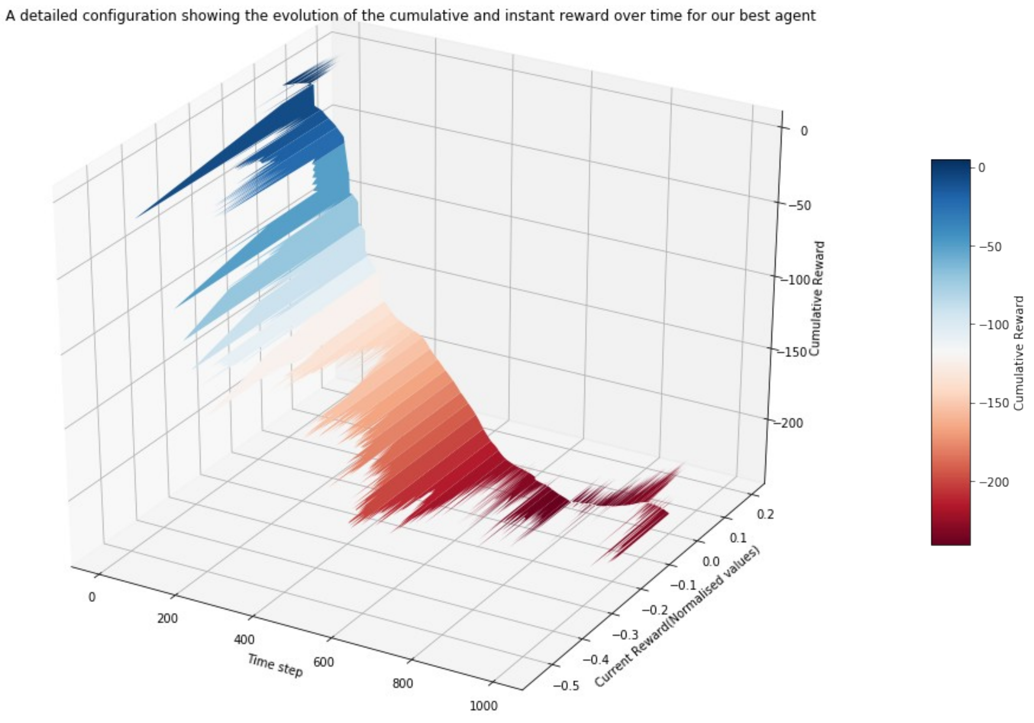
Illustration 10: part of positive reward

The 9th scheme shows an other interesting thing about the reward system. The doNothing agent, despite making a very good score, never has positive reward. It means that only real actions can be positively rewarded.

Greedy has much more positive reward that Mistrust with credits. That’s quite surprising, because their final score are pretty close. It let us suppose that despite being good, Mistrust doesn't choose the finest action for increasing the next reward.

Illustration 11 & 12 gives us deeper vision of what is happening during the test. In the first graph we can see both cumulative reward and instant reward on the same in two different dimensions.

The second one compare compare all the agents in term of cumulative rewards fluctuating over the time.



llustration 11: The mistrust agent with credit system

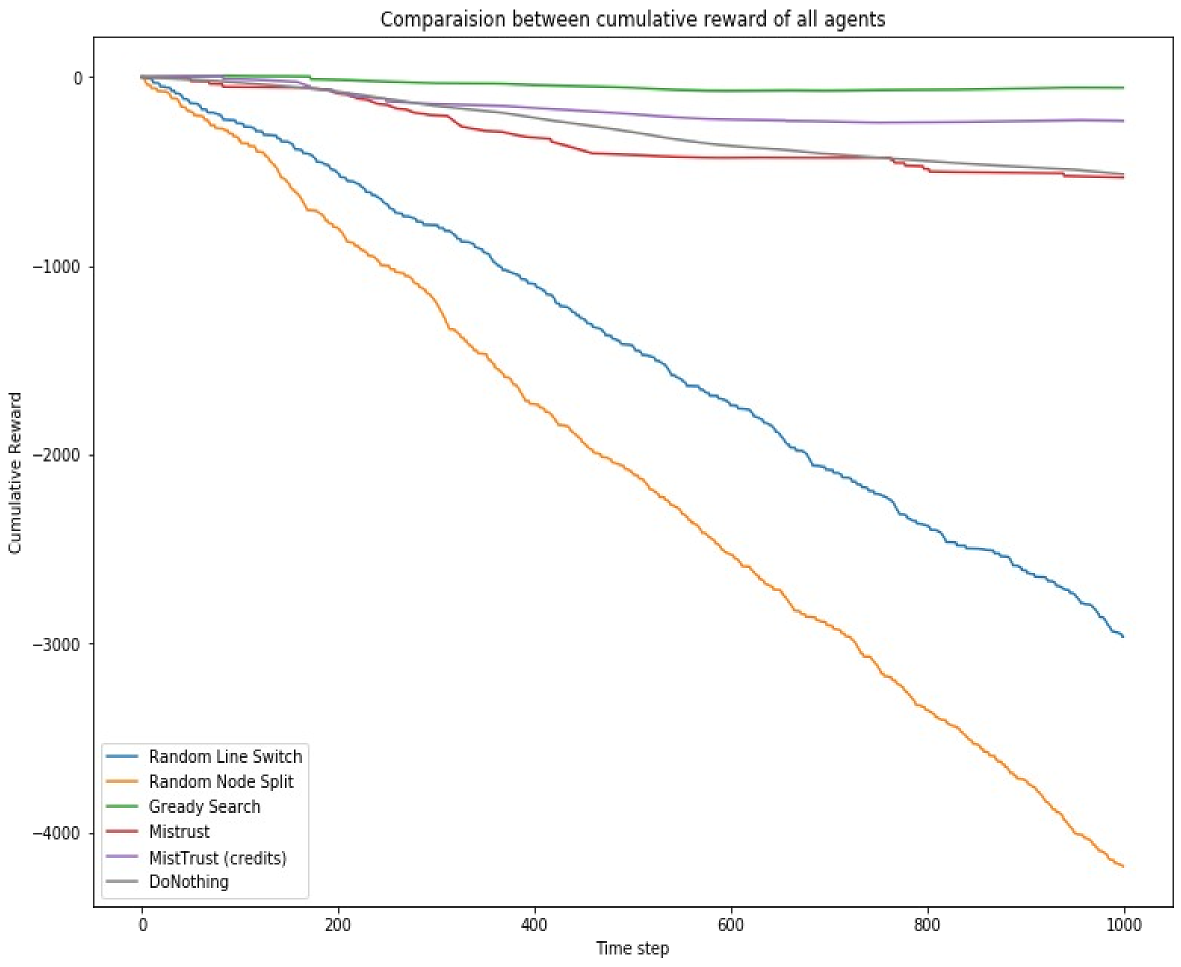


Illustration 12: All agents

Discussion and conclusion :

We succeeded in our task of improving the time's performance, while preserving the a good final score.

Trying with different supervised learning methods would have been interesting. But we chose to explore new approaches to the problem rather than spending time on this technical issue.

However, we can regret that the agent we try to imitate, a tree search agent, only maximise the next step reward. There is neither long term strategy, nor even tactics to avoid the game over.
The project could be continued by increasing the steps range of the tree search agent (greedySearch). The literature also suggest us to implement deep reinforcement learning. As far a we understood, it means using a supervised learning methods to approximate the states, and then using reinforcement learning.
We could then imitate those agents, which would probably be even slower and more efficient that greedySearch.

Going back to the topic of power transmission, machines are not ready to rule so sensible network for now. But we're convinced that it could become an amazing tool for the dispatchers in the coming years.

Bonus :

Cross-validation is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting.

overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.

Sources :

https://pypownet.readthedocs.io/en/latest/?fbclid=IwAR3LbtabX2uOgn0HnW_GFc0S2iExUHhAl3WrRjfSj1vQKEkAmSGk6H3x4u8
[2] : http://www.numpy.org/
https://github.com/MarvinLer/pypownet

https://buildmedia.readthedocs.org/media/pdf/pypownet/latest/pypownet.pdf

https://www.rte-france.com/fr/la-carte-du-reseau