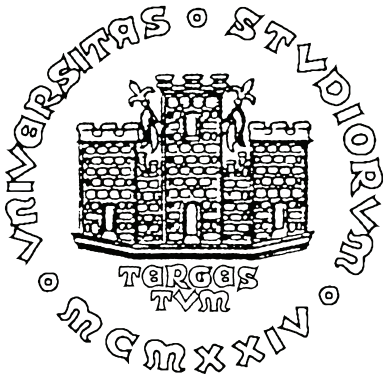

Università degli studi di Trieste

Dipartimento di Matematica e Geoscienze

Master's Degree in
Data Science and Scientific Computing

Optimizing fault search in power grid outages through Reinforcement Learning

Candidate:

Angela Carraro

Supervisor:

Antonio Celani, ICTP

Co-Supervisors:

Andrea Zancola, AcegasApsAmga

Luca Bortolussi, UniTS

Academic year 2020/2021

Abstract

Riassunto in italiano.

Contents

Acronyms

AAA AcegasApsAmga S.p.A.. 1

DTP Decision-Theoretic Planning. 6

MDP Markov Decision Process. 6, 7, 9–11

ML Machine Learning. 5

POMDP Partially Observable Markov Decision Process. 9–12

RL Reinforcement Learning. 5, 6, 9

Chapter 1

Introduction

1.1 The problem

AcegasApsAmga S.p.A. (AAA) is a company based in Trieste and subject to the direction and coordination of Hera S.p.A., called also Hera Group, which is a multiutility company based in Bologna, Italy. Hera operates in the distribution of gas, water, energy, and waste disposal in some Italian provinces.

The project involves AAA Trieste power grid.

1.2 The power grid

Describe the power grid and all its elements.

Chapter 2

Data Management

2.1 Introduction

2.2 Raw to bronze

2.3 Bronze to silver

2.3.1 The circuit graph

2.3.2 The electrical graph

2.3.3 The substations graph

2.4 Bisection

2.5 Weighted bisection

Chapter 3

Reinforcement Learning

3.1 Introduction

As described in [1], Reinforcement Learning (RL) is a paradigm of Machine Learning (ML), along with others, like supervised learning and unsupervised learning. It analyzes how it is possible to learn the best course of action in an environment based on maximizing some given numerical reward. The learner is not told which actions to take, but learns empirically which are the ones that lead to a bigger reward. It is also possible that the current action influences not only the immediate reward, but also the next status of the environment, and, through that, all the other rewards to come. In this case, we speak of delayed reward, which, together with trial-and-error search, form the two distinctive features of RL.

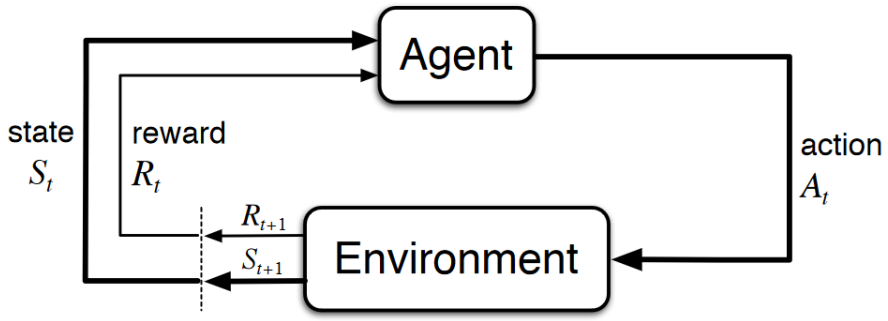

Figure 3.1: The agent-environment interaction [1]

In an RL problem we have a learning *agent*, the decision-maker, that interacts over time with the *environment* in which it is placed — which includes everything outside the agent — in order to achieve a *goal*, which is to maximize the total reward it receives over the long run. As we see in Figure 3.1, the agent can sense aspects of its environment through the *states* — which can also be seen as a representation of the environment itself, it can choose *actions* to influence the environment, and it receives a *reward* based on the outcome of its actions. We consider to be a reinforcement learning method any method that is well suited to solving problems framed in this way. Other elements of the RL system are a *policy*, a *value function*, and a *model* of the environment. A *policy* describes the behavior of the agent at a given time, using a mapping from states to actions to be taken in those states. A *value function* determines the value of a state as the total amount of reward an agent can expect to accumulate over the future, starting from that state. Instead, a *model* of the environment reproduces the behavior of the latter, and it is used for planning actions before actually experiencing them. For example, given to the model a state and action, it might predict with a certain probability the resultant next state and next reward.

3.2 Finite Markov Decision Processes

The treatment of this topic will closely follow the one presented in [1].

Markov Decision Processes, or MDPs, are used to formalize sequential decision-making, where actions influence the given rewards and the next states of the system, and through the latter also future rewards can be affected. Thus, MDPs need to balance both immediate and delayed rewards. This formalism is used both in decision-theoretic planning (DTP), RL, and other learning problems in stochastic domains [2].

We assume that the process evolves through a sequence of discrete time steps, $t = 1, 2, 3, \ldots$, even if it is possible to extend to the continuous case. These steps do not need to reflect fixed intervals of real time, but they can refer to arbitrary successive stages of decision-making and acting. At each time step t, the agent detects the *state* of the environment, $S_t \in \mathcal{S}$, based on which it selects an action $A_t \in \mathcal{A}(s)$ (if the action set is the same in all states, we will write it simply as $\mathcal{A}$). Then, as a consequence, the agent receives, one time step later, a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and ends up in a new state S_{t+1}. Thus, given this sequential

process, it emerges a sequence, or *trajectory*, given from

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, \dots \tag{3.1}$$

In a *finite* MDP we have that the set of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ have finite cardinalities, respectively $|\mathcal{S}|, |\mathcal{A}|$, and $|\mathcal{R}|$.

Since it is a *finite* MDP, the random variables R_t and S_t have well-defined discrete probability distributions, and since we are considering a *Markov* process, it holds the *Markov property*, also called *memorylessness property*, and thus the *dynamics* of the process through the space of states, or the *model* of the environment, depend only on the preceding state and action:

$$p(s', r \mid s, a) := \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}\,, \tag{3.2}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

One could compute also the *transition probabilities*, defined (with a little abuse of notation, since we continue to use the letter p) as

$$p(s'|s, a) := \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\,. \tag{3.3}$$

The expected rewards for state–action–next-action triples can be computed as $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$,

$$r(s, a, s') := \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \cdot p(s', r|s, a)\,. \tag{3.4}$$

The MDP can also be seen as the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $r = r(s, a, s')$ is the expected immediate reward that the agent receives for taking action a in state s [3, 4].

To describe how likely it is for an agent to take any given action from any given state, we use the notion of policy. A *policy* is a class of probability distributions $\pi(a|s)$ over an action $a \in A(s)$ for each state $s \in \mathcal{S}$:

$$\pi(a|s) = \Pr\{A_t = a | S_t = s\}\,, \tag{3.5}$$

which describes the probability that the agent chooses that action being in that state at that time step t.

The agent's goal is to maximize the *expected return*, o *cumulative reward*, it receives in the long run given a certain policy π:

$$\mathbb{E}[G_t] = \mathbb{E}_{p(\cdot,\cdot|s_t,a_t),\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]. \tag{3.6}$$

where $\gamma \in [0, 1]$ is the *discount factor*, and it determines how much we value future rewards. With $\gamma = 0$ the agent considers only immediate rewards and discards future ones, while with $\gamma = 1$ the agent will take into account, in the same way, every reward it receives, considering a very long series of events.

Actually, we can take this average with respect to the distribution of rewards themselves, so as to ignore the dependency on the distribution of rewards. This is a distinctive property of the fact that we are working with a *known model of the environment*. So, since we are only interested in optimizing averages, we do not care any longer about what is the actual distribution of the rewards. Thus, we can rephrase the agent goal as

$$\mathbb{E}[G_t] = \mathbb{E}_{p(\cdot|s_t,a_t),\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s, a, s')\right]. \tag{3.7}$$

To reach its goal, the agent must estimate how good it is to be in a given state if it is following a certain policy. This value is stored in the *(state-)value function*. Formally, the value of state s under policy π is

$$V_\pi(s) := \mathbb{E}\left[G_t \mid S_t = s\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k R_t \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}\,, \tag{3.8}$$

which is the expected return when starting in state s and following policy π thereafter. If there exists a terminal state, which causes the process to terminate if reached, its value is always zero.

Similarly, the value of taking action a in state s under policy π is stored in the *action-value function*, or *quality*, which is expressed with

$$\begin{aligned} Q_\pi(s, a) &:= \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k R_t \,\middle|\, S_t = s, A_t = a\right], \end{aligned} \tag{3.9}$$

which is the expected return starting from state s, taking action a and following policy π thereafter.

We have that the following relation holds:

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s, a)\,. \tag{3.10}$$

The value function possesses a fundamental property, which is extensively used

in RL: it satisfies the following recursive relation

$$\begin{aligned} V_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V_\pi(s') \right] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma V_\pi(s') \right] , \end{aligned} \tag{3.11}$$

which is called *Bellman equation for* V_π.

Similarly, we have the *Bellman equation for* Q_π:

$$\begin{aligned} Q_\pi(s, a) &= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a') \right] \\ &= \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a') \right] . \end{aligned} \tag{3.12}$$

3.3 Partially Observable MDPs

The presentation of this topic will mainly follow the one in [5].

What happens if the agent does not know anymore with full certainty the state of the environment, maybe due to imperfections or limitations in the agent's sensors, or to errors in the interpretation of the environment, or simply to unknown aspects of the environment itself?

If some features of the state are hidden from the agent, and the latter can sense only a part of the real state of the environment, the resultant state signal will no longer be Markovian, breaking a key assumption of most RL methods, like the MDP formulation. Luckily, we can consider an extension of the (fully observable) MDP framework, that can deal with the uncertainty originated from the imperfect states perceived by the agent or the uncertain action effects.

A *partially observable Markov decision process*, or POMDP, is a framework that considers environments that are only partially observable to the agent, but allows anyway for optimal decision-making [3], contrary to the requirement of full knowledge of the environment of MDPs.

The fact that the environment is partially observable can derive mainly from two reasons:

- different states generate the same observation, due to the same sensor reading, caused by the agent's limited knowledge of the environment;

- sensor readings are noisy, so the same state can generate different observations due to different sensor readings.

Thus, we have that the state of the environment is not uniquely identified by the agent's observations, and situations that appear similar to the agent may require instead different actions.

Interestingly, we can consider fully observable MDPs as a special case of POMDPs, in which the observation function maps each state to its correct unique observation deterministically [6].

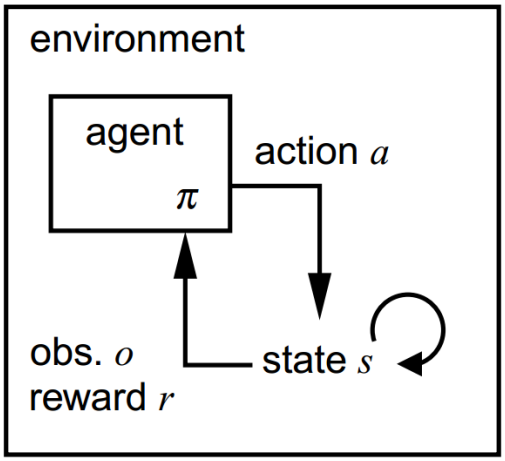

Figure 3.2: The agent-environment interaction in a POMDP [5]

Since a POMDP is an extension of an MDP, they share many elements. Also in POMDPs the time is divided into different steps, and in each one of them the agent has to take an action. As for the MDPs, we will consider discrete and finite models, which are simpler than the continuous ones. So the environment is represented with a finite set of states $\mathcal{S} = \{s_0, s_1, s_2, \ldots, s_N\}$, and the finite set of possible actions is $\mathcal{A} = \{a_0, a_1, \ldots, a_K\}$. But instead of perceiving directly the state in which the environment is, the agent senses an *observation* of it — a signal that depends on the true state of the system but provides only partial information about it. The set of observations is discrete and finite: $\mathcal{O} = \{o_0, o_1, \ldots, o_M\}$, and represents all the possible sensor readings the agent can receive.

If the system is in state s and the agent performs action a, we have that the system transitions to state s' according to the transition probability $p(s'|s, a)$ as in a MDP, but the agent receives observation o' with probability

$$z(a, s', o') = \Pr(o'|a, s')\,, \tag{3.13}$$

using the notation of [6].

Figure 3.2 represents the interaction between these elements in the POMDP.

Thus, a POMDP can be described as the tuple $(\mathcal{S}, \mathcal{A}, p, r, \mathcal{O}, z)$, where $r = r(s, a, s')$ is the expected immediate reward, as in an MDP [3].

The goal of the agent is, once again, to find the optimal policy π that maximizes the expected return in (3.7). To solve both MDPa and POMDPs, one can use several different methods, which are classified according to the dimensions of the state and action spaces as well as based on whether the model of the environment is available or not. In fact, there are *tabular methods* — which are exact and use arrays to store value functions, or *approximate solution methods* — which employ function approximators, using limited computational resources; then there are *model-based methods* — which make use of the known model of the environment and of planning to find the exact optimal policy, or *model-free methods* — which are trial-and-error learner.

Often finding an exact optimal policy for POMDPs can be computationally very expensive, so even for relatively small problems it is required to use approximate methods [7]. They are divided into two main classes: *value-function methods* — or *action-value methods* as in [1], that attempt to learn an approximate value function of the belief states (which are probability distributions over the states); and *policy-based methods*, that search for a good policy within a fixed class of parameterized policies. We will focus on the second ones, since it is what we will implement to solve our problem.

3.4 Policy gradient methods

A *policy gradient method* attempts to optimize the parameters of a parametrized policy using either gradient ascent or gradient descent in the parameters' space, based on the expression of the agent's goal. In fact, it tries to maximize (or minimize) the expected return of a policy inside the policy class dictated by the parametrization. This method then uses directly the policy to choose the actions, instead of consulting a value function, which however might still be used to learn the parameters.

Policy gradient methods present some perks that make them highly valuable: the parametrized policy allows for direct incorporation of potential environment insights, they can naturally switch from the discrete setting to the continuous one, and they converge to at least a locally optimal policy, even if, as a drawback, the convergence can be very slow and finding the global optimum instead of a local one

can be hard [8]. Another distinctive trait of these methods is that they can also naturally handle partial state information. In fact, if a state variable can not be observed, then we can choose the policy parametrization such that the parameters do not depend on that state variable. Thus, we can directly apply these methods to POMDPs, without having to make any changes.

We will denote the policy's parameters' vector with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, and we will write, as in (3.5),

$$\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}\,, \tag{3.14}$$

for the probability of taking action a being in state s at time t with parameters $\boldsymbol{\theta}$ [1]. The policy can be parametrized in any way, as long as it is differentiable with respect to its parameters, which means that the column vector of partial derivatives of π w.r.t. the components of the parameters' vector $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta})$, exists and is finite for all $s \in \mathcal{S}, a \in \mathcal{A}$ and $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. In particular, as described in [8], for discrete problems is often used the (exponential) soft-max distribution (i.e., Gibbs or Boltzmann distribution)

$$\pi(a|s, \boldsymbol{\theta}) = \frac{e^{\phi(s,a)^\top \boldsymbol{\theta}}}{\sum_{b \in \mathcal{A}} e^{\phi(s,b)^\top \boldsymbol{\theta}}}\,, \tag{3.15}$$

while for continuous problems it is used the Gaussian distribution

$$\pi(a|s, \boldsymbol{\theta}) = \mathcal{N}(\phi(s, a)^\top \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\,, \tag{3.16}$$

where $\boldsymbol{\theta}_2$ is an exploration parameter, and $\phi(s, a) \in \mathbb{R}^{d'}$ is a feature vector characterizing state s and action a [9].

To learn the policy parameters $\boldsymbol{\theta}$ we will use the gradient of some scalar *performance measure* $J(\boldsymbol{\theta})$ — which can also be directly the expected return — with respect to the policy parameters. If the performance $J(\boldsymbol{\theta})$ measures the rewards that the agent receives, we want to maximize it, so we update the parameters approximating gradient *ascent* in $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}\,; \tag{3.17}$$

instead, if the performance measures the costs that the agent encounters, we seek to minimize it, so we use approximate gradient *descent* in $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}\,; \tag{3.18}$$

where, in both cases, $\alpha > 0$ is a *step-size parameter* or the *learning rate*.

As done before, we will treat the episodic case, for which we define the *performance measure* $J(\boldsymbol{\theta})$ as the average value of the starting states of the process:

$$J_\pi(\boldsymbol{\theta}) = \sum_s \rho_s(s) V_{\pi_{\boldsymbol{\theta}}}(s)\,. \tag{3.19}$$

We define as $\rho_0(s')$ the probability that an episode begins in state $s' \in \mathcal{S}$, and then we define $\eta_\pi(s')$ as the average number of time steps that the agent spends in state s' before the process dies. Time is spent in a state s' if episodes start in that state s', or if the environment transitions into s' from a previous state s in which time is spent. Thus, we have that the formula for η_π is

$$\eta_\pi(s') = \rho_0(s') + \sum_s \eta_\pi(s) \sum_a \pi(a|s) p(s'|s,a), \text{ for all } s' \in \mathcal{S}. \tag{3.20}$$

This is a *linear system* of equations, one for each state $s' \in \mathcal{S}$, and it can be solved for the expected number of visits $\eta_\pi(s)$.

The *policy gradient theorem* provides an analytic expression for the gradient of the performance J with respect to the policy parameters, which is what we need to approximate for gradient ascent in (3.17) (or descent in (3.18)):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_s \eta_\pi(s) \sum_a Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s). \tag{3.21}$$

See [1] for the demonstration in the episodic case, which involves just elementary calculus and re-arranging of terms in the expression of the value function V_π.

Thus, solving the expression for (3.21) and using it in equations (3.17) – (3.18) allows optimizing the parameters of the policy, building an optimal policy in the chosen parametrization class. Note that to find both η_π and Q_π we need to solve two linear systems, (3.12) and (3.20), and, moreover, they depend on the policy π, thus they must be solved at every iteration.

3.4.1 Natural policy gradient

See Natural actor critic, Peters, Vijayakumar, Schaal, European Conference on Machine Learning (2005).

See (Hennes, Neural Replicator Dynamics: Multiagent Learning via Hedging Policy Gradients) here (page 8, appendix A4) for proof of natural gradient.

Amari, S. I. (1998). Natural gradient works efficiently in learning. Neural Computation, 10(2):251–276.

Chapter 4

The model

4.1 The mathematical model

4.2 Implementation of the model

Chapter 5

Conclusion

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts (MA): MIT Press, 2018. URL: http://incompleteideas.net/book/the-book.html (cit. on pp. 5, 6, 11–13).

[2] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning: State of the Art*. Springer Verlag, 2012. ISBN: ISBN: 978-3-642-27645-3 (cit. on p. 6).

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), pp. 99–134. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(98)00023-X (cit. on pp. 7, 9, 11).

[4] William Uther. "Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 642–646. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_512 (cit. on p. 7).

[5] Matthijs T. J. Spaan. "Partially Observable Markov Decision Processes". In: *Reinforcement Learning: State of the Art*. Ed. by Marco Wiering and Martijn van Otterlo. Springer Verlag, 2012, pp. 387–414 (cit. on pp. 9, 10).

[6] Pascal Poupart. "Partially Observable Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 776–776. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_642 (cit. on pp. 10, 11).

[7] D. Aberdeen and Jonathan Baxter. "Scaling Internal-State Policy-Gradient Methods for POMDPs". In: *International Conference on Machine Learning*. 2002 (cit. on p. 11).

[8] Jan Peters and J. Andrew Bagnell. "Policy Gradient Methods". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 774–776. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_640 (cit. on p. 12).

[9] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 2000 (cit. on p. 12).

Index