The L^{2}F Tennis Challenge

Matteo Caorsi and Umberto Lupo
L^{2}F Research Department[*]

ABB Corporate Research, 9 September 2019

Contents

[*] `{m.caorsi, u.lupo}@l2f.ch`

1 Introduction

The Applied Machine Learning Days (AMLD) are one of the most important events about artificial intelligence and its applications. At AMLD 2019, we had the chance to organise a virtual tennis challenge: a great occasion to get in touch with many talented people, all sharing our passion for machine learning and artificial intelligence. We review here the content of the challenge and stress some of its subtlest points. Section 2 is dedicated to explaining the Q-learning algorithm, the technique that led one of the teams to victory. All the material we provided for the challenge can be found at: `https://github.com/ulupo/L2F_Tennis_Challenge`.

1.1 The dataset

The dataset we provided was a simplified version of Jeff Sackmann's freely available[1] crowdsourced dataset: it is a great source of tennis data!

In our version of the dataset, we provide only a subset of the entire available list of rallies between Roger Federer and Rafael Nadal. Each rally is described as a sequence containing the individual shots and the final rally outcome; we also provide information on who is serving that particular point and who actually won that point. That's it!

1.2 The goal of the challenge

The goal of the challenge was not a standard predictive task – typical of most machine learning competitions: we asked the participants to provide us with a strategy for Roger Federer to beat Rafael Nadal. What exactly do we mean by strategy? We asked for a stochastic matrix that will determine the best shots by Roger Federer given Rafael Nadal's incoming "prompt". For reasons which will become clearer later in this document, we will refer to shots as "actions" and denote a generic one by A, and to prompts as "states" and denote a generic one by S – hence, the stochastic matrix encodes the conditional probabilities $\pi(A|S)$. Of course, this strategy matrix can be fully deterministic if we believe that for each state A we can find a corresponding action $\varpi(S)$ which is the ouright "best" response every time. In this case, we just define the strategy matrix as a family of "delta distributions", by setting $\pi(A|S) = \delta_{A,\varpi(S)}$.

[1] `https://github.com/JeffSackmann/tennis_MatchChartingProject`

1.3 Testing the strategy

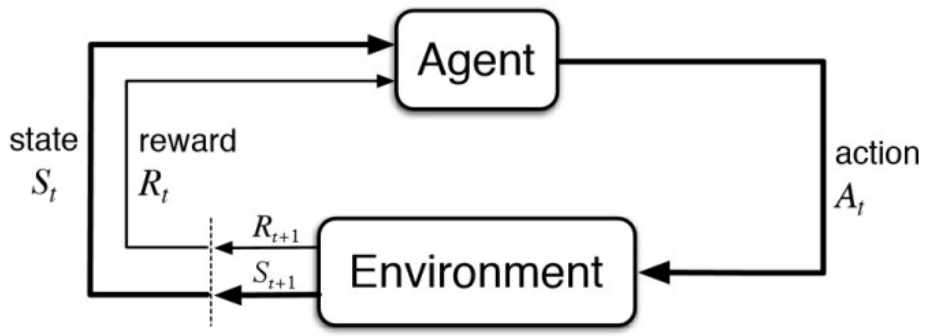
In order to rank solutions by different teams, we had to build up an environment to make the optimised Roger Federer and the historical Rafael Nadal play against each other. Each of the competing teams provided a strategy for Roger Federer which was tested against a "Rafael Nadal simulator" playing according to the same shot statistics as the historical Rafael Nadal. There was one very subtle trap here: the dataset with which *we* computed Rafael Nadal's statistics was not exactly the same as the one we gave to the workshop participants in order to train their models: our set was only 60% similar to the training set used in the workshop. Overfitting was an important issue to be reckoned with!

The final score for each team was simply the ratio of the points won by the optimised Roger Federer over all the points played.

2 The winning solution: *Q-learning*

2.1 Basic RL setup

All groups provided a different solution to the problem: some used very detailed counting methods to analyse the statistics of the response shots, weighted by the distance from the end of the rally. Some others created their own model to estimate and optimise the winning probability of each shot. In the end, the winning group used *reinforcement learning* (RL): this allowed them both to attain the best performances and to generalise well. We will revise here the basics of *Q-learning*, a very powerful "model-free" RL algorithm which suits our specific problem very well. The interested reader can find more details in [1]. The idea is as simple as it is old: an agent $\mathscr{A}$ finds itself, at time t, immersed in an environment $\mathscr{E}$ described by the state S_t. "Prompted" by this state, $\mathscr{A}$ performs an "action" A_t, to which $\mathscr{E}$ "reacts" according to a rule which may depend on some or all of the preceding history. As a result, $\mathscr{E}$'s state is updated to a new value S_{t+1}.



The *reward* R_{t+1} tells us how well the agent did by performing action A_t on state S_t. Hence, broadly speaking, maximising rewards should lead to the agent learning to "do the right things". What exactly should we maximise? Our guiding principle in what follows is that, at each time t, $\mathscr{A}$ should seek to maximise a suitable weighted sum of *all* future rewards! Technically, we define the (total discounted) *return*:

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+1+k},$$

where $\gamma \in [0,1]$ is called the *discount factor.* If $\gamma < 1$, we give higher weight to more immediate rewards.

2.2 From returns to quality

The return G_t is a stochastic variable (since each future reward is) and it is therefore not possible to maximise it as is. Instead, we consider the expectation value of the return, mediated over the strategy $\pi(A|S)$ (often referred to as the *policy* in the literature). So what $\mathscr{A}$ really should do is maximise the so-called *quality function*:

$$Q(S,A) := \mathbb{E}_\pi(G_t \mid S_t = S, A_t = A).$$

There are different approaches to this problem: if we had full knowledge of the model we could go for *dynamic programming.* Otherwise, Monte Carlo methods are a straightforward alternative. But the real innovation of RL is in the *temporal difference* algorithm.

2.3 Q-learning

The simplest and most famous algorithm to maximise the expected return is *Q-learning.* To understand why the algorithm works requires a deep understanding of the Bellman equation (and we are not diving into it here). One can get the gist of it by simply looking at the update equation for Q:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \cdot \max_A Q(S_{t+1}, A) - Q(S_t, A_t) \right], \quad (1)$$

where S_{t+1} is the state returned by $\mathscr{E}$ when $\mathscr{A}$ reacts to S_t with action A_t according to a policy which is "good" given current knowledge of the expected returns when the state is S_t. Thus, in a simple version of this algorithm we would pick, for each t,

$$A_t = \arg\max_A Q(S_t, A). \quad (2)$$

You can think of this as locally following a policy function which precisely adheres to the maximisation principle described in Section 2.2. Notice that the simple rule in Equation (2) is deterministic, but this need not be the case – we return to this briefly below. The first term in the right-hand side of (1) can be thought of as the quality the agent currently attributes to the state-action pair at time t, whereas the term multiplied by the *learning rate* α describes what $\mathscr{A}$ learns from the present and future experiences.

One runs this iterative algorithm "until convergence" (hopefully!) of Q and, at the very end, the Q-learning solution is the deterministic policy function determined by

$$S \mapsto \arg\max_A Q(A, S),$$

again according to the ideas of Section 2.2.

One final remark is due: when we initialize (as we must!) the Q function, A does not know what to do: it is worth spending some time exploring some possibilities (say by playing tennis at random). Some random strategies may lead to unexpected victories and thus they will influence the final policy. A trade-off between the *exploration* phase just described and the *exploitation* phase (i.e. following the $\arg\max_A Q(A, S)$ of the current definition of Q) is usually achieved by randomly selecting one or the other option according to a Bernoulli variable with parameter ε. Typical values for ε may hover around 0.1, but this is largely dependent on the details of the specific problem.

The story of RL goes far beyond Q-learning. . . but that is the material for a new challenge!

3 Comments and Conclusions

The attentive reader may argue that Q-learning might not have been the best choice: in this simplified tennis situation, the full model of Rafael Nadal – i.e. the “environment” $\mathscr{E}$ – was known! So, why not using the old and mathematically proven approach of *dynamic programming*? We tried this approach and the scoring over the training set was amazingly high: above 88%. Unfortunately, this model is calibrated on the training set only and it has no generalisation power: over the test set, the scoring of this model dropped down to 60%.

The Q-learning algorithm, on the other hand, does not require precise knowledge of the model governing the environment: the performances on the training set were less spectacular than with dynamic programming, but its generalisation power was much higher. The winning group at AMLD 2019, who also took time to perform a standard hyper-parameter tuning via *cross-validation*, reached a score of 74.02%!

References

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