

Spring 2020

Prof. Dr. Renato Renner

Master's Thesis

Designing experiments with neural networks

Eduardo Gonzalez Sanchez

Advisor: Raban Iten

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Zurich, April 14, 2020

Eduardo Gonzalez Sanchez

Abstract

(Mock version, not definitive) In this work we propose a new model that mixes reinforcement learning and deep learning to create agents able to design strategies for experiments, using as feedback only the quality of the predictions about properties of the physical system made exclusively from the data the agents collect. This is could be summarized as 'agents able to do science'; since science checks its validity by making predictions over the physical world. In the thesis we also put the first building blocks of a theoretical model for the scientific method in machines. This last part is important in the long-term goal of developing a variant of quantum theory that can consistently describe agents who are using the theory. If the science of the future is done by machines, we must ensure consistency in the underlying principles driving automated science.

Keywords: Reinforcement learning, deep learning, automated science, feature representation, experiment design, artificial intelligence

Contents

1	Preface	1
2	Minimal model for science	5
3	Machine Learning theory	6
4	Experimenter-analyzer model	7
4.1	General elements and definitions	10
5	E-A architecture in different scenarios	16
5.1	Scenario 1: Modified multi-armed bandit	16
5.1.1	General multi-armed bandit problem	16
5.1.2	Simple Reinforcement Learning algorithm for non-stationary multi-armed bandit	18
5.1.3	Modified multi-armed bandit problem	20
5.1.4	Summary of the multi-armed bandit	30
6	E-A architecture for sequential experiments	32
6.1	Sequential experiments	32
6.2	E-A for sequential experiments	34
6.3	Example	35
6.3.1	The experiment	35
6.3.2	Mathematical characterization of the experiment	37
6.3.3	Technical details of the agents	38
6.3.4	Results and discussion	39
A	Appendix ?	41

Chapter 1

Preface

Many of the limitations of humans at doing science come from their biological condition:

- Humans have a limited lifespan: 72.6 years on average [9]. When experienced scientists die their knowledge and expertise die with them.
- From the limited lifespan, humans can dedicate only a fraction to do science. This is inevitable since humans need to eat, sleep and deal with social interactions, among other things. Moreover, humans need decades of study and training to start making contributions to the scientific knowledge.
- Humans are susceptible to suffer from diseases and other limiting biological conditions that hinder their scientific production.
- Humans' understanding of the physical world is tightly linked to their limited sensorial perception and other inherited or acquired factors like the language or the cognitive capacity.

Other limitations are indirectly caused by the need to satisfy their biological necessities. For example, a person who wants to dedicate their life to science needs some type of financial support to satisfy the basic human necessities. This support usually comes from a greater institution like a state, a company or a patron. This financial relationship ties inextricably science to the economical structure of the society. Profitable discoveries are encouraged while resources for unprofitable science are scarce. It can be argued that any form of scientific research, human or not, will require an investment of energy and resources in a society in which those are limited. This can be true, but a more efficient way of doing science will increase science independence from the economy.

At the same time, scientific discoveries influence drastically modern society and its economical structure. They provide new knowledge that allows humanity to develop new tools and protocols to improve human well being. In the last centuries, science has changed society by setting the theoretical and experimental grounds of a technological transformation. It is of public interest to boost and improve scientific production.

During the last century, the amount of available scientific literature has been growing exponentially [13, 3], with a yearly growth rate of ~9% in the last decade. Scholars read on average almost 240 articles per year [15]. Some authors [1] suggest that science is in the midst of a data crisis. Although the available literature grows exponentially the cognitive capacity of human beings remains constant. This forces scientists to derive hypotheses from an exponentially smaller fraction of the collective knowledge. This will lead to scientists increasingly asking questions that already been answered and reducing further the efficiency of scientific production.

Some areas of science are starting to suffer from a reproducibility crisis [12, 2] in which scientists are generally unable to reproduce their peers' findings. Some voices in the physics community [7] point out that foundational physics has been stagnated during the last decades. However, some authors defend that there is not such a crisis [6]. Nonetheless, it is clear that to sustain an exponential growth of reliable scientific production with no exponentially increasing human effort is impossible, and the crisis is thus, unavoidable.

However, the lack of efficiency in scientific production is not the only drawback produced by the biological limitations of human beings. Humans' intuition and understanding of the physical world is conditioned by the percepts collected by their sensory system. This limitation becomes evident when trying to intuitively understand physical systems that show behaviors that differ from those susceptible to be collected by the sensory system. This is the case of, for example, quantum theory. Humankind has developed tools to overcome the limitations of the sensorial system to observe new properties of physical systems that are out of reach for our biological receptors. For instance, using infrared cameras to map infrared signals to a representation in the visible spectrum, humans can detect infrared radiation. But these tools don't allow to build an intuitive understanding of the phenomena without analogies to the phenomena perceived by the sensory system. For example, people that are blind from birth have never had any input to their visual cortex, so they have no visual intuition which limits their ability to understand some physical concepts. Similarly, the lack of receptors for other arbitrary physical properties

limits human understanding of the physical world and likely hinders scientific advance.

Modern science requires from agents with complex cognitive abilities. So far humans are the only known material structure able to perform it. It is true that some animals perform scientific behavior, like Crows or monkeys solving puzzles by trial and error. But those anecdotal examples are far from the formalized version of the scientific method employed by humans. However, humans are also the living f of the possibility of agents performing sophisticated science. There is no reason to think that there's anything special in humans that makes them the best possible form of a scientific agent. Rather it is reasonable to think that there is plenty of space for improvement, since the human brain was designed solely by millions of years of random mutations and natural selection.

Recent advances in artificial intelligence, yet far from achieving an artificial general intelligence, open the door to an automation of science. In the recent years, a vast amount of effort has been dedicated to the development of machine learning techniques to help scientists of the physical sciences to process data to create new better models [4]. However, these machine learning based techniques are just tools to help human scientists to interpret complex data to provide new predictions, and not efforts towards an automation of science. Nonetheless, the potential role that artificial intelligence might play in the process of scientific production has been getting growing awareness. In [10], the authors use a projective simulation model to design complex photonic experiments that produce high-dimensional entangled multiphoton states. According the authors, the system autonomously discovers experimental techniques which are a standard in modern quantum optical experiments. In [8] the authors explore the use of variational autoencoders to extract autonomously physically relevant parameters from physical data without prior assumptions about the physical system. In [11] they expand the work to present an architecture based on communicating agents that deal with different aspects of a physical system and show that it can be combined with reinforcement learning techniques. More work on similar directions can be found in [citas articulo de Raban II].

However, scientific agents need to be designed carefully to minimize the inherited biases and limitations from their human creators. In this thesis we present a minimal axiomatic model for science and a new model architecture that mixes reinforcement learning with deep learning to create agents capable to design strategies

for experiments. Using as feedback only the quality of the predictions about properties of the physical system made exclusively from the data the agents collect from their sensorial available receptors.

Here I will write a couple of paragraphs explaining the structure of the thesis and pointing out to the repository, where to find the code, etc.

Chapter 2

Minimal model for science

Chapter 3

Machine Learning theory

TO BE WRITTEN.

Chapter 4

Experimenter-analyzer model

In this section, we are going to present our proposal for a basic model for a machine learning set-up that designs an experiment applying the scientific method. It is a model that mixes reinforcement learning and deep learning to create agents able to design strategies for experiments, using as feedback only the quality of the predictions about properties of the physical system made exclusively from the data the agents collect.

One of the cornerstones of our model is the principle of modularity: in order to be applicable to a wide range of set-ups it needs to be constituted by simple individual elements that can be combined to build complex learning loops. The agent that will learn to design experiments and make predictions with them is composed by two different kinds of sub-agents that complement each other:

- **Experimenters:** this kind of sub-agents are reinforcement learning based agents. Its goal is to take decisions about influencing the dynamics of the physical system (e.g by modifying a parameter of a controlled experiment) or about what measurements to make (eg. by choosing what property of the physical system to measure). It could be that the choice of measurement modifies as well the dynamics of the physical system (eg. by choosing which slit to "look at" in the double slit experiment). To choose what actions to take they can consider previous observations and other kind of information.
- **Analyzers:** this kind of sub-agents are learning agents that take the information gathered by the experimenters to make predictions about the environment. Then, based on the accuracy of those predictions compared to the real values ¹ of the environment a reward is generated to train all the sub-agents.

¹One may ask how these real values are obtained. Those values could be obtained by measure-

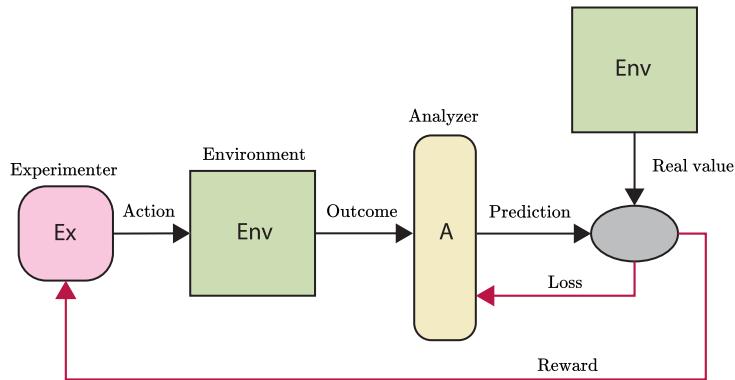


Figure 4.1: Diagram of the simplest combination of elements. The feedback loop consists on the following: 1. The experimenter takes an action. 2. The environment gives back an outcome resulting from the action. 3. The action is taken by the analyzer and used to make a prediction. 4. The prediction about the environment and the real value of the environment are compared to generate the loss and the reward to train the analyzer and the experimenter.

To get a better understanding of the purpose of each part of the model let us take a look to the simplest structure that can be formed with the model (Fig. 4.1). To illustrate it let us imagine a simple physical set-up. Suppose we have a simple gravity pendulum of length L and mass M and the goal is to find the oscillation period on Earth's surface. We know the period is a function only of the length L and the value of the Earth's gravity (which we assume to be constant): $T \approx 2\pi\sqrt{\frac{L}{g}}$, so the value of the mass M is irrelevant for the purpose, but in principle this is unknown to the agent. In this set-up we give the experimenter two possible actions to take (but only one try): either measuring the mass M (eg. by using a calibrated spring) or measuring the length L of the string (e.g by using a calibrated ruler). When the experimenter takes an action over the Environment, it produces an outcome (either the mass M or the length L). This outcome is passed to the analyzer which tries to make a guess of the period based on the value of the outcome of experimenter's action. Then the real period of the pendulum is observed and compared to the value predicted by the analyzer. Depending on the result of

ments made by other experimenters that form part of the learning agent. However, for simplicity in the work presented here we assume that the real values needed to check the validity of the prediction are always accessible with unlimited accuracy to the agent. In the context of this modular framework we could assume that we have an experimenter that its only possible action is to measure the real value of the target property.

the comparison a loss and a reward are generated to train the experimenter and the analyzer. In general: the closer the value of the prediction to the actual value the smaller the loss and the higher the reward. After the iteration is completed, a new pendulum with new values of M and L (ideally generated uniformly at random under the i.i.d. assumption) is generated and the process starts again.

The goal in the set-up depicted in Fig. 4.1 is to form a feedback loop between the experimenter and the analyzer from the, at first unknown, correlations between the outcomes and the values to be predicted. The better the experimenter gets, the better the data available for the analyzer to make better predictions that will generate better rewards for the experimenter. Hopefully, this feedback loop will converge to an optimal experiment strategy and an optimal model for the predictions. At the beginning both sub-agents will start by giving random outputs since they are not trained. This is what we call the *exploration phase* and it is of fundamental importance for the feedback loop to start. It can be artificially enforced, for example with ϵ -greedy algorithms that we will discuss later in this work. If there exists any correlation between the choices of the experimenter and the target value to be predicted, the analyzer's training algorithm (usually Stochastic Gradient Descent) can exploit those correlations to start the descent to some local or global minimum. This working principle is somehow similar to the use of artificial neural networks for feature selection in classical machine learning theory.

The alert reader may have noted that in 4.1 the analyzer has no apparent way of telling which action the experimenter took, since it only receives the outcome of the experiment. Therefore the analyzer does not know if what is receiving is the value of M or L . Although it may appear counter intuitive, the analyzer does not care in this particular set-up. The optimal strategy is to treat everything as if it were L since knowing the value of M does not provide any information about T . However, it may make sense for the analyzer to know when M is provided if the objective is to minimize the loss function (e.g. if the analyzer knew that the value corresponds to M the optimal prediction would be the random guess that minimizes the loss). In practice, unless the distributions where M and L are generated from are the same, the analyzer learns to differentiate them with high confidence just by value of the outcome. However, in other set-ups this may not be the case and the information about what actions were taken is useful for the analyzer to make the predictions. In this case, we should also feed the actions as inputs to the analyzer. As we will see, these Experiment-analyzer loops can grow very quickly in complexity when several sub-agents are involved. This is why we introduce the concept of *buffer*.

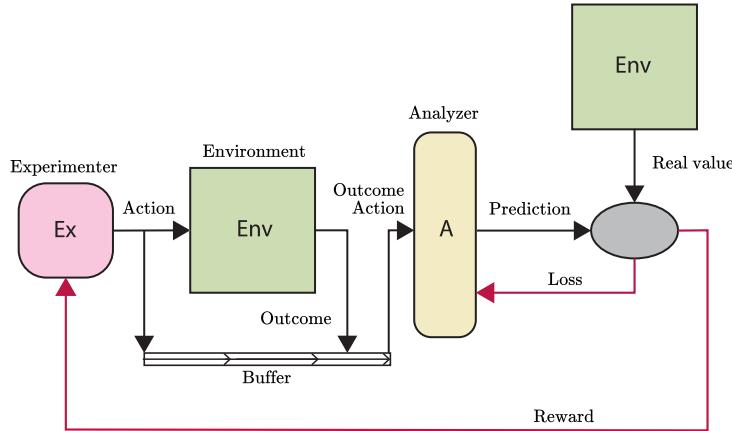


Figure 4.2: Diagram of a simple learning loop with buffer. In this case, the process is identical to the process depicted in Fig. 4.1 with the difference that the values of the action and the outcome are stored in the buffer, and then passed to the analyzer. In this case the information about which action was taken is available to the analyzer.

In our model we can use a buffer to store everything that we might want to feed to the agents. Usually this will be the actions taken and the outcomes obtained in every observation of the environment, although we have the freedom to add whatever we might need. It is just a variable to store information. This variable is not strictly needed, since we could establish directly the connections between the different subagents and environment outcomes. However, when we have a few sub-agents with several outcomes it becomes very difficult to keep track of each connection and to represent them. In these cases, the concept of buffer becomes highly useful from an illustrative point of view. Usually, the buffer is emptied at the end of each episode or iteration of the learning loop. We could not do so, but in practice, keeping the data in the buffer does not provide any easy advantage since to use it we should solve many challenges, for example an increase on the number of features fed to the agents each episode. We won't refer the buffer as *memory* in this work, since the word *memory* is reserved for an element of some versions of sub-agents that does not restart in each iteration.

4.1 General elements and definitions

In this section let us define more formally some of the elements and concepts of our model.

- **Simulated physical environment (SPE):** it represents the physical world that determines the outcomes of all the measurements realized by the agent. It is completely represented by a state $s \in S$, where S is the set of all possible states in which the SPE can be. For this particular model we assume that the evolution of the system is assumed to be given by a Markovian function $f : S \rightarrow S$ such that $f(s_\tau) = s_{\tau+1}$. This function maybe deterministic or not.
- **Agent:** the agent is formed by the composition of two kind of different sub-agents:
 - **Experimenter:** this part has the task of making decisions that will have an effect on the observations. This effect could be due to the influence of the decisions in the evolution of the system (for example, if the agent decides to shoot a bullet to a box with a certain velocity) or due to the fact that the decisions themselves could be what observations to make (for example, choosing to measure the position of a box within a spatial range). This agent will be implemented using reinforcement learning techniques. Mathematically it can be represented by a trainable function $E : \mathcal{X} \rightarrow \mathcal{A}$ where \mathcal{A} represents the space of possible actions and \mathcal{X} is an arbitrary space that can represent any accessible information that might be useful to make a choice of action. In the context of reinforcement learning, \mathcal{X} is the set of states of the sub-agent's environment².
 - **Analyzer:** the goal of this part of the agent is to process the data collected in the observations after the action of the experimenter to predict a physical property of the SPE. This property could be a physical parameter of the system or a prediction of the dynamics of the environment. It will usually consist of a regular regression neural network or an autoencoder, depending on the specific set-up. But we could use any trainable function like SVMs or any kind of regression. Mathematically it can be represented by a trainable function $A : \mathcal{M} \rightarrow \mathcal{P}$, where \mathcal{M} is the space of measurements and \mathcal{P} is the space of predictions.
- **Orchestration:** when coding we will call Orchestration everything that connects the different parts to complete a feedback loop. For example, one of

²Not to be confused with the SPE.

the tasks of the Orchestration is to transform the state of the SPE into the measurements received by the analyzer. The Orchestration is needed when we simulate fictitious physical situations in a computer to test our models. It represents somehow something analogous to the sensors and wires that connect the external environment to the processor of an autonomous robot.

Example 4.1.1. Let us complete the example of the pendulum to fully illustrate the architecture. First, we define the SPE. Let us assume that the mass of the pendulum and the length of the string are sampled uniformly at random from the intervals $0.1 \text{ kg} < M < 1 \text{ kg}$ and $0.1 \text{ m} < L < 1 \text{ m}$. The space of states of the SPE is then:

$$S \equiv \{(m, l) : m \in [0.1, 1], l \in [0.1, 1]\} \quad (4.1)$$

Now, let us define an experimenter. In this case, the experimenter's environment space \mathcal{X} is the empty set \emptyset , since there is no input. This means that the agent takes the action based solely on the rewards obtained. The action space \mathcal{A} is just $\{0, 1\}$, with 0 representing the action of measuring the mass and 1 the action of measuring the length.

We can therefore set a very basic decision rule for our experimenter agent:

- Let $\mathbf{Q} = (q_0, q_1)$ be the value table, where q_0 and q_1 are the values associated with the actions 0 and 1 respectively. Q can be also understood as a trainable function $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$. Initially $Q(a) = 0 \forall a \in \mathcal{A}$.
- Every episode, the agent takes an action a and receives a reward r after taking the action. The update rule for \mathbf{Q} is the following:

$$q_a \leftarrow q_a + r \quad (4.2)$$

This is, the value q_a is just the cumulative reward obtained by the action a .

- The agent's policy $E : \emptyset \rightarrow \mathcal{A}$ or decision rule is:

$$a = \arg \max_{x \in \mathcal{A}} Q(x) \quad (4.3)$$

This is, the experimenter takes the action a with the highest q_a , and E is trained by updating \mathbf{Q} each episode.

Note that the choice of (4.2) is just to illustrate that we can build Reinforcement Learning agents with very simple rules. We could have chosen instead of the

cumulative reward the average reward obtained with the action, but the working principle would be identical.

However, our policy is still flawed, since the first action that gets a reward would always be elected afterward independently of its optimality. To solve this we can force the agent to initially explore different options ignoring the decision rule, and slowly, when the values q_a are more reliable, let the agent choose according to (4.3). The easiest way to achieve that is with an ϵ -greedy decision algorithm: we set an exploration rate ϵ that decreases in each episode until it reaches a minimum value $\epsilon_{\min} \geq 0$. The experimenter is set to take the (4.3) policy with probability of $P_{\text{greedy}} = 1 - \epsilon$ and a random action with a probability of $P_{\text{random}} = \epsilon$. We have the freedom to choose how to decrease the value of ϵ , and it may have a crucial impact on the training. A very popular rule is to decrease the value constantly by subtracting $\epsilon = 1/N$, where N is the number iterations of the training. So in each episode:

$$\text{if } \epsilon > \epsilon_{\min} \quad \epsilon \leftarrow \epsilon - \epsilon \quad (4.4)$$

$$\text{else} \quad \epsilon \leftarrow \epsilon_{\min} \quad (4.5)$$

For the analyzer, we set a trainable real function $A : S \rightarrow \mathbb{R}$. We can use any trainable function, but to keep it simple let's use a regular feed-forward neural network with one hidden layer of two neurons, trained with the Stochastic Gradient Descent (SGD) algorithm and a the Mean Squared Error (MSE) loss.

We need now to define how to calculate the reward. In our case we want the reward function to be high when the analyzer predicts a period T_{pred} close to the real period of the pendulum and low when the period prediction is far. So, ideally we would like a function that:

$$r(T_{\text{pred}} \approx T) \approx 1 \quad (4.6)$$

$$r(T_{\text{pred}} >> T \text{ or } T_{\text{pred}} << T) \approx 0 \quad (4.7)$$

One natural way to achieve this is using a Gaussian distribution over the relative error $\Delta = (T - T_{\text{pred}})/T$. This is:

$$r(\Delta) = \exp \left(-\frac{1}{2} \left(\frac{\Delta}{\sigma} \right)^2 \right) \quad (4.8)$$

and σ allows us to modify the precision we want in order to achieve a reward.

Now we have every element defined. Let us outline the full algorithm in pseudocode:

Algorithm 1 Simple learning loop

```

1: procedure PENDULUM
2: count = 0
3: while count < N :
4:    $M, L \leftarrow$  random sample  $M, L$ 
5:    $T \leftarrow T(L)$ 
6:   pendulum  $\leftarrow [M, L, T]$ 
7:   action  $\leftarrow \arg \max Q$ 
8:   measurement  $\leftarrow$  pendulum[action]
9:   prediction  $\leftarrow$  analyzer(measurement)
10:  Apply SGD to analyzer with data point (measurement,  $T$ )
11:  reward  $\leftarrow$  reward(prediction,  $T$ )
12:   $Q[\text{action}] \leftarrow Q[\text{action}] + \text{reward}$ 
13:  count  $\leftarrow$  count + 1

```

Let us see how this simple algorithm performs in the task. Ideally, the experimenter would find the optimal policy which consists in choosing always to measure the length and the analyzer would learn to estimate accurately T from the data obtained by the experimenter. In our experiments, with this simple structure of only two neurons and the very naive reinforcement learning policy the agent consistently finds the optimal action (to measure L). It also accurately predicts the value of T in less than 10,000 episodes. With less episodes it is less consistent in finding the optimal policy.

In the figures Fig. 4.3 and Fig. 4.4 we can observe the evolution of different parameters during the training. All the parameters show successful learning of both, the optimal policy and the prediction of the period. More complex neural network structures and longer trainings can achieve much lower errors. However, this is not a surprise since once the optimal policy is obtained the process just consists on fitting a continuous single variable function. A task that can be solved optimally by a single layered sigmoidal neural network [5]. This pendulum example, although trivial, is useful to appreciate the working principle behind the experimenter-analyzer architecture and how the feedback loop is formed. In the next section we are going to explore less trivial examples with more complex learning loops.

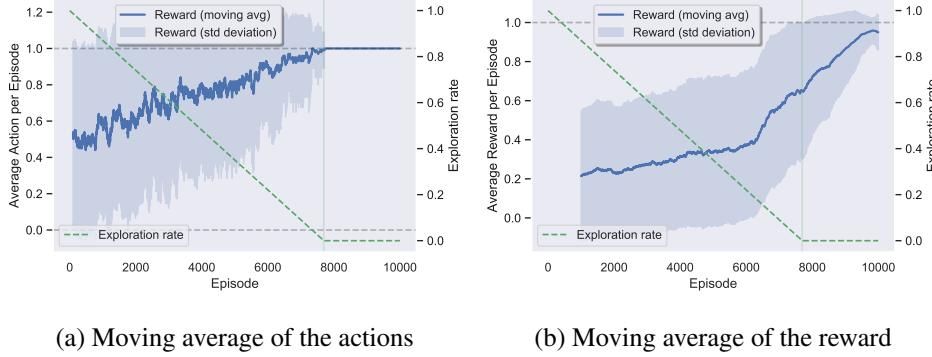


Figure 4.3: (a) The moving average of the actions taken in each episode. We can see how it starts taking both actions evenly with an average of 0.5. This is expected since at the beginning $\epsilon \approx 1$. As ϵ decreases the experimenter starts to take more greedy actions and gets biased towards the action 1 (measure L). This bias appears as a result of the existing correlation between L and T , but not between M and T . This correlation is exploited by the SGD algorithm for the neural network of the analyzer. (b) The moving average of the reward obtained in each episode. At the beginning the reward obtained is low and similar to those obtained with random guesses. However, it increases slowly, apparently because the analyzer starts to exploit slightly the correlation between L and T . This slight improvement over the prediction for the outcomes of Action 1, creates the bias that starts the feedback loop. Values: $\sigma = 0.05$, $N = 10000$, $lr_{\text{analyzer}} = 0.01$, Optimizer = Adam

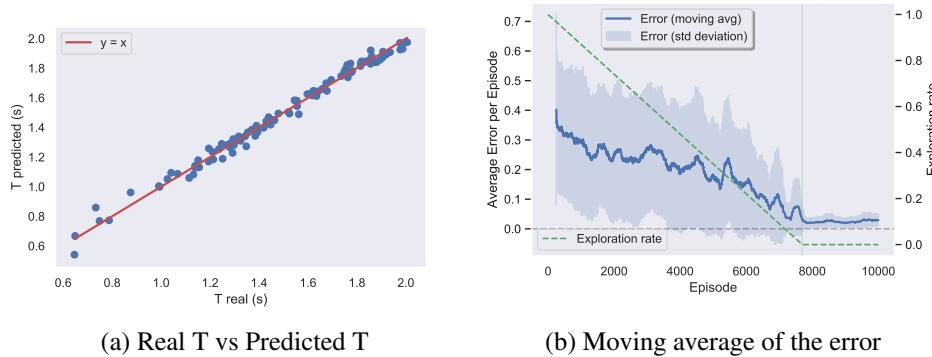


Figure 4.4: (a) A sample of the predictions made by trained agent. We can see how it fits the $y = x$ line, showing that it learned the relation between L and T . (b) In this figure we can see how the error decreases to almost zero with very low deviation. Values: $\sigma = 0.05$, $N = 10000$, $lr_{\text{analyzer}} = 0.01$, Optimizer = Adam

Chapter 5

E-A architecture in different scenarios

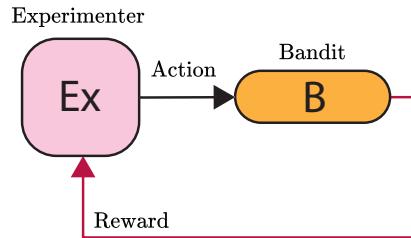
5.1 Scenario 1: Modified multi-armed bandit

In the first section of this chapter we are going to study a modified version of the classical Multi-armed bandit problem, a classical problem of Reinforcement Learning and Probability Theory. Let us start by introducing the general problem.

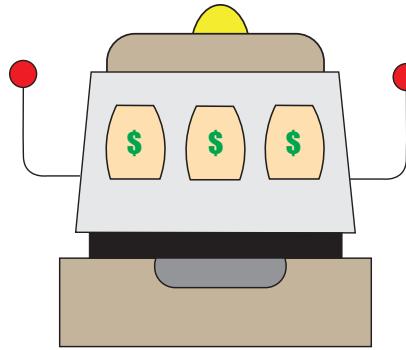
5.1.1 General multi-armed bandit problem

Suppose that we have an agent that has access to a k -armed bandit. Each time the agent pulls one of the k levers, the bandit produces a reward. The objective of the agent is to maximize the obtained reward in a determined number of trials or episodes N . The agent does not know how the bandit produces the rewards. Its only way of getting information about the reward system is by trial and error by pulling the levers and obtaining rewards.

Mathematically we can model our agent as an experimenter $E : \mathcal{X} \rightarrow \mathcal{A}$, where the action space is $\mathcal{A} = \{0, 1, \dots, k\}$ and the state space \mathcal{X} is again the empty set \emptyset . The bandit can be modelled as a probabilistic function $R : A \times \dots \rightarrow \mathbb{R}$, with “...” meaning that in principle the generated reward can depend on anything: the previous actions, the previous rewards, the number of trials, the number of times a lever has been pulled, etc. This is an extremely general problem. Often authors make assumptions on R . For example, R being Markovian if depends only on the last rewards and/or actions. Or stationary if R can be modeled as a set of immutable distributions $R = \{R_1, \dots, R_k\}$, with each distribution being associated



(a) Simple diagram portraying the multi-armed bandit problem



(b) Draw of a two armed bandit

Figure 5.1: Multi-armed bandit problem

with the rewards delivered by each of the arms of the bandit.

We define the value $Q^*(a) = q_a^*$ of an action $a \in \mathcal{A}$ as the true expected or mean reward associated to the action. We denote by a_n the action taken in the n^{th} episode, and r_n the reward obtained. This is:

$$Q^*(a) = \mathbb{E}[r_n | a_n = a] \quad (5.1)$$

Note that, although we don't write it explicitly, $Q^*(a)$ can also depend on any of the subsets of the domain of R . The optimal strategy for the k-armed bandit problem is to always choose $a^* = \arg \max Q^*(x)$.

Most reinforcement learning approaches to this problem look for an estimate of $Q^*(a)$, $Q(a)$, and base their choice on $a = \arg \max Q(x)$. In general, the best algorithm depends deeply on the characteristics of R , e.g., an algorithm that works well for stationary bandits can perform very poorly on non-stationary bandits. The

interested reader can read a gentle introduction to the multi armed bandit problem in the Chapter 2 of [14]. We are not going to discuss here many of the approaches to solve the multi-armed bandit problem. However, let us outline a very simple algorithm that works particularly well for non-stationary bandits.

5.1.2 Simple Reinforcement Learning algorithm for non-stationary multi-armed bandit

Let us outline a simple algorithm that we will use later in this section. This algorithm can be understood as a particular case of the Q-Learning algorithm, where the state space of the experimenter \mathcal{X} is the empty set \emptyset and the discount factor γ is set to zero. The update rule for the estimate $Q(a)$ each time the action a is taken is:

$$Q(a) \leftarrow Q(a) + \alpha (r - Q(a)) \quad (5.2)$$

where $\alpha \in (0, 1]$ is the called the learning rate, $Q(a)$ is the estimate value for the action a and r is the reward obtained after taking the action a .

If we name by Q_n the estimate of $Q^*(a)$ for an arbitrary action a at the $(n)^{\text{th}}$ time the action a was taken, then due to (5.2) we have that:

$$Q_{n+1} = Q_n + \alpha (r_n - Q_n) \quad (5.3)$$

$$= \alpha r_n + (1 - \alpha)Q_n \quad (5.4)$$

$$= \alpha r_n + (1 - \alpha) [\alpha r_{n-1} + (1 - \alpha)Q_{n-1}] \quad (5.5)$$

$$= \alpha r_n + (1 - \alpha)\alpha r_{n-1} + (1 - \alpha)^2 Q_{n-1} \quad (5.6)$$

$$= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} r_i \quad (5.7)$$

The last equation, (5.7), is just a weighted average of the rewards obtained, where the weights give exponentially less importance to rewards coming from distant actions in the past. Sometimes it is known as the *exponential recency-weighted average*.

Claim 5.1.1. *If α is constant on n , the exponential recency-weighted average is also a convex sum of the rewards obtained.*

Proof: Since $\alpha \in (0, 1]$ all the coefficients are no greater than 1. We just need to prove that $\forall n$ they sum to one. If we write the sum S_n of the coefficients at the time n :

$$S_n = (1 - \alpha)^n + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} \quad (5.8)$$

We also have that:

$$(1 - \alpha)^n = (1 - \alpha)(1 - \alpha)^{n-1} \quad (5.9)$$

$$= (1 - \alpha)^{n-1} - \alpha(1 - \alpha)^{n-1} \quad (5.10)$$

Inserting (5.10) in (5.8) we get:

$$S_n = (1 - \alpha)^{n-1} - \alpha(1 - \alpha)^{n-1} + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} \quad (5.11)$$

$$= (1 - \alpha)^{n-1} + \alpha \sum_{i=1}^{n-1} \alpha(1 - \alpha)^{n-1-i} \quad (5.12)$$

If we call $l = n - 1$ we obtain exactly (5.8) but with l instead of n . We can repeat this process until the new index happens to be 1. Then we have:

$$S_n = (1 - \alpha) + \alpha = 1 \quad \square \quad (5.13)$$

Lets outline with pseudo-code how this agent would work:

Algorithm 2

```

1: procedure NON-STATIONARY MULTIARMED-BANDIT AGENT
2: count = 0
3: while count < N :
4:   action  $\leftarrow \arg \max Q$ 
5:   reward  $\leftarrow \text{bandit}(\text{action})$ 
6:   Q[action]  $\leftarrow Q[\text{action}] + \alpha (\text{reward} - Q[\text{action}])$ 
7:   count  $\leftarrow \text{count} + 1$ 

```

If we pay attention to the pseudo-code, we see that to take the first action we need an initial estimate Q_1 for each action. These initial estimates are not irrelevant. They play an important role on the convergence of the algorithm. For example, if we set $Q_1 = 0 \quad \forall a \in \mathcal{A}$, the $\arg \max Q_1$ will output a random action. Therefore, the training can be biased towards that random action depending on the initial reward. Sometimes a good strategy is to set high values of Q_1 for all actions. This will make the agent overoptimistic in the estimate, forcing it to explore all options until getting a more realistic estimate of the reward.

It can be shown that choosing the following learning rate give us an exponential recency-weighted average without initial bias:

$$\alpha_n = \beta/o_n \quad (5.14)$$

where β is a constant parameter and o_n is a constant step size updating parameter so that:

$$o \leftarrow o + \beta(1 - o) \quad (5.15)$$

with an initial value of $o_1 = 0$ for all actions.

5.1.3 Modified multi-armed bandit problem

If we come back to the pendulum example of Chapter 4, we can map the physical set up to a 2-armed bandit problem. We just need to assign the action of measuring L to one of the arms and measuring M to the other. Then we set the function R of the bandit to be $R : \mathcal{A} \times S \rightarrow \mathbb{R}$ as a composition of Gaussian reward function r with the analyzer predicting the period from the outcome. Basically, we assume that the bandit is the result of the interaction between the Environment, the analyzer and the Gaussian comparator (Fig. 5.2).

Note that this multi-armed bandit problem is quite difficult: it is non-stationary since the analyzer and the Environment change each episode and it is non-Markovian since the performance of the analyzer depends on the training over all the previous episodes. However, as we have seen, even very simple agents perform surprisingly well for this simple case.

If we can map the simple pendulum example to a multi-armed bandit, it is evident that many other physical set-ups can be mapped too. So instead of transforming physical examples to a multi-armed bandit problem, we can design a customizable multi-armed bandit to represent an infinite variety of physical situations. However, this bandit is only useful to represent those situations in which a discrete set of actions is chosen at the beginning of the experiment without further intervention until the end of the experiment.

In this kind of set up we only have a single experimenter at the beginning of the learning loop. This means that the state space \mathcal{X} of the experimenter's environment is again the empty set \emptyset . The action space \mathcal{A} is finite and discrete with $k \in \mathbb{N}$ possible actions. The SPE, in the state $s \in S$, after the action a of the experimenter, generates an outcome $\mathbf{O}(a, s) \in \mathbb{R}^m$, and the value of the target physical property $\mathbf{f}(s') \in \mathbb{R}^l$, where $s' \in S$ is the state of the SPE after the action a . Then, the analyzer A is asked to guess $\mathbf{f}(s')$ using the value of $\mathbf{O}(a, s)$ and potentially the

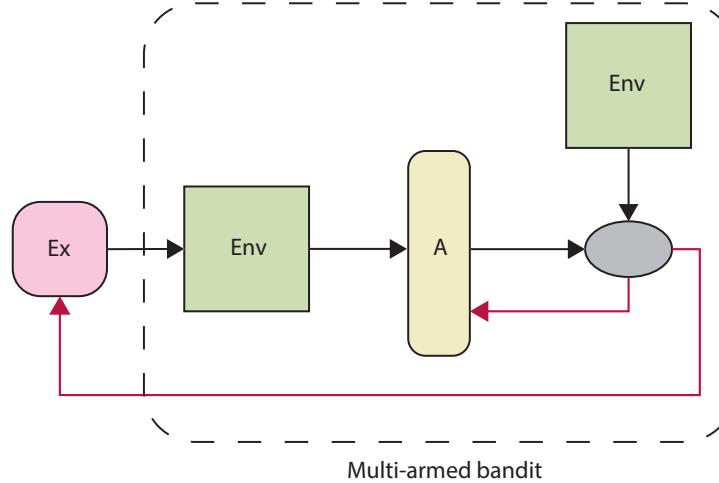


Figure 5.2: Diagram of how the simplest combination for the E-A architecture (4.1) can be modelled as a multi-armed bandit.

action a and some additional auxiliary information $I \in \mathbb{R}^p$. The prediction $\mathbf{P} \in \mathbb{R}^l$ of the analyzer is compared to the actual value of the physical property $\mathbf{f}(s')$ by means of a reward generating function $r : \mathbb{R}^l \times \mathbb{R}^l \rightarrow \mathbb{R}$. The output of $r(\mathbf{f}(s'), \mathbf{P})$ is the output reward of the bandit.

Example 5.1.2. We can illustrate again what each element of the bandit is with the familiar pendulum example. In the pendulum example:

- $k = 2$, since there are only two possible actions.
- $s' = s$ since the actions do not change the state of the pendulum represented by the tuple (M, L) .
- $f(s') = T(L) = 2\pi\sqrt{L/g}$ and therefore $l = 1$.
- $O(a = 0, s = (L, M)) = M$ and $O(a = 1, s = (L, M)) = L$. Therefore $m = 1$, since the outcome is just a real number.
- r is defined by (4.8)
- There's no auxiliary information I .

Now let us explore different versions of this modified multi-armed bandit.

Modified multi-armed bandit I

In this example, the multi-armed bandit is going to be very simple. There are k arms. One of them, the arm t (target) is special. This arm is generally selected at random in the first episode. At the beginning of each episode, the state of the SPE is generated by associating to each arm a real number v_k . Each value v_k is sampled from the same uniform distribution $\mathcal{U}(v_{min}, v_{max})$. Formally, the SPE state space is:

$$S \equiv \{s = (v_0, v_1, \dots, v_k) \in \mathbb{R}^k : \quad (5.16)$$

$$\forall i \quad v_i \sim \mathcal{U}(v_{min}, v_{max})\}$$
 (5.17)

Each time a lever a is pulled, the outcome $O(a, s)$ is just the the value v_a associated to the arm a . We choose the function $f(s)$ to be any real function of only the value of the target arm. For instance, $f(v_t) = v_t^2$. Then the analyzer is fed the outcome $O(a, s) = v_a$ to try to predict $f(v_t)$. It produces a prediction $A(v_a) = p$ that is used to generate a reward $r(p, f(v_t))$ according to (4.8). Then the analyzer and the experimenters are trained with their respective algorithms.

It is clear that the analyzer would be able to make a prediction of $f(s)$ better than random guessing only if the experimenter chooses the target arm t . Note also that the pendulum example is just a particular case of this example. We just need to set t as the arm that outputs L , and $f(v_t) = T(L)$.

Let's run a few different cases to see how does it perform with different configurations.

Example 5.1.3. In this example let us set:

- The number of arms to $k = 8$.
- The function $f(v_t)$ to be the identity $f(v_t) = v_t$
- The experimenter is based on the algorithm described in 5.1.2, equipped with an ϵ -greedy exploration phase with exponential decrease of the exploration rate.
- The analyzer is a two hidden layer feed-forward neural network, with each layer consisting in 16 fully connected neurons activated with the ReLU function (Fig. 5.3). The optimizer is just t qhe gradient descent algorithm. This analyzer is of unnecessary complexity for this task, but it will be useful later for more complex functions $f(v_t)$.

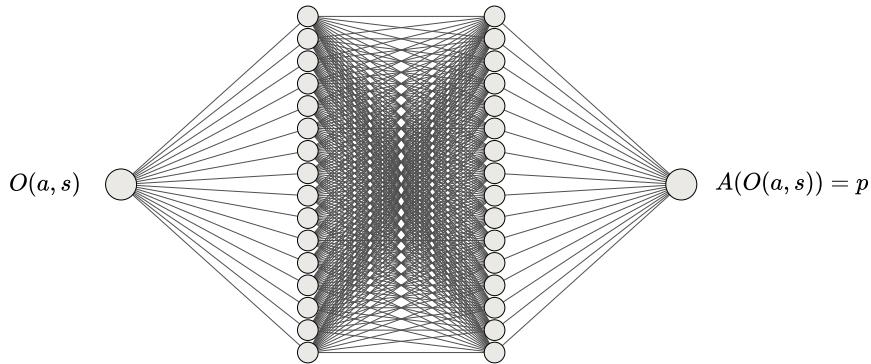


Figure 5.3: Two hidden layer feed-forward neural network, with each layer consisting in 16 fully connected neurons.

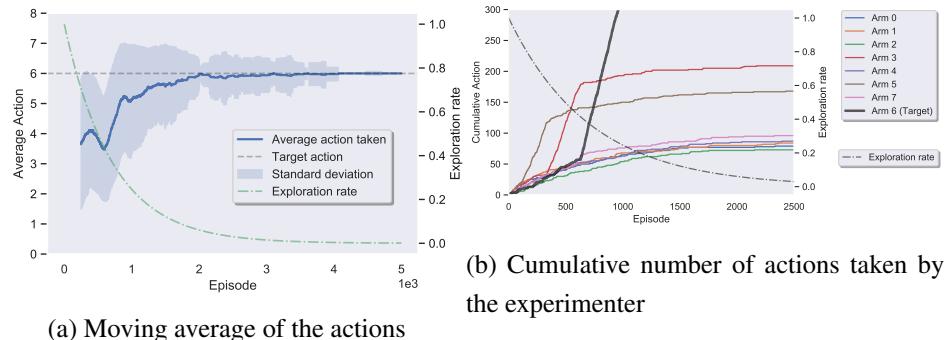


Figure 5.4: Training of example 5.1.3. These two figures shows us how the agent realizes that in order to get reward it needs to take the target action (the lever 6 in this case). In a) we can see the average value of the action taken in each episode how it converges to the target lever. In (b) we can see the cumulative number of actions taken by the agent and how around the 600th episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 5000$, $\beta_{\text{experimenter}} = 0.01$, $\text{lr}_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

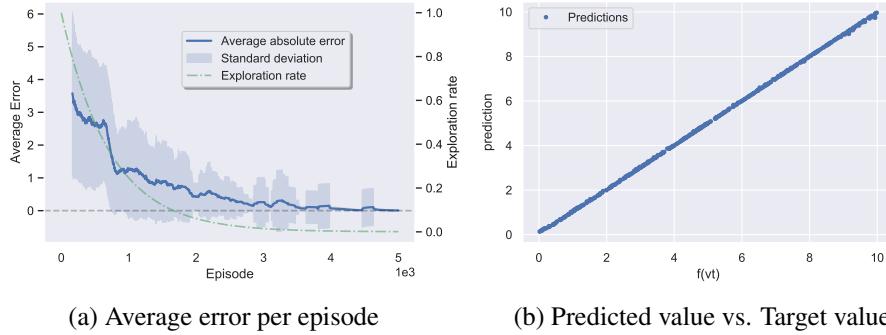


Figure 5.5: Performance of the example 5.1.3. These two figures shows us how the agent learns to predict the identity function. In a) we can see the average value of the error drops quickly to zero as the agent identify the correct lever and the exploration rate vanishes. In (b) we can see the predictions of the agent after 5000 episodes and how it fits nearly perfectly the identity function. Information: $N = 5000$, $\beta_{\text{experimenter}} = 0.01$, $lr_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

- The reward generating function r is defined as in q (4.8).

Running a few tries we find that by setting the number of episodes to $N = 5000$ the agent always finds the target lever and correctly learns the identity function. In the Fig. 5.4b we can observe how the agent discovers the target lever around the episode 600, and once the lever is found, as it is expected, the analyzer converges very quickly to the identity function (Fig. 5.4a). Although this example might look trivial, is not evident that it should work. Note that at the beginning, neither the experimenter or the analyzer has any information about the target lever or the function $f(s')$. In the exploration phase, the experimenter chooses random levers and the analyzer tries to fit the function with most of the times useless inputs but 1/8th of the times a useful value. Then the Gradient Descent algorithm is applied and results to be surprisingly robust to the noise produced by the useless inputs.

Example 5.1.4. In this example we keep the exact same configuration than in the previous example but changing only the function $f(s')$. Instead of the identity we are going to ask for a complicated function and test if the agent finds the lever. The chosen function is:

$$f(s') = f(v_t) = e^{\sin(v_t)} \quad (5.18)$$

Keeping the same hyper-parameters we observe that 5,000 episodes are not enough for the agent to succeed. However, trying with 50,000 episodes we observe that

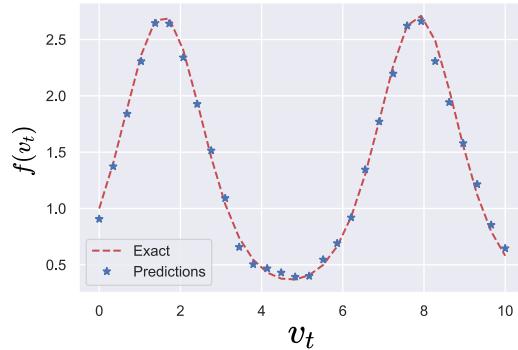


Figure 5.6: Graphical representation of the predictions made by the analyzer after the training compared to the exact value $e^{\sin(v_t)}$ for the interval $[v_{min} = 0, v_{max} = 10]$ after 50,000 episodes of training.

the agent finds without problems the correct lever (Fig. 5.7a) and successfully learns the function $f(v_t)$ (Fig. 5.6). This reveals that non-trivial correlations can proportionate enough bias to initiate the feedback loop, although the duration of the exploration phase for it to start increases with the complexity of the correlation.

Example 5.1.5. In this example we are going to see what happens if instead of a small number of arms, we have a large number of arms in which only one gives the relevant value. If we set $k = 100$ and a simple but not trivial polynomial function $f(v_t) = v_t^2 - v_t$ we find that the agent finds the correct lever also in less than 50,000 episodes (Fig. 5.9a). The predictions of the analyzer after the training are nearly perfect due to the simplicity of the function and the high number of training steps.

These last two simple examples show us that the agent performs well for both, complex functions and large action spaces. There is no reason to think that larger action spaces or more complex correlations would impose any limitations besides an increase on the number of episodes required or the need of a more flexible analyzer. As we will see in another example later, even with a more complex configuration and 2,000 different levers the agent is able to find the correct lever (although the training took around 5h in an average laptop processor).

In the next examples we will try different configurations in which the outcome are multidimensional. We will also introduce the concept of auxiliary parameters, that would be very useful for more complex configurations of the E-A architecture.

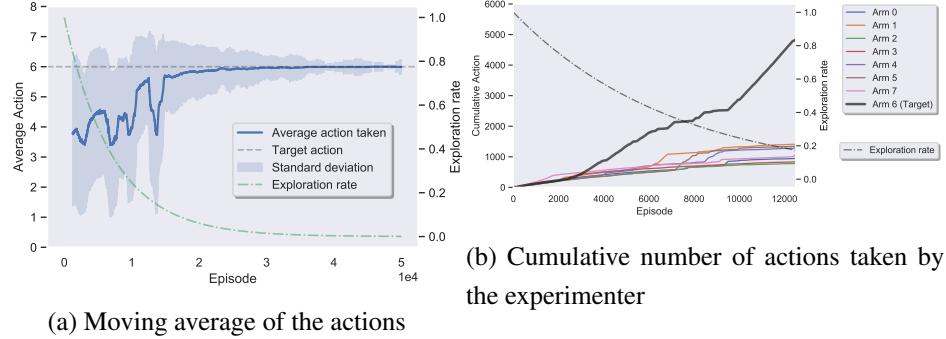


Figure 5.7: Training of example 5.1.4. These two figures shows us how the agent realizes that in order to get reward it needs to take the target action (the lever 6 in this case). In (a) we can see the average value of the action taken in each episode and how it converges to the target lever. In (b) we can see the cumulative number of actions taken by the agent and how around the 2000th episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 5000$, $\beta_{\text{experimenter}} = 0.01$, $lr_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

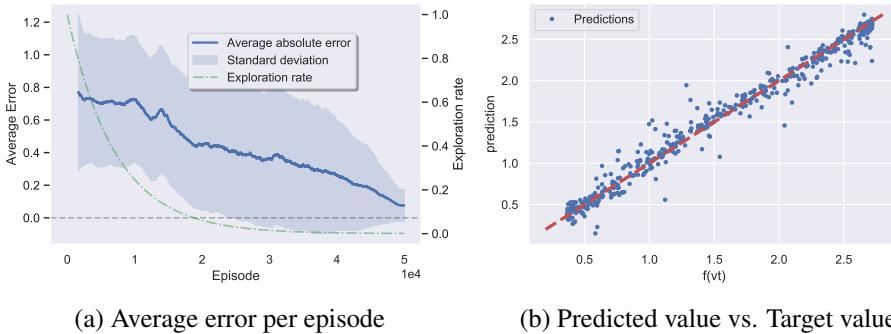


Figure 5.8: Performance of the example 5.1.4. These two figures shows us how the agent learns to predict the identity function. In (a) we can see the average value of the error drops to zero as the agent identify the correct lever and the exploration rate vanishes. In (b) we can see the predictions of the agent after 50,000 episodes and how it fits with acceptable accuracy the target function. Information: $N = 50,000$, $\beta_{\text{experimenter}} = 0.01$, $lr_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

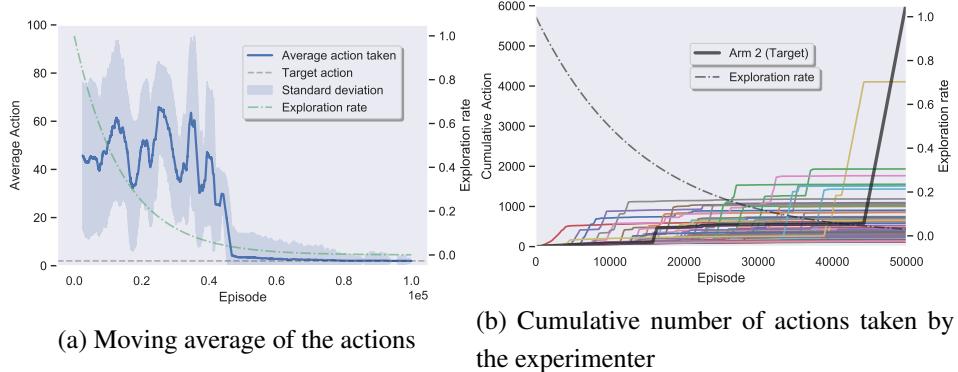


Figure 5.9: Training of example 5.1.5 . These two figures shows us how the agent realizes that in order to get reward it needs to take the target action (the lever 2 in this case). In a) we can see the average value of the action taken in each episode and how it converges to the target lever, despite of the large number of levers. In (b) we can see the cumulative number of actions taken by the agent and how around the 40,000th episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 10^5$, $\beta_{\text{experimenter}} = 0.01$, $\text{lr}_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

Modified multi-armed bandit II

In this example we are going to set $k = 8$ again. In this case, each lever is going to have three values associated instead of only one. But we keep only a valid target lever t and the rest output useless values. At the beginning of each episode, the state of the SPE is generated by associating to each arm three real numbers $\{x_a, y_a, z_a\}$. The values ares sampled from the uniform distributions $\{\mathcal{U}(v_{\min}, v_{\max})\}_{v=x,y,z}$. Formally, the SPE state space is:

$$S \equiv \{s = (x_0, y_0, z_0, \dots, x_k, y_k, z_k) \in \mathbb{R}^{3k} : \quad (5.19)$$

$$\forall v, i \quad v_i \sim \mathcal{U}(v_{\min}, v_{\max})\}$$

Now we can set $f(s')$ to be 3-variable function, for example:

$$f(x_t, y_t, z_t) = x_t y_t z_t \quad (5.21)$$

Note that in this case at the input of the analyzer we will feed three parameters instead of one, maintaining the same hidden layers. Keeping the rest of the agent

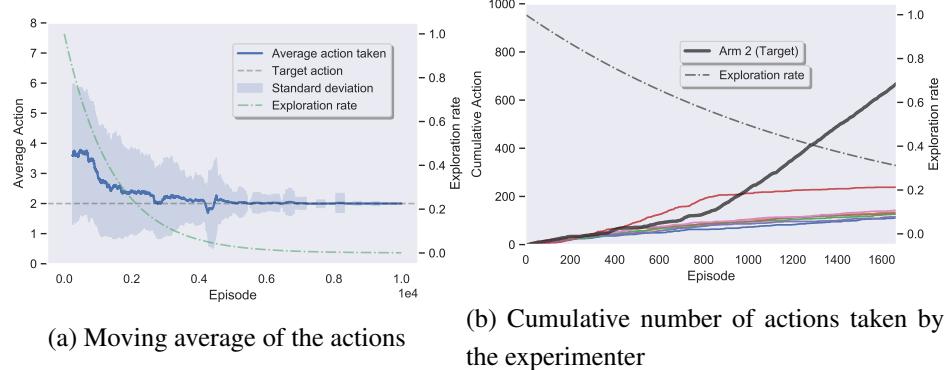


Figure 5.10: Training of example for the 3-variable example. These two figures shows us how the agent realizes that in order to get reward it needs to take the target action (the lever 2 in this case). In (a) we can see the average value of the action taken in each episode and how it converges to the target lever. In (b) we can see the cumulative number of actions taken by the agent and how around the 10,000th episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 10^4$, $\beta_{\text{experimenter}} = 0.01$, $\text{lr}_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

settings like in the previous examples we find that it discovers the correct action and learns to predict $f(s')$ in less than 10,000 episodes. This behavior is almost as good as for the single variable case (Fig. 5.10a). This means that adding more elements to the outcomes does not affect significantly the performance of the algorithm. This was to be expected, since the addition of more parameters adds more tools to recognize the correlations at the expense of a maybe more complicated computation of the function. However, the used analyzer has enough flexibility to fit to those kind of functions.

A question that one may ask is: "What if the agent is allowed to pull more than one lever per episode, for example, n levers, to then be asked for a prediction that involves the parameters of $l \leq n$ levers?" In such a case the problem can be mapped to a single-lever bandit like the ones we have seen by associating a lever to each one of the possible combinations of pulls. This is, for a bandit with k levers and n pulls each episode, the number of possible combinations is just $\binom{k}{n}$. And each new bandit would output the n parameters that would result from all the pulls of the associated combination. A more interesting, but also more difficult problem, is to create an agent that also optimizes in the number of pulls per

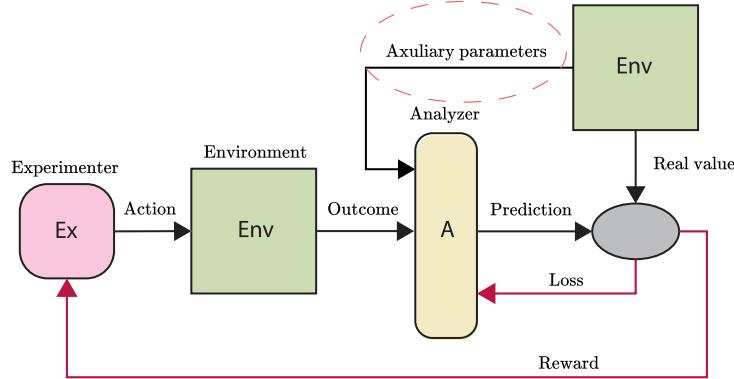


Figure 5.11: To add the auxiliary parameters to the learning loop we just need to make a small modification to the diagram.

episode, to find the minimum number l of pulls needed to maximize the precision of the predictions. This kind of agent would be very interesting in the context of minimal representation learning. The main challenge here is that changing the number of pulls changes also the number of features fed to the analyzer. A possible solution would be to add a penalization to the reward proportional to the number of pulls performed per episode and solving the issue of the number of features by using placeholders to keep the neural network architecture working. However, the design of this architecture is beyond the scope of this work and it is left for future investigations.

Modified multi-armed bandit III

In this version of the multi-armed bandit we are going to introduce a slight change in the architecture of the bandit. This consists in adding to the input of the analyzer some auxiliary information needed to make the prediction. In addition to the outcome $\mathbf{O}(a, s)$ and potentially the action a , we can also feed to the analyzer more parameters that might be relevant to make the prediction. Let us illustrate again this concept by using the pendulum example.

Suppose that, instead of directly asking the analyzer for the period T of the gravity pendulum, what we want the analyzer to predict is the angle $\theta(\tau)$ of the pendulum at a given time τ . The accuracy of the analyzer predicting the angle would determine the reward. However, the dynamics are determined by a second order differential equation. In order to predict the angle $\theta(\tau)$ correctly, the analyzer

needs at least three extra parameters. The time τ and the values $\theta(0)$ and $\dot{\theta}(0)$. It is, in general, a good practice to sample the extra features of the analyzer uniformly at random from a selected interval. This would force the analyzer to learn without bias within the given interval. We can see how the learning loop is modified with the new addition (Fig. 5.11). We shouldn't expect any drastic change in the performance compared to the multi-parameter case. We are just adding parameters, the only difference is that they are not determined by the action of the experimenter. This means that the analyzer has some degree of information even if the experimenter takes wrong choices. We can expect the analyzer performing better than random guessing regardless the performance of the experimenter.

The results of running this example are summarized in Fig. ???. Like we expected, it worked similarly to previous examples.

Example 5.1.6. This example will complete the pendulum example. Instead of asking the analyzer to predict the value of the period T , we are going to ask for the angle $\theta(\tau)$ of the pendulum. To do it we are also going to feed to the analyzer the instant τ and the initial values $\theta(0)$ and $\dot{\theta}(0)$. All these values are generated from a uniform distribution randomly each episode. We assume that the pendulum behaves like an simple harmonic oscillator and therefore:

$$f(L, \tau, \theta(0), \dot{\theta}(0)) = \theta(\tau) = \theta(0)\cos(\omega\tau) + \dot{\theta}(0)\sin(\omega\tau) \quad (5.22)$$

where $\omega = \sqrt{g/L}$. Results on Fig. 5.12.

Example 5.1.7. In this short example let us see how our agent can solve a multi-armed bandit problem with $k = 2000$, with an auxiliary parameter and a non-trivial function to predict. There is only a target lever t that outputs the value v_t . Let u be an auxiliary parameter sampled form a uniform distribution and fed to the analyzer. The function asked to predict is:

$$f(u, v_t) = \sqrt{uv_t}\sin(v_t) \quad (5.23)$$

We ran it for $N = 2 \times 10^6$ and we found that the agent successfully discovered the correct lever in around 6×10^5 episodes (Fig. 5.13a).

5.1.4 Summary of the multi-armed bandit

TO BE WRITTEN

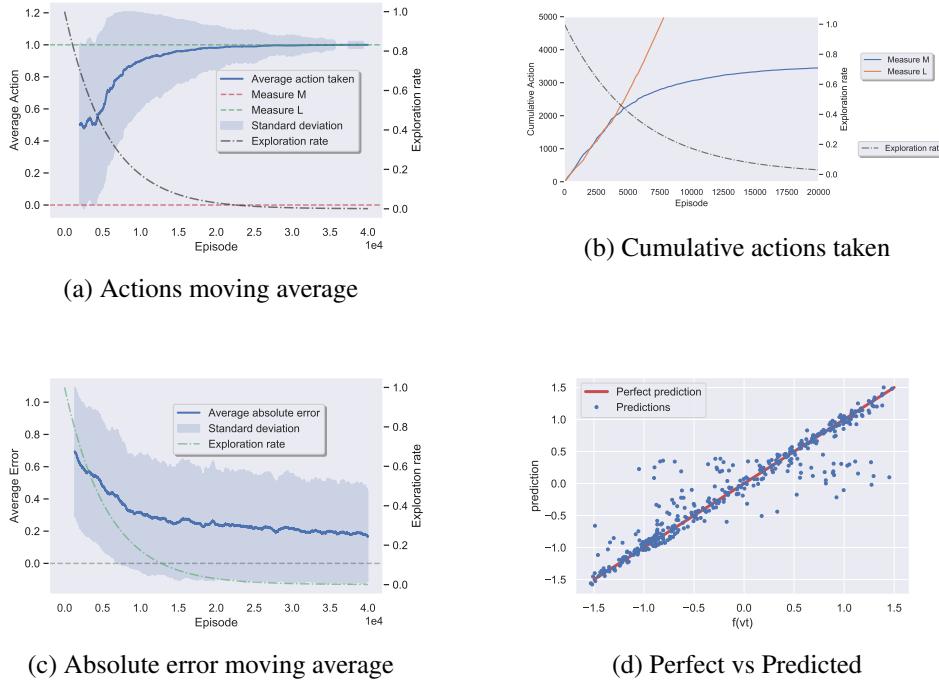


Figure 5.12: Summary of the training for the example 5.1.6. We observe that the actions converge like in previous examples. The analyzer does not completely fit the function, although shows clear signs of learning. This could be due to an inadequate Neural Network architecture or insufficient training. Information: $N = 40,000$, $\beta_{\text{experimenter}} = 0.01$, $lr_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

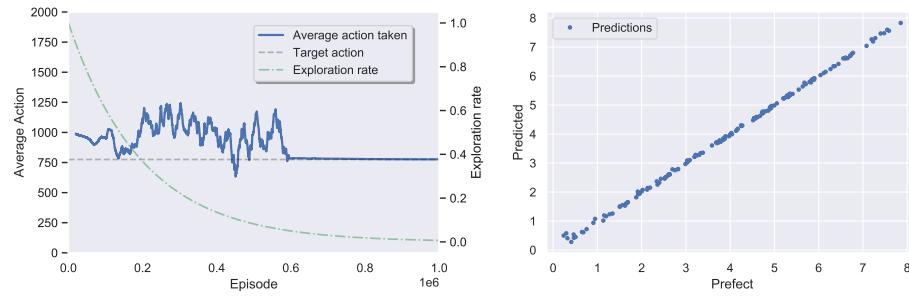


Figure 5.13: Summary of the training example (5.1.7) Information: $N = 2 \times 10^6$, $\beta_{\text{experimenter}} = 0.01$, $lr_{\text{analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

Chapter 6

E-A architecture for sequential experiments

In the previous section we showed how the simplest version of the E-A architecture can find the optimal strategy for experiments that can be expressed as a multi-armed bandit problem. This is: experiments with a discrete number of action that can be predetermined before the experiment is started. However, in general it is not possible to express every experiment as a multi-armed bandit problem. For example, if the action space is continuous (e.g. selecting the position of a detector) or if the experiment requires from a dynamical intervention, for example, if in an experiment consisting of sequential actions, it is needed the result of the preceding actions to take the correct choices. In this section we are going to show how one could tackle this kind of experiments using the E-A architecture.

6.1 Sequential experiments

So far, all the E-A models we have used consists of an experimenter $E : \emptyset \rightarrow \mathcal{A}$ whose action a over the environment in the state s produce some outcome $\mathbf{O}(a, s)$, and then the outcome and maybe some additional information are fed to the analyzer $A : \mathcal{M} \rightarrow \mathcal{P}$ to output a prediction about a physical property of the system. Then such property is observed (or in our case, simulated) and compared to the prediction to generate a reward and a loss, to train both E and A .

This simple structure is enough for experiments in which we can group all actions and measurements to be taken in a single action at the beginning of the experiment. For example, if the experiment consists on measuring different pa-

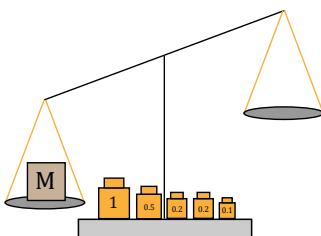
rameters of an electric circuit to know the power consumed by a section of the circuit. We can group all those choices of measurements, e.g. measuring the intensity at node A and the voltage at node B, in a single action taken at the beginning of an episode. This is because the outcomes of the measurements don't condition each other. However, in other experiments the outcomes of previous actions may condition the actions to be taken in order to get the correct parameters for a prediction. We will call this experiments *sequential*, since a number of actions need to be taken sequentially. Let us illustrate the concept with an example of a sequential experiment:

Example 6.1.1. Imagine an experiment that has the objective of determining the mass M of an object with the following tools:

- A beam balance with two plates that leans to the side of the heavier plate.
- A set of calibrated weights consisting of:
 - A mass of 100 g
 - Two masses 200 g
 - A mass of 500 g
 - A mass of 1 Kg

If the mass M to be weighted is less than 2Kg we can find value of the mass M , with a uncertainty of 100 g, in only five uses of the scale. For example, if the mass is 1.3 Kg a procedure would be the following:

1. M vs 2 Kg → right
2. M vs 1 Kg → left
3. M vs 1.5Kg → right
4. M vs 1.2 Kg → left
5. M vs 1.4 kg → right
6. We know that the mass is:
 $1.3 \text{ Kg} \pm 100 \text{ g}$



However, in order to find the value of M in five uses of the scale or less we can't have a prefixed strategy at the start of the experiment. We need to observe

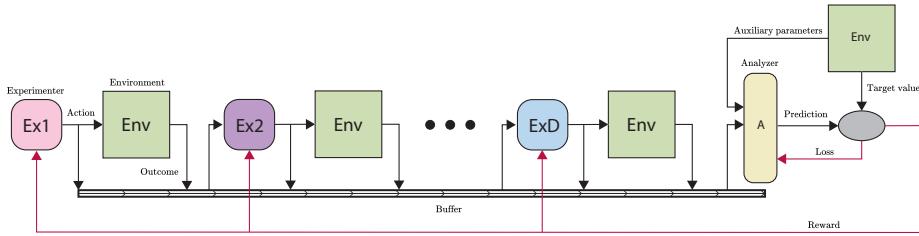


Figure 6.1: Diagram of the learning loop for a E-A architecture for a sequential experiment with D steps. The structure is the same as for the simple case but concatenating D experimenters instead of a single one.

the outcomes of each measurement individually and respond accordingly. We can not express then this experiment as a multi-armed bandit decision problem and therefore is sequential.

6.2 E-A for sequential experiments

In this section we are going to explain how to use the E-A architecture to tackle sequential experiments. The basic idea behind this approach is to concatenate as many experimenters as sequential actions we need to take and each experimenter is fed with the information gathered by the actions and outcomes gathered by the preceding experimenters.

Suppose we have a sequential experiment that needs from D sequential actions. Then we would have D different experimenters $E_n : \mathcal{X}_n \rightarrow \mathcal{A}_n$ with $\mathcal{X}_0 \equiv \emptyset$. Each \mathcal{A}_n can be different depending on the nature of the experiment, since the experimenters E_n could be different reinforcement learning agents with different architectures. The experimenters' environments \mathcal{X}_n can consist of any information stored in the buffer at the moment they are fed to E_n . The SPE is in the state $s_n \in S$ when the action $a_n \in \mathcal{A}_n$ is taken and it produces the outcome $\mathbf{O}_n(a_n, s_n) \in \mathbb{R}^{d_n}$ where d_n is the number of parameters that are outputted after the action a_n . Then, as in the single experimenter version, all the information in the buffer together with auxiliary parameters can be stored into $m \in \mathcal{M}$ to feed the analyzer $A : \mathcal{M} \rightarrow \mathcal{P}$. Then the prediction is compared with the observed/simulated value of the property to be predicted with a reward generation function r_n . The reward could be different for each experimenter depending on the nature of the architecture. However, in this work only explored the architecture with a single reward generation function r for all experimenters.

One of the main limitations of this approach is that we need to fix the number D of actions beforehand. In some experiments this may not be relevant since the number D is fixed by the experiment itself. However, in others this fact might be important. For example, in the context of the example 6.1.1, we may ignore what is the minimum number of actions needed to determine the mass with an uncertainty of less than 100 g. The agent will only search for the best strategy for the D actions given, which may not be optimal.

6.3 Example

In this section we are going to show an example of a sequential experiment in which the A-E architecture finds the optimal strategy.

6.3.1 The experiment

We have a box of mass M on an infinite horizontal surface with friction coefficient μ respect to the mass M . The SPE is programmed so that a canon shoots a bullet of mass m with velocity v that collides inelastically with M when an episode is started. The mass M is the only parameter restarted each episode by sampling uniformly at random from the interval $[M_{min}, M_{max}]$. We can calculate the distance traveled by the box using the conservation of momentum. The moment of the bullet is $p_b = mv$. Applying the conservation of momentum we can obtain the velocity of the box after the impact:

$$v_M = \frac{p_M}{M} = \frac{m}{M}v \quad (6.1)$$

We know that the friction force is $F = \mu N = \mu mg$. When the box stops, due to the conservation of energy, the work realized by the friction force must be the kinetic energy K of the mass $M+m$ of the box after the impact of the bullet. Since the friction force is assumed to be constant, the work done is $W = Fx = \mu mgx$, where x is the distance traveled until it stops. Equating this term to the kinetic energy we obtain:

$$x = \frac{1}{2g\mu} \left(\frac{vm}{M+m} \right)^2 \quad (6.2)$$

We give the agent access to two sensors to measure the position of the box:

- **A low precision but wide range (LPWR) position sensor:** it works as follow: the landing zone is divided into four regions or zones of equal size.

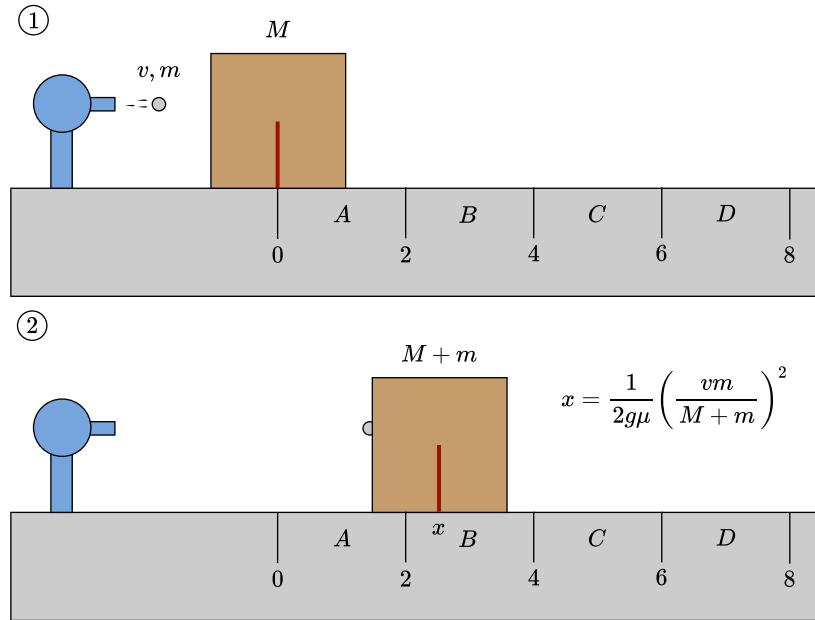


Figure 6.2: Diagram explaining the physical environment. The canon shoots a bullet of mass m with velocity v . When it collides inelastically with the box of mass M , the box displaces. Due to the friction force it stops in one of the four zones A, B, C or D .

This sensor outcomes the middle point of the zone in which the box has landed. For example, in the situation depicted in Fig. 6.2 this sensor would produce the outcome: 3, since it is the middle point of the zone B . This allows to know roughly where the box landed, but with low accuracy.

- **A high precision but narrow range (HPNR) position sensor:** It works as follow: to activate it you need also to specify a region and if the box landed on the specified region the outcome of the sensor is the exact position of the box. Otherwise it outputs a constant negative value (set to -1) representing a failed measurement. For example, in the situation depicted in Fig. 6.2 this sensor is placed in the zone B it will output the value of x . If placed in any other zone it outputs -1 .

The goal of the agent is to predict the position in which the box will land for a shooting with velocity v_{test} and bullet mass m , but only with **two uses of the sensors**. The velocity v_{test} is selected randomly each episode from a uniform distribution for the interval $(v - \frac{v}{2}, v + \frac{v}{2})$. This velocity v_{test} would be the auxiliary parameter

fed to the analyzer.

Using the same procedure that we did for the first shooting, we can find that the new displacement d of the box after the shooting is:

$$d = \frac{1}{2g\mu} \left(\frac{v_{test}m}{M + 2m} \right)^2 \quad (6.3)$$

Since d depends on the mass M , and M can be inferred from the first displacement x using (6.2), the agent has enough information to predict the displacement of the test shooting if it detects correctly the position x of the box.

This experiment is sequential because to always obtain the exact position of the box, the agent needs to use sequentially the sensors in a specific way. This strategy consists on choosing in the first try the LPWR sensor, to detect in which zone the box landed, and in the second use of the sensors to choose the HPNR sensor and place it in the zone inferred from the first outcome.

6.3.2 Mathematical characterization of the experiment

Let us contextualize each element of the training using the mathematical formulation used in the text.

- The state of the SPE is determined by the tuple $s = (M, v_{test})$.
- The E-A agent has two experimenters: $E_1 : \mathcal{X}_1 \rightarrow \mathcal{A}$ and $E_2 : \mathcal{X}_2 \rightarrow \mathcal{A}$ where $\mathcal{X}_1 = \emptyset$ and $\mathcal{X}_2 = \mathcal{A} \times \mathbb{R}$, since we feed the second experimenter the action a_1 of E_1 and the outcome $O(a_1)$ of its action.
- The action space for both experimenters is $\mathcal{A} = \{0, 1, 2, 3, 4\}$, with the action 0 representing the action of using the LPWR sensor, and the actions $\{1, 2, 3, 4\}$ represent the actions of using the HPNR and placing it in the zones A, B, C and D respectively.
- The analyzer is $A : \mathcal{M} \rightarrow \mathbb{R}$, where each element of \mathcal{M} is a tuple $(a_1, O(a_1, s), a_2, O(a_2), v_{test})$.
- The quantity to be predicted is $f(s) = d = \frac{1}{2g\mu} \left(\frac{v_{test}m}{M + 2m} \right)^2$
- The reward function is $r(\Delta) = \exp \left(-\frac{1}{2} \left(\frac{\Delta}{\sigma} \right)^2 \right)$, where $\Delta = \frac{f(s)-p}{p}$ and p is the output of the analyzer.
- The loss function for the analyzer is $L(f(s), p) = (f(s) - p)^2$

6.3.3 Technical details of the agents

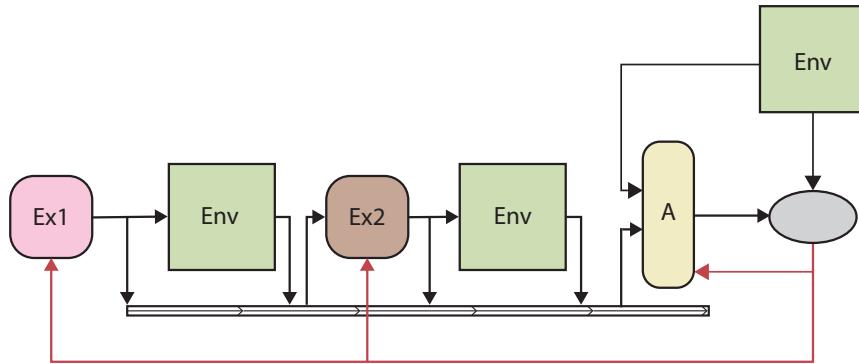


Figure 6.3: Diagram of the learning loop for the experiment.

Experimenter 1

E_1 is the same kind of experimenter that we used in the previous examples (2), since for E_1 everything looks like a multi-armed bandit. The agent was subjected to an ϵ -greedy exploration phase with linear decrease of the exploration rate.

Experimenter 2

For the experimenter 2 now we have inputs, so the reinforcement learning algorithm of E_1 is not valid. Since some of the inputs are real numbers, we opted to use a standard Deep Q Network algorithm with online training (See Appendix [TODO]). We also tried a modified version of the Deep Q Network (DQN) algorithm equipped with a memory and two neural networks (see appendix) to improve the performance. We could have opted for a normal Q-learning algorithm as well, but would have required some amount of hardcoding, which is undesirable in our purpose of finding agents as independent as possible.

The structure of the neural network for this agent is a two hidden layers (16x16) fully connected sigmoidal neurons for both versions. We ignore if this is an appropriate structure since we haven't performed an optimization of the hyperparameters.

The agent was subjected parelly to the same ϵ -greedy exploration phase than E_1 .

Analyzer

The analyzer in this case is a sequential feedforward neural network with 3 hidden layers ($20 \times 10 \times 2$) fully connected ReLU neurons.

6.3.4 Results and discussion

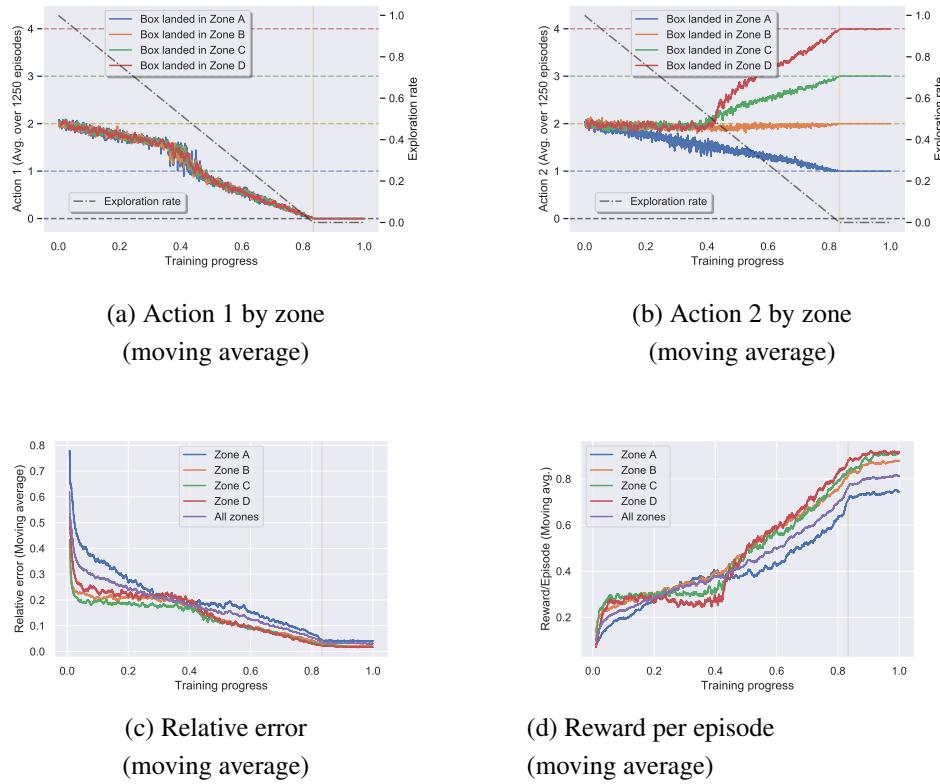


Figure 6.4: Summary of the training for the simple DQN agent. In (a), we can see how the first experimenter learns to take the action 0, this is, using the LPWR sensor always. In (b) we see how the second experimenter learns the optimal strategy converging to the correct action for every zone. This implies the agent learns to interpret correctly the outcome obtained by E_1 and places the HPNR sensor in the corresponding zone. In (c), we observe the decrease on the relative error, that goes to nearly zero for every zone, meaning that the analyzer learns to interpret correctly the outcomes. In (d), we see the reward obtained by the agent per episode. It gets close to the maximum of one at the end of the training for all zones. Number of episodes: $N = 2.5 \times 10^6$

Since $x \propto 1/M^2$, if we sample uniformly at random M , it means that the zones closer to the canon are going to be quadratically more frequent than zones further away. With the used parameters, we obtain that the box lands in the zone $A \sim 54\%$ of the times, $\sim 27\%$ in B , $\sim 12\%$ in C and $\sim 7\%$ in zone D . Then our training data for the agent is strongly biased to the closer zones A and B .

In our tests, we found that with not enough training, the agent in an attempt to maximize the reward and minimize the loss learns to exploit this imbalance by choosing always to place the HPNR sensor in zones A and B . This is actually a clever strategy, since it obtains the exact position of the box for 80% of the episodes. However is suboptimal, since by placing first the LPWR sensor and then using its outcome to place the HPNR sensor in the correct place it is possible to obtain the exact position a 100% of the times.

We observed, that if we let the agent train long enough (2.5×10^6 episodes), it finds the optimal strategy even for an unbalanced scenario. This shows that the E-A architecture is potentially valid for getting information about rare events by showing robustness against imbalanced environments. In Fig. 6.4 we have a summary of the training. As we can see in Figs. 6.4a and 6.4b both experimenter learn to cooperate, the experimenter 1 taking always the action 0, and the experimenter 2 using the outcome of the action taken by experimenter 1 to place correctly the HPNR sensor. It results surprising that this can be achieved only with the rewards generated from the at first untrained analyzer. The working principle is supposed to be the same than for the multi-armed bandit: to let the analyzer exploit the correlations during the exploration phase to start a feedback loop in which all agents improvement improves each other performance.

Changing the DQN algorithm of the experimenter 2 to a more sophisticated Dual DQN algorithm with experience replay reduced the number of episodes needed to find the optimal strategy by a factor of 10, needing only 2.5×10^5 episodes to show almost identical results. However, both algorithms took a similar amount of computation time (~ 5 h in an average laptop). This is because the Dual DQN algorithm fits several points from the memory in each episode, taking longer to compute each episode.

This shows us that there is a lot of room for improvement by choosing different reinforcement learning agents or hyper-parameters. It also manifest one of the strength of this architecture: its modularity. With very few changes on the learning architecture we can try completely different agents.

Appendix A

Appendix ?

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliography

- [1] Ahmed Alkhateeb. Can scientific discovery be automated? *The Atlantic*.
- [2] C Glenn Begley and Lee M Ellis. Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012.
- [3] Lutz Bornmann and Rudiger Mutz. Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references. *Journal of the Association for Information Science and Technology*, 66(11):2215–2222, 2015.
- [4] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), Dec 2019.
- [5] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [6] Daniele Fanelli. Opinion: Is science really facing a reproducibility crisis, and do we need it to? *Proceedings of the National Academy of Sciences*, 115(11):2628–2631, 2018.
- [7] Sabine Hossenfelder. The crisis in physics is not only about physics. *Back-ReAction*.
- [8] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical Review Letters*, 124(1):010508, 2020.
- [9] Esteban Ortiz-Ospina Max Roser and Hannah Ritchie. Life expectancy. *Our World in Data*, 2020. <https://ourworldindata.org/life-expectancy>.

- [10] Alexey A. Melnikov, Hendrik Poulsen Nautrup, Mario Krenn, Vedran Dunjko, Markus Tiersch, Anton Zeilinger, and Hans J. Briegel. Active learning machine learns to create new quantum experiments. *Proceedings of the National Academy of Sciences*, 115(6), Jan 2018.
- [11] Hendrik Poulsen Nautrup, Tony Metger, Raban Iten, Sofiene Jerbi, Lea M. Trenkwalder, Henrik Wilming, Hans J. Briegel, and Renato Renner. Operationally meaningful representations of physical systems in neural networks, 2020.
- [12] Andrea Saltelli and Silvio Funtowicz. What is science’s crisis really about? *Futures*, 91:5 – 11, 2017. Post-Normal science in practice.
- [13] Roberta Sinatra, Pierre Deville, Michael Szell, Dashun Wang, and Albert-László Barabási. A century of physics. *Nature Physics*, 11(10):791–796, 2015.
- [14] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.
- [15] Carol Tenopir, Lisa Christian, and Jordan Kaufman. Seeking, reading, and use of scholarly articles: An international study of perceptions and behavior of researchers. *Publications*, 7(1), 2019.