Chapter 5

E-A architecture in different scenarios

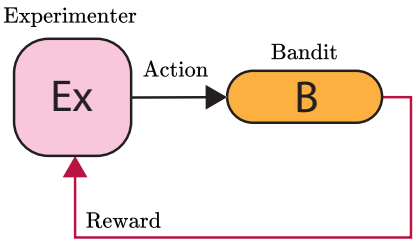
5.1 Scenario 1: Modified multi-armed bandit

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 Probablity Theory. Let us start by introducing the general problem.

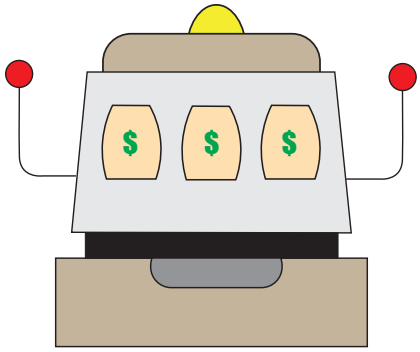
5.1.1 General multi-armed bandit problem

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 doesn't 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 $\mathcal{A} = \{0, 1, ..., k\}$ and the state space $\mathcal{X}$ is again the empty set $\varnothing$. The bandit can be modelled as a probabilistic function $R : A \times ... \rightarrow \mathbb{R}$, 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 $R = \{R_1, ..., R_k\}$, 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:

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

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 $\varnothing$ 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:

$$Q(a) \leftarrow Q(a) + \alpha\left(r - Q(a)\right) \tag{5.2}$$

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:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha\left(r_n - Q_n\right) && (5.3)\\
&= \alpha r_n + (1-\alpha)Q_n && (5.4)\\
&= \alpha r_n + (1-\alpha)\left[\alpha r_{n-1} + (1-\alpha)Q_{n-1}\right] && (5.5)\\
&= \alpha r_n + (1-\alpha)\alpha r_{n-1} + (1-\alpha)^2 Q_{n-1} && (5.6)\\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} r_i && (5.7)
\end{aligned}$$

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:

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

We also have that:

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

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

Inserting (5.10) in (5.8) we get:

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

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

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:

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

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

Algorithm 2

```
procedure NON-STATIONARY MULTIARMED-BANDIT AGENT
count = 0
while count < N :
    action ← arg max Q
    reward ← bandit(action)
    Q[action] ← Q[action] + α (reward - Q[action])
    count ← 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 aren't irrelevant. They play an important role on the convergence of the algorithm. For example, if we set $Q_1 = 0 \ \ \forall a \in \mathcal{A}$, 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:

$$\alpha_n = \beta/o_n \tag{5.14}$$

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

$$o \leftarrow o + \beta(1 - o) \tag{5.15}$$

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 $\varnothing$. 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

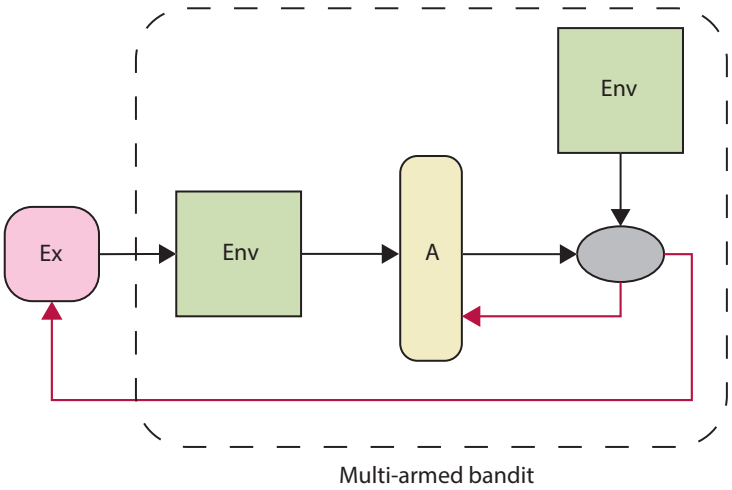


Figure 5.2: Diagram of how the simplest combination for the E-A architecture (4.1) can be modelled as a multi-armed bandit.

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 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 $r : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$. 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:

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

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

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 $r(p, v_t)$ 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 $f(v_t)$.

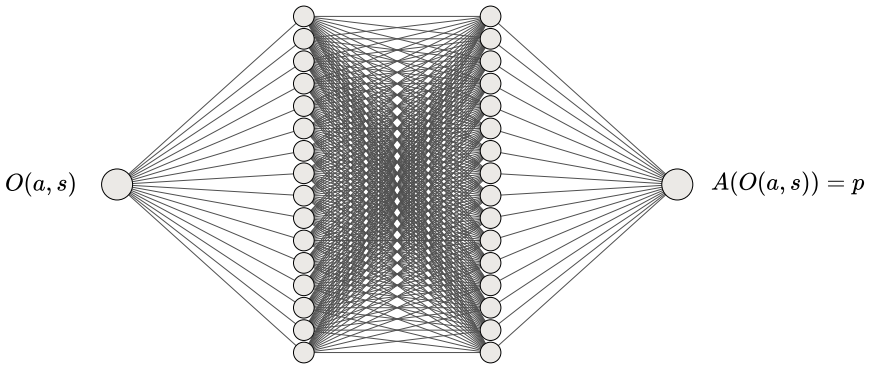
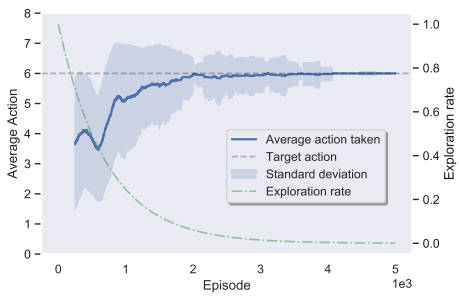
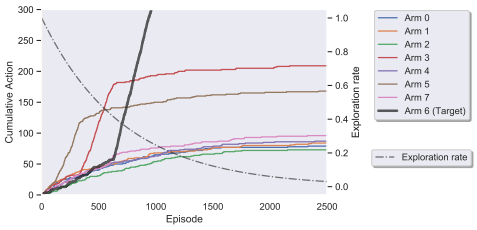


Figure 5.3: Two hidden layer feed-forward neural network, with each layer consisting in 16 fully connected neurons.

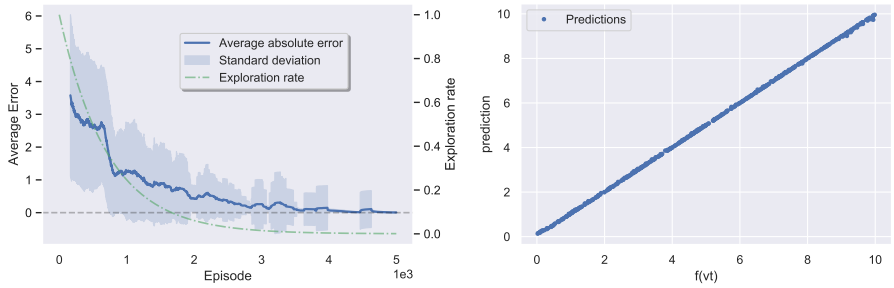


(a) Moving average of the actions



(b) Cumulative number of actions taken by the experimenter

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 600^{th} episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 5000$, $\beta_{\text{Experimenter}} = 0.01$, $\text{lr}_{\text{Analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$



(a) Average error per episode (b) Predicted value vs. Target value

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: $N = 5000$, $\beta_{\text{Experimenter}} = 0.01$, $\text{lr}_{\text{Analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

- 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 $1/8^{\text{th}}$ 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:

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

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

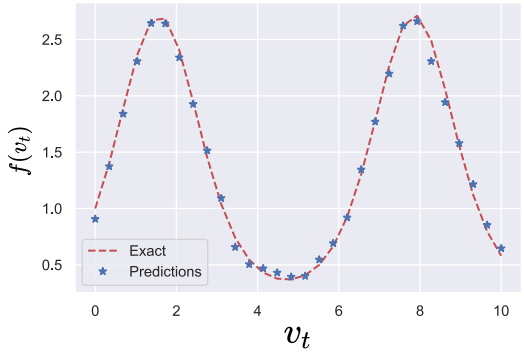


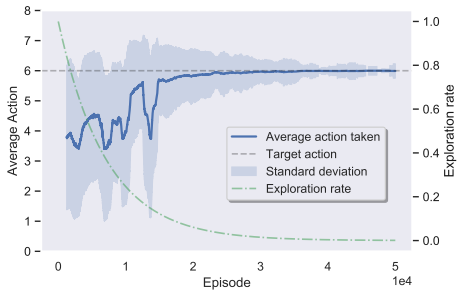
Figure 5.6: Graphical representation of the predictions made by the analyzer after the training compared to the exact value $e^{\sin(v_t)}$ for the interval $[v_{min} = 0, v_{max} = 10]$ after 50,000 episodes of training.

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.

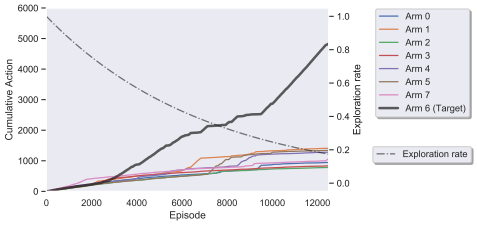
Example 5.1.5. In this example we are going to see what happens if instead of a small number of arms, we have a large number of arms in which only one gives the relevant value. If we set $k = 100$ and a simple but not trivial polynomial function $f(v_t) = v_t^2 - v_t$ we find that the agent finds the 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.

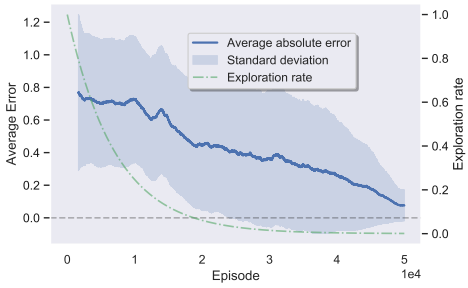


(a) Moving average of the actions

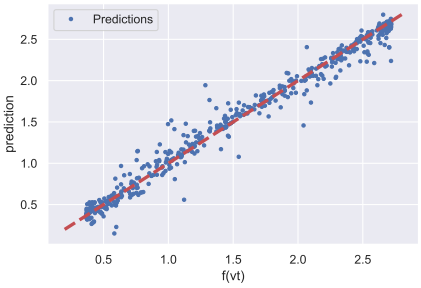


(b) Cumulative number of actions taken by the experimenter

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 2000^{th} episode the agent finds the target arm and chooses it whenever a greedy action is taken. Information: $N = 5000$, $\beta_{\text{Experimenter}} = 0.01$, $\text{lr}_{\text{Analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$

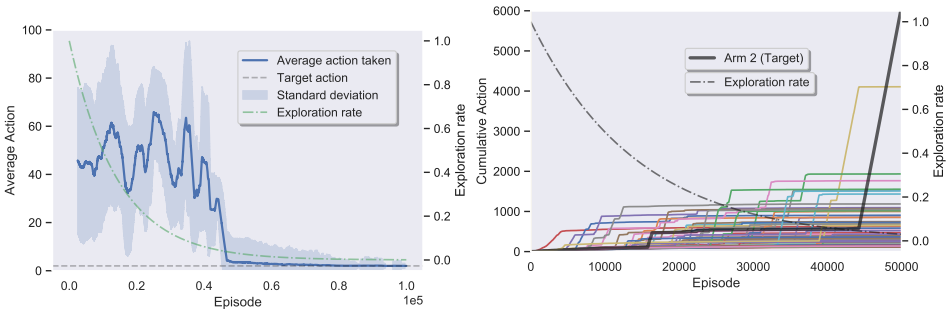


(a) Average error per episode



(b) Predicted value vs. Target value

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: $N = 50,000$, $\beta_{\text{Experimenter}} = 0.01$, $\text{lr}_{\text{Analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$



(a) Moving average of the actions

(b) Cumulative number of actions taken by the experimenter

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 $40,000^{\text{th}}$ 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:

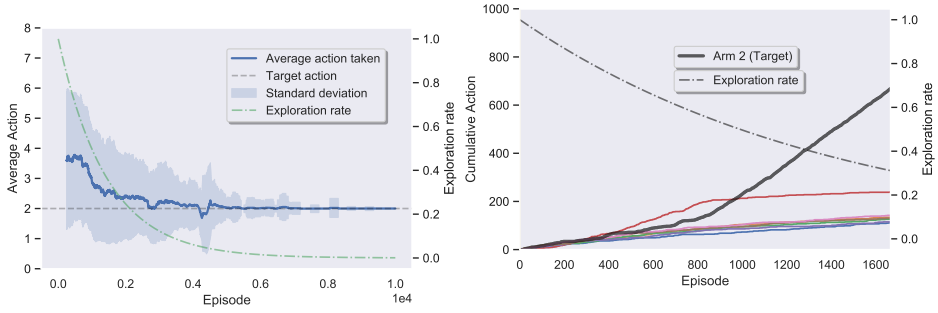
$$S \equiv \{s = (x_0, y_0, z_0, ..., x_k, y_k, z_k) \in \mathbb{R}^{3k} : \tag{5.19}$$

$$\forall v, i \;\; v_i \sim \mathcal{U}(v_{min}, v_{max})\} \tag{5.20}$$

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

$$f(x_t, y_t, z_t) = x_t y_t z_t \tag{5.21}$$

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



(a) Moving average of the actions

(b) Cumulative number of actions taken by the experimenter

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 $10,000^{\text{th}}$ 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 doesn't 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 $l \leqslant n$ 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

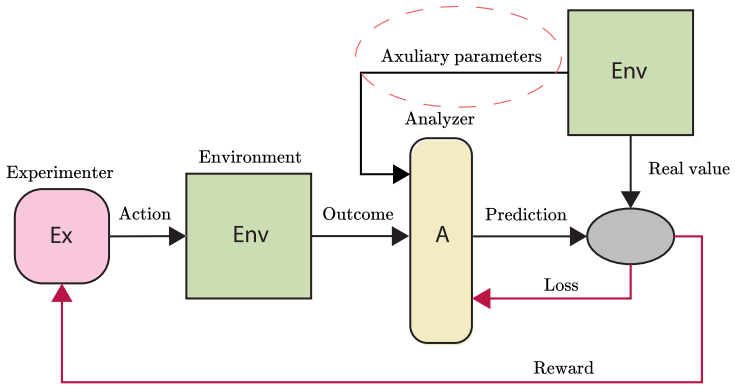


Figure 5.11: To add the auxiliary parameters to the learning loop we just need to make a small modification to the diagram.

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 feeded 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 let 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 An-

alyzer 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 aren't 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:

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

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

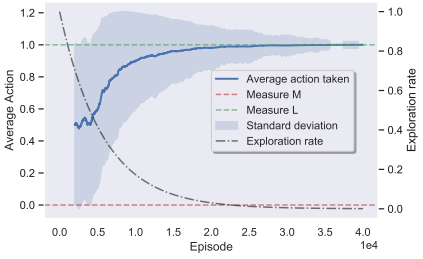
Example 5.1.7. In this short example let us see how our agent can solve a multi-armed bandit problem with $k = 2000$, with an auxiliary parameter and a non-trivial function to predict. There is only a target lever t that outputs the value v_t. Let u be an auxiliary parameter sampled form a uniform distribution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

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

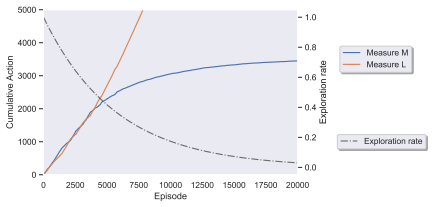
We ran it for $N = 2 \times 10^6$ and we found that the agent successfully discovered the correct lever in around 6×10^5 episodes (Fig. 5.13a).

5.1.4 Summary of the multi-armed bandit

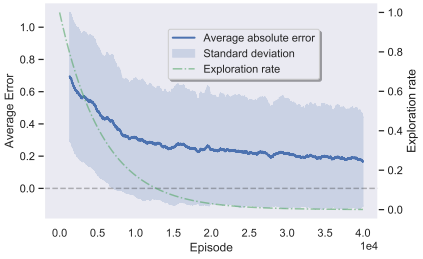
TO BE WRITTEN



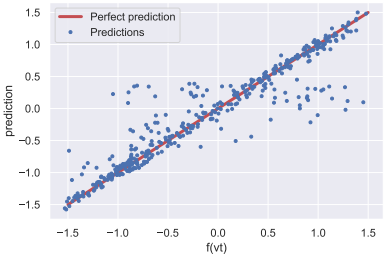
(a) Actions moving average



(b) Cumulative actions taken

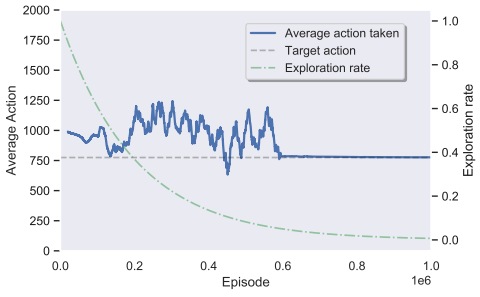


(c) Absolute error moving average

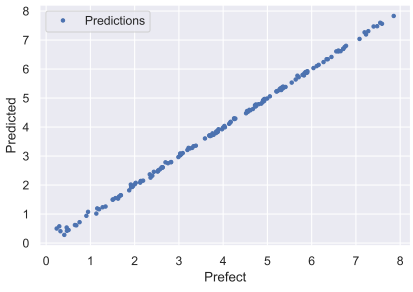


(d) Perfect vs Predicted

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 doesn't completely fit the function, although shows clear signs of learning. This could be due to an inadequate Neural Network architecture or insufficient training. Information: $N = 40,000$, $\beta_{\text{Experimenter}} = 0.01$, $\text{lr}_{\text{Analyzer}} = 0.01$, $Q_1 = 0$, $\sigma = 0.05$



(a) Moving average of the actions taken



(b) Predicted value vs. Target value

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

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 ZdeborovÃ¡. 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. *BackReAction*.

[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.