POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Homework report

ODE control with Deep Q-learning

Scientific computing tools for advanced mathematical modelling

Authors: Daniela Zanotti, Davide Carrara and Francesco Romeo

Academic year: 2021-2022

---

1. Mathematical formulation of the problem

The goal of the project is to determine a controlling procedure for the evolution of an Ordinary Differential Equation in a Deep Reinforcement Learning Framework, with the particular approach of Deep Q-Learning.

1.1. Lotka-Volterra Equations

The Lotka-Volterra system of equations, also known as Predator-Prey equations, are commonly used to describe the evolution of a number interacting populations in a simplified natural scenario. Mathematically speaking, the system is composed by n first-order nonlinear differential equations, here represented in the chosen case of two populations.

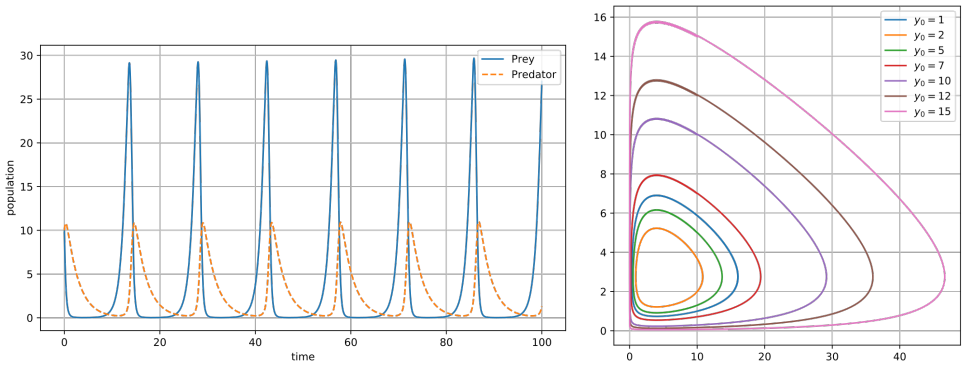
$$\begin{cases} \dot{x} = \alpha x - \beta xy \\ \dot{y} = \gamma xy - \delta y \end{cases} \tag{1}$$

The variables $\boldsymbol{x}$ and $\boldsymbol{y}$ represent respectively the number of individuals of the prey and the predator population, while $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta}$ are positive real parameters.
It is worth noting that the preys are assumed to have infinite food resources at their disposal and thus their reproduction rate is determined only by the parameter $\boldsymbol{\alpha}$, without any suffering due to scarcity in case of overpopulation. On the other hand, the parameter $\boldsymbol{\beta}$ determines the predation rate of the prey due to the predator: this is the only case of death for the prey in the system.
$\boldsymbol{\gamma}$ represents the growth rate of the predator population, which is entirely defined by its interaction with the prey: notice that the term is similar to the predation rate, but the two coefficients are different. $\boldsymbol{\delta}$ represents the natural decay of the predator population, either due to death or emigration.

1.2. Solution and the atto-fox problem

The Lotka-Volterra system of equations admits a periodic solution for any tuple of parameters, even though they are not easily expressible using trigonometric functions.
The orbits presented in Figure 1.2 can be described as level curve of the quantity $V = \delta x - \gamma \ln(x) + \beta y - \alpha \ln(y)$. In this case V can be interpreted as a form of energy for the system, which is always conserved. It is also worth noticing that for any feasible tuple of parameters is it possible to find an initial state that is also an equilibrium, meaning that the derivatives of both populations are always null.

Observing Figure 1.2 we can note a serious problem when using the Lotka-Volterra system as a model for the evolution of populations. Since the values of $\boldsymbol{x}$ and $\boldsymbol{y}$ are continuous it happens often that one population reaches extremely low values, recovering after some time. This is mathematically correct but biologically false,



(a) Example of solution of Lotka-Volterra (b) Example of orbit varying y0

Figure 1: Example of Lotka-Volterra

since any value smaller than one for the prey population would imply the extinction of the population itself. This phenomenon is known as the *atto-fox* problem, with atto-fox being the 10^{-18} part of a fox, a value that is completely senseless from a biological point of view but mathematically acceptable.

1.3. Our Problem

The main goal of our project will be to create a model able to solve the *atto-fox* problem, by tuning one or more parameters during time. In more practical terms, we will consider a population to be extinct if its number of individuals drops below a certain threshold, set to 4 and we will teach our model how to modify the parameter in order to keep both populations constantly over this threshold.
When considering only one parameter we will modify α, the growth rate of the prey. In the multivariate case we will also include δ, the death rate of the predator. The choice of the parameters is due to a possible translation into an applicative setting: it is reasonable to imagine evolution studies where the experimenter does not want one of the populations to be extinct. In such a case, controlling α and δ via the constant addition or subtraction of some individuals would be much easier than controlling the predation rate of the groups.

In order to act in a realistic scenario reasonable starting values for the four parameters are needed: we will thus refer to the standard case of the two populations of lynxes and snowshoe hares in Hudson Bay in Canada. The values for the parameters are reported here:

$$\alpha = 0.55 \quad \beta = 0.028 \tag{2}$$
$$\gamma = 0.026 \quad \delta = 0.84 \tag{3}$$

2. Methods

The problem will be solved using Reinforcement Learning techniques, with a particular focus on Deep Q-Learning.

2.1. Reinforcement Learning

A classical Reinforcement Learning scenario is determined by two main entities: the Environment and the Agent. The Environment represents all the knowledge about the system, its state and evolution, while the Agent is in charge of choosing an action at each step.
A standard reinforcement learning iteration begins with the Environment providing the current state of the system to the agent. Based on this state and on his previous knowledge of the problem the Agent will then choose one action among the ones available.
The environment receives the action and computes the new state of the system together with the reward, a quantity representing the quality of the new state. The Agent receives the reward and updates its decision policy accordingly.

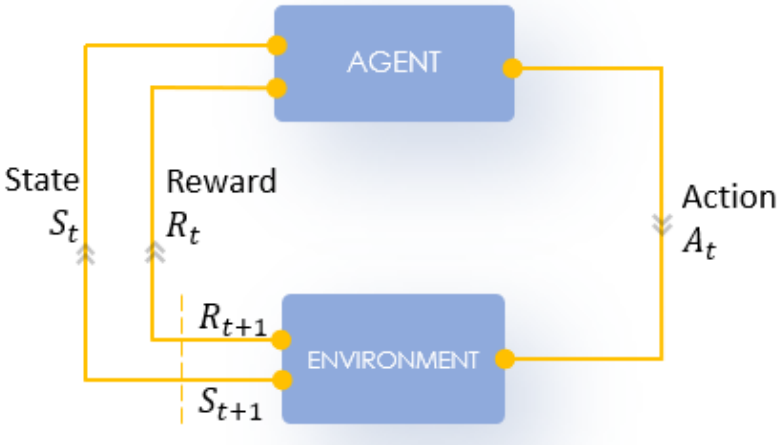


Figure 2: Reinforcement Learning Model

2.1.1 Q-learning

In the case of Q-Learning, the decision policy of the Agent is encapsulated in the so called Q function, from which the method takes its name, and that is defined as:

$$Q : S \times A \longrightarrow \mathbb{R} \tag{4}$$

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

The function Q represents the quality of each state-action combination, and will lead the model in its decision: the Agent will in fact choose the action that maximises the Q function given the state.
The function is initialized randomly and is then updated at each iteration taking into consideration the current value, the reward and the maximum possible value at the next state.
The parameters α, also known as learning rate, can be modulated and represents the importance given to the current value of the function with respect to the improvement term. High values of alpha will lead to a flexible Q, very sensitive to changes occurred during training, while small values will lead to a more stable Q.
On the other hand, the parameter γ represent the importance that the model will give to the future prediction of the state. Low levels of gamma will lead to a very short-sighted model, while higher values will force the model to focus more on future outcomes.

2.1.2 Deep Q Learning

In the case of vanilla Q-Learning, the values of the Q function are stored into a matrix having states on the rows and actions as columns: in this way it is possible to compute it for each possible combination. However, as the number of possible states increases, or assumes continuous values, the matrix approach is not feasible anymore. In this case it becomes necessary to introduce a more general approximator, which is best found in a Neural Network.
As shown in Figure 2.1.2, the Neural Network will receive in input the state and will have as many output neurons as the number of actions. In this way given a state it is always possible to compute all the correspondent q-values, provided by the network. The agent will subsequently choose the best action maximising the function or through an epsilon greedy policy.

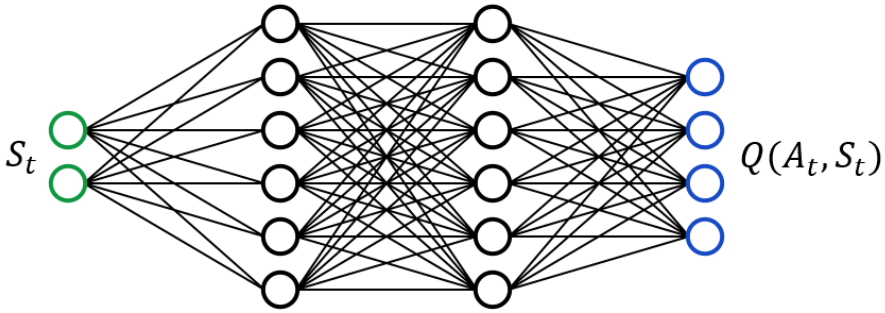


Figure 3: Neural Network Approximator

2.2. Our Model

Once we described the theory behind our model, we can start defining its main components: the state S_t, the set of available actions A_t and the reward R_t.

- We chose as state variables the numerosity of the two populations:

$$\mathbf{S_t} = \begin{bmatrix} X_t \\ Y_t \end{bmatrix}$$

- The actions that the agent can perform in order to modify the state variables consist in varying the parameters of the Lotka-Volterra system of equations in given ranges:

$$\alpha \in [0.5, 1.5] \tag{5}$$
$$\beta \in [0.001, 0.05] \tag{6}$$
$$\delta \in [0.5, 1.5] \tag{7}$$

  To simplify the problem we decided to discretize these intervals, creating grids of equally spaced values. The numerosity of the grid, i.e. the number of available actions, changes based on how many parameters the agent is allowed to vary: if only one parameter is free, then the grid consists in 15 points, while if two parameters are free, then for each parameter there are 5 possible actions, i.e. 5 points on the grid.
  Each time the agent has to choose an action, he selects one of the values on the grids.
- The reward reflects the goal of the agent: keeping the two populations alive. Based on this, we decided to give to each action the following rewards:
  - if one of the two populations becomes extinct, we give a reward of -1000. A population is considered extinct if its numerosity drops below 4 units
  - if the agent manages to keep the two populations alive for 400 time steps of the Lotka-Volterra system of equation, he obtains a reward of +500
  - for the other steps, the reward is equal to -1

  The choice of the reward aims at creating a global maximum for the Q function.

2.3. The algorithm

In this section we describe the steps of our algorithm. The training phase consists in 2000 episodes. Every episode covers an entire simulation of the Lotka-Volterra dynamic, which ends if the extinction of one population is reached or if the goal is accomplished, i.e. if the two populations don't become extinct for at least 200 iterations (equal to 400 time steps of Lotka-Volterra dynamic).

- For each episode:
  - Initialize the state randomly as follows:

$$S_0 = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 100 \\ 10 \end{bmatrix}, \begin{bmatrix} 10 & 0 \\ 0 & 2 \end{bmatrix} \right)$$

  - While the two populations are alive and we performed less than 200 iterations:
    - Feed the state S_t to the Q-network, obtaining the values of the Q-function in correspondence of the state and all the available actions.

– Select the next action A_t through ϵ-greedy policy:

$$A_t = \begin{cases} \textit{random action} & \text{with probability } \epsilon \\ \text{argmax}_{a'} Q(S_t, a') & \text{with probability } 1-\epsilon \end{cases}$$

– Change the values of the parameters according to the selected action and run the Lotka-Volterra system of equations for two time steps to obtain the next state S_{t+1}. Compute the reward associated to this action.
– If one of the two population becomes extinct or if the goal is reached, the episode ends and we go to the next one; otherwise we increment the iteration counter and we go to the next step.
– Use the *update rule* to compute the new value of the Q function in correspondence of S_t and A_t: $Q^{new}(S_t, A_t)$.
– Train the neural network by means of the following loss function:

$$Loss = (Q^{new}(S_t, A_t) - Q(S_t, A_t))^2$$

This loss function quantifies the discrepancy between the expected value of the Q function and its observed value in correspondence of the state action pair S_t, A_t.
– Set $S_t = S_{t+1}$ and go to the next iteration.

This is the main track that we followed while designing the algorithm. However, it is suggested from literature to adopt some adjustments to improve its the performance.
First, instead of always selecting the most recent observation to update our network, we store all of our past observations and sample from this replay buffer whenever we update. This is done to break the correlation between subsequent observations, given the fact that non-linear approximations (such as neural networks) tend to deal poorly with such correlations. This mechanism is called **experience replay**.
Another improvement that we adopted consist in updating the weights of the Q-network after few iterations and not every single observation, that is, we deploy **mini batches**, whose length is equal to 10 observations. Doing so we are able to increase the speed of the algorithm and to improve the policy of the agent by intertwining observing and updating. Moreover, it's often the case that from one single observation the network can't learn that much, thus is better to wait few observations to update the weights.
Particularly combined with experience replay, this is a powerful technique to get stable updates based on a vast pool of previous observations.
Always regarding the training phase, an important parameter that characterises the network is the **learning rate**. We decided to decrease this parameter with a strategy based on the number of iterations reached by the episodes. In particular, after the fifth episode, we compute the mean of the number of iterations of the last five episodes, and if this mean is above a certain threshold (that we set to 180) we decrease the rate by a factor of 0.7. The idea behind is that if the algorithm is able to keep the two populations alive sufficiently long, then it is likely that the network is close to the global maximum of the Q-function, which corresponds with the accomplishment of the desired goal.
Last but not least, we deployed a **target network** in order to compute the observed value of Q that appears in the *update rule*. A target network is a periodic copy of the Q-network. In particular, the copy is performed every 20 observations. This adjustment solves an issue of correlation that otherwise may arise due to the fact that the observed Q-value and the expected Q-value would be computed by the same network.

3. Numerical results

The presented results have been tested on 200 new episodes with stochastic initial conditions equivalent to those used during training. Even if they present different behaviours they achieve the maximum requested simulation length on all episodes, once assured that the starting condition are over the extinction threshold.

3.1. Parameters: α

We started considering only one parameter: α.
The model reaches the goal and it is able to avoid the extinction of both species, however we are not really satisfied with our results, shown in Figure 4. The evolution of the states is too noisy, does not present any periodic pattern and α does not stabilize, but changes almost at every iteration. This is due to the fact that the model learns to overreact, while we want it to find a stable solution.

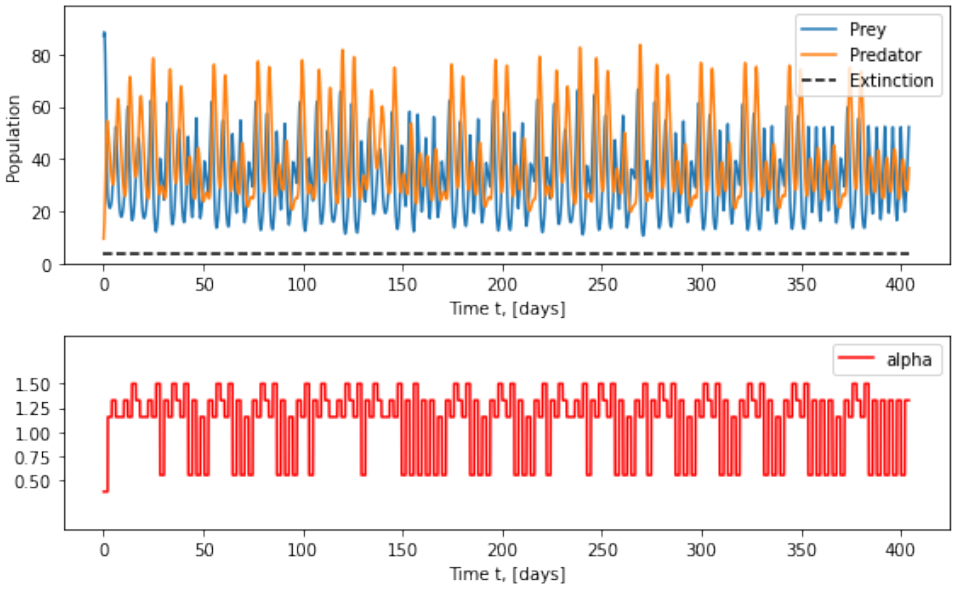


Figure 4: Results with α

To overcome the previous behavior, we modified the reward and added a penalization term:

$$\max\{range(X), range(Y)\} > threshold$$

The model receives the penalization when either the range of the prey or the range of the predators, from the initial time step to the current one, are above a certain threshold.
The new reward is:

- -1000000, if one of the states is below 4
- -200, if maximum range above 100
- -50, if maximum range above 50
- +1, for each step
- +50000, if the goal is reached

Now the results (Figure 5) are good, since the evolution of the states is smooth and alpha stabilizes after a certain time step. This is correct because a stable solution satisfying our constraints does not exist given the initial condition that we provide to the model.

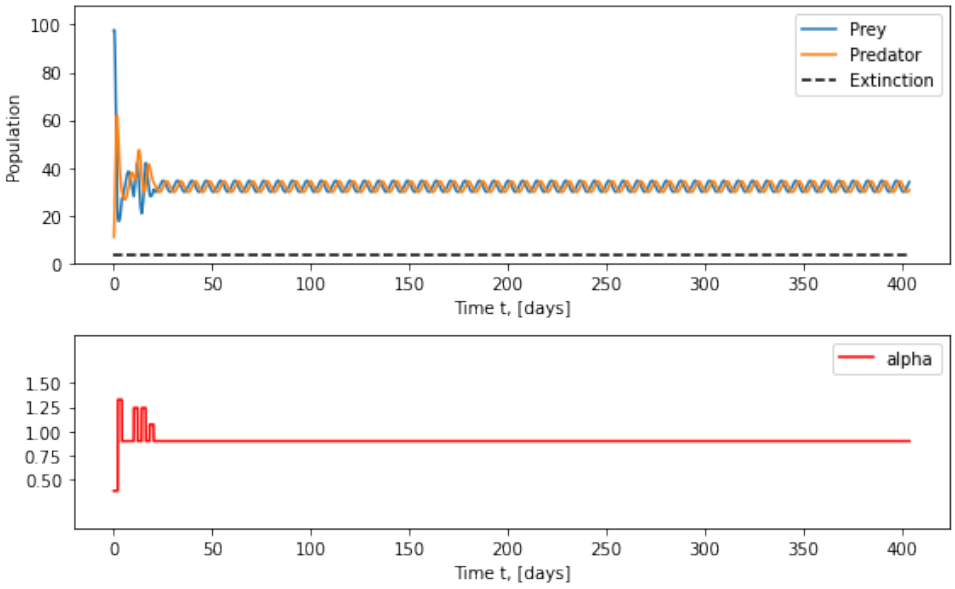


Figure 5: Results with α with modified reward

Another evidence that the model works as expected is given by the fact that the system converges to a stable orbit. (Figure 6)

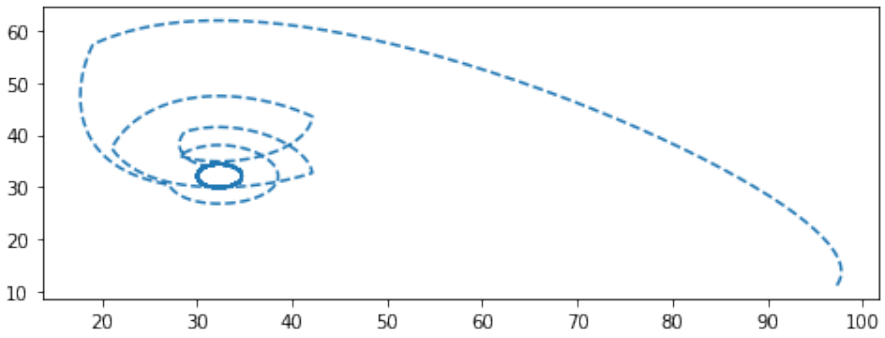


Figure 6: Orbit of the system

3.2. Parameters: α and δ

As next step we increased the complexity of the problem considering two parameters, in particular α and δ. We made this specific choice of parameters because, in case of a real simulation of the problem, they can be controlled and we can intervene to modify them without too much effort, indeed they represent the growth rate of the prey and the death rate of the predators.
The model is able to find a stable pair of parameters after just a few iterations, even without the introduction of the additional penalization term.

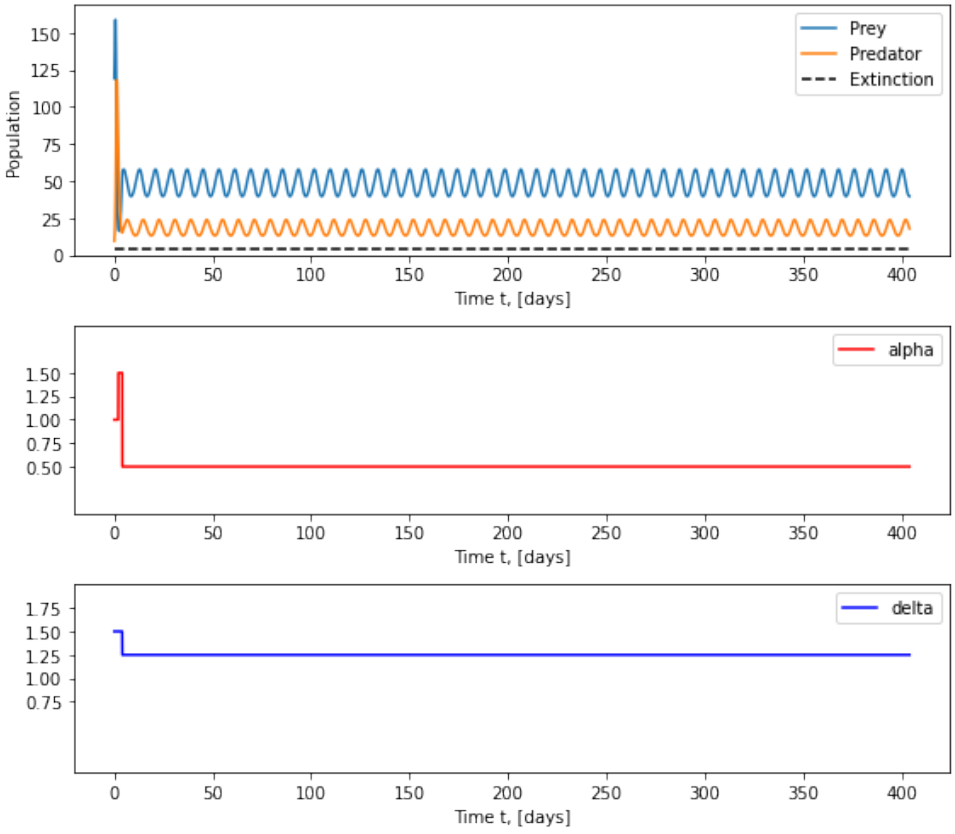


Figure 7: Results with α and δ

In this setting the algorithm benefits the possibility of controlling two parameters, however to make the training more feasible we reduce the number of possible values that each parameter can take, going from 15 to 5.

4. Conclusions

When controlling just one parameter and without adding an additional reward term it is easier for the model to learn an overreacting behaviour rather than a stable one. The efficacy of the result is confirmed on the test set, but the effect is not particularly satisfactory, since the variance of the population is extreme. A *smoothing* reward is thus required.

On the opposite hand, adding parameters widens the possible action space, but at the same time increases the range of available stable orbits, thus simplifying the training procedure.

4.0.1 Possible improvements

Even if controlling more than two parameters is technically possible the impact of this approach would probably of limited use in the case of the Lotka-Volterra problem, when it is much easier to control the growth and death rate of a population rather than the interaction terms.
On the other hand a first useful improvement would include an increase in the number of possible actions, in order to create a more flexible controlling system. This is most likely possible by increasing the dimension of the Neural Network and by designing a more computationally demanding training procedure. The extreme case limit of a continuous alpha is achievable, but would involve a different construction of the model.
Lastly, the training procedure can be easily modified to consider a higher variance on the initial conditions. This modification would surely lead to longer training procedures but to a more representative model as well.

References

[1] Chang Sun et al. Deep reinforcement learning for optical systems: A case study of mode-locked lasers. *Mach. Learn.: Sci. Technol.*, 2020.

[2] Swagat Kumar. Balancing a CartPole System with Reinforcement Learning, 2020.

[3] Wouter van Heeswijk. A Minimal Working Example for Deep Q-Learning in TensorFlow 2.0, 2021.