Department of Physics & Astronomy

Heidelberg University

Bachelor Thesis in Physics

submitted by

Fabian Krautgasser

born in Salzburg

2018

Of Hunters and Gatherers –

a Deep Reinforcement Learning Approach

to Population Dynamics

This Bachelor Thesis has been carried out by

Fabian Krautgasser

at the Department of Physics & Astronomy
Institute of Environmental Physics (IUP)

Reviewer:	Prof. Kurt Roth
Second reviewer:	Prof. Fred Hamprecht

30th of June 2018

Of Hunters and Gatherers –
a Deep Reinforcement Learning Approach to Population Dynamics

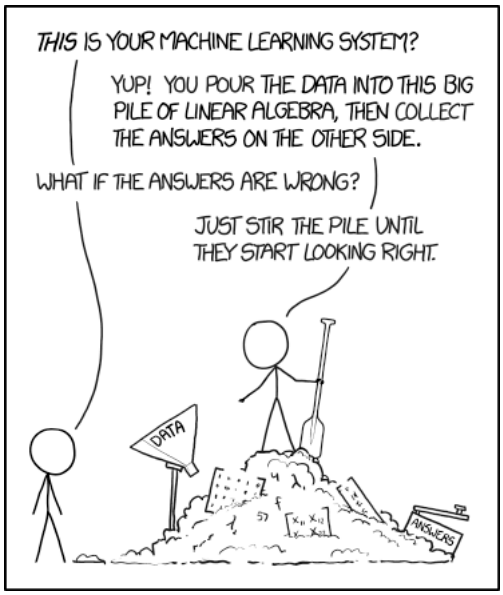
Stochastic and deterministic formulations to model the dynamics of a population have been around for decades. The stochastic models utilize probabilities to describe the interactions within a population, although often without considering the adaptability of the individuals. Adaption can incorporate everything from evolution, e.g. in the form of individuals developing traits that help them survive, to learning survival strategies, e.g. in the form of effective hunting.

This thesis is an approach to implement the latter in conjunction with a known population dynamics model, using deep convolutional neural networks as policy for decision strategies. Two experiments are deducted, each with a different set of interactions between the individuals. This thesis discusses the different strategy outcomes, in terms of the interactions and model restrictions.

Von Jägern und Sammlern –
ein Deep Reinforcement Learning Ansatz für Populationsdynamiken

Stochastische und deterministische Formulierungen, um die Dynamik einer Population zu modellieren, gibt es seit Jahrzehnten. Die stochastischen Modelle nutzen Wahrscheinlichkeiten um die Interaktionen innerhalb einer Population zu beschreiben, allerdings oft ohne auf die Anpassungsfähigkeit der Individuen einzugehen. Anpassung kann von Evolution, z.B. Entwicklung von Merkmalen, die die Überlebenschance erhöhen, bis zum Erlernen von Überlebensstrategien, z.B. effizientes Jagen, Vieles beinhalten.

Diese Arbeit ist ein Ansatz, Letzteres im Zusammenspiel mit einem bekannten Populationsdynamikmodell zu implementieren, wobei ein Deep Convolutional Neural Network als Methode für Entscheidungsstrategien benutzt wird. Zwei Experimente sind durchgeführt, jedes mit einem anderen Satz an Interaktionen zwischen den Individuen. Diese Arbeit erörtert die unterschiedlichen Ergebnisse der Strategien, in Abhängigkeit der Interaktionen und Modelleinschränkungen.



©XKCD, *Munroe* [2017]

Table of Contents

1

Introduction

Spontaneous decisions depend almost exclusively on their very moment. Past experiences may have allowed for a particular decision, but these influences pull the strings mostly unnoticed and subconscious. Random choices may lead to a desirable outcome, although often enough they worsen the situation. So purely random behaviour is dissatisfactory; but given a rough set of rules to obey, random decision making may be welcome, e.g. in a cellular automaton to model the dynamics of a population of microorganisms [see *Weber* 2015].

While restricted randomness may be applied to models of some lifeforms in this manner, their actual rules may be more complex than our model's capabilities. In other words, a lifeform's decision to move in a specific direction may be triggered by some notion of sensory input, e.g. changing light conditions or sensing imminent danger, or by an internal process, e.g. the need to search for food or a potential mating partner. This input to the lifeform's decision process influences present and potentially future behaviour. Adapting to an ever changing environment seemed to be beneficial for survival, as *Darwin and Wallace* [1858] postulated in their work *On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection.*

A seeing-eye-dog, who learned to guide a blind person throughout the day, was trained for that specific purpose. Its decision to act according to a set of rules was not inherited, but the result of a learning process. Trying to understand and model such a learning process is a topic of current research. A playful example may be the work of *Mnih* et al. [2013], who trained a convolutional neural network to play Atari. In this specific work, a set of possible decisions was given, but the neural network takes control of the decision process.

In this thesis we will focus on exploring an ansatz for modeling behaviour development in a complex system.

Generally speaking, the dynamics of a population may be seen as a game of survival. The "good" players will outlive the "bad" players with a high probability. In this case, a good or bad player is characterised by the ability to make adequate or poor choices in a crucial situation, e.g. a rabbit not fleeing from an approaching fox most probably made a poor decision. In this thesis, we will explore a possible way to model a survival game and its players, more generally called agents, as well as the differences to known models building on restricted random behaviour.

The aforementioned comic raises one of the issues in machine learning, especially neural networks: although it is indeed a rather technical field of study, a good intuition for choosing parameters may lead to better results, because the optimal solution can only be approximated. For a class of problems previously unexplored even a machine learning approach would mean trial-and-error at first.

2

Theoretical Background

In this chapter we will lay the foundation of this thesis. First, we will walk through the concepts of population dynamics and predator-prey interactions (section 2.1), hike across the fields of decision making (section 2.2) and finally climb the hills of reinforcement learning (section 2.3). We will only consider the subtopics necessary for this thesis, with occasional outlooks into neighbouring regions.

2.1. Population Dynamics

Before we can look deeper into population dynamics, let us first define what a population is:

> A *population* is a large set of interacting individuals, where an *individual* is an "atomic" unit, i.e., a clearly distinguishable entity that has an existence of its own.
> — *Roth* [2017, p. 335]

For instance, every animal species with the single specimens as individuals make up such a population. Their dynamics, however, may depend on various influences: changes in their environment caused by natural catastrophes (or humankind), an invasive second species claiming its new territory or just the intraspecies competition over a common resource.

With its origin rooted in mathematical biology more than 200 years ago, the branch of population dynamics aggregated a rather substantial scientific body since. Its initially most influential contributor was *Malthus* [1798] with his *Essay on the Principle of Population*, where he introduced the law of exponential growth. About 40 years later, *Verhulst* [1838] published his work on the logistic model for limited population sizes. The building blocks of modern multi-species interaction models were published by *Volterra* [1926], without knowledge on the previously done work of *Lotka* [1920].

We will focus on a subset of the population dynamics' territory, centered around predator-prey interactions and subsequent dynamics.

The overall inspiration was drawn from *Roth* [2017, chapter 9], who followed the work of *Weber* [2015].

2.1.1. Predator-Prey-Model

In this section we will use certain nomenclature (states s, actions a) similar to the vocabulary used in reinforcement learning. The relevant terminology will be treated in more detail in section 2.2.

Hierarchy The most fundamental predator-prey-models (PPMs) define a *hierarchy* for possible interspecies interactions (Figure 2.1) with R being a resource for species 1 to feed on, and species 1 solely sustains species 2. In detail we have:

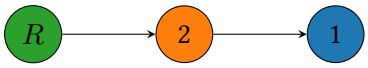


Figure 2.1. Concept of an s-process for 2 species and a resource. Such a process could be a grassy landscape, providing nurturing for rabbits, which in turn act as prey for foxes.

(i) An abundant resource R, that (depending on the model) may temporally run out or persist and which incorporates the peculiarities of the environment.

(ii) A species that feeds on this resource (species 2), which we will call preys.

(iii) Another species, only feeding on preys, which we will call predators.

(iv) All intra- and interspecies interactions are considered to be stochastic processes with constant probabilities.

Model foundation Sticking to the species-wise enumeration, a dynamical two-species predator-prey system may be written in the general form:

$$\begin{aligned} \dot{n}_1 &= n_1 f_1(n_1, n_2) \\ \dot{n}_2 &= n_2 f_2(n_1, n_2) \end{aligned} \tag{2.1}$$

with $f_{\{1,2\}}$ being effective growth rates (comprised of probabilities for dying, procreating, finding food, etc.) and $n_{\{1,2\}}$ the populations size in numbers of individuals.

Every (natural) environment holds a specific carrying capacity, i.e. the environments ability to sustain N individuals in total (space- or food-wise). This allows for f to be considered a population density, meaning the fraction of a populations size n with the environments carrying capacity. Further assuming a constant growth rate $\mu > 0$ for preys, meaning unlimited resources R, and a constant death rate $\nu > 0$ for predators, we can assemble the classical *Lotka-Volterra model*:

$$\begin{aligned} \dot{n}_1 &= n_1 \left[a_{12} n_2 - \nu \right] \\ \dot{n}_2 &= n_2 \left[\mu - a_{21} n_1 \right] \end{aligned} \tag{2.2}$$

where $a_{12}n_2$ and $a_{21}n_1$ are the interaction terms[1] between the species. Note, that in this scenario preys can only die by falling prey to a predator. This system has two fixed points, with the trivial one being $(0, 0)$ and the second one

$$(n_1^*, n_2^*) = \left(\frac{\mu}{a_{21}}, \frac{\nu}{a_{12}} \right). \tag{2.3}$$

Applying this knowledge from the second fixed point, we can reformulate (2.2) to an even simpler form

$$\begin{aligned} \dot{u}_1 &= u_1 \left[u_2 - 1 \right] \\ \dot{u}_2 &= \gamma u_2 \left[1 - u_2 \right] \ , \end{aligned} \tag{2.4}$$

[1] Interaction terms describe the interspecies dependence on each others population size. a_{ij} are scalar factors, specifying the strength and sign of the dependence.

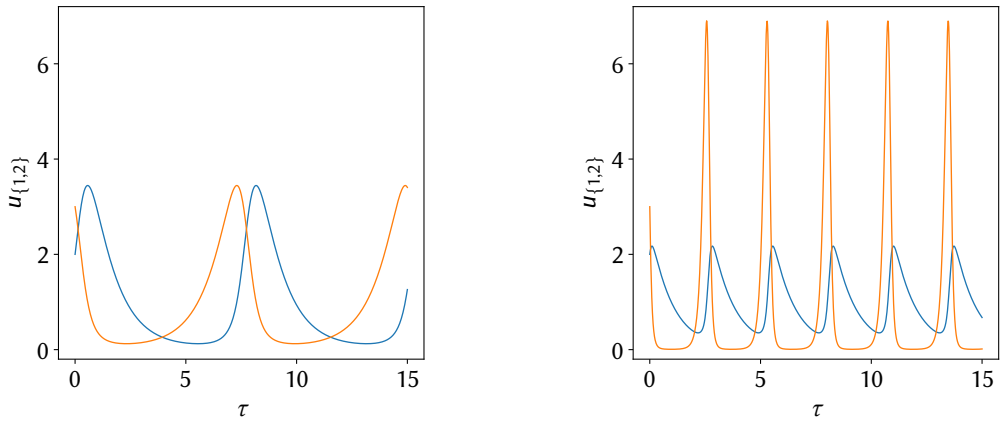


Figure 2.2. Dimensionless trajectories of the classical Lotka-Volterra model (2.4) for $\gamma = 1$ (left) and $\gamma = 10$ (right). Initial value for the predator (blue) is 2, and 3 for the prey (orange).

with the dimensionless variables

$$u_{\{1,2\}} := \frac{n_{\{1,2\}}}{n^*_{\{1,2\}}} \qquad \tau := \nu t \,, \qquad \gamma := \frac{\mu}{\nu} \,. \tag{2.5}$$

Trajectories of $u_{\{1,2\}}$ capture some of the systems characteristics (Figure 2.2); the decrease of the prey population as the predator population increases continues only up to a certain point, with a subsequent decrease in the predator population and a recovery phase for both species. In case of higher values of γ we can see the larger ratio of prey growth rate μ to predator death rate ν as a more rapid growth of prey populations.

Limited resources In (2.4) we considered the resource R to be unlimited, i.e. the only cause of death for preys was a predator and thus only the predators are controlling the preys growth. Introducing a limiting environment creates a new playground with different dynamics. The system is now no longer conservative and the fixed point of both species coexisting becomes attractive. Counterintuitively, the newly emerged population is more stable, than in an unlimited environment (Figure 2.3).

To incorporate the limiting behaviour in our formula, we introduce an effective growth rate μ^{eff} instead of a constant growth rate μ:

$$\mu^{\text{eff}} = \mu \left[1 - \frac{n_2 a_{22}}{q}\right]$$

with q/a_{22} being the specific carrying capacity for species 2. Inserting the new effective growth rate into (2.2) and identifying the fixed points, we find: (i) once again a trivial fixed points, where both species vanish, (ii) a vanished predator population with the preys population grown to the environments carrying capacity and (iii) the stable coexistence of both species:

$$(n^*_1, n^*_2) = \left(\frac{\mu}{a_{21}}[1-\beta], \frac{\nu}{a_{12}}\right) \,, \tag{2.6}$$

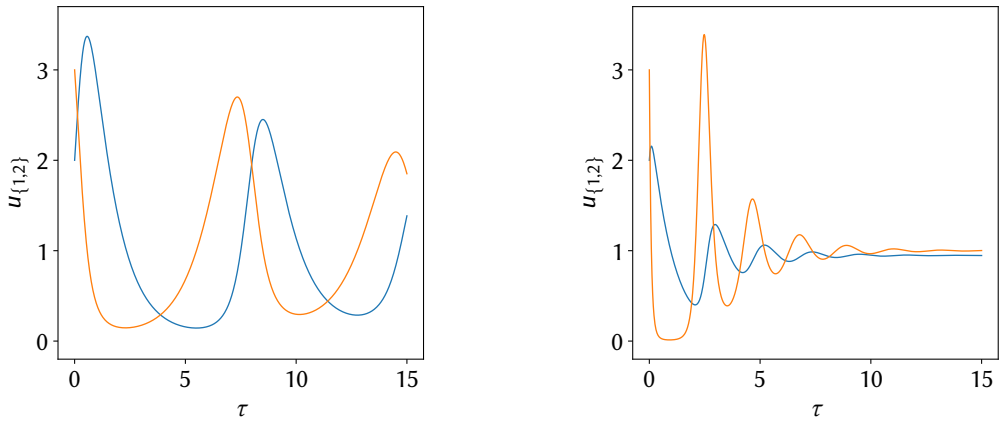


Figure 2.3. Dimensionless trajectories of the Lotka-Volterra model with limited environment (2.7) for $\gamma = 1$ (left) and $\gamma = 10$ (right) and $\beta = 0.05$ in both cases. Initial value for the predator (blue) is again 2, and 3 for the prey (orange).

with an additional new variable $\beta := \frac{\nu}{a_{12}} \frac{a_{22}}{q}$. In this context, $1 - \beta$ can be understood as the *predation pressure* on the prey.

Defining the same dimensionless variables as in (2.5), using the fixed point-solution for coexistence, we reformulate (2.4) to

$$\begin{aligned} \dot{u}_1 &= u_1 \left[u_2 - 1\right] \\ \dot{u}_2 &= \gamma u_2 \left[1 - u_1 - \beta[u_2 - u_1]\right] \ . \end{aligned} \tag{2.7}$$

Note the difference from Figure 2.2 to Figure 2.3; trajectories in the latter converge to a stable point, where both species coexist. For a more thorough derivation and discussion of the fixed points' stability properties please refer to [*Roth* 2017, pp. 356–361].

Predator-Prey Model Conceptually, we set the stage for the model from a theoretical point of view. To consolidate the theoretical background we implement the atomic interactions and ascertain whether the macroscopic behaviour can be formulated with the Lotka-Volterra model. For this, we need to consider some vital details.

Firstly, our environment of choice is a regular (not necessarily square) grid, where each cell can only carry one specimen of any species. With this, we have set the limitation of the environment to its geometrical extent (e.g. for a 10×10 grid we would have at most 10^2 individuals in the environment). To circumvent unwanted behaviour near the borders of the grid, we consider its shape to be toroidal, i.e. an individual trying to leave the grid on the left/top side, will appear again (in the same row/column) on the right/bottom side and vice versa.

Secondly, each individual carries its own set of internal states. These internal states may describe, for example, the individuals ability to survive and procreate (food reserve f_R and maximum food reserve $f_R^{\max}$), its orientation on the grid (orientation o, see section 3.4) or its chance to flee from a predator (p_{flee}). More details for the model will be investigated throughout this thesis.

Thirdly, if we were to equip this setting with a complete set of rules for all combinations of possible configurations of individuals within the environment, we would have defined a cellular

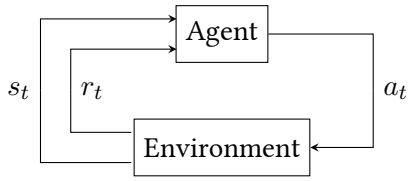


Figure 2.4. The reinforcement learning problem. The agent observes state s_t from the environment, then chooses a_t (following its policy π_t) and receives r_t in return. Modified from *Sutton and Barto* [1998, p. 52].

automaton (CA)[2]. With the goal of letting our individuals become self-deciding agents, a more agent-based approach is beneficial. Thus, we only define the rules governing the model in a probabilistic way; a given state s of the environment may only lead with a certain probability to a subsequent state s', i.e. a predator chooses a certain prey and tries to eat it with $1 - p_{\text{flee}}$. We will discuss state transitions and agent decisions more formally in section 2.2.

2.2. Decision Making

2.2.1. Definitions

After our first encounter with the reinforcement learning nomenclature, we formally define what we actually mean by these terms. Let it be noted, that we assume discretized timesteps t for all cases. We will mostly follow the notation of *Sutton and Barto* [1998], adjusted to our specific case.

State A state s_t is some representation of the environment at timestep t, with $\mathcal{S}$ being the set of all possible states at all times. Sometimes we conveniently may only consider a state s and its subsequent state s' (instead of s_t and s_{t+1}).

Agent and environment An agent in our context is an entity that learns how to make decisions, while it's interacting with the environment (Figure 2.4). The decision process depends on the agent's policy π (see next paragraph) and may depend on the agent's internal state. The environment is everything that provides states s for the agent to observe, takes the agent's action $a \in \mathcal{A}$ (see next paragraph) and returns a reward $r \in \mathcal{R}$, the set of all possible rewards. The border between agent and environment can be understood as an abstraction layer; the agent is just the deciding organ, whereas e.g. sensorial input, mechanical joints etc. belong to the environment.

Action and policy To be a bit more flexible in our notation, we set $\mathcal{A}$ as the set of all possible actions, and $\mathcal{A}(s_t)$ as the set of all possible actions in state s_t. The agent's decision is governed by the policy[3] π, which is a mapping:

$$\pi \colon \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1], \quad (s, a) \mapsto \pi(s, a) \,, \tag{2.8}$$

with $\pi(s, a)$ being the probability of taking action a when in state s. In the literature (e.g. [*Kochenderfer* et al. 2015]) one may also define the action a as the policies output to state s, i.e. $a \coloneqq \pi(s)$.

[2] There still seems to be an ongoing debate on what actually defines a CA. For instance, *Adamatzky* [2010] defines a CA as an infinitely extended grid, with some cells called "alive". At every discrete time unit, called *generation*, a fixed rule is applied to the ensemble of "alive" cells, to arrive at the configuration for the next generation.

[3] The policy changes during the learning process, so we may refer to it by designating the current timestep: π_t.

Reward The environments reaction to an agent's action a at timestep t is the immediate reward r_t (for the rewards I followed the notation of *Kochenderfer* et al. [2015], because it describes the problem in a more natural way[4]). The rewards are vital for the whole reinforcement learning machinery, because they steer the agent's decisions, as it gradually improves in its task.

As a general line of thought, we can think of the rewards as feedback for our actions, i.e. a positive feedback if we do something (clearly defined as) good, and a negative feedback/ penalty, if we act in a wrong way.

2.2.2. Selecting actions

So far, we just used the policy as tool, but did not discuss any of its possible internals. How should an agent "know", when to choose a specific action? One of the easiest methods is the *ϵ-greedy method*. But first, we need to think about what we expect from a selected action.

Let us for now, completely unbiased, call the *true* value of the outcome of an action $Q^*(a)$, and the *estimated* value of the outcome of an action $Q(a)$ for any action a. By true, we mean the actual value we are going to receive, and by estimated, we mean the value we assume we will receive. A natural way to estimate $Q(a)$ is to just average every reward of action a we received so far, i.e.:

$$Q(a) = \frac{1}{n_a} \sum_{i=1}^{n_a} r_i \ ,$$

where n_a is the number of times, action a has been chosen prior to this point. Of course, we have to set an initial value, e.g. $Q_0(a) = 0$, if a has never been chosen. As n_a grows towards ∞ and with the law of large numbers, $Q(a) \to Q^*(a)$.

Now that we have an idea of true and estimated value of an action, we establish the greedy-method. Formally we have:

$$a^* = \arg\max_{a \in \mathcal{A}} Q(a) \ , \tag{2.9}$$

where a^* is the greedy action, and $\mathcal{A}$ is the set of all possible actions. The greedy action is one of the actions, which maximize the estimated action value $Q(a)$. An agent utilising this method means it is exploiting its current knowledge (of Q) to maximize the immediate reward. In other words, all momentary inferior actions will not be considered at all, even if they might turn out superior in the long run. We introduce a technique, where an agent acts greedy most of the time, but considers other actions every now and then, circumventing the "myopic" behaviour of the agent. This presents us the ϵ-greedy method, with ϵ being the small probability for an agent not to act greedy. Assuming that we have a large number of trials, the number of not greedy actions will also be very large, and for the number of trials growing infinitely, the estimated action values converge to the true values, $Q(a) \to Q^*(a)$.

One of the downsides of ϵ-greedy is, that it chooses uniformly between all non-greedy actions; so the worst action is equally probable as the second best action. We can weight the probability to choose an action a_j with its action value $Q(a_j)$:

$$p(a_j) = \frac{\exp\left[Q(a_j)/\tau\right]}{\sum_{i=1}^{n} \exp\left[Q(a_i)/\tau\right]} \ , \tag{2.10}$$

[4] As *Sutton and Barto* [1998] defined it, reward r_{t+1} is received as return for action a_t, conjoined with the next state s_{t+1}. While that may be the case for many reinforcement learning problems, an immediate reward seemed more feasible in my multi-agent environment.

where $n = \#\mathcal{A}$ and $\tau > 0$, a parameter called the temperature[5]. The above expression is called *softmax* action selection; it takes the action values at a given point and squeezes their values into the range of $[0, 1]$ where the sum over all squeezed values results in 1. These probabilities can then be used to represent a categorical distribution, with a defined sampling method to directly extract an action.

In this thesis, the softmax action selection with $\tau = 1$ was used (section 3.2).

2.2.3. System dynamics

Up to now, we can define what an agent should be capable of in theory, and how certain parts of the process are defined, but we still don't know anything about the workings behind the whole reinforcement learning machinery. So in the following, we will define the *Markov property*, *Markov decision processes (MPD)* and their special cases for our problem, as well as the system dynamics in form of *transition probabilities* and *immediate rewards*.

We assumed discrete timesteps earlier; in this section we even assume a finite number of states and reward values. This is a rather convenient way to be able to work with sums and probabilities, rather than integrals and probability densities.

Markov property Think of the response of any arbitrary environment to an action a_t at timestep t. In the most general cases, the environment's history plays a role in how the environment responds to a_t. To understand the dynamics, we would have to specify and construct the complete (joint) probability

$$P\left(s_{t+1} = s', r_t = r \;\middle|\; s_t, a_t, r_{t-1}, s_{t-1}, a_{t-1}, \ldots, r_0, s_0, a_0\right)$$

for all $s',\ r$ and all possible combinations of past events. Think of a human's (clearly defined) emotional reaction to a situation he or she is in: everything this human being has experienced in his or her life influenced his or her reaction in that situation.

We say a state s_{t+1} has the *Markov property*, if it only depends on the state and action of the previous timestep (same goes for the reward). This simplifies our (joint) probability drastically; we can now describe the systems dynamics only by specifying

$$P\left(s_{t+1} = s', r_t = r \;\middle|\; s_t, a_t\right) \tag{2.11}$$

for all $s',\ r$ and preceding events s_t, a_t. If (2.11) is true for all $s',\ r$ and their histories, then the environment and the task are said to have the Markov property. This property will turn out to be crucial in our further discussion about reinforcement learning, because decisions and values are assumed to be functions only of the current state.

Markov decision process If our reinforcement learning task satisfies the Markov property as mentioned above, it is called a *Markov decision process (MDP)*. If additionally the state and action spaces $\mathcal{S},\ \mathcal{A}$ are finite (i.e. $\#\mathcal{A} < \infty$), then it's a *finite Markov decision process (finite MDP)*.

[5] High values of τ cause the probabilities to be almost equally probable, whereas low values of τ turn out in greater differences in the resulting probability, for actions that differ in their action values. The name *temperature* stems from the analogy to Maxwell-Boltzmann statistics as used in statistical mechanics.

For given (and finite) state and action spaces, we still need to prescribe the one-step dynamics of the environment, to define our finite MPD. We call

$$\mathcal{P}^a_{ss'} = P\left(s_{t+1} = s' \,\middle|\, s_t = s, a_t = a\right) \tag{2.12}$$

the *transition probability*[6] for a given state s and action a to proceed to s'. Analogously, the expected *immediate reward* for any given pair of state and action is defined via

$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_t \,\middle|\, s_t = s, a_t = a\right] \ . \tag{2.13}$$

So the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$ almost[7] completely specifies the dynamics of a finite MDP.

Linking our line of thought back to section 2.1 we can identify the transition probabilities as the probability to procreate, p_{breed}, and the probability for a predator to successfully eat a prey, $1 - p_{\text{flee}}$. Without the notion of rewards as a measure for the performance of the agents, the expected reward $\mathcal{R}$ may also be the increase in food reserve after consuming a resource. We will discuss rewards in detail in the next section.

If the problem turns out to be surprisingly simple, i.e. the state and action spaces are tiny, one could set up a look-up table containing $\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$ $\forall\ s, s', a$. For more complex scenarios, e.g. an agent learning to move in 4 directions on a 10×10 grid would have $4^{100} \approx 10^{60}$ entries in such a table. Thus, such a table is not feasible for larger state and action spaces.

2.3. Reinforcement Learning

Venturing further towards the foot of the hill we want to scale, we will extend the formalism so far; value and state functions, V^π and Q^π, respectively and Bellmann equations mark the entrance into steeper terrain. Along the way to the top we will come across partially observable Markov decision processes and hints at the solution methods for the reinforcement learning problem so far. And finally, an approximative method and neural networks at the culmination.

2.3.1. Extended formalism

A function that is fundamental to almost all reinforcement learning algorithms is the state-dependent (or state-action pair dependent) *value function* V^π. The value function is a measure for "how good" it is for an agent to be in the given state (or how good it is to choose a certain action in the given state). Clearly, the "goodness" needs to be defined more formally. We will define it in terms of expected *return*.

Return and discount The quantity a reinforcement learning agent tries to maximize (at a given timestep t) is the *return* R, or to be precise: the expected return $\mathbb{E}[R]$. The return itself is often defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k} \ , \tag{2.14}$$

where γ is the so called *discount factor*, $\gamma \in [0, 1]$, and r_{t+k} are the rewards received at timesteps $t + k$. This definition also holds for infinite horizon reinforcement learning tasks, i.e. for $T \to \infty$, but only if $\gamma < 1$; otherwise, the return would grow infinitely. The discount factor governs the

[6] Transition probabilities are sometimes denoted with $\mathcal{T}^a_{ss'}$.

[7] We only lose information about the distribution of rewards around the expected value.

present value of future rewards: a reward received k timesteps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. Further on, we will denote the horizon with T, but with the possibility for $T \to \infty$.

State-value and action-value Recall the definition (2.8) of a policy π. We define the *state-value function* V^π as

$$V^\pi(s) \coloneqq \mathbb{E}_\pi \left[R_t \,\middle|\, s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \,\middle|\, s_t = s \right] . \tag{2.15}$$

We can interpret this as the expected return, given that the agent is (or starts) in state s at timestep t and follows the policy π thereafter. If we let the agent decide to take an action a_t, we define the *action-value* Q^π as

$$Q^\pi(s,a) \coloneqq \mathbb{E}_\pi \left[R_t \,\middle|\, s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a \right] . \tag{2.16}$$

Similar to the state-value, we can interpret this as the expected return, given s_t *and* a_t and following the policy π thereafter.

Both value functions can be estimated from experience; if an agent keeps track of every return R that followed a state s, then the long term average will converge to V^π. If the agent additionally keeps track for each action separately, then the different sets will converge to Q^π [*Sutton and Barto* 1998]. As we can imagine, these tables of values grow large for $T \to \infty$ and even faster if we consider multiple agents. A feasible solution is to keep an approximation of V^π and Q^π as parameterized functions, adjusting the parameters gradually to fit the observed returns. We will discuss this in greater detail in section 2.3.2.

Bellmann equation for V^π We defined the Markov property in section 2.2. With that, we introduce a consistency condition that holds for a state s and its successor s', the *Bellmann equation for* V^π. Starting with the definition of the state-value function (2.15), we find

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi \left[R_t \,\middle|\, s_t = s \right] \\
&= \mathbb{E}_\pi \left[r_t + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \\
&= \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,\middle|\, s_{t+1} = s' \right] \right] \\
&= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right] .
\end{aligned} \tag{2.17}$$

Before we go any further, let us dissect the steps. (i) We insert the definition of R_t and split off the first summand, followed by an index shift. (ii) The additive property of the expected value allows us to rephrase r_t as $\mathcal{R}^a_{ss'}$, but we then need to consider all actions and following states with r_t in between. And by considering we mean average over all possibilities, weighting each by its probability of occuring. (iii) Lastly, we identify the expected value on the right side to be the state-value of the subsequent state s'.

To summarize, the Bellman equation expresses the relationship between the value of a state s and its successor s'.

Optimality The existence of policies raises the question whether there exists an *optimal* policy, i.e. a policy that achieves the biggest return in the long run, for a given reinforcement learning problem. In case of MDPs we can exactly define such an optimal policy. For this, we define an ordering for two different policies π, π':

$$\pi \leq \pi' \text{ , if and only if } V^{\pi}(s) \leq V^{\pi'}(s) \; \forall s \in \mathcal{S} \; .$$

This ordering together with the finite state space implies, that there is always a policy that is better than or equal to all other policies. We will call such a policy an *optimal policy* π^*. If more than one optimal policy exists, we can consider the set of all optimal policies and pick one to our liking. Since all are optimal, they share the same *optimal state-value function* V^{π^*} and *optimal action-value function* Q^{π^*}, defined by

$$V^{\pi^*}(s) \coloneqq \max_{\pi} V^{\pi}(s) \; \forall s \in \mathcal{S} \; , \tag{2.18}$$

and

$$Q^{\pi^*}(s,a) \coloneqq \max_{\pi} Q^{\pi}(s,a) \; \forall s \in \mathcal{S} \; , \forall a \in \mathcal{A}(s) \; , \tag{2.19}$$

respectively.

The formal difference between V^{π} and Q^{π} is that for Q^{π} we choose an action a and follow π subsequently. This implies, that we receive a certain reward r_t for our decision of a_t. Using this thought and the second line in (2.17), we can formulate Q^{π} in terms of V^{π} and in further consequence Q^{π^*} in terms of V^{π^*}:

$$Q^{\pi^*}(s,a) = \mathbb{E}_{\pi^*} \left[r_t + \gamma V^{\pi^*}(s_{t+1}) \;\middle|\; s_t = s, a_t = a \right] \; . \tag{2.20}$$

Since the policy π^* is an optimal policy, we do not need to refer to any specific policy, and can thus drop the π^*-index of the expected value.

Let us consider (2.17), this time for the optimal state-value function V^{π^*}:

$$\begin{aligned}
V^{\pi^*}(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_{a} \; \mathbb{E}_{\pi^*} \left[R_t \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_{a} \; \mathbb{E}_{\pi^*} \left[r_t + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_{a} \; \mathbb{E} \left[r_t + \gamma V^{\pi^*}(s_{t+1}) \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_{a} \; \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi^*}(s') \right] \; .
\end{aligned} \tag{2.21}$$

Again, we dissect the steps. Initially, we take a step backwards from the action-value function; applying the max-operator over all possible actions, we get the optimal state-value. (i)-(iii) We follow the steps in unison with (2.20), where we too do not need the reference for any specific policy (in the expectancy) anymore. (iv) Same procedure as in the last step of (2.17), although we take the maximum here instead of averaging over all actions. Inserting this result in (2.20), we

obtain

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^{\pi^*}(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a\right]$$
$$= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^{\pi^*}(s', a')\right] . \tag{2.22}$$

Equations (2.21) and (2.22) are called *Bellman optimality equations*. For finite MDPs, (2.21) has a unique solution indepenent of the policy. If the problem has N states, then the bellmann optimality equation actually is a system of N equations in N unknowns. If we knew the dynamics of the problem in the form of $(\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$, we would in theory be able to solve for V^{π^*} [*Sutton and Barto* 1998].

Partially observable MDP The knowledge we built up so far assumes that an agent always has a perfect observation o of the environment, i.e. that the state the environment is in and the agent's observation match. This is no longer the case, if we consider e.g. a small 5×5 grid-like environment, and an agent's field of view that just covers its 5- or 9-neighbourhood (Figure 3.1). With the previous settings, we then have an MDP with an observation model. In the following we will use inspiration from *Kochenderfer* et al. [2015] and *Cassandra* [1994].

Here, $\mathcal{O}(o \mid s)$ is the probability to observe o given state s, and Ω the set of all observations, $\Omega = \{o_i\}_i$. Alternatively, we can also define $\mathcal{O}(o \mid s, a)$ as the probability to observe o given state s and last-selected action a. $\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$ define the transition probabilities and immediate rewards, respectively. Thus, our partially observable Markov decision process (POMDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \mathcal{O}, \Omega)$, for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$.

The decision an agent would make at timestep t depend on the past observations, at least the last one. But instead of keeping arbitrarily long histories of observations, a common approach is to define *belief states* b. A belief state b is a distribution over the states s, where $b(s)$ is the probability of being in state s given belief state b. Thus, the policy mapping changes from (2.8) to

$$\pi \colon \mathcal{B} \times \mathcal{S} \longrightarrow [0,1], \quad (b,a) \mapsto \pi(b,a) , \tag{2.23}$$

where $\mathcal{B}$ is the set of all belief states. If $S = \#\mathcal{S},\ S < \infty$, then $\mathcal{B} \subset \mathbb{R}^S$. In this context now, we can see that a POMDP actually is a MDP in which the states are belief states, called a *belief state MDP*.

How do the belief states transit into each other? Clearly, we need something other than $\mathcal{P}^a_{ss'}$ to describe the believed states; let us denote the transition probability for $b \xrightarrow{a} b'$ with $\mathcal{T}^a_{bb'}$. We find[8] $\mathcal{T}^a_{bb'}$ to be:

$$\mathcal{T}^a_{bb'} = \sum_o P(b' \mid b, a, o) \sum_{s'} \mathcal{O}(o \mid s') \sum_s \mathcal{P}^a_{ss'} b(s) . \tag{2.24}$$

The last sum depicts the expected transition $s \to s'$ given the belief of being in state s. The middle sum is the expected observation of o, given state s'. The first sum is the expectancy of transitioning from belief state $b \to b'$, given observation o and executed action a. This transition though is tricky; given an initial belief state, say b_0, we can update the current belief state using recursive Bayesian estimation based on the last pair of action and observation (a, o). And as *Kochenderfer* et al. put it: "The update can be done exactly for problems with discrete states and for problems with linear-Gaussian dynamics and observations. For general problems with continous spaces, we often

[8] For a thorough derivation, please refer to [*Kochenderfer* et al. 2015, p. 136].

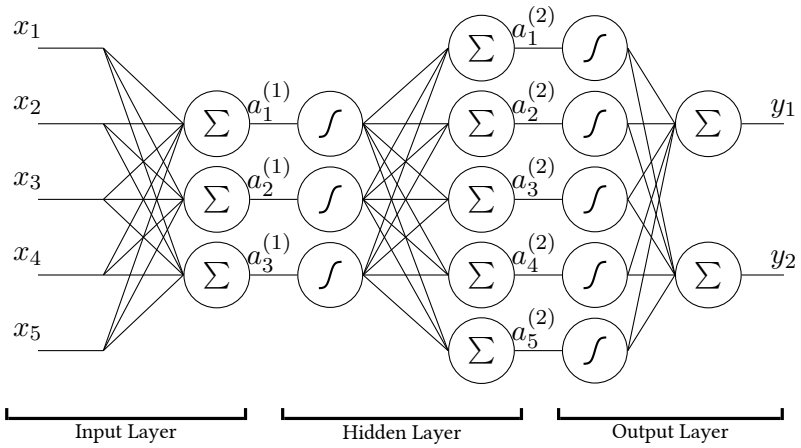


Figure 2.5. Schematic depiction of a multilayer perceptron (without biases). $\{x_i\}_i$ are entries of the input vector $\mathbf{x}$, $\{y_j\}_j$ are entries of the output vector $\mathbf{y}$. Lines leading to a summation joint Σ are weighted. These connections are summed and result in $a_k^{(\ell)}$, where k is the summation joint index and (ℓ) is the layer number. The nodes with a sigmoid icon signify a non-constant activation function.

have to rely on approximation methods[9]." So this transition probability needs more consideration, depending on the problem. A good instruction on exact solution methods can be found in the sources mentioned at the beginning of this paragraph.

2.3.2. Deep Reinforcement Learning

Given the system's dynamics $\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$ and enough computing time and resources we can find optimal solutions to our (PO)MDP based problems. Oftentimes it is much more feasible and less time consuming to aim for approximate solutions, which can still satisfy our demands. One way may be to approximate the state- and action-value functions with parameterized functions, adjusting the parameter in a yet to be defined learning procedure. *Hornik* [1991] showed, that a multilayer feed forward network (also called multilayer perceptron) is a universal approximator[10], i.e. we can use a multilayer perceptron to approximate the state- and action-value functions.

We will briefly define the basic concepts of multilayer feed forward networks and their terminology. Then we proceed to the *actor-critic method* used for reinforcement learning in this thesis.

Multilayer perceptron A multilayer perceptron (MP) is made up of layers of nodes (Figure 2.5). The layers can be divided in *input, hidden* and *output* layers. The layers are interleaved with weighted connections and activation units. A weighted connection is defined via

$$a_j^{(\ell)} = b^{(\ell)} + \sum_i w_{ji}^{(\ell)} z_i^{(\ell)} \,, \tag{2.25}$$

where $b^{(\ell)}$ is a constant parameter, the *bias* and $z_i^{(\ell)}$ is the activation,

$$z_i^{(\ell)} = h\left(a_i^{(\ell)}\right) \,, \tag{2.26}$$

[9] e.g. Discrete state filter, linear-Gaussian filter or particle filter.

[10] The first version of this proof was done by *Cybenko* [1989] for sigmoid activation functions. *Hornik* expanded the theorem to the general case for non constant activation functions.

and $h(\cdot)$ is a non constant activation function, e.g. the sigmoid function

$$h(x) = \sigma(x) \coloneqq [1 + \exp(-x)]^{-1} \tag{2.27}$$

or the rectified linear unit (ReLU),

$$h(x) = \mathrm{ReLU}(x) \coloneqq \max(0, x) \ . \tag{2.28}$$

The weights $w_{ji}^{(\ell)}$ describe "how strong" the link between node i of the activations and node j of the summation joints in layer ℓ should be.

With precisely adjusted weights and biases, a multilayer perceptron can approximate any (in our context well behaved) function. How do we know, which values to choose for the weights and the biases?

Learning from mistakes One technique to find optimal (or "good enough") parameters for the MP is to start with randomly initialized parameters, propagate a set of input vectors $\{\mathbf{x}_n\}_n$ through the network and compare the resulting set of output vectors $\{\mathbf{y}_n\}_n$ with a set of target vectors[11] $\{\mathbf{t}_n\}_n$. The target vectors are the results we want the network to achieve. We calculate a measure of difference between output and target, and use this *error function* $E_n(\mathbf{w})$ of the network to adjust its parameters, where we defined $\mathbf{w}$ to be the matrix containing all weights and biases. We then iterate over the next set of input vectors and repeat. This procedure is known as *supervised learning*. A measure of difference may be the Euclidean distance between output and target vector or the smooth L^1 loss (section A.1).

Once we obtain the error function[12] $E_n(\mathbf{w})$ for a triple $(\mathbf{x}_n, \mathbf{y}_n, \mathbf{t}_n)$, we can adjust the parameters $\mathbf{w}$ that lead the network from $\mathbf{x}_n \to \mathbf{y}_n$. This is done by calculating the gradient of E_n with respect to the weights, $\nabla_{\mathbf{w}} E_n(\mathbf{w})$. This gradient is actually a matrix with the partial derivatives of E with respect to the respective entry in $\mathbf{w}$; for now, let us consider one such partial derivative:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \ , \tag{2.29}$$

where we just applied the chain rule. This is possible, because the error function depends on w_{ji} only in the form of (2.25). We can use the very same equation to identify the second factor as z_i. We refer to the first factor as the *error* δ_j of this layer. So the gradient for the final layer f can be reformulated to

$$\frac{\partial E}{\partial w_{ji}^{(f)}} = \delta_j^{(f)} \cdot z_i^{(f)} . \tag{2.30}$$

Going a layer backwards in the network to $f - 1$, consider its error:

$$\begin{aligned} \delta_j &= \frac{\partial E}{\partial a_j} \\ &= \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} \\ &= h'(a_j) \sum_k w_{kj} \delta_k \ , \end{aligned} \tag{2.31}$$

[11] The target vectors or targets are also called "ground truth".

[12] Remark: in this passage I use a similar notation as *Bishop* [2006, pp. 241–245].

where we used (2.25) and (2.26) in the last step. h' denotes the derivative of the activation function. The sum in the second-last step runs over all nodes k to which node j is connected. The beauty about this is, that we can repeat this for all layers and thus calculate their respective errors δ. The outlined process is called *error backpropagation*, because we propagate the networks error backwards, starting from the error at the output. The collectivity of errors make up the gradient $\nabla_{\mathbf{w}} E(\mathbf{w})$. Finally, we can use the gradient to adjust the parameters $\mathbf{w}$ towards a better result, using techniques like e.g. stochastic gradient descent. We repeat this procedure for every input vector $\mathbf{x}_n$ and thus iteratively update and improve the parameters. A thorough example of error backpropagation and more techniques on how to tweak the networks parameters can be found in e.g. [*Bishop* 2006] or [*Goodfellow* et al. 2016].

Having a set of target (or ground truth) vectors eases the training of a network a lot. If we recall our setting of an agent in an partly observable environment, thinking of any ground truth to compare the agent's decision with becomes nearly impossible. In this case, we need a different concept of error. In the previous sections we already discussed the reward an agent receives for selecting a specific action and the return; if we now consider the expected return R as ouput value and the true (actual) return R^* as target value, we can again define an error function $E(\mathbf{w})$ as measure of difference between expected and actual return.

Rewards and penalties Until now, rewards were just a concept; a quantity, every reinforcement learning agent tries to maximize by finding an optimal policy π^*. In terms of supervised learning, we actually know what the network inherent in the agent should learn. Without this training data, we have to set up a table to define the immediate rewards, i.e. a table for $\mathcal{R}^a_{ss'}$. The crux of the matter is, that we do not know, what the optimal behaviour for the agent is – hence the network, to approximate V^π and Q^π. We just have an understanding of what the agent should and should not do, e.g. a self-driving[13] car should stop at red lights and should not exceed the speed limit. The balancing act is, that we would like to penalise the agent for unwanted behaviour and reward it for wanted. But during the process of learning, the agent may find an optimal policy we inadvertently prohibit with too strictly defined rewards, e.g. the self-driving car should always stay on the road, but may leave the road to avert a fatal crash with an approaching vehicle or pedestrian; if the penalties for leaving the road and crashing would be equal, the agent might decide to just stay on the road, because the outcome would be the same anyway.

> It is thus critical that the rewards we set up truly indicate what we want accomplished.
> — *Sutton and Barto* [1998, p. 56]

More details and discussion on the chosen rewards for this thesis in chapter 4.

Actor-Critic Reinforcement Learning So far we discussed a deep feed-forward network, i.e. a multilayer perceptron. "Deep" in this context refers to one or more hidden layers in the network. A multilayer perceptron can also be called a *neural network (NN)*, although the class of neural networks contains a multitude of (more complex) neural network types, e.g. feed-backward NN, recurrent NN, just to name a few. I used a feed-forward neural network with a convolutional[14] input layer as function approximator (more detail on the topology in section 3.2).

[13] The "self" here would of course be an agent.

[14] A wonderful introduction to convolutional neural networks are the lecture notes on *CS231n: Convolutional Neural Networks for Visual Recognition* at Stanford University by *Karpathy* [2018].

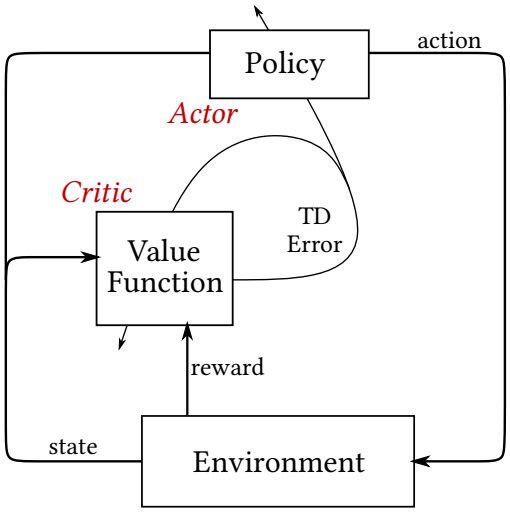


Figure 2.6. Actor-Critic model schematic. The *actor* acts out a certain policy and is then criticised by the *critic*. TD Error is the *Temporal Difference Error*, which is roughly a measure whether a chosen action lead the agent in a state that is better or worse than expected. Modified from [*Sutton and Barto* 1998, p. 151].

An actor-critic[15] agent (Figure 2.6) takes in the environment's state, which may be the actual state of the environment s or a belief state $b(s)$ due to partial observability. We will refer to the state just as s in the following. From this state, the actor-critic calculates two outputs: (i) the estimated state-value $\mathbb{E}[V(s)]$, and (ii) a recommendation of what action to take; the policy π.

In terms of neural network topology this means two outputs (see Figure 3.4); one output returns a single number, the estimate, and the other output returns a vector $\mathbf{p}(s)$ of probabilities regarding the choice of actions (if a softmax unit is used).

After each action selection $(s_t \xrightarrow{a_t} s_{t+1})$, the new state is evaluated to determine, whether the action turned out in a good or bad way for the agent. This evaluation is the *TD error*[16]:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \,, \tag{2.32}$$

where V is the current value function implemented by the critic. A positive value of δ_t suggests a stronger tendency to choose a_t in the future, whereas a negative value of δ_t suggests a decreased tendency. If the j$^{\text{th}}$ entry in $\mathbf{p}(s_t)$ should be adjusted, one way would be:

$$\mathbf{p}_j(s_t) \leftarrow \mathbf{p}_j(s_t) + \beta\delta_t \,,$$

where $\beta > 0$ is a step-size parameter, regulating the strengthening or weakening of the tendency.

[15] A picturesque introduction to (advantage-)actor-critic reinforcement learning may be found in [*Gilman* 2018], *Intuitive RL: Intro to Advantage-Actor-Critic (A2C)*.

[16] Not to be confused with the layer-wise error $\delta^{(\ell)}$ used for backpropagation.

3

Model Implementation

All models were implemented in Python 3.6 in conjunction with the modules numpy and matplotlib. The pytorch v0.4 framework [*Paszke* et al. 2017] provided the means to create, train and work with neural networks.

At first, a code basis was established to test and reproduce the well-known patterns of predator-prey-models. The first experiment builds on top of this basis in terms of action space $\mathcal{A}$ (section 2.2) and execution, with learning agents in addition. During this process, many new ideas arose and ultimately converged to the second experiment, with a different action space, but almost similar execution.

3.1. Basis

As a conceptual as well as code-wise basis for the two carried out experiments, a regular predator-prey-model was implemented[17], with inspiration drawn from the work of *Roth* [2017, pp. 364–378]. To create a better environment for reinforcement learning, this model follows the approach of an agent-based model (ABM) as opposed to the cellular automaton (CA) approach. This allows for a set of agents, each carrying its own set of properties, e.g. food reserve f_R and probabilities to flee and procreate p_{flee} and p_{breed}.

Common ground Aside from their differences in action space and configuration parameters (section 2.1 and 2.2), all models – that is the basis and the experiments – share a common set of features:

(i) The agents are initially randomly distributed across the grid. The number of agents per species is given by a density, 1 being a grid fully populated by one species.

(ii) Each cell is either empty or occupied by only a single agent.

(iii) There exist only two different species at any given time – if one of the species dies out, the simulation stops.

(iv) Each agent has access to its 5- or 9-neighbourhood (Figure 3.1) (but may be able to see a bigger snippet of its surroundings).

(v) The grid has toroidal shape, i.e. periodic boundary conditions.

[17] The code base for this thesis can be downloaded at [*Krautgasser* 2018]. The state of the code base is the submission date of this thesis.

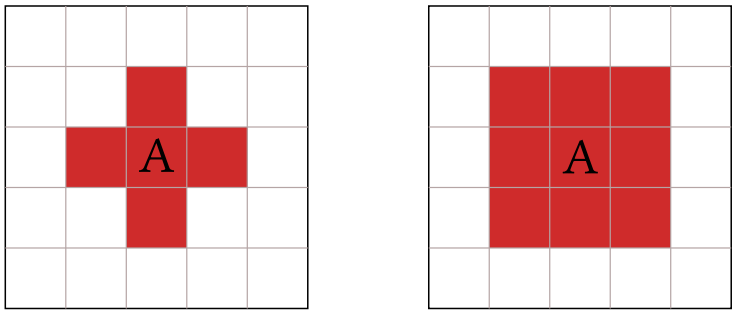


Figure 3.1. The 5- (left) and 9-neighbourhood (right), each centered around an agent denoted by A.

Storyline We will try to conceptionally visualise the different experiments along certain storylines. One can think of a variety of suitable examples, but we will use a biological storyline of microorganisms or bacteria. For the basis model we think of a culture medium (the resource R) and two species of microorganisms; the first is able to extract nutrients from the culture medium, the second can only survive by consuming the first species. Both types are able to reproduce asexually, that is they do not need a partner to create an offspring.

Basis specification The basis stochastic model, i.e. every action is chosen with a given probability. We will consider:

- a quadratic grid with periodic boundary conditions,
- no potential obstacles for the agents,
- a 9-neighbourhood to interact with,
- probabilities p_{breed} and p_{flee} different from 0 and 1,
- a per-turn reduction of the food reserve f_R by 1,
- asexual reproduction (offspring can spawn with p_{breed} from any agent with sufficient f_R)
- and all agents start with a food reserve of three and a maximum food reserve $f_R^{\max} = 8$.

The set of possible actions $\mathcal{A}$ in this scenario is the product of the neighbourhood size with the three actions, *moving*, *eating*, and *procreating*, and thus $\#\mathcal{A} = 27$. While moving seems clear, what does it mean for an agent to eat? In case of the prey, it just means an increase of f_R by 1, because there is an infinite amount of edible (and stationary) resource growing on the whole grid. In the other case, for the predator, eating is a bit trickier: it has to hunt and eat a prey to increase its food reserve. This is only possible, if a prey is in the 9-neighbourhood[18] of a predator, and even then, the prey can flee with probability p_{flee}. Upon a successful hunt the predator's f_R is increased by three. Procreation is possible for both species, given that the specimen has enough energy, i.e. a sufficiently high food reserve ($f_R > 5$), to create an offspring.

[18] The same applies for the 5-neighbour-hood.

Definition of a turn After going through these preliminaries, we will now look into the order of events during a single turn, which is the frame for a single agent to act in. To speed up the dynamics of this basis model, every agent has the chance to execute multiple actions within one turn. Let us inspect the structure of such a turn, to shine some light on this:

(i) An agent is randomly selected (without replacement) from the list of all agents that currently populate the grid.

(ii) The agent's food reserve is reduced by 1. If $f_R \leq 0$, the agent dies and is removed from the grid. Otherwise, the agent has now the following (sequential) options:

  (α) The search for food; a prey will always find food, predator has to actively search and catch a prey. If a prey is within the 9-neighbourhood, it is consumed with probability $1 - p_{\text{flee}}$.

  (β) If $f_R \geq 5$, the agent tries to reproduce. In the course of successful reproduction, the agent's f_R is reduced by three. The offspring is placed in an empty cell in the agent's 9-neighbourhood, but is not yet added to the list of agents[19] (though it can be eaten, if it is a prey's offspring).

  (γ) If the agent is prey, it can then even move to an unoccupied cell within its reach.

(iii) Continue with (i) if the list of all agents is not empty. Otherwise, create a new list of agents, now including the newborns.

The iteration over the list of agents composes a timestep in the simulation. The basis model typically runs for 500 timesteps at maximum, because the most dynamic behaviour resides in the first few hundred timesteps. Once the system reaches equilibrium, no major change occurs in either the patterns or species-wise densities [*Roth* 2017].

Testing the basis Running a simulation with the above defined settings (Listing B.1) leaves us with the expected results (Figure 3.2) , e.g. emergent cauliflower-like patterns and the settling of a stable equilibrium of predator-prey-densities [*Roth* 2017, pp. 368–371]. Using these same parameters recreates a similar behaviour.[20]

3.2. A word about learning

What distinguishes humans from agents is our ability to observe our environment and *decide*, which action is most appropriate in our current situation. The agents' decisions are predetermined by (i) implemented rules and by (ii) probability. We may not be able to solve the second restriction, but we may learn how to disobey rules to find an even more appropriate action.

To enable our agents to actively choose their actions in response to their *observed state*, we would like to bestow an individual neural network to each agent roaming the grid. Unfortunately, this is not possible due to computational and temporal expenses;[21] furthermore, the lifespan of

[19] This is a mere implementation detail. If newborn agents were directly added to the list of all agents, a timestep could grow to large lengths. This is solved by adding the newborn agents to the list of all agents at the end of every timestep.

[20] Refer to video ppm.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

[21] The sequential nature of the models made an implementation setting for multiple CPUs difficult. Furthermore, I had no discrete GPU permanently at hand. Running a single simulation for $\mathcal{O}(10000)$ episodes took up to two weeks.

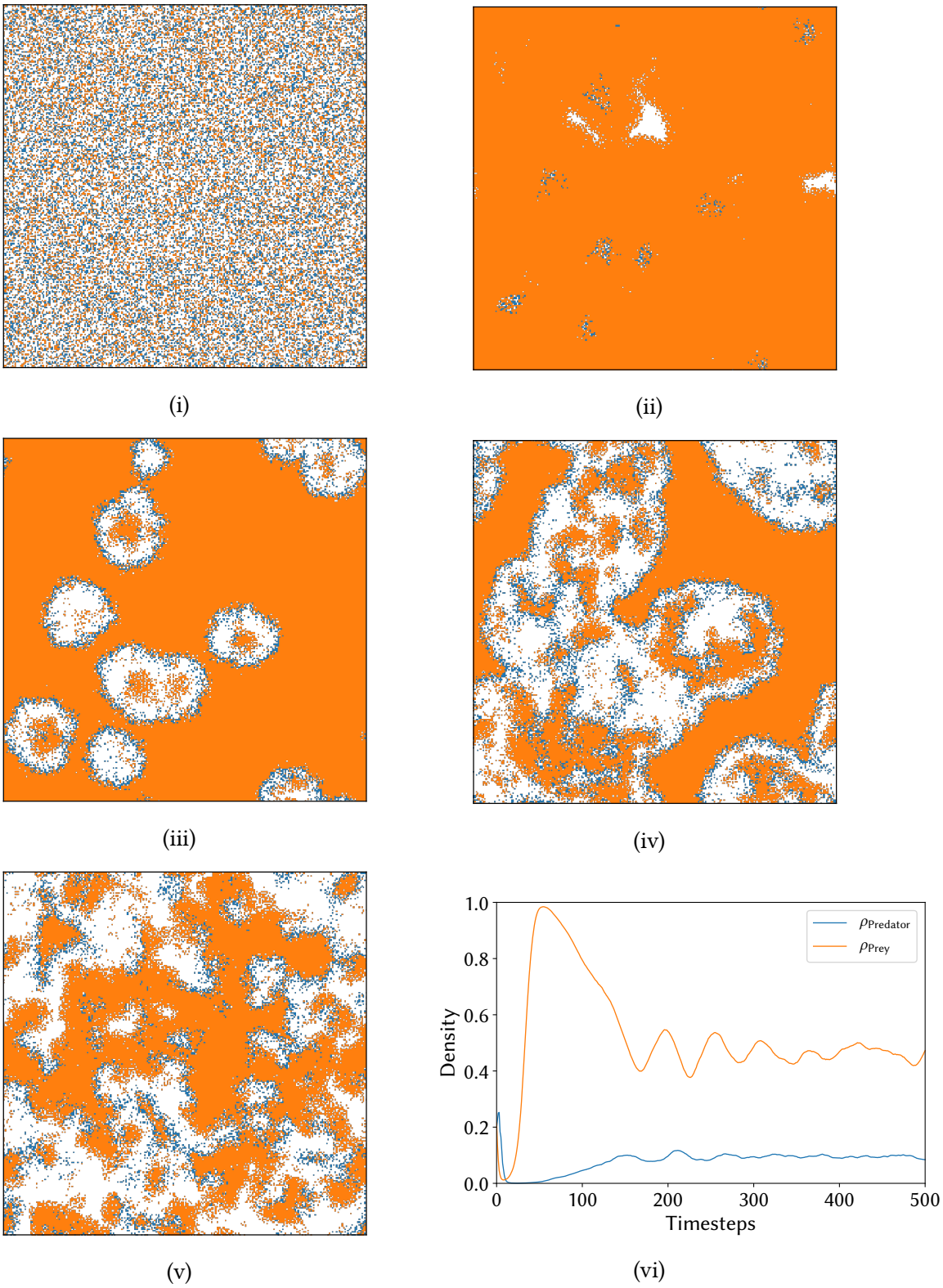


Figure 3.2. Views of the environment of the basis simulation at different time steps on a 256×256 grid; preys are orange, predators blue. (i) initial random distribution, (ii)-(v) at timesteps 50, 100, 150 and 500 respectively. (vi) shows the dynamics in terms of species density fluctuations.

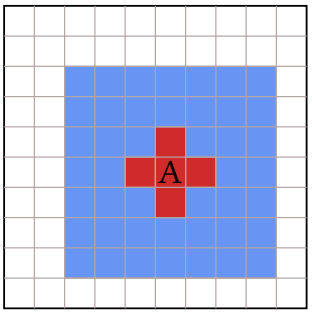


Figure 3.3. A 7×7-snippet of a 10×10 grid. Everything in the blue square is observed by the agent and it can interact with everything inside the red cross.

an agent is rather short (compared to a timescale of $\lessapprox 500$ timesteps for a simulation) and thus its experience – the set of states, actions and rewards – to train the network is too small. To handle this issue, we want to follow a simple line of thought: all individuals of a given species should react similar, when confronted with the same situation (with a small probability to react differently). What does this mean for our experiments? We begin by only using two topologically identical neural networks, one for each species. Each specimen carries its own experiences, and the entire species-wise experiences are then used to train the network.

Training To help the agents train and find the optimal action, I used a smaller set of actions, reducing all possible actions down to the ones accessing only the 5-neighbourhood ($\#\mathcal{A} = 15$) for the first experiment and an even smaller set for the second (see section 3.4). A smaller action space provides less room for exploration, so the agents should need less time to find the optimal actions. Using probabilities to decide whether an agent is able to flee or procreate turned out to be rather harmful to the learning process[22], at least when the probabilities were small. To circumvent this issue, the probabilities were removed, i.e. set to non-affecting values: $p_{\text{flee}} = 0$, $p_{\text{breed}} = 1$, and the agents received a bigger viewing range. So instead of just the 9-neighbourhood – which is a 3×3-snippet of the environment – every agent now has a view of a 7×7-snippet of its environment (Figure 3.3). In theory, the agents could decide to take a certain action (e.g. running away or moving in the direction of a prey), based on their farther afield surroundings.

As a second input we are free to choose from the agents' properties. To keep the model as simple as possible, only the food reserve f_R was used for the first experiment, and a set of f_R and orientation o (see section 3.4) for the second experiment. Since we are situated in experimental waters, I wanted to keep the input information as small as possible; a measure for the "hungriness" of an agent, and its physical orientation in the environment looked promising.

As introduced in section 2.3, agents receive a numerical reward for their actions, where they try to maximize their outcome, i.e. minimize the difference between expected and received return. Before we can dive deeper into how this process works in detail, we need to establish an understanding of our neural network structure (Figure 3.4), that is, what do we actually use as observed state, how is the state propagated through the structure and, of course, what is the mapping from the networks output to an actual action?

Walkthrough We now take an agent's view of the world, which is just a 7×7 snippet of the actual environment, centered around the agent, and a set of values representing that agent's internal state. The view contains only numerical information about whether a cell is empty (0), occupied

[22] Resulted in very short episodes and thus very short histories to learn from. More on this in chapter 4.

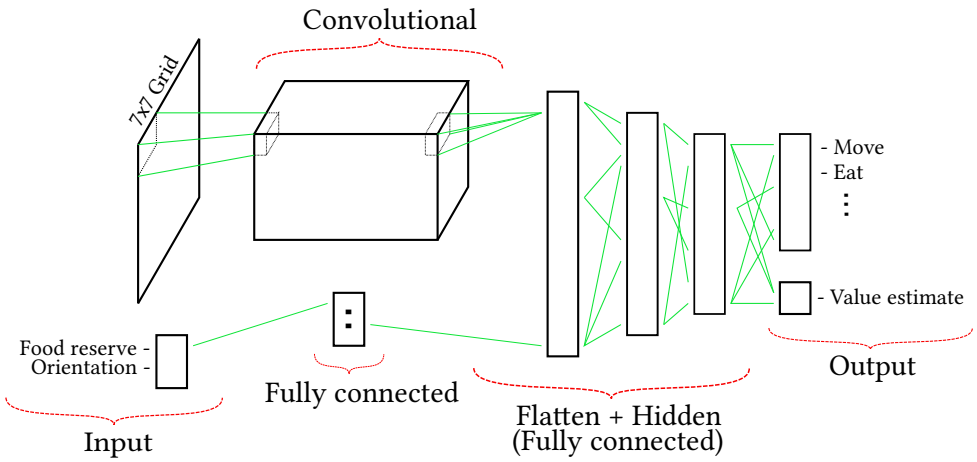


Figure 3.4. Schematic of the networks' topology. Inspiration for this configuration was taken from [*Leibo* et al. 2017]. Green lines indicate the connections between the layers, although for a more comprehensive visualisation not all connections are outlined.

by a predator (-1) or a prey (1). The usage of a distribution of belief states (section 2.3) was omitted, again with the model simplicity in mind. These two pieces of information need to be handled separately: (i) The view is the input for the convolutional layer, since the agent would like to recognize patterns. We do not have any colour-coded input information, i.e. a grid with colours representing agents and empty cells[23], so a single input channel and a number of three output channels is sufficient. This number is arbitrarily chosen, but remains constant throughout the experiments. (ii) The additional information about the internal state is fed into a regular, fully-connected layer with the same number of in- and outputs (recall Figure 3.4)[24].

So after we propagate through the first multi-input layer, their activations are concatenated to a single 1D vector $v \in \mathbb{R}^{147+m}$, to be passed on to the three hidden layers, with m being the size of additional information, e.g. 1 for only the food reserve. Reaching the last multi-output layer, we end up with two parts of information: an optimal action and an expected outcome of the optimal action. The number of output values (probabilities) p_i in the first part resemble the possible actions $a_i \in \mathcal{A},\ i = 1, \ldots, 15$. The highest value signifies the (most likely) optimal action choice from the network's current point in training[25]. It is crucial to notice here, that we get the probabilities p_i only, because we inserted a softmax unit (section 2.3) between the last layer and the actual output. The second part of information is just a single, real number – the network's estimate of the actions outcome, also called state value $V^\pi(s)$.

If the action selection process stops here, the network will not explore the action space too well – rather, it will stick to actions it already knows. More flexibility is introduced by e.g. an ε-greedy strategy (see section 2.2.2). We use a softmax action-selection process: a categorical distribution[26]

$$f(\mathbf{a}|\mathbf{p}) = \prod_{i=1}^{k} p_i^{a_i} \tag{3.1}$$

[23] Unlike *Leibo* et al. [2017] and *Mnih* et al. [2013], who used an RGB and gray-scale representation of the agent's observed state, respectively.

[24] All layers are interleaved with ReLU-units (section 2.3).

[25] An untrained network will just give random suggestions, whereas a trained network should choose optimal actions.

[26] Note that the superscript a_i is a label, not a power.

is constructed, with $k = 15$, $\mathbf{p}$ and $\mathbf{a}$ being the probability and action vectors for the probabilities and actions p_i and a_i under the constraint $\sum_i p_i = 1$. The probability vector $\mathbf{p} \in \mathbb{R}^k$ contains the output of the softmax unit. Note here, that $\mathbf{a}$ is a k-dimensional vector containing zeros, except for a single entry equal to 1, representing the chosen action. Sampling the categorical distribution finally provides the mapping from network output to action. The expected value of action a_i, the i^{th} entry of $\mathbf{a}$, is $\mathbb{E}_{a_i \sim f(\mathbf{a}|\mathbf{p})}[a_i] = p_i$, but there is enough variation to search the action space for better outcomes.

Learning procedure Lastly, how does the network increase its performance from a collection of state-action-reward tuples ((s, a, r)-tuples)? For that, see the pseudo-code instruction (Algorithm 3.1) of the actual implementation (see codebase[27]); same terminology as in section 2.3.

Algorithm 3.1 The *learning procedure*, a method to iterate over the set of all action-reward sequences of a species, the history $\mathcal{H}$. The action itself is again a tuple, containing the log-probability of the chosen action and the (estimated) state value $V^\pi(s)$. γ is the discount factor. $\mathbf{R}$ and $\mathbf{A}$ are the temporally ordered per-agent reward-action pairs. The machine epsilon ϵ is an upper bound on the relative error due to rounding in floating point arithmetic, used to prevent infinite values in case of zero variance. For type double or float64: $\epsilon = 2^{-52}$.

1: $\text{PolicyLoss} = \text{list}(),\ \text{StateValueLoss} = \text{list}(),\ \mathcal{L} = 0,\ \text{losses} = \text{list}()$
2: **for** $\mathbf{R},\ \mathbf{A}$ in $\mathcal{H}$ **do**
3: $\quad\mathcal{R} = 0,\ \text{rewards} = \text{list}()$
4: $\quad$reverse $\mathbf{R}$ ▷ temporal order is reversed
5: $\quad$**for** r in $\mathbf{R}$ **do**
6: $\quad\quad\mathcal{R} \leftarrow r + \gamma\mathcal{R}$
7: $\quad\quad$append $\mathcal{R}$ to rewards from left
8: $\quad$**end for**
9: $\quad\text{rewards} \leftarrow \frac{\text{rewards} - \mathbb{E}[\text{rewards}]}{\sqrt{\text{Var}[\text{rewards}] + \epsilon}}$ ▷ standardised score
10: $\quad$**for** $\text{logprob},\ V,\ r$ in $[\mathbf{A},\ \text{rewards}]$ **do**
11: $\quad\quad\text{reward} \leftarrow r - V$
12: $\quad\quad$append $(-\text{logprob} \cdot \text{reward})$ to PolicyLoss
13: $\quad\quad$append $L^1(V, r)$ to StateValueLoss ▷ L^1 is the smooth L^1-loss (A.3)
14: $\quad$**end for**
15: $\quad$Zero existing gradients of the optimizer.
16: $\quad$append $(\sum \text{PolicyLoss} + \sum \text{StateValueLoss})$ to losses
17: **end for**
18: $\mathcal{L} \leftarrow \mathbb{E}[\text{losses}]$ ▷ Average the agent-wise losses
19: Backpropagate $\mathcal{L}$ and update network parameters

The procedure between lines 3-16 + 19 in algorithm 3.1 reflect the regular update procedure of the network's parameters. The difference here is that we calculate the loss for every agent (of a given species) separately, and then average the loss to get a statistically backed loss-value $\mathcal{L}$ for the update.

[27] Codebase can be found in [*Krautgasser* 2018] in directory actor-critic.

3.3. Experiment I: Hunters and Gatherers

This experiment is (action-wise) almost completely based on the stochastic model described in the section above.

Storyline Similar to the basis model, we think of a culture medium carrying two different species of microorganisms. In this scenario, the organisms developed some sense for their surroundings, i.e. they can sense further than they can take direct action. Such a sense may be e.g. optically, chemically, etc.

In section 4.1 we will discuss some effects of the chosen rewards and initial conditions.

Episodes and timesteps The setup for simulations yet is slightly different: we expect the agents' behaviour to change over time as they learn to cope/interact with their environment. Thus, running a single simulation of at maximum n timesteps does not tell us anything – especially, if one of the species dies out within a few 10 timesteps. What we actually want is a gradual training, meaning that we let a simulation run, use the agents' collected experiences to update the neural networks' parameters and restart the simulation, but this time with the pretrained networks. To have a clear nomenclature, we call a single simulation that runs for a fixed number of timesteps an *episode* (also sometimes called an *epoch* in the literature). We will often consider developments of the agents' behaviour over the course of episodes, rather than timesteps.[28]

Experiment specification Recall for a moment what we already mentioned earlier: we disabled the probabilistic properties for *eating* and *procreating*, enhanced the agents' field of view, utilized a categorical distribution (3.1) to account for the exploration of the action space $\mathcal{A}$ and set up two topologically identical neural networks to approximate the decision policy for each species. Furthermore, in experiments I and II, every agent can only act once during its turn. If an action fails, because the wrong decision was made, the agent ends up doing nothing, but receiving a negative reward. An overview of a sample configuration file is given below in Listing 3.1.

The temporal order of the (s, a, r)-tuples is crucial to the learning process. A prey has to learn, that it got eaten in the last timestep, so the last state s a prey "observes" is centered around the successful predator. This means code-wise that the random order of the shuffled agents list is intermitted once a predator's hunt is fruitful, to let the prey observe its environment a last time and receive its penalty in form of a negative reward.

```
Model:
    dim:                !!python/tuple [16, 16]  # dimensionality
    densities:          !!python/tuple [0.3, 0.35]  # Predator, Prey
    food_reserve:       4
    max_food_reserve:   8
    generation:         0  # initial value for first generation
    neighbourhood:      49  # only squares of odd numbers, e.g. 9, 25, 49
    p_breed:            1.0
    p_flee:             0.0
    p_eat:              1.0  # = 1-p_flee
    mortality:          True  # if False, agents cannot starve
    instadeath:         0.00  # more than 1 predator -> dying probability
```

[28] Since a policy does not change within the course of an episode.

```
14  rewards:
15    wrong_action:       -1
16    default_prey:       1
17    default_predator:   1
18    indifferent:        0
19    succesful_predator: 5
20    offspring:          20
21    death_starvation:   -5
22    death_prey:         -15
23    default:            1
24    instadeath:         0
```

Listing 3.1. Sample configuration snippet for experiment I. The parameters in lines 12 and 13 will be discussed in section 4.1. The parameter p_{eat} is introduced for convenient implementation. All reward parameters starting with `default` reflect a motivation for moving, and eating in the case for preys. A wrong action is rewarded with a small negative reward, whereas starving and falling prey are weighted much stronger; a succesfull hunt for a predator presents a high reward for his effort. The highest reward is given for procreation, with the simple thought that every lifeform should safeguard its survival.

3.4. Experiment II: a Sense of Orientation

During the progress of the first experiment, I started to explore my personal action space in creating a different model to experiment with. The new model shares a common ground with the basis model and the first experiment as mentioned in section 3.1, as well as an almost similar network topology: the orientation o as additional input in conjunction with the food reserve f_R. The main goal was to find out, whether this smaller set of actions would down-scale the need for exploration adequately and thus increase the "learning speed".

Storyline The baseline here is again a culture medium with two species of microorganisms, one preying on the other. The microorganisms in this thought experiment are no longer able to move in all four directions arbitrarily, rather just the one they are oriented in. They are able to rotate and then move forward by e.g. using a flagellum. This sense of orientation may be explained by chemotaxis; assuming all organisms in the medium leave a trace of some detectable (semio)chemical.

Oriented actions This experiment features an action space of size 8, with mostly new defined actions. In both previous models, an agents choice of action corresponded directly to a certain direction: up, down, left, right (except for the actions that don't move the agent). Now, every agent has an intrinsic sense of direction/orientation o. This is represented by a 2-tuple, which can be one of the canonical unit vectors $\{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}$ of 2D-space or their negative counterparts. Initially, every agent's orientation is chosen randomly. Now, 3 out of the 8 actions are solely dedicated to change the agent's orientation: *turn left, turn right* and *turn around.* The actual turning can be easily achieved by multiplying the orientation-vector with the corresponding 2D-rotation matrix. The by this time well known trio *move, eat* and *procreate* comprises another 3 members of the action space, each of them directed by the agent's orientation. The remaining two actions are *do nothing* i.e. do not move, and *eat while standing still.* The (theoretically) possible action to procreate onto the agents own cell was left out, since it is not provided for both species[29]. Of course, different actions need different sets of rewards. This will be explained in detail in section 4.

[29] Recall: only a single agent per cell.

The incipiently mentioned small change in network topology is due to the new internal property o. To actively use the new variable, the additional information for the policy is now the set of food reserve f_R and a simplified measure of the direction, $o^\flat$ (o-*flat*), with $o^\flat \in [0, 1)$ and a map $\flat$ with:

$$\flat \colon \mathbb{R}^2 \to [0, 1), \quad \flat(o) \mapsto o^\flat, \tag{3.2}$$

which is just a transformation for the 4 directions to equally spaced values between 0 and 1, with 0 and 1 overlapping.

Experiment specification Now that we have our actions defined, let us summarize this experiment. With the action space $\mathcal{A}$, disabled probabilistic properties for *eating* and *procreating*, slight adjustments to the network topology with unchanged update procedure and some newly defined parameters[30] (Listing 3.2), everything is conceptually set up.

```
Model:
    dim:                !!python/tuple [16, 16]  # dimensionality
    densities:          !!python/tuple [0.2, 0.35]  # Predator, Prey
    food_reserve:       3
    max_food_reserve:   8
    metabolism:  # see description!
      OrientedPredator:
        fast:           0.25  # decrease every timestep
        satiety:        3  # get for eating
        exhaust:        3  # for mating
      OrientedPrey:
        fast:           0.5
        satiety:        2
        exhaust:        3
    view:               !!python/tuple [7, 7]  # supersedes neighbourhood
    generation:         0  # initial value for first generation
    p_breed:            1.0
    p_flee:             0.0
    p_eat:              1.0  # = 1-p_flee
```

Listing 3.2. The biggest structural change is given by the dictionary `metabolism`. It holds the parameters that describe an agent's decrease of f_R due to hunger (`fast`) and creation of an offspring (`exhaust`), and an agent's increase after consuming a resource (`satiety`). Additionally, the extent of the observation an agent receives is given by the parameter `view`.

[30] With the new model idea a new structure for configuration files arose. Unfortunately, there was not enough time to introduce the new structure to the existing code base.

4

Results

With the patterns of Figure 3.2 in mind, what do we expect from not randomly acting agents? As pointed out in the theoretical background (section 2.3.2), choosing the rewards will inevitably shape the policy's outcome. Unfortunately, the infinite possibilities to tweak the rewards and study their effects supported by the time-expensive simulation runs prevented a more elaborate study.

Network The parameters for the neural network topology that was used throughout both experiments and their variations is listed in Listing 4.1. The value 148 might seem ominous and arbitrary, but it is just the product of the input dimension (in our case always 7×7) with the number of output channels in the convolutional layer plus the number of additional inputs (in our case f_R in experiment I, and $(f_R, o^\flat)$ in experiment II).

```
Network:
    kind:               'conv'  # first layer is conv, see below
    mode:               'cpu'  # can also be 'gpu'
    layers:             # overview of layers
      conv1:
        in_channels:    1
        out_channels:   3
        kernel_size:    3
        padding:        1
        stride:         1
      affine1:          !!python/tuple [1, 1]  # for food reserve
      hidden1:          !!python/tuple [148, 64]  # 148 = 3*7*7 + 1 [concat]
      hidden2:          !!python/tuple [64, 32]
      hidden3:          !!python/tuple [32, 32]
      action_head:      !!python/tuple [32, 15]  # action prob. output
      value_head:       !!python/tuple [32, 1]
    gamma:              0.9  # discount factor
```

Listing 4.1. Parameters of the neural network topology as used in both experiments. The tuples are the number of in- and output nodes for each layer.

4.1. Experiment I

The "classical" predator-prey model was the cornerstone of this experiment. Many of the parameters were left untouched as in Listing B.1, e.g. f_R or initial densities ρ_{Predator} and ρ_{Prey}. The parameter for timestep-wise reduction of the food reserve was adjusted from 1.0 to 0.25, because with an initial $f_R = 4$ there was not enough time to train the predators to go hunting; the result was, that almost all predators chose not to move at all, which resulted in starvation after four

timesteps. This behaviour did not change at all over the course of a few thousand episodes, repeatably returning the same loss-values. The change to a timestep-wise reduction of 0.25 of the food reserve widened the space of possibilities sufficiently.

As a baseline for discussion we will consider the macroscopic patterns emerging during an episode, as well as the density fluctuations of that episode. A second measure to capture a notion of learning is the average reward over episodes: we consider the average reward the agents of a species received in an episode over the respective number of that training episode. With this measure we can relate changes in the plot and behavioural developments. A further source of information is the length of episodes in timesteps: all episodes are limited by either 500 timesteps or the extinction of a species. In conjunction with the average reward within a species, we can relate the predators ability to hunt or the preys ability to flee.

4.1.1. On Rewards I

During the exploration phase of finding working values for the rewards (Listing 3.1, lines 15-24), some parameters influenced the behaviour more drastically than others:

- `wrong_action: -5` resulted in predators who did not want to procreate; a sufficiently high f_R is needed to create an offspring, so trying to do that with a low food reserve will be penalised with a negative reward. After a few thousand episodes of training, the predators learned to survive by hunting; although they did not procreate at all. The preys did learn to procreate, because they had an easier access to food.
- Interestingly, setting `succesful_predator: 15` had an almost similar effect. The reinforcement learning predator tries to maximize its expected return; so if just hunting is more feasible than procreating, why should it bother doing it?
- Rewarding the agents not for moving (all rewards starting with `default`), but only for eating and procreating made no real impact on the preys; the predators however resigned and stopped moving, again waiting for starvation. Occasionally, a predator tried to hunt, but all conducted simulations showed no sign of improvement in this regard.
- The prey agents had an easier life. In every simulation they learned that basically sitting in one spot, eating all the time and periodically creating an offspring (if there is enough space) was everything they needed. However, once the predators realized how to effectively hunt (e.g. by combing through the grid in two directions), the struggle to survive became apparent.

Apart from these overall insights, we now look into two simulations in more detail.

4.1.2. Isolated Agents

To look more closely at the learning process of the agents, I implemented a modified version of the hunters and gatherers experiment. The four introduced parameter changes are

(i) `dim: !!python/tuple [10, 10]`

(ii) `densities: !!python/tuple [0.01, 0.01]`

(iii) `mortality: False`

(iv) `instadeath: 0.005`

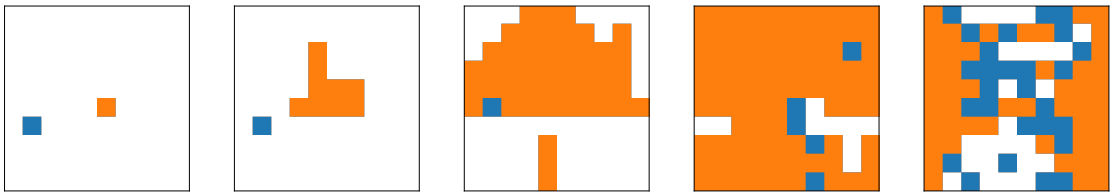


Figure 4.1. Experiment I – Isolated Agents, frequent path; views of the environment at different time steps on a 10×10 grid; preys are orange, predators blue. (i) initial random distribution, (ii)-(v) at timesteps $10,\ 20,\ 30$ and 50 respectively. In (i)-(iii) we can observe how the first predator just moves downward in one column of the grid, until it finds prey. Once the reproduction starts, given enough space, the predators move to the left or right, and create offspring in the down or up direction.

Let us discuss the implications for a moment. (i) Is just the extent of the grid, so we consider a smaller grid in this simulation than in all others (like 16×16 or 32×32). Recall that the grid size determines the maximum of individuals on it. In conjunction with (ii), we just have an initial number of one agent per species. Since the placement for these two agents on the grid is random, a larger grid would result in longer periods where the predators need to actually find the prey on the one hand, and in longer simulation runtimes, because of the growing population size on the other hand. (iii) Is a debatable point. This parameter removes the option that agents with $f_R \leq 0$ actually die. Thus, a prey can only die by being eaten by a predator, but predators cannot die, since they cannot starve. This brings us to (iv), which matter is twofold: firstly, as long as there is more than one predator on the grid, every predator may die with this probability. The reason for this probability to come into effect once there are two or more predators, is solely for the gain of information. A lone predator that survived for 100 timesteps will be dead at timestep 100 in half the cases, even with a probability of 0.5% per turn to die spontaneously. Secondly, predators have no other way of dying in this scenario; so once they learn how to hunt, which happens relatively fast, they will cover the grid immediately, so this probability reduces this growth rate slightly.

Storyline The points (iii) and (iv) may best be represented with a storyline and let us stay consistent with the central theme of bacteria and microorganisms in a culture medium. Some microscopic lifeforms inherit the ability to set themselves in a ”state of hibernation”: a state, where almost every metabolic activity is shut down, and they can prevail inhospitable conditions, e.g. droughts, strong radiation or lack of nutrition. For the prey organism this would not really be necessary, because the culture medium provides enough nutrition for one hundred individuals. But the predator organism might have trouble finding another lifeform it can consume, so the best way to survive in this condition is by shutting down almost all metabolic activity, with the ability to move around, in case any prey comes by.

With a training period of ≈ 40000 episodes, two rather different possible outcomes arise (Figure 4.1 and Figure 4.2), although both look similar in the first few timesteps: the lone predator searches for prey by moving in one direction; the lone prey feeds on the culture medium and reproduces when its food reserve is sufficient. Depending on the initial separation of both agents, the predators needs a certain amount of time, to find prey. This is where the two paths split:

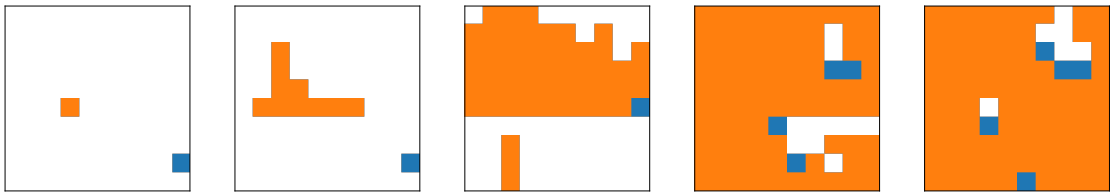


Figure 4.2. Experiment I – Isolated Agents, less frequent path; views of the environment at different time steps on a 10×10 grid; preys are orange, predators blue. (i) initial random distribution, (ii)-(v) at timesteps $10,\ 20,\ 30$ and 50 respectively. Note the similarity to Figure 4.1; the predators effort to eat all preys may be interfered by not allowing them to have space for reproduction.

- in the more frequent path[31], the predator finds enough preys to consume within a short time, typically around 10 timesteps. With the preys distribution up to this point (which usually is only one big cluster of preys, e.g. Figure 4.1 (ii)-(iii), and their exponential growth[32], the lone predator can leave its "state of hibernation" and go for a hunt with subsequent reproduction. In this case, there is enough physical space for the predators to spread their offspring, so all preys will be eaten within the next 30 to 60 timesteps.
- in the less frequent path[33], the two agents are separated farther apart initially, and the predators guess in which direction it should move is not optimal. While the predator moves, the preys can spread on the whole grid. Once the predator encounters its prey, there is almost no space left for a predator's offspring. In some cases, the predator and its descendants are able to establish a population size of up to 5 individuals. Still, there is not enough space to create more offspring, because every cell can only be occupied by a single individual. In this dilemma, the best choice for the predators is to keep eating; and hope for space to place an offspring.

Comparing the two prey populations (Figure 4.3), we can see the increasing difference in density, starting around timestep 25. It is up to that point crucial for the predators to establish a population big enough to be able to cope with the growing number of preys. Including the predator in the comparison (Figure 4.4), we find that the initial density progress is quite similar. With only a few empty cells around to place offspring into, the predator population will rise to consume all predators in the environment, but will otherwise decline to a single individual.

The development of average rewards over episodes (Figure 4.5) reveals a lot about the learning process. The behaviour in the first few thousand episodes is rather random, thus all episodes run the maximum number of timesteps, since no species dies out. From about the 4000^{th} episode forth the predators develop their effective hunting technique, thus resulting in a drastic drop in timesteps per episode. More interesting are the three "branches" of episode durations (around 0, 60-100 and at 500 timesteps). In the list above, I pooled the lower two branches into one: the

[31] Refer to video isolation_frequent.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

[32] Preys increase their f_R by 2 by eating, and decrease by 3 by reproducing. Since they start at $f_R = 0$ in this case, it takes the first prey at least 3 timesteps to eat enough and produce an offspring. So the population size doubles approximately every 3 timesteps $\left(N_{\text{Prey}}(t) \approx 2^{t/3}\right)$ in the first 10-15 timesteps), although this process is damped by limited space.

[33] Refer to video isolation_infrequent.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

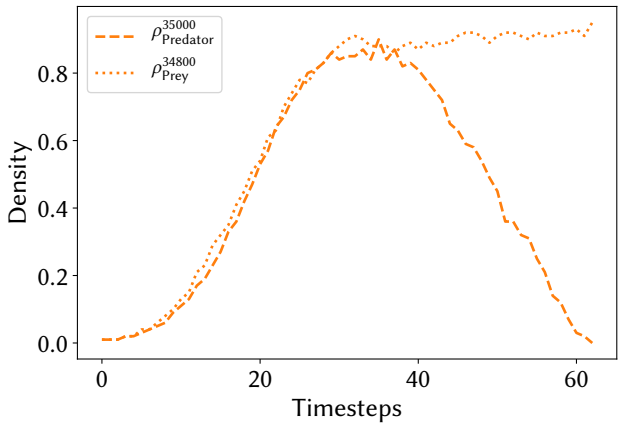


Figure 4.3. Experiment I – Isolated Agents; development of the prey population for the two possible paths. Superscripts indicate the respective episode. The dashed line shows the development for the frequent path, the dotted for the less frequent. Note, that the simulated data stem from two different points during the training, but the 200 episodes do not account for this drastic change in behaviour; this can also be observed, using the same neural networks, and only adjusting the initial distribution of the two agents.

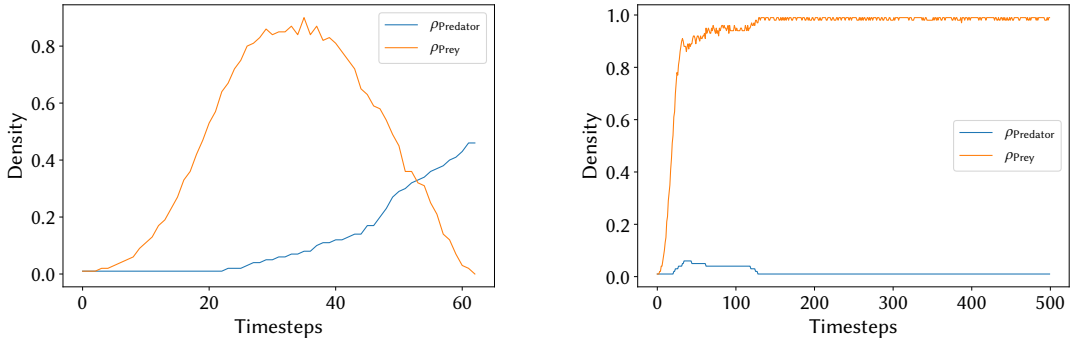


Figure 4.4. Experiment I – Isolated Agents; comparison of the population density development of the frequent path (left) and the less frequent path (right). Note the decline in predator population in the less frequent path due to stochastic death and the lack of space to place offspring.

frequent path, i.e. predators find food fast enough to cover the whole grid within only a few timesteps. Of course this also contains the possibility of a predator and prey spawned next to each other in the first step, which will result in a rather short episode. The top branch is then the infrequent path.

The overall continous increase in the preys average reward is the result of a strategically better spread and reproduction.

What distinguishes this result from the other simulations in both experiments are the apparent constant lines of reward; often enough the average reward for either predators or preys or both is a an integer, i.e. $\mathbb{E}[r] \in \mathbb{Z}$. These patterns cannot be found in any other results. Apparently, if the initial agent takes actions in a particular sequence, the average reward comes down to an integer. The two most prominent examples would be an average reward of 5.0 for the predators and an average reward of 1.0 for the preys. Interestingly, these specific values of average rewards can be found in all three branches of episode durations. My guess here would be, that this is a result of the choice of values of the rewards, but this would need a more rigorous investigation. Additionally, there are other lines at non-integer values; but they are not as prominent as the integer valued ones.

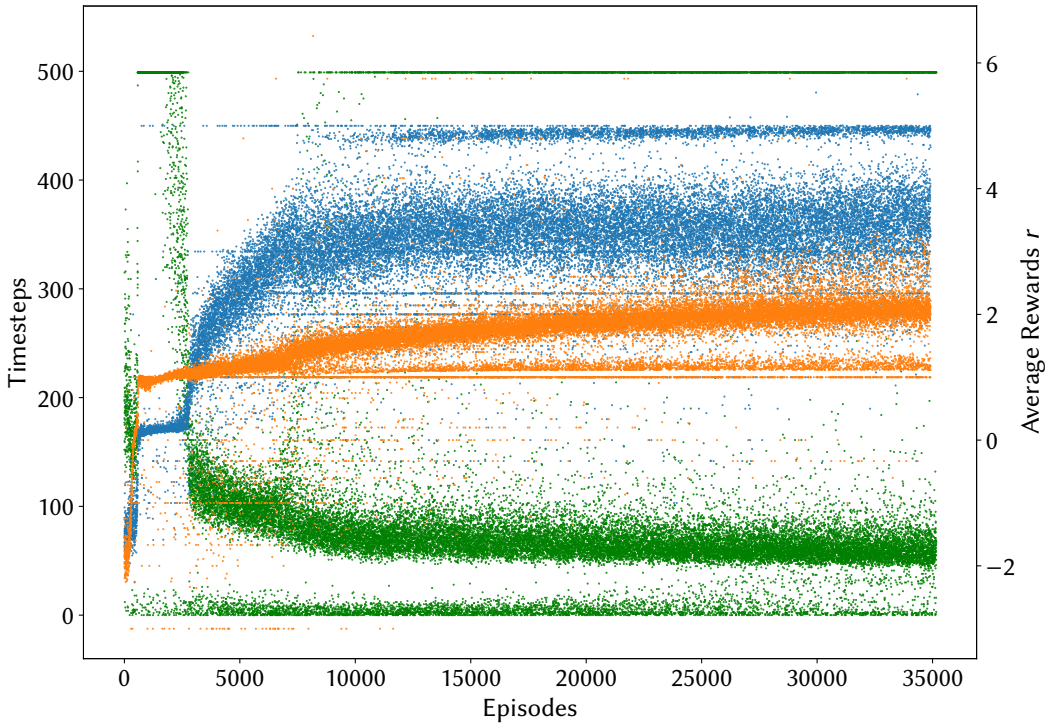


Figure 4.5. Experiment I – Isolated Agents; average rewards of predators (blue) and preys (orange) per agent per episode & number of timesteps (green) per episode.

4.1.3. Hunters and Gatherers

In this section, we turn from isolated agents to a regularly populated grid, with parameters similar to the basis model. The configuration used in this section can be found in Listing C.2

The training of the neural networks was conducted on a 16×16 grid. For a better grasp of the macroscopic patterns, a simulation an a 128×128 grid was run with the pretrained networks from the smaller grid. Both simulations had a maximum of 500 timesteps per episode.

Let us start the discussion with the smaller grid.[34] With Figure 4.6 in mind, we can roughly divide the training phase in three parts:

(i) Initial phase: agents taking actions randomly, first steps of learning; up to episode ≈ 2000.

(ii) "Reign of predators": the predators learned an effective way to hunt, so they are superior to the preys; episode 2000-18000.

(iii) Equilibrium: the preys adapted to the predators hunting methods, so an equilibrium sets in; episodes > 18000

Let us walk through these parts, step by step. (i) This part is rather boring, because the behaviour is mostly random and erratic, due to the random initialisation of the neural networks. Although we can see, that the preys learn rather quickly to receive a stable amount of rewards, until the

[34]Refer to video hunters_gatherers_small.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

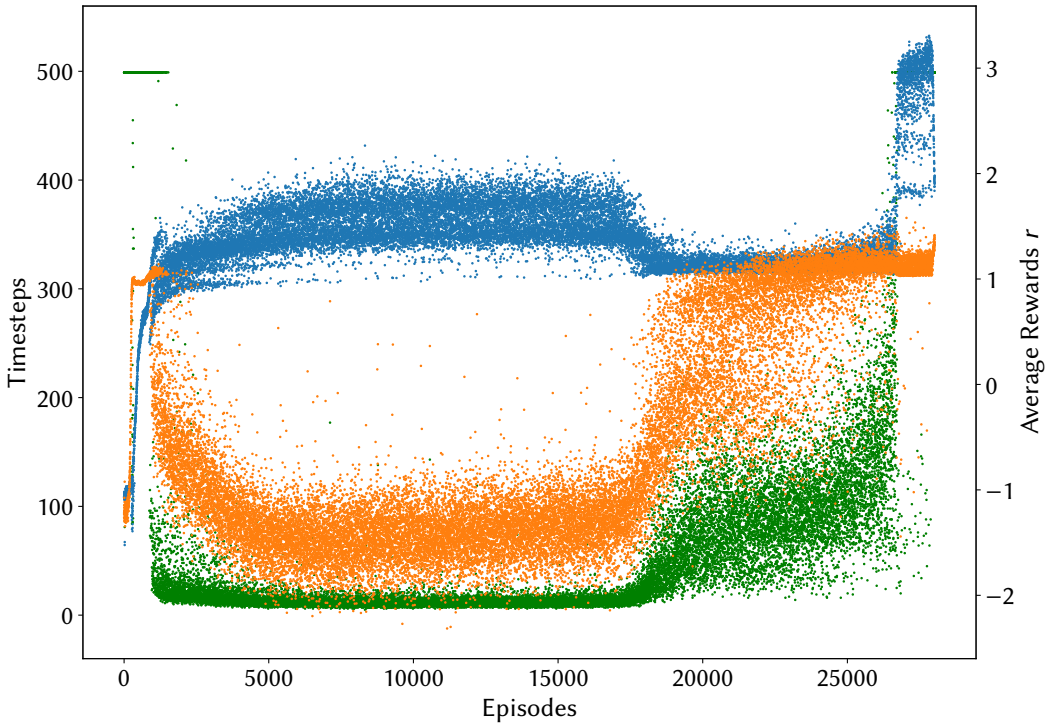


Figure 4.6. Experiment I – Hunters and Gatherers; average rewards of predators (blue) and preys (orange) per agent per episode & number of timesteps (green) per episode. Training on a 16×16 grid.

predators follow. (ii) The predators developed a technique to effectively find and consume prey. This technique consists of moving to the right,[35] gradually taking steps downwards if prey is there, thus "combing through" the grid. Considering the number of timesteps per episode,[36] this technique links to the almost sudden drop in episode length right after the initial phase. This effective method of hunting posed a long period of struggle for the preys. (iii) A key element of survival in this game is moving away from an approaching threat. After the long part of struggling with superior predators, the preys adapted to the predators effective method by running away, still able to eat and reproduce (Figure 4.7). Thus, a stable equilibrium emerged (Figure 4.8).

[35] In a simulation on a 32×32 grid, but else same parameters, the predators developed the same technique, only moving to the left and down. The choice of direction is rather accidental than biased, because all directions are initially equiprobable.

[36] Recall: an episode is over if either the maximum number of timesteps is reached or if one of the species died out.

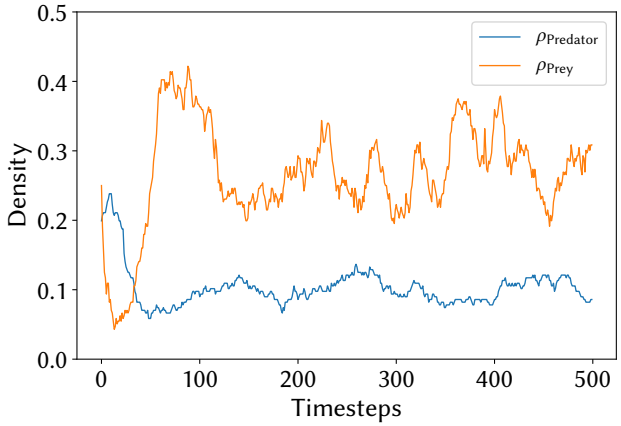


Figure 4.8. Experiment I – Hunters and Gatherers; dynamics in terms of species density fluctuations on a 16×16 grid.

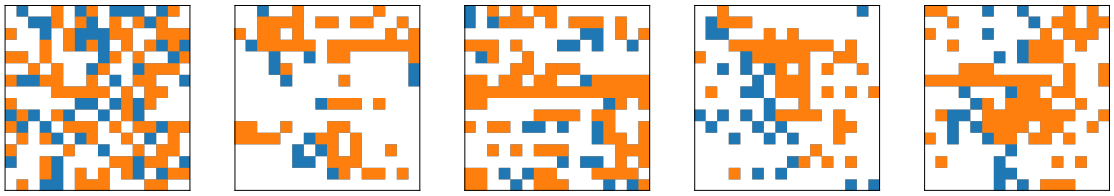


Figure 4.7. Experiment I – Hunters and Gatherers; views of the environment at different time steps on a 16×16 grid; preys are orange, predators blue. (i) initial random distribution, (ii)-(v) at timesteps 50, 100, 150 and 500 respectively.

Macroscopically, we see no resemblance of Figure 4.7 to Figure 3.2, although the shape of the density fluctuations looks promising, apart from much larger variations in this experiment.

Let us now consider the larger grid, 128×128.[37] Since the pretrained networks of the 16×16 simulation were used, we will just focus on the macroscopic patterns and the densities.

Generally, we expect the same density fluctuations, although a "smoother" graph (Figure 4.9 (vi)), because of a higher number of agents. Interestingly, the larger grid is more robust against the predators combing through, than the smaller grid.

A larger grid opens the possibility for better pattern formation. The equilibrium of part (iii) manifests in a wave-like pattern (Figure 4.9 (iii)-(v)) that slowly propagates through the grid. If we compare it to the base model (Figure 3.2), we see a completely different macroscopic pattern, but a similar outcome for the densities, although the total amplitudes are smaller throughout the whole episode.

[37] Refer to video hunters_gatherers_large.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

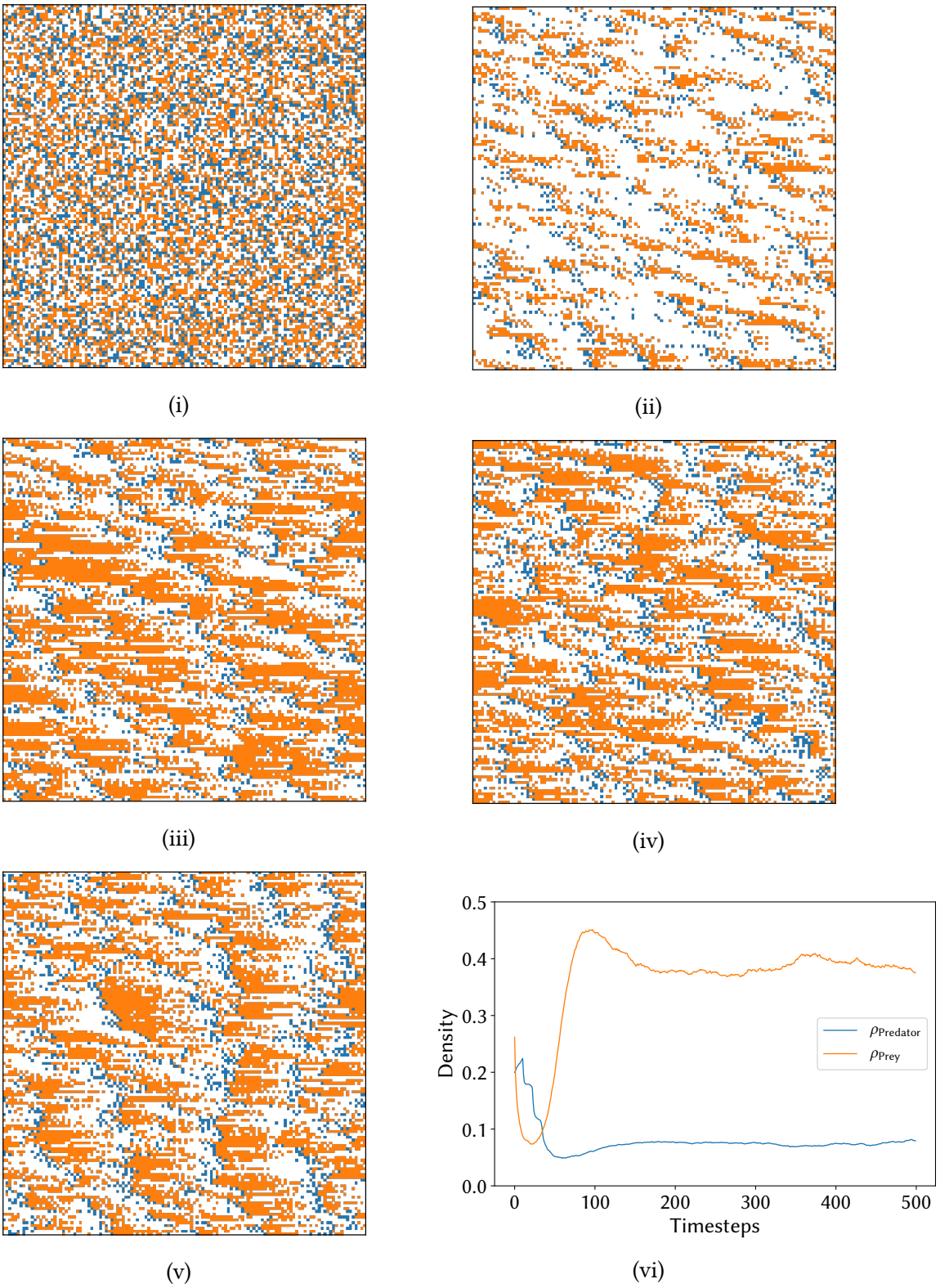


Figure 4.9. Experiment I – Hunters and Gatherers; views of the environment at different time steps on a 128×128 grid; preys are orange, predators blue. (i) initial random distribution, (ii)-(v) at timesteps 50, 100, 150 and 500 respectively. (vi) shows the dynamics in terms of species density fluctuations.

4.2. Experiment II

Every simulation conducted for experiment I showed a directed movement of the predator population in one or two directions to search the grid effectively for prey. We can now ask ourselves, whether similar techniques will arise, if the agents had to actively turn in a specific direction, to create such a polarized movement.

As previously stated (Listing 3.2), some improvements on configuration went in the implementation. A few trial runs were needed to determine sensible values for the `fast` and `satiety` parameters for both species; higher rates for both just resulted in too rapidly dying predators without any detectable change over the course of a few thousand episodes of training.

The only difference in network topology here is an additional input value for orientation, which is done with `affine1: !!python/tuple [2, 2]` and `hidden1: !!python/tuple [149, 64]` in lines 52 and 53 in Listing 4.1, respectively.

4.2.1. On Rewards II

The table of rewards is given below in Listing 4.2. The values are almost similar to the previous simulation, with a small difference in the `indifferent` parameter. Previously, this reward was given only for the action *do nothing*. In this implementation, all actions regarding the orientation, i.e. *turn left, turn right* and *turn around* were also rewarded using this value. This created a rather silly habit of the predators: during the first few thousand episodes of training, they often just remained in their cell and turned left multiple times, basically "idle spinning", instead of searching for food. The networks decision to turn left developed accidentally, because initially all actions were equiprobable. This habit almost disappeared in the later part of the experiment, but rarely it could still be observed.

```
30 rewards:
31   wrong_action:       -1
32   default_prey:       1
33   default_predator:   1
34   indifferent:        0.1
35   succesful_predator: 5
36   offspring:          20
37   death_starvation:   -5
38   death_prey:         -10
```

Listing 4.2. Table of rewards for experiment II.

4.2.2. A Sense of Orientation

We will now explore the properties of this implementation and see, whether we find similar or rather different results. For the used configuration refer to Listing C.3. Let us review the important aspects here again (for more details see section 3.4):

(i) Extent of environment is 16×16 cells.

(ii) Action space $\mathcal{A}$ contains only eight possible actions for both species.

(iii) All individuals inherit a sense of orientation o, which directs their actions.

(iv) The orientation o is randomly initialised in every agent.

(v) This single value piece of information $(o^\flat)$is the second additional input for the neural network (see Figure 3.4).

Considering the setup, we would expect a random behaviour in the first phase, same as before. Following that, I was a bit unsure, what to expect; would the single value input $o^\flat$ be sufficient, to signal the agent's orientation? It turns out, for a period of about 20000 episodes (Figure 4.12), the agents do not seem to utilize their orientation that much (Figure 4.10), however they do manage to survive and a stable equilibrium sets in[38] (Figure 4.11). This again can be seen in the small variation of the average reward distribution during this period, and because all episodes in this episode run the maximum number of timesteps.

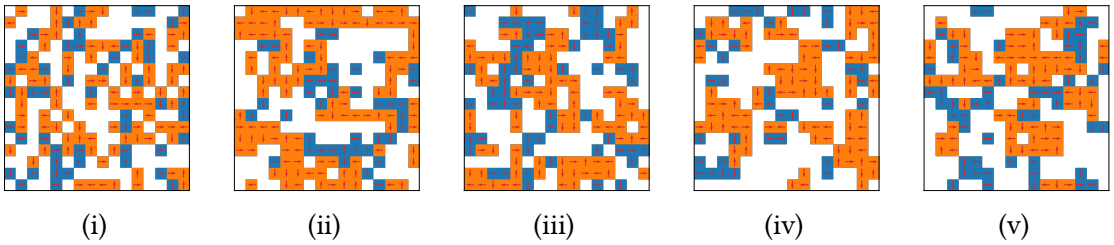


Figure 4.10. Experiment II – a Sense of Orientation, episode 20000; views of the environment at different time steps on a 16×16 grid; preys are orange, predators blue, orientation indicated by red arrows. (i) initial random distribution, (ii)-(v) at timesteps $50,\ 100,\ 150$ and 500 respectively.

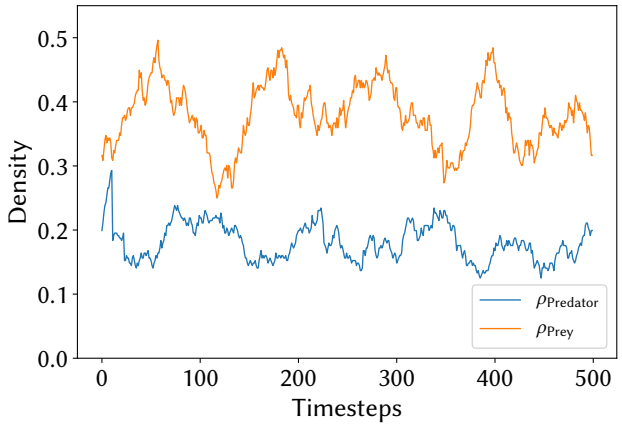


Figure 4.11. Experiment II – a Sense of Orientation, episode 20000; dynamics in terms of species density fluctuations. Note, that there is no transient phase in the beginning: the distribution of agents in the environment is always more or less random, with some fluctuations and clustering.

There appears only one more transition phase in this exact setting around episode 25000. The predators learn again to "comb through" the grid, effectively using their orientation to steer in the optimal direction. As Figure 4.12 indicates, the preys did not learn how to cope with the new predatory scenario. This is also due to the initial population density of the grid: 0.2 and 0.35 for predators and preys, respectively. An abundance in prey population will inevitably[39] lead to an

[38] As in the previous experiments, the first few thousand episodes are rather random, but with first steps of learning visible. Refer to video oriented_episode20000.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

[39] Of course, only if the predators are reasonable hunters.

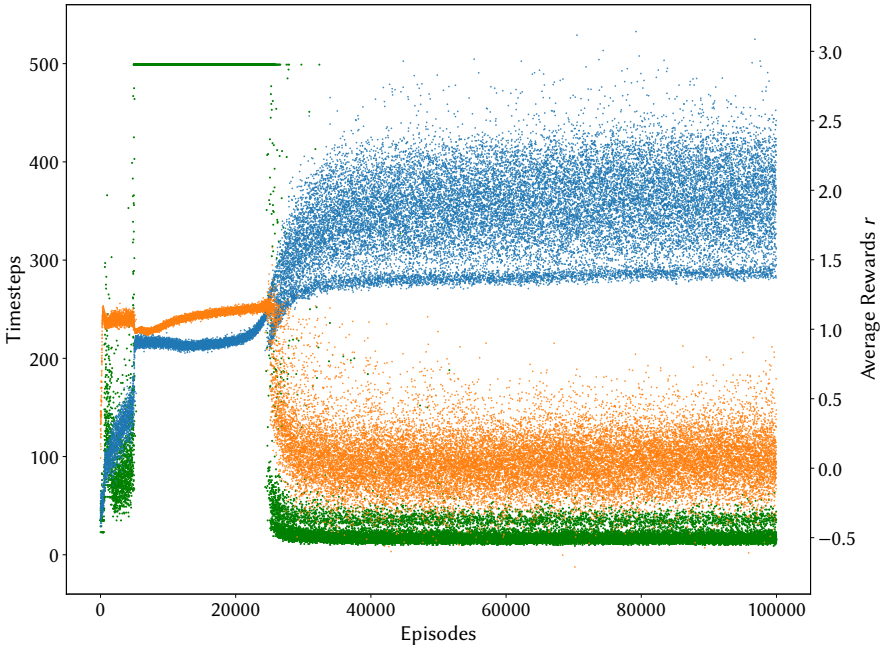


Figure 4.12. Experiment II – a Sense of Orientation; average rewards of predators (blue) and preys (orange) per agent per episode & number of timesteps (green) per episode.

increase in predators at short sight (Figure 2.3) with a subsequent decline of the same, due to a marauded resource[40] (Figure 4.13 and Figure 4.14).

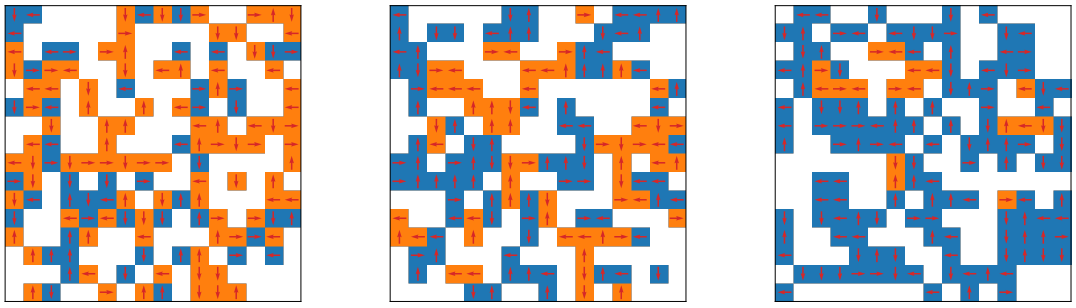


Figure 4.13. Experiment II – a Sense of Orientation, episode 100000; views of the environment at different time steps on a 16×16 grid; preys are orange, predators blue, orientation indicated by red arrows. (i) initial random distribution, (ii)-(iii) at timesteps $5,\ 10$ respectively.

Using the pretrained networks of the 100000 episode run, a small change in initial density of the preys but otherwise unchanged parameters offered no significant change for the preys (Figure 4.15) within subsequent 20000 episodes. The overall average reward for preys increased, while the predators' decreased.

A second change in initial density for both species offered more possibilities: for a few thousand episodes the preys could withstand the predators hunting pressure, resulting again in episodes of maximal length. Towards the end of the additional training phase of 20000 episodes (Figure 4.16

[40]Refer to video oriented_episode100000.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

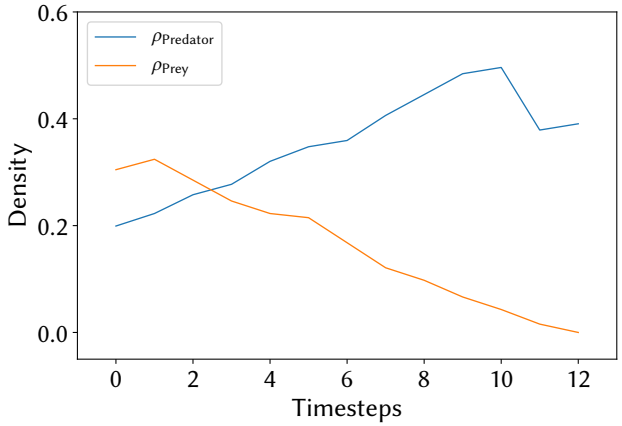


Figure 4.14. Experiment II – a Sense of Orientation, episode 100000; dynamics in terms of species density fluctuations.

and Figure 4.17) the predators tuned their hunting techniques again, effectively clearing the grid of preys again.[41] The data points towards the end show a decreasing trend for preys and increasing trend for predators in terms of average rewards; the behaviour following this trend would have to be studied separately, but was beyond the scope of this thesis unfortunately.

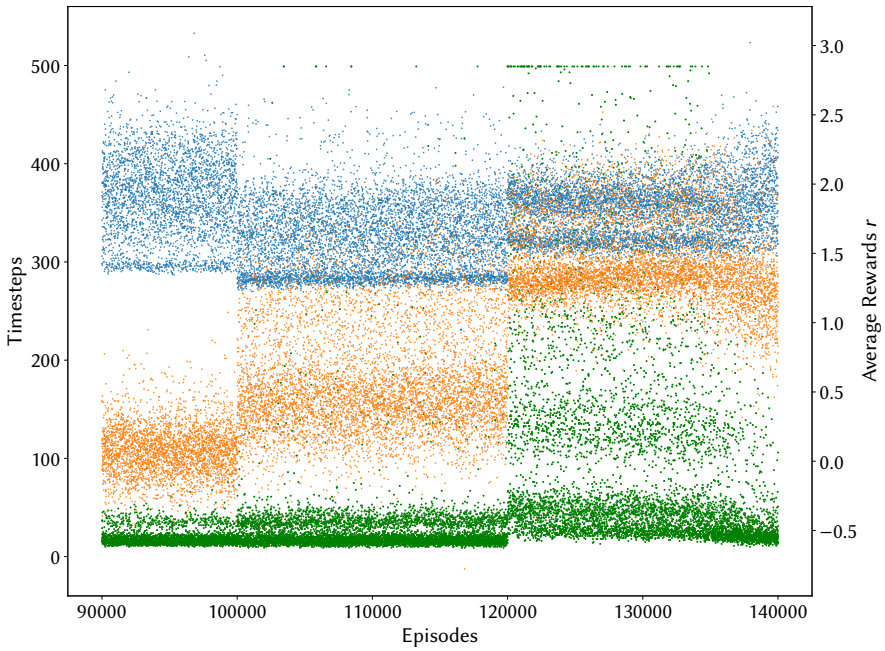
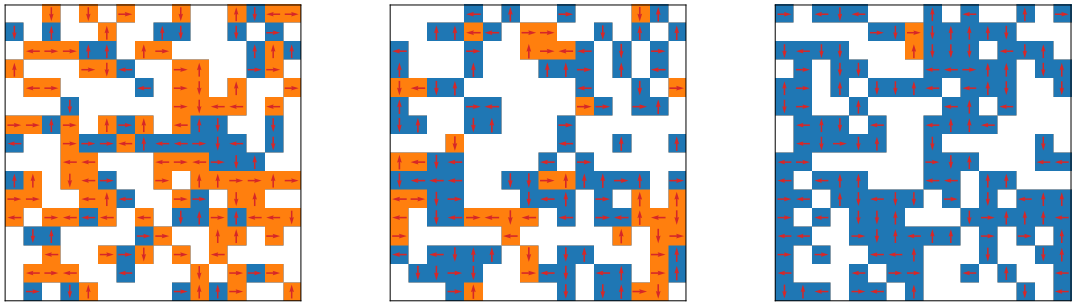


Figure 4.15. Experiment II – a Sense of Orientation; average rewards of predators (blue) and preys (orange) per agent per episode & number of timesteps (green) per episode. The sudden jumps at episode 100000 and 120000 are due to a change in initial density for predators and preys from `[0.2, 0.35]` to `[0.2, 0.2]` and from there to `[0.1, 0.1]`, respectively.

[41] Refer to video oriented_140000.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.



(i) (ii) (iii)

Figure 4.16. Experiment II – a Sense of Orientation, episode 140000; views of the environment at different time steps on a 16×16 grid; preys are orange, predators blue, orientation indicated by red arrows. (i) initial random distribution, (ii)-(iii) at timesteps 5, 10 respectively.

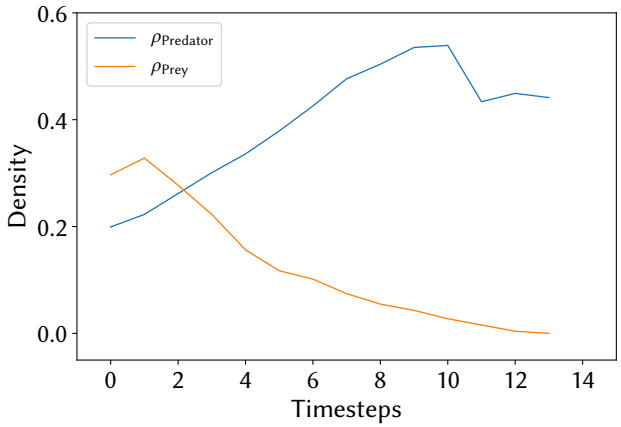


Figure 4.17. Experiment II – a Sense of Orientation, episode 140000; dynamics in terms of species density fluctuations.

4.2.3. Orientation in Isolation

With the same thought in mind as for section 4.1.2, and by using the same parameter changes, we want to look into the behaviour of agents in isolation.[42] The pretrained networks from the first part of experiment II were used to deduct this simulation. Although we have to note, that the agents did not train in this particular environment, with no possibility of starving and a stochastic death. Figure 4.18 captures the result quite nicely: the overall population growth is rather slow; the predator moves in just one direction, but turns around around timestep 150. Although the predator should "see" the preys due to the periodic boundary conditions, it keeps moving straight. Once the prey cluster grows close enough to the predator, in timestep 170, the predator starts to hunt. Then, within just 30 timesteps it reproduces a few times, thus reducing the prey population down to four individuals, which also find a quick end. The densities (Figure 4.19), show a certain similarity to the densities in the previous isolated agents simulation, (left) in Figure 4.4. Apparently, the predators tend to search for prey in the sense of "moving straight", until a prey is close enough, i.e. in one of the neighbouring cells.

[42]Refer to video oriented_isolation.mp4 on the supplied medium or on https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil/tree/master/videos.

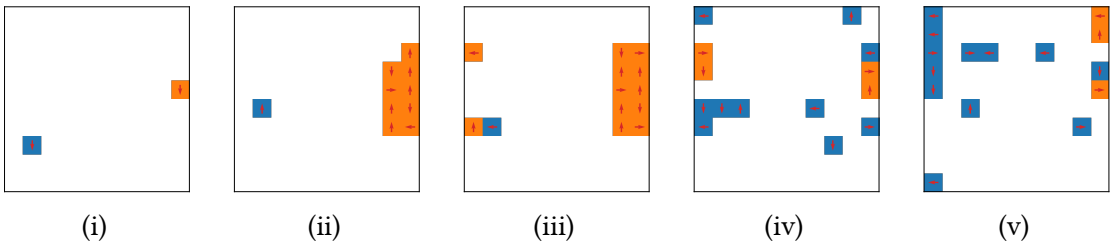


Figure 4.18. Experiment II – Orientation in Isolation; views of the environment at different time steps on a 10×10 grid; preys are orange, predators blue, orientation indicated by red arrows. (i) initial random distribution, (ii)-(v) at timesteps $150,\ 170,\ 200$ and 210 respectively. Timestep 210 actually is the last in this simulation, because the predators eat the three remaining prey within the next timestep.

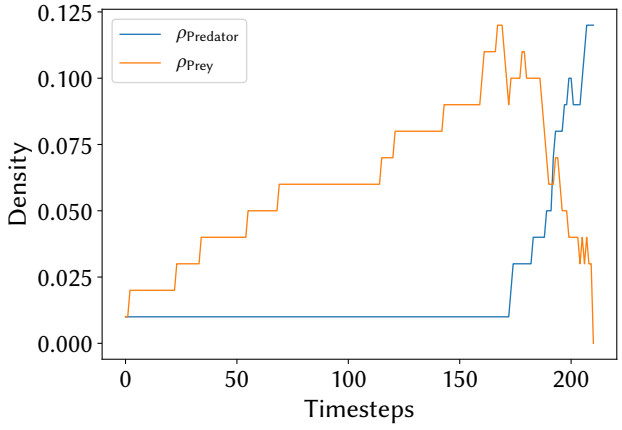


Figure 4.19. Experiment II – Orientation in Isolation; dynamics in terms of density fluctuations. Note the similarity to Figure 4.4 (left), although the graph is more discontinuous here.

5

Discussion & Outlook

The first real point of discussion would be the network topology. The used implementation was inspired by the work of [*Leibo* et al. 2017] and adjusted to the needs of the experiments, in terms of inputs and outputs. More research and testing should be done, to investigate a better network architecture. In the current state, agents are not able to remember the last state they were in; as explained in chapter 2, this is not necessary. But it may help the agents in their decision, if they could remember their last steps.

All training runs were conducted on rather small grid sizes, e.g. 32×32, 16×16 and 10×10. There is no guarantee that the learned behaviour would develop in a similar way on larger grids. Due to limited computational resources, training on larger grids was not possible unfortunately.

It is difficult to say when to stop the training process. The agents might learn something in the far future or may have already found their optimum. Take the example of experiment I: the increase in average reward for the preys was tiny, over the course of 10000 episodes. Being patient paid off for me in all cases, but would being more patient also result in further interesting observations?

The value set of rewards was shaped by all examples of reinforcement learning I found one the one hand, and by tweaking the values to test their effect on the other hand. Using differently scaled values ma result in different strategies.

To test the networks confidence, i.e. the distribution of output probabilities $\mathbf{p}$ of actions $\mathbf{a}$ for a given state s, the entropy of the network for that state can be calculated using the *Shannon Entropy* $H(s) = -\sum_i p_i \log_b p_i$, where p_i are the components of the probability vector $\mathbf{p}(s)$, and b is the base of the logarithm (usually 2, e or 10). An untrained network will have a high entropy, because all actions are equiprobable; as the network improves in its performance, the distribution will develop a maximum for the optimal action in the given state, and thus yielding a smaller entropy.

Implementing more than two species or a non-abundant resource R will affect the dynamics directly; a non-abundant resource R represents a changing environment. These changes could be more drastic, e.g. some parts of the grid become inhabitable. The question is, whether the agents would be able to adapt to a changing environment. Furthermore, this model is restricted to a single individual per cell. For more than two species it may be a good idea to lift this restriction, to allow for possible different patterns to emerge.

One of the largest issues was computing time. The codebase was written in Python, without any low level C-programming with Cython. Reimplementing the model using either Cython or C/C++ would decrease the computational effort significantly. The issue would then shift to the usage of PyTorch, which would have to be replaced by e.g. Caffe.

With shorter simulation runtimes, better parameter sweeps could be conducted for described experiments. Additionally, the above mentioned changing environment would pose a smaller issue in regard to computational effort again.

Generally, the goal of this thesis was to explore possible ways of implementing learning agents in the setting of a predator-prey model. This turned out to be possible, although with a few obstacles in the way regarding the machine learning part. As already mentioned and hinted at in the comic on page ii, it is very difficult to make a choice about the network topology and the rewards table, especially for someone who is new in the field.

There are a lot of possible paths following this one, with variations in the network topology leading the way. Introducing either LSTM-units into the existing structure or rebuilding the structure using a recurrent neural network may be good starting points. Some notion of memory would be beneficial for both species.

Bibliography

Adamatzky, A. (2010). *Game of life cellular automata* (1st). Springer Publishing Company, Incorporated.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Berlin, Heidelberg: Springer-Verlag.

Cassandra, A. R. (1994). *Optimal policies for partially observable markov decision processes*. Providence, RI, USA: Brown University.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314. doi:10.1007/BF02551274

Darwin, C. & *Wallace*, A. (1858). On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection. *Zoological Journal of the Linnean Society*, *3*(9), 45–62. doi:10.1111/j.1096-3642.1858.tb02500.x

Gilman, R. (2018). Intuitive rl: Intro to advantage-actor-critic (a2c). Retrieved June 30, 2018, from https://hackernoon.com/intuitive-rl-intro-to-advantage-actor-critic-a2c-4ff545978752

Goodfellow, I., *Bengio*, Y. & *Courville*, A. (2016). *Deep learning*. http://www.deeplearningbook.org. MIT Press.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. doi:https://doi.org/10.1016/0893-6080(91)90009-T

Karpathy, A. (2018). Cs231n: Convolutional neural networks for visual recognition. Retrieved June 30, 2018, from https://cs231n.github.io/convolutional-networks/

Kochenderfer, M. J., *Amato*, C., *Chowdhary*, G., *How*, J. P., *Reynolds*, H. J. D., *Thornton*, J. R., … *Vian*, J. (2015). *Decision making under uncertainty: Theory and application* (1st). The MIT Press.

Krautgasser, F. (2018). Code base of bachelor thesis. Retrieved June 30, 2018, from https://ts-gitlab.iup.uni-heidelberg.de/fkrautga/Imazalil

Leibo, J. Z., *Zambaldi*, V. F., *Lanctot*, M., *Marecki*, J. & *Graepel*, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, *abs/1702.03037*. arXiv: 1702.03037. Retrieved from http://arxiv.org/abs/1702.03037

Lotka, A. J. (1920). Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences*, *6*(7), 410–415.

Malthus, T. (1798). *An essay on the principle of population*. London: J. Johnson.

Mnih, V., *Kavukcuoglu*, K., *Silver*, D., *Graves*, A., *Antonoglou*, I., *Wierstra*, D. & *Riedmiller*, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, *abs/1312.5602*. arXiv: 1312.5602. Retrieved from http://arxiv.org/abs/1312.5602

Munroe, R. (2017). Machine learning. Retrieved June 30, 2018, from https://xkcd.com/1838/

Paszke, A., *Gross*, S., *Chintala*, S., *Chanan*, G., *Yang*, E., *DeVito*, Z., … *Lerer*, A. (2017). Automatic differentiation in pytorch. In *Nips-w*.

Roth, K. (2017). *Chaotic, complex & evolving environmental systems – lecture notes* (Version 0.3.). Institute of Environmental Physics, Heidelberg University, Germany.

Sutton, R. S. & *Barto*, A. G. (1998). *Introduction to reinforcement learning* (1st). Cambridge, MA, USA: MIT Press.

Verhulst, P. R. (1838). *Correspondance mathématique et physique*. Impr. d'H. Vandekerckhove. Retrieved from https://books.google.at/books?id=8GsEAAAAYAAJ

Volterra, V. (1926). Fluctuations in the abundance of a species considered mathematically. Nature Publishing Group.

Weber, J. (2015). *Simulation of biological two- and three-species systems using cellular automata* (Msc. thesis). Institute of Environmental Physics, Heidelberg University, Germany.

Appendix

A. Deep Reinforcement Learning

A.1. Smooth L^1 loss

A commonly used loss function is the L^1-loss. It is used to minimize the error which is the sum of all absolute differences between ground truth (GT) and predicted values; more formally:

$$L^1(y_{\mathrm{GT}}, y_{\mathrm{predicted}}) \coloneqq \sum_{i=1}^{N} |y_{\mathrm{GT}} - y_{\mathrm{predicted}}| \tag{A.1}$$

In analogy, the L^2-loss is used to minimize the error which is the sum of all squared differences between GT and predicted values:

$$L^2(y_{\mathrm{GT}}, y_{\mathrm{predicted}}) \coloneqq \sum_{i=1}^{N} [y_{\mathrm{GT}} - y_{\mathrm{predicted}}]^2 \tag{A.2}$$

In most cases the L^2-loss is preferred, but if the data set contains outliers the loss performs bad because of the squared difference. In this case, the L^1-loss is the better choice, as it's not affected (to the same extent) by the outliers.

Going further, we can combine the positive sides of both losses, and define the *smooth L^δ loss* (also called the *Huber loss*):

$$L^\delta_{\mathrm{smooth}}(\mathbf{y}_{\mathrm{GT}}, \mathbf{y}_{\mathrm{predicted}}) \coloneqq \sum_{i=1}^{N} z_i(\delta) \tag{A.3}$$

$$\text{with } z_i(\delta) = \begin{cases} \frac{1}{2}\left[y^i_{\mathrm{GT}} - y^i_{\mathrm{predicted}}\right]^2 & \text{if } |y^i_{\mathrm{GT}} - y^i_{\mathrm{predicted}}| < \delta \\ \delta|y^i_{\mathrm{GT}} - y^i_{\mathrm{predicted}}| - \frac{1}{2}\delta & \text{otherwise.} \end{cases}$$

Superscript i is an index.

In this thesis, I just used $L^\delta_{\mathrm{smooth}}$ with $\delta = 1$, so for convenience I dropped the subscript label and call this loss function *smooth L^1-loss* in the following.

B. Model Implementation

```yaml
Grid:
    NX: 256             # number of X gridcells
    NY: 256             # number of Y gridcells

Sim:
    FastForward: 100    # plot just every nth time step
    Timesteps: 500      # number of timesteps per epoch
    RhoPred: 0.2        # density of predators
    RhoPrey: 0.215      # density of preys

Prey:
    Pflee: 0.4          # probability to flee
    Pbreed: 0.2         # prob to reproduce
    FoodReserve: 4      # initial food reserve
    FoodReserveMax: 8   # maximum food reserve

Pred:
    Pbreed: 0.2         # prob to reproduce
    FoodReserve: 4      # see above
    FoodReserveMax: 8   # see above
```

Listing B.1. Parameters for basis simulation.

C. Results

```yaml
Model:
    dim:                !!python/tuple [16, 16]  # dimensionality
    densities:          !!python/tuple [0.2, 0.35]  # Predator, Prey
    food_reserve:       3
    max_food_reserve:   8
    generation:         0
    neighbourhood:      49  # only squares of odd numbers, e.g. 9, 25, 49
    p_breed:            1.0
    p_flee:             0.0
    p_eat:              1.0  # = 1-p_flee
    mortality:          True
    instadeath:         0.000
    rewards:
      wrong_action:     -3
      default_prey:     1
      default_predator: 1
      indifferent:      0
      succesful_predator: 5
      offspring:        20
      death_starvation: -5
      death_prey:       -10
      default:          1
      instadeath:       0
```

```
Network:
    kind:               'conv'
    mode:               'cpu'  # can also be 'gpu'
    layers:
      conv1:
          in_channels:  1
          out_channels: 3
          kernel_size:  3
          padding:      1
          stride:       1
      affine1:          !!python/tuple [1, 1]
      hidden1:          !!python/tuple [148, 64]
      hidden2:          !!python/tuple [64, 32]
      hidden3:          !!python/tuple [32, 32]
      action_head:      !!python/tuple [32, 15]
      value_head:       !!python/tuple [32, 1]
    gamma:              0.9  # discount factor
```

Listing C.2. Parameters for experiment I – Hunters and Gatherers.

```
Model:
    dim:                  !!python/tuple [16, 16]  # dimensionality
    densities:            !!python/tuple [0.2, 0.35]  # Predator, Prey
    food_reserve:         3
    max_food_reserve:     8
    metabolism:
      OrientedPredator:
        fast:             0.25  # decrease every timestep
        satiety:          3  # get for eating
        exhaust:          3  # for mating
      OrientedPrey:
        fast:             0.5
        satiety:          2
        exhaust:          3
    view:                 !!python/tuple [7, 7]  # supersedes neighbourhood
    generation:           0
    p_breed:              1.0
    p_flee:               0.0
    p_eat:                1.0  # = 1-p_flee
    mortality:            True
    instadeath:           0.000
```

```
kind:               'conv'
mode:               'cpu'  # can also be 'gpu'
layers:
  conv1:
      in_channels:  1
      out_channels: 3
      kernel_size:  3
      padding:      1
      stride:       1
  affine1:          !!python/tuple [2, 2]
  hidden1:          !!python/tuple [149, 64]  # 3x7x7 + 2
  hidden2:          !!python/tuple [64, 32]
  hidden3:          !!python/tuple [32, 32]
```

```
  action_head:      !!python/tuple [32, 8]
  value_head:       !!python/tuple [32, 1]
gamma:              0.9  # discount factor
```

Listing C.3. Parameters for experiment II – a Sense of Orientation.

Acknowledgements

This thesis would not have been possible without the help of many people. First, I'd like to thank all people in the group, especially Yunus, Benni, Josi, Chris and Hendrik, for wonderful discussion rounds and input, regarding almost every aspect of this thesis. Furthermore, I'm grateful for all coffee meetings with Johannes, who's knowledge about machine learning and neural networks was a much appreciated help.

Of course, my gratitude goes to my supervisor Prof. Kurt Roth for being available for many weird questions and possible interpretations of my results, during the time of the thesis and to Prof. Fred Hamprecht for being the second reviewer.

And last but not least, a big thank you to my family and my flat mates, for bearing with me and enduring my frequent grumpiness because of dysfunctional pieces of code.

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 1. Juli 2018

. . . . . .

(Fabian Krautgasser)