Double Deep Q-Networks for Finding Sustainable Policies in the World3 Model

Emil Hed and Ivar Karlsson

***Abstract*—This project explores the integration of reinforcement learning with the World3 model, as detailed in the book Limits to Growth [1], and aims to provide insights that could guide further research for decisions towards global well-being and extended life expectancy. The World3 model breaks down global dynamics into subsystems such as agriculture, capital, pollution, population, and resources. Utilizing PyWorld3, a Python-based implementation of the World3 model, we facilitate dynamic simulations of the model. By integrating reinforcement learning techniques, specifically Double Deep Q-learning, we can assess and potentially alter the current global condition. Our results show that our reinforcement learning agent successfully interacts and learns effectively within the models framework. However, the outcomes are subject to limitations, including the selection of hyper-parameters, computational constraints, the design of the reward system, and because of simplifications and constraints within the World3 model itself.**

***Sammanfattning*—Denna rapport utforskar integrationen av Reinforcement Learning med World3-modellen, som beskrivs i boken Limits to Growth [1], och syftar till att ge insikter som kan vägleda ytterligare forskning för beslut mot globalt välbefinnande och förlängd livslängd. World3-modellen bryter ner global dynamik i delsystem som jordbruk, kapital, föroreningar, befolkning och resurser. Genom att använda PyWorld3, en Python-baserad implementering av World3-modellen, utför vi dynamiska simuleringar av modellen. Genom att integrera Reinforcement Learning tekniker, särskilt Double Deep Q-learning, kan vi bedöma och eventuellt förändra det nuvarande globala tillståndet. Våra resultat visar att vår agent för Reinforcement Learning framgångsrikt interagerar och lär sig effektivt inom modellens ramar. Resultaten är dock föremål för begränsningar, inklusive valet av hyperparametrar, beräkningsbegränsningar, utformningen av belöningssystemet och på grund av förenklingar och begränsningar inom själva World3-modellen.**

***Index Terms*—Reinforcement learning, World3, system dynamics, DQL,**

Supervisors: Matthieu Barreau, Jonas Mårtensson, Elisa Bin

TRITA number: TRITA-EECS-EX-2024:133

I. Introduction

The intersection of environmental sustainability and public health is a critical area of research, especially in the context of the United Nations Sustainable Development Goals (SDGs) [2]. Among these, SDG 3, "Good health and well-being", is necessary for societal progress. This project presents an approach to addressing SDG 3 using reinforcement learning (RL). In this project the goal is to find policies leading to a sustainable outcome of the World3 model, a simulation model that depicts the interconnections between global population, resource utilization, and pollution generation, to name a few, originally developed as part of the Limits to Growth study [3]. The hope is that the results in this study can serve as pointers on how different policies affect the world and serve as a pre-study for continued research.

The World3 model has been a powerful tool for understanding the long-term patterns of human civilization and their impact on the planet's ecological systems. However, with advanced computational techniques and the growing urgency to address complex global challenges, there is an opportunity to enhance the predictive capabilities and relevance of the model. In this project RL, a subset of machine learning focused on making sequences of decisions, is employed to dynamically adjust the World3 model parameters in response to evolving conditions to optimize health and well-being outcomes.

The primary contribution of this study lies in its methodology, which combines the complex, dynamic world3 system modeling with the adaptive, goal-oriented optimization power of RL. In doing so, the objective is to uncover new insights and strategies for achieving global health objectives.

In the following sections, the technical foundations of the World3 model and RL are introduced, the methodology for integrating these tools is described, the results from the simulation experiments are presented, and the implications of the findings for policy-making and future research are discussed.

II. Problem Statement

The objective of this work is to employ control theory within a model that mimics global dynamics, aiming to optimize outcomes related to a specific United Nations sustainability goal. We have opted to use RL for this purpose. RL is a highly effective method for managing dynamic systems, as evidenced by numerous previous studies [4].

The challenge involves developing a method to assess the current global status within the model by analyzing key variable values. Additionally, it is necessary to determine the most appropriate policies to implement in various simulated states to enhance overall well-being and life expectancy.

III. The World3 model

The World3 model is a dynamic system designed to project future global conditions by breaking down essential characteristics into five distinct subsystems: agriculture, capital, pollution, population, and resources. Each subsystem is defined by several metrics intended to reflect its current status. Between these subsystems, there are intradependencies as well as interdependencies.

The model that is used in this project is an implementation of the World3 model in Python called *PyWorld3* [5]. The original World3 model that PyWorld3 is based on is from the book: *Dynamics of Growth in a Finite World.* [3].

The PyWorld3 model comprises 314 distinct constants, initial values, and variables, which are collectively referred to as metrics. At the start of a simulation, these initial values are used to set up all variables. Once initialized, the model iteratively updates each variable at each time step, the duration of which can be adjusted according to the simulation requirements.

The update of each variable during a cycle is calculated using a mathematical formula that integrates other variables and constants. Every variable is governed by a unique equation that illustrates its dependence on other variables and constants. For instance, the variable representing the land used for industrial and urban purposes (*uil*) is updated as follows:

$$uil[k] = uil[k] + dt \cdot lrui[k]$$

where [k] represents the variable's value at the simulation's time step k. This formula demonstrates how *uil* is adjusted based on its previous value and land removal for industrial uses (*lrui*), scaled by the time elapsed (*dt*) since the last update.

Similarly to *uil*, every variable in the model is described by an equation that details how it evolves based on the value of other variables in the model. While many relationships in the model, such as *uil's*, are linear and intuitive, not all real-world interactions are straightforwardly linear. For example, changes in healthcare funding don't immediately impact life expectancy because it takes time to build the infrastructure that will improve health outcomes. The model captures these delays using first- and third-order delay filters, which help to phase in the impacts of certain variables gradually, thereby enhancing the model's realism.

Furthermore, the model incorporates table functions to establish non-linear relationships. These functions use a set of data points and interpolate between them to create a mapping from the x values to the y values, allowing simulation of nonlinear dynamics.

In this project, we were given a modified version of the World3 model. In this version feedback loops were introduced into some of the mathematical equations. This adjustment introduced the ability to manipulate outcomes by altering control values, offering enhanced interaction with the simulated world dynamics.

IV. Reinforcement Learning

A. Fundamentals of Reinforcement Learning

RL is a subset of control theory focused on learning through trial and error. In this framework, an agent is placed within an environment characterized by various states, denoted as s. In each state, the agent can take an action, a. For every every action the agent takes it ends up in a new state. Each state is associated with a reward, r, which guides the agent's learning process. This action then leads to a new state, s', continuing the cycle. Through this iterative process, the agent learns to navigate the environment by maximizing cumulative rewards over time, effectively learning the best actions to take in different states based on the outcomes of its previous actions. An illustration of RL in action can be seen when someone learns to play tic-tac-toe for the first time. Initially, the player is only informed about the basic objective: to align three marks in a row, column, or diagonal to win the game. However, with each subsequent game, the player gains valuable experience. For example, they might discover that placing the first mark in the center of the grid often increases the chances of winning. Over time, through trial and error, the player starts to identify and adopt more effective strategies, learning from past outcomes to improve future performance. This process of iterative learning and strategy refinement is analogous to how RL algorithms optimize decisions over time in various applications.

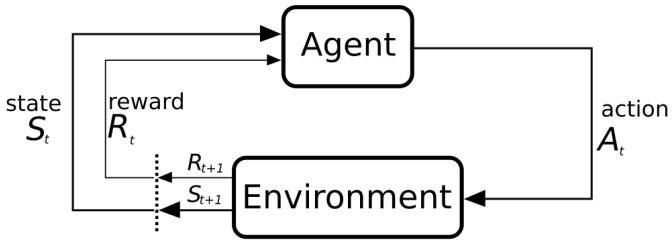

Fig. 1. Figure of Agent-Environment interaction. Source: https://commons.wikimedia.org/wiki/File:Markov_diagram_v2.svg

B. Markov Decision Process

A fundamental prerequisite in RL is the use of Markov Processes (MPs). What defines an MP is the Markov Property. This property implies that future dynamics are independent of the past, given the present state. Hence making the process memoryless [6]. In its simplest form, an MP can be mathematically described by state transition probabilities:

$$P(s'|s) = \Pr(S_{t+1} = s'|S_t = s) \tag{1}$$

Where $P(s'|s)$ is the probability of transitioning to state s' from state s.

Extending this concept, a Markov Decision Process (MDP) incorporates actions and rewards into the framework, making it suitable for modelling decision-making tasks where each action taken by an agent affects the state transitions and outcomes. The MDP can be described using the transition probabilities:

$$P(s', r|s, a) = \Pr(R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a) \quad (2)$$

Here, $P(s', r|s, a)$ represents the probability of transitioning from state s to state s' and receiving reward r after taking action a. This transition probability is a core component of the dynamics in RL [6].

In the context of the World3 we can define the system's state using variables such as *population*, *industrial output*, etc. Actions respresented by the different control signals incorporated into the model equations such as *fraction of industrial output allocated to agriculture* and the rewards designed to align with the set objective. Because of inertia in the model caused by the delay functions built in to the model, it does not strictly have the Markov property. The next states will always depend on the the current state and some previous state. Therefore, in this project we approximate the World3 model as an MDP for the purpose of applying reinforcement learning.

C. Bellman Equation

MDPs are often combined with the Bellman equation in RL contexts. The Bellman equation is a fundamental recursive optimization equation that is used to compute a scalar-value function in an MP [6]. It can be expressed as:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s'). \quad (3)$$

where $V^{\pi}(s)$ is the expected reward for being in state s and following a fixed policy π, $R(s, \pi(s))$ represents the immediate reward received when entering state s, and γ is the discount factor, see IV-F. The equation for following a optimal policy for maximized reward can be expressed as:

$$V^{\pi *}(s) = \max_{a} \left\{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{*}(s') \right\}. \quad (4)$$

Here $\pi *$ is some optimal policy and $V^{\pi *}(s)$ is the expected reward. This equation fundamentally describes how an action taken in a specific state influences the variable $V(s)$, balancing immediate and future rewards.

To utilize the Bellman equation effectively, one must know the rewards associated with different state-action pairs, which require defined state and action spaces.

D. Q-learning

Q-learning is a RL algorithm seeking the best action to take given the current state by storing the value function for each state-action pair in a Q-table. The Q-learning agent learns from interacting with its environment. By exploring different actions given the current state, it develops a decision making policy that can be used at a later stage. The objective of Q-learning is to learn a policy, and tell the agent what action to take in what state.

Q-learning finds an optimal policy by maximizing the expected value of the total reward over all steps, starting from the current state. Q-learning works by estimating the values of the optimal action-value function, which is the maximum expected return achievable after taking an action in a given state [6]. The action-value function $Q(s, a)$ is updated as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \quad (5)$$

where:

- s is the current state,
- a is the action taken,
- r is the reward received after taking action a in state s,
- s' is the new state after action a,
- a' is the next possible action,
- γ is the discount factor, and
- α is the learning rate.

This equation is applied every time an action is taken and the agent receives feedback from the environment. Over time, $Q(s, a)$ converges to the optimal action-value function $Q^*(s, a)$ which prescribes the best possible action for every state [6].

1) Discount Factor γ : The discount factor, denoted by γ, assumes a value between zero and one, shaping the agent's perception of the value of future rewards. A γ near zero renders the agent myopic, focusing solely on the immediate rewards. Conversely, a γ closer to one equips the agent with a farsighted approach, emphasizing the significance of long-term rewards and the strategic planning of actions. This balance is crucial for guiding the agent's behaviour toward a blend of immediate and future objectives.

2) Learning Rate α: The learning rate α is a crucial hyperparameter in the update process of the Q-function within Q-learning. It dictates the degree to which new information impacts and adjusts previously learned Q-values. When α is set low, the agent incrementally assimilates new experiences, resulting in minimal adjustments to the Q-values and a consequently gradual learning trajectory. Conversely, an excessively high α may cause drastic alterations in Q-values from single experiences, potentially resulting in an unstable learning process where the agent continuously oscillates and struggles to converge to the optimal policy.

3) Exploration - Exploitation: The end goal in Q-learning is for the agent to work under a policy that maximizes the reward over a long chain of decisions. To learn this policy the agent needs to explore its environment to learn this pogrouplicy. When an agent is exploring, it does not take the Q-value of an action into account when performing that

action. The action is taken purely based on learning the long-term and short-term consequences of that action.

To find the universally best chain of decisions, the agent would need to explore the whole environment. Exploring all the different permutations of the sequence of decisions can be computationally inefficient. Therefore, it is necessary to limit the agent to a certain amount of exploration before it can start to be utilized in the sought-after way of trying to choose the optimal action at every step of the decision chain.

However, if the agent is not allowed to explore enough, it will not have the information needed to make informed decisions on what the best action would be. This balance act is known as the exploration-exploitation dilemma, a prevalent challenge in data-driven decision-making [7].

4) ϵ - greedy policy: The epsilon-greedy policy is a common strategy used to address the exploration-exploitation dilemma. In this approach, the agent primarily exploits by choosing the action that it currently believes to have the highest expected reward. However, with a probability epsilon ϵ, it will instead randomly select any available action, thereby exploring the environment. The value of ϵ typically starts high to encourage more exploration at the beginning when the agent knows little about the environment. Over time, ϵ is gradually reduced, increasing the focus on exploitation as the agent becomes more confident in its knowledge of the rewards associated with different actions [6].
This method ensures a balance between exploring enough to discover potentially better strategies and exploiting known strategies to accumulate rewards. It integrates both greedy actions, which always select the best-known option for immediate and known future reward, and non-greedy actions, which explore other possibilities that might lead to higher future rewards. Thus, the epsilon-greedy policy provides a practical framework for managing the exploration-exploitation trade-off effectively.

E. Deep Neural Networks

Deep Neural Networks (DNN) are sophisticated computational frameworks designed to mimic the human brain's architecture, using interconnected nodes or neurons in a layered structure [8]. Central to the field of machine learning, neural networks excel in managing complex problem domains such as image recognition, natural language processing, and advanced decision-making systems, thereby serving as the foundation of deep reinforcement learning techniques.

DNNs comprise multiple layers of interconnected nodes, known as neurons, arranged in a structured way. To illustrate the communication process among neurons in the body, consider the following example: sensory neurons in the eye are stimulated by environmental input. These neurons transmit signals to adjacent neurons, activating them. This activation

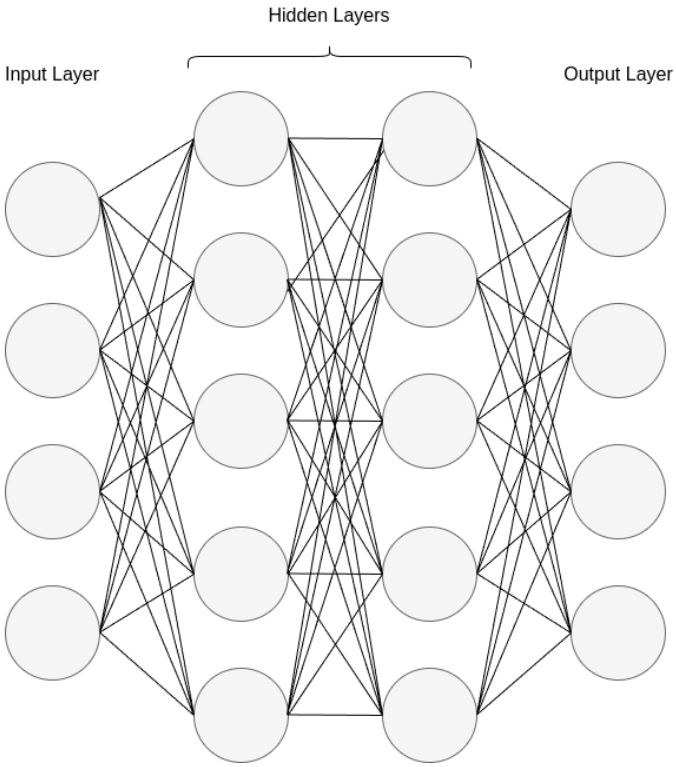

Fig. 2. Structure of a Neural Network Architecture

continues along a lengthy chain of interconnected neurons until the signal reaches the brain. Upon receiving the signal, the brain interprets it and generates a visual representation. This sequential interaction among neurons is the basis for the design of neural networks that aim to replicate this complex pattern of neural connectivity.

In a neural network, there is an input layer that takes in an arbitrary number of inputs. These inputs are then passed through a layer of nodes called the hidden layer. If there is more than one hidden layer, the network is often referred to as a deep neural network (DNN). At the end of the network, there is an output layer that outputs the now-processed input signals. In the human analogy, the neurons were described as activating each other. The activating of neurons in the human anatomy is emulated by a mathematical expression in the neural network that depending on the inputs and the inner properties outputs a specific signal. Every node has associated weights and a bias that creates a linear combination together with all its input signals. The bias is a constant that is independent of the inputs. This linear combination is then passed through an activation function. The activation function is non-linear, allowing there to be non-linear dependencies between the input and output signals. The values of each node's weights are what decide the final output of the neural network. To create a network that can successfully produce the sought-after input from a certain output, they need to be modified to a specific combination. This combination is achieved after training the neural network. Training the neural network is done by fitting a specific input to a specific output. Fitting is done by back-

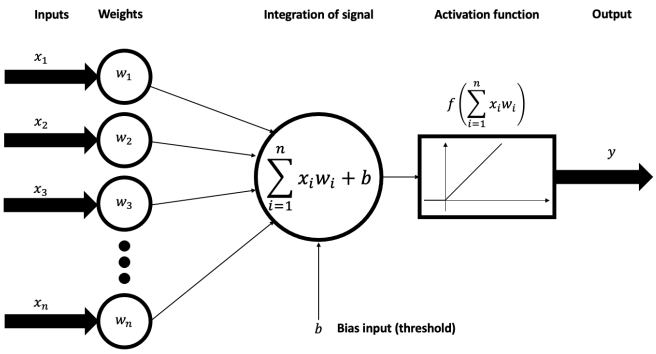

Fig. 3. The inner architecture of a node in a neural network.

propagating through the network, starting from the output signals. Back propagating is a concept that can be briefly described by adjusting the weights of each node depending on the gradient of that node concerning the output nodes. The process of fitting data is repeated until the network can correctly predict the output from an input with an acceptable margin for error. When this process is done, the weights in the network are changed to their desired values. The network is at this stage ready to be used on any data, not just data on which it has been trained.

F. Deep Q-Learning

The core idea behind Q-learning is to learn the Q-value associated with taking a particular action in a given state, capturing these values in what is called a Q-table. However, for environments where states and actions are numerous or continuous, maintaining a discrete Q-table becomes impractical.

Deep Q-Learning (DQL) addresses this challenge by using a Deep Neural Network (DNN) to approximate the Q-values. To train a Deep Q-Network (DQN), the agent needs to interact with the environment to collect experiences throughout a number of episodes. Given a state s the agent takes an action a determined through a epsilon-greedy policy. The current state, reward received from the taken action, the action taken and the next state are saved every time an action is taken. A dataset that contains this information is called an experience. All experiences are stored in a replay memory, managed as a first-in-first-out queue to maintain relevance, with older experiences removed as new ones are added. When the agent has collected enough experiences it will then train itself on those experiences.

The training process consists of periodically retrieving batches of experiences from the replay memory to update the Q-value estimates using the following loss function:

$$L_{s,a,s'} = \left((r + \gamma \max_{a'} Q(s', a', w')) - Q(s, a, w)\right)^2 \quad (6)$$

Where r is the reward received after taking action a in state s, leading to state s. The loss function optimizes action selection by using the maximum value of the potential actions for the subsequent state's value.

The gradient of this loss function is backpropagated through the DQN, updating its weights w and incrementally improving the Q value predictions. [9] However, using the same network parameters w for both current and target Q-value estimations can lead to instabilities in the loss, which might cause an overestimation of Q-values. To mitigate this, the double-DQN approach introduces a target network whose weights w^- are the same as the online network at initialization, but are updated less frequently to provide a stable comparison for the online network [10]. Updates to w^- can be either hard updates or soft updates:

$$w^- \leftarrow \begin{cases} w & \text{hard update} \\ w^-(1-\tau) + w\tau & \text{soft update} \end{cases} \quad (7)$$

Where $\tau << 1$, this slowly changes the weights of the target network and promotes a steady learning process for the online network [11]. τ is a factor that can be updated exponentially or set to a constant value throughout the training process.

Practical implementation of DQN involves careful selection and tuning of hyper-parameters such as the learning rate, batch size, and the architecture of the neural network.

V. METHOD

A. Deciding Between Q-learning and Deep Q-learning

Given the complexity of the World3 model, deciding between a Q-learning agent and a Deep Q-learning (DQL) agent involves considerations based on the complexity of the state space and action space, and the required performance.

Q-learning is a traditional form of RL that works well with discrete, limited state spaces, but struggles with high-dimensional environments as it stores the value of each state-action pair in a matrix known as a Q-table. Given the dynamic and potentially large state space of the World3 model, Q-learning could become impractical due to the computational requirements of the state- and action space, making it hard to manage the Q-table effectively.

DQL, on the other hand, integrates neural networks with Q-learning, using the network to approximate the Q-value function. This approach is beneficial for handling large or continuous state spaces, making it well suited for the World3 model where inputs from sectors like agriculture or population can vary significantly. DQL can generalize across states and learn efficient representations through training, which alleviates the curse of dimensionality associated with traditional Q-learning.

Therefore, for a model as complex as World3, DQL is a more appropriate approach. Efficiently managing large-scale data and learning from complex patterns that are essential for simulating and optimizing dynamic interactions over time. This allows for more robust decision-making capabilities in scenarios that involve complex and interdependent variables.

B. Deep Q-Learning Algorithm and Implementation

To determine the structure of the DQL algorithm we conducted various simulations with different structures to compare the outcomes, see V-F. But some hyper-parameters was possible to approximate before. Considering the dynamics of the World3 model, and our objective to better global well-being, we know that we need to largely consider future rewards, and that instant rewards might have negative impact on the model in the future, thus the discount factor γ needs to be large. Since we want to explore the environment as much as possible to find optimizer policies we want to start with ϵ being large, and slowly decay to explore as much as possible, and later exploit the policies learned. Since the World3 model is very volatile in its outcomes, and small changes in the control signals can lead to large variations, we have adapted the idea of using a target network to update our main networks Q-values, this provides a more stable learning process and reduce overestimation [10]. Below is a pseudo algorithm for how the agent is implemented.

Algorithm 1 Deep Q-learning with experience replay

Initialize replay memory D to capacity N, action-value function Q with random weights w, target action-value function $\hat{Q}$ with weights $w^- = w$, and parameters ϵ, ϵ_{min}, ϵ_{decay}, τ, γ
for episode $= 1$ to M **do**
  Reset environment and get initial state s_t
  for $t = 1$ to T **do**
    Select action a_t either randomly with probability ϵ,
    or as $a_t = \arg\max_a Q(\phi(s_t), a; w)$
    Execute a_t, observe (r_t, s_{t+1}),
    store (s_t, a_t, r_t, s_{t+1}) in D
    if D is ready **then**
      Sample minibatch, set y_j based on done signal, perform a gradient descent step on $(y_j - Q(\phi_j, a_j; w))^2$
      Hard/Soft Update w^- every C steps,
      decay ϵ if above ϵ_{min}
    end if
  end for
end for

C. Network Architecture

The network architecture is an integral part of the DQL architecture. There are a lot of hyper-parameters that can be adjusted, which will yield networks with vastly different characteristics. The number of hidden layers, the structure of the network, the number of nodes in each hidden layer, the optimizer used and the learning rate of the network are the hyper-parameters that we have experimented with to find the most suitable architecture for the neural network.

The structure of the network was a straightforward choice. The network used is a dense sequential neural network. This is the most used structure in RL. The chosen optimizer for the architecture is the adaptive moment estimation, more commonly referred to as the ADAM-optimizer. This optimizer is one of the most frequently used compilers.

The learning rate was set to a default of 0.001. The amount of layers and how many nodes there are in each layer is the hyper-parameter that has been experimented with the most. Tests with a wide variety of setups have been conducted. The structure that performed the best was the structure pictured in 4. The network structure features an input layer that has as many nodes as there are states, s, followed by six hidden layers—three with 256 nodes each and another three with 128 nodes each. Finally, there's an output layer with several nodes that matches the total number of possible actions, a.

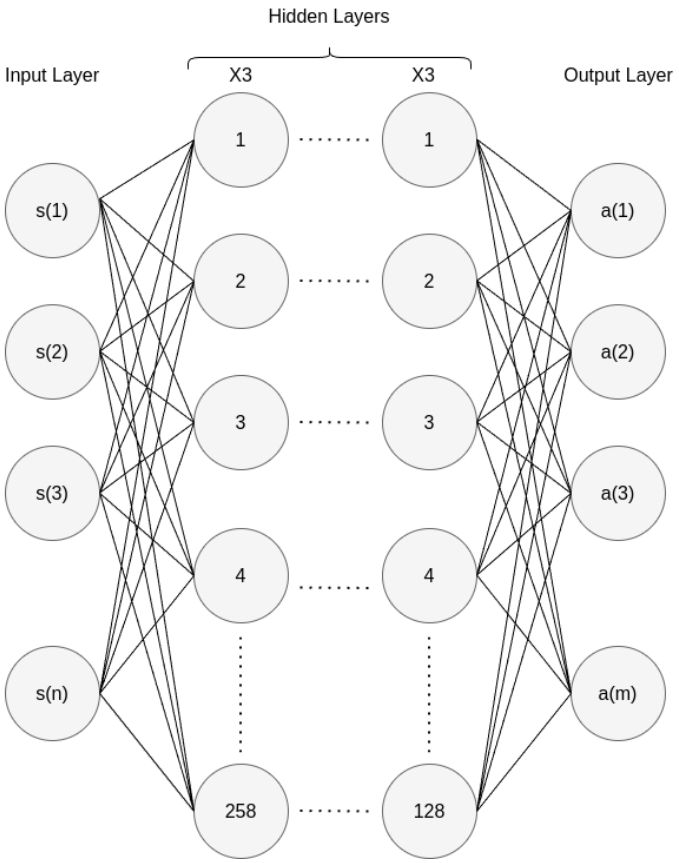

Fig. 4. Network Architecture

In the section IV-F it was mentioned that the frequency the network is updated affects the training process. And in V-B it was mentioned that, to avoid overestimation and enable a more stable learning process, we are using a soft update for the target network, see 5. The update frequency that was settled on was an update every fifth step.

D. Defining The Model, States and Actions

The goal of this project is to enhance the general well-being of the world by creating and training an RL agent that sets policies that affect well-being. The purpose is for it to serve as a tool that can be used to give pointers on the effect that specific policies have on the real world. From these

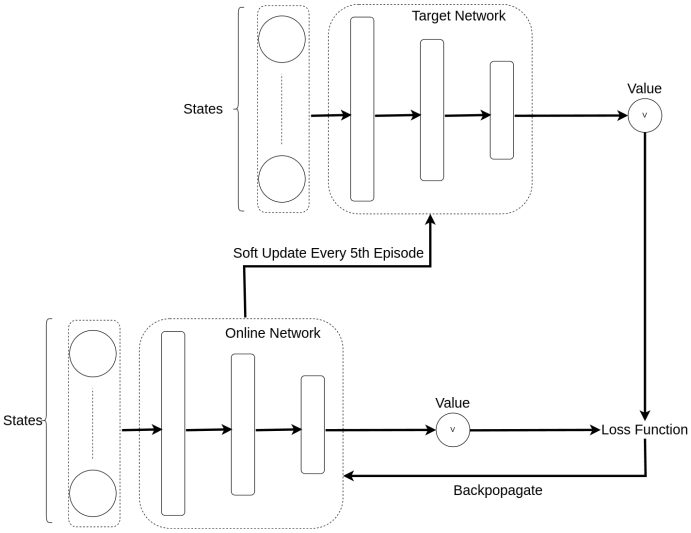

Fig. 5. Soft update of the target network

two guidelines the policy setting constraints, state space and action state have been designed.

In an attempt to emulate the policy setting to policy setting in the real world, the agent is constrained to take an action every fifth year. A time period of five year in the World3 model will therefore be referred to as one step. Each learning episode for the agent spans from the years 1905 to 2225 and consists of 64 steps.

In this project the following state variables were chosen: life expectancy (*le*), the derivative of life expectancy (*dle*), population (*pop*), industrial output per capita (*iopc*), service output per capita (*sopc*), agricultural input (*ai*) and persistent pollution (*ppol*). The state variables *le*, *dle* and *pop* are self-explanatory meanwhile the other variables *iopc*, *ppgr*, *sopc* and *ai* are ambiguous at first glance. The variable *iopc* is defined as: "[...] the stream of real products (consumer goods and investment goods)" [1]. The service section of an economy is the section that is made up of businesses related to services such as retail. The variable *sopc* captures how much money that is put in the service sector of the economy and is measured in dollars/year per person. Agricultural input is measured in dollars per year and translates to how much money is put into the agricultural sector. The *ppgr* variable describes the rate at which pollution units are generated per year. It is not specified in the book what a pollution unit is. With these seven state variables, it was deemed possible to assess the current state of the world in a relatively accurate way in a global scope.

In a standard run of the World3 model (a run where the initial conditions have not been changed from the original) changes in the capital and resource sectors are what drive changes in the other sector [12]. Because of this, the selected control signals were: industrial output capital ratio (*icor*), fraction of input allocated to consumption (*fioac*) and fraction of industrial output allocated to agriculture (*fioaa*). The control signals *icor* and *fioac* are in the capital sector and *fioaa* is indirectly connected to the capital sector but is defined in the agricultural sector. Another criterion for the control signals was that they were supposed to be implementable as a policy in the real world. Therefore, all these control signals are related to different budget allocations, which are easily implementable in the real world.

Each action consists of multiplying the initial value of the chosen control signals with a constant. For example, with the given control signals, *icor*, *fioac*, *fioaa*, the agent chooses a constant to multiply each control signal with individually.

$$\begin{aligned} \mathit{icor}[k] &= \mathit{icor}[0] \cdot C_1 \\ \mathit{fioac}[k] &= \mathit{fioac}[0] \cdot C_2 \\ \mathit{fioaa}[k] &= \mathit{fioaa}[0] \cdot C_3 \end{aligned} \tag{8}$$

Where *icor[k]* is the control signal at the next state, and C_i are the chosen constants by the agent.

E. Reward System

The objective of the agent is to maximize the variable *le*. To be able to achieve this, a reward system is needed that constructively reflects this. From simulating the World3 model, without any outer interaction, and plotting the *le* variable, we could see that the maximum *le* value is 61.44 years, and the lowest was 26.67 years. Another thing we noticed is that there is a steep decline in *le* around the year 2020 6. We decided on a reward system that promotes *le* above 60 and small variations in *le*, and punishes drastic variations and *le* below 20 and 30 years 2.

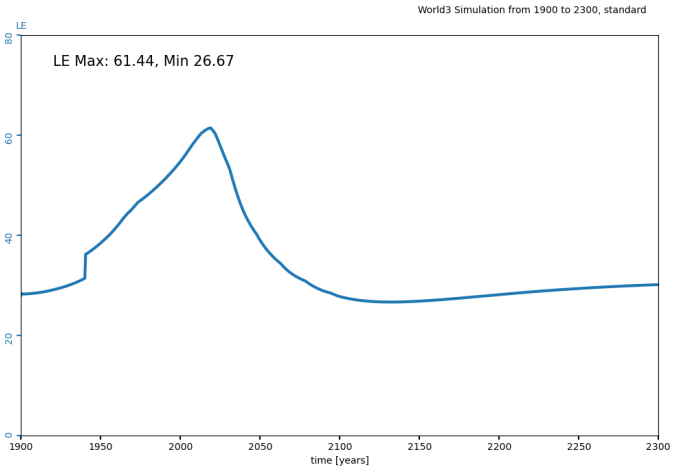

Fig. 6. Standard run showing the life expectancy

F. Grid-Search, Hyper-Parameters

To determine the most suitable configuration for the model, we conducted several simulations using all possible permutations of the different setups shown in Table I and Table II . Each simulation ran for 500 episodes. By plotting the total rewards per episode and comparing the

Algorithm 2 Calculate Reward Based on Life Expectancy

$reward \leftarrow 0$
if $le < 20$ **then**
  $reward \leftarrow reward - 100$
else if $le < 30$ **then**
  $reward \leftarrow reward - 50$
else if $le < 60$ **then**
  $reward \leftarrow reward + 10 \times \frac{dle}{dt}$ ▷ Proportional reward
else if $le \geq 60$ **then**
  $reward \leftarrow reward + 50$
  if abs$(\frac{dle}{dt}) < 0.1$ **then**
    $reward \leftarrow reward + 50$ ▷ Big reward for stability
  else if abs$(\frac{dle}{dt}) < 0.2$ **then**
    $reward \leftarrow reward + 10$ ▷ Smaller reward for less stability
  else
    $reward \leftarrow reward - 100 \times$ abs$(\frac{dle}{dt})$ ▷ Penalty for instability
  end if
end if
if $\frac{dle}{dt} < -0.2$ **then**
  $reward \leftarrow reward - 100 \times$ abs$(\frac{dle}{dt})$ ▷ Penalty for rapid decrease
end if
return $reward$

simulations, the two best results were chosen to perform further simulations.

The two setups that showed the most promise in terms of accumulated rewards over episodes can be seen in Table III.

The final Setup for the model's parameters is given in IV and the only difference in the two setups are the action space, where setup 1 and setup 5 are using action spaces 1 and 2, respectively.

TABLE I
CONTROL SIGNAL SETUP

Control signal setup	Multiplying constants
A	[0.5, 1, 1.5]
B	[0.5, 0.75, 1, 1.25, 1.5]
C	[0.1, 0.5, 1, 1.5, 2]
D	[0.1, 1, 2, 3, 4]

TABLE II
TARGET NETWORK UPDATE FREQUENCY

Target Network setup	Update frequency
A	Every episode
B	Every tenth episode
C	Every second step
D	Every fifth step

TABLE III
FINAL SETUP

Setup	Control signal setup	Target Network setup
1	A	D
2	B	D

TABLE IV
HYPER PARAMETERS

Parameter	Value
Learning Rate (α)	0.001
Discount Factor (γ)	0.95
Soft Update (τ)	0.01
Epsilon Start (ϵ)	1.0
Epsilon Decay	0.9974

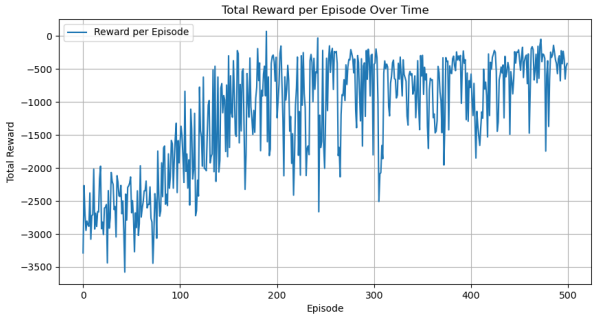

Fig. 7. Rewards over 500 episodes with Setup 1 III.

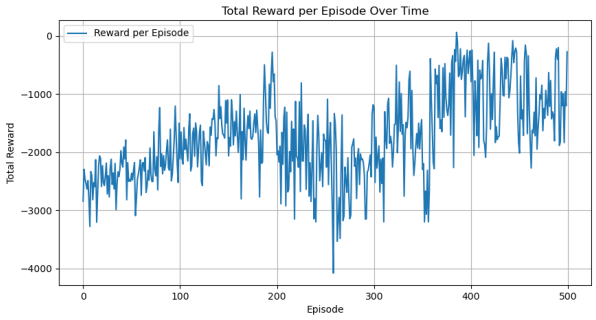

Fig. 8. Rewards over 500 episodes with Setup 2 III.

VI. RESULTS

The results from running the simulation for 2000 episodes with setup 1 and 2, see III, are displayed in the graphs below. During the simulation the moving average for the reward gained in any episode was derived and plotted. In these graphs we can see that the agent is gradually learning to find the state-action pairs that leads to a higher reward over time. This tells us that our agent is working and adapting to our designed reward system. From our simulations with the given reward function, 2, we know that in the case of "do nothing", meaning we do not implement any changes in the system, we get a reward of approximately -2000. In the figures 9 and12 we can see that our agent gets about 0 and -500, respectively, towards the end of the simulation.

When the training of the agent were done the agent were tested in two different scenarios. The first scenario was to allow the agent to control the model from 2020 and the other one was to let it control from 1900. These two tests were compared to the life expectancy of the normal run, the "do nothing" case. The results of the tests are presented in the graphs below, see figures 11 10 and figures 14 13. From these figures we can see that the outcome did not align with our main objective, to stabilize the life expectancy around 60 years, but rather postpone the growth, or stop it completely, as in both cases when the simulation is done from 1900. And for the case of running the simulation from 2020, we can see that the difference in neither setup 1 or setup 5 is controlling the model in a tangible way in comparison to the normal run.

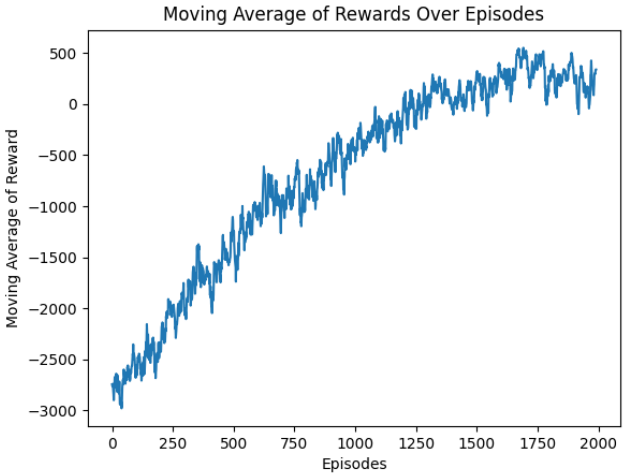

Fig. 9. The moving average of the reward with setup 1 III

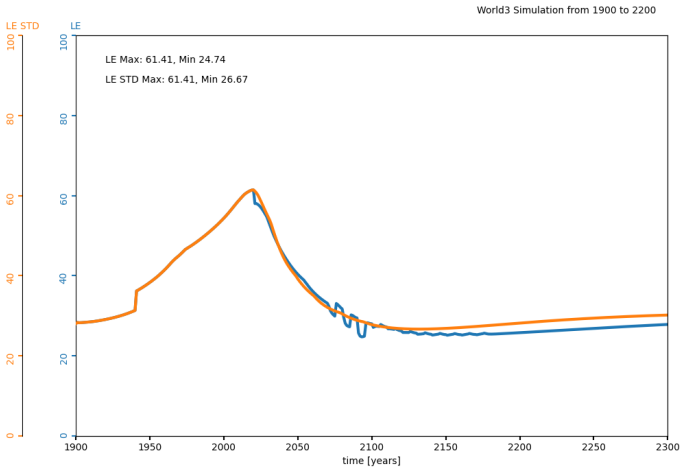

Fig. 11. The controlled Life Expectancy (LE) starting control in 2020, and the Life Expectancy in the standard run (LE STD) with setup 1 III

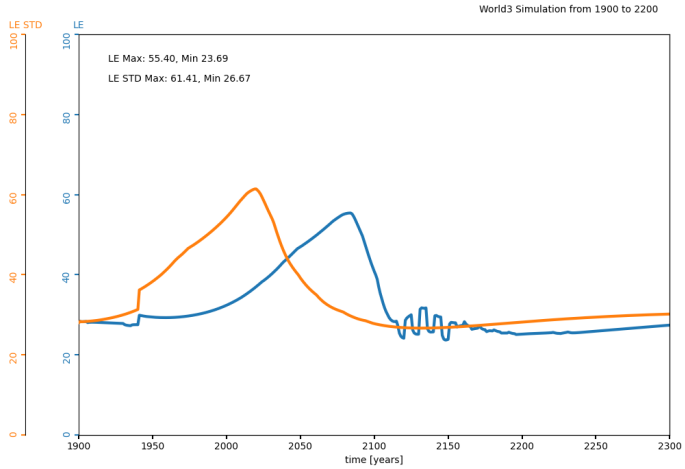

Fig. 10. The controlled Life Expectancy (LE) starting control in 1900, and the Life Expectancy in the standard run (LE STD) with setup 1 **??**

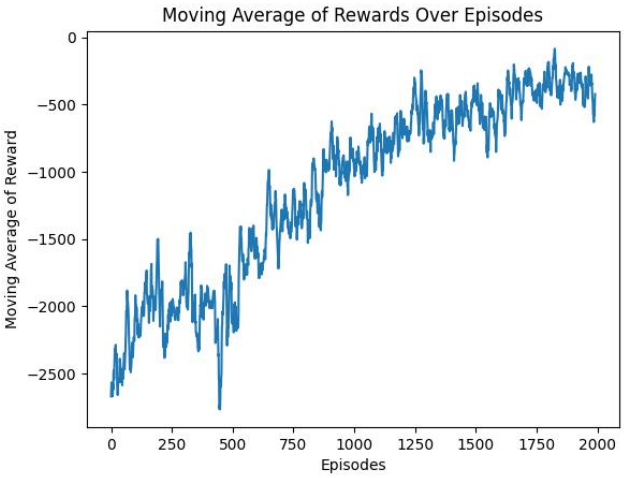

Fig. 12. The moving average of the reward with setup 2 III

In all the figures plotting the comparison of the life expectancy we can see that there are oscillations in the controlled case. This is due to the way the reward system was designed, where we are rewarding a positive change and punishing a negative change in life expectancy. Given the almost constant time difference between the oscillations in all the figures, the agent seems to have found a way of only gaining the positive reward for a positive change, and avoiding the negative reward.

If the agent is permitted to control the model from an earlier point in time, 10 and 13, it can affect the life expectancy. The agent prevents rapid changes in the derivative, which leads to a more stable life expectancy. The agent was not able to stabilize the life expectancy to a higher standard than in the standard run.

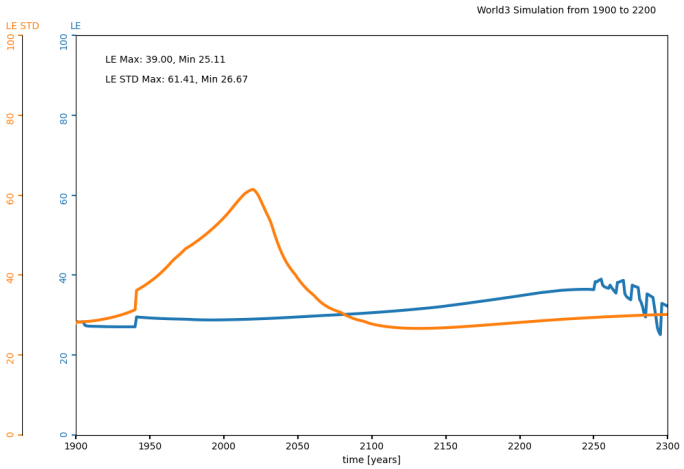

Fig. 13. The controlled Life Expectancy (LE) starting control in 1900, and the Life Expectancy in the standard run (LE STD) with setup 2 III

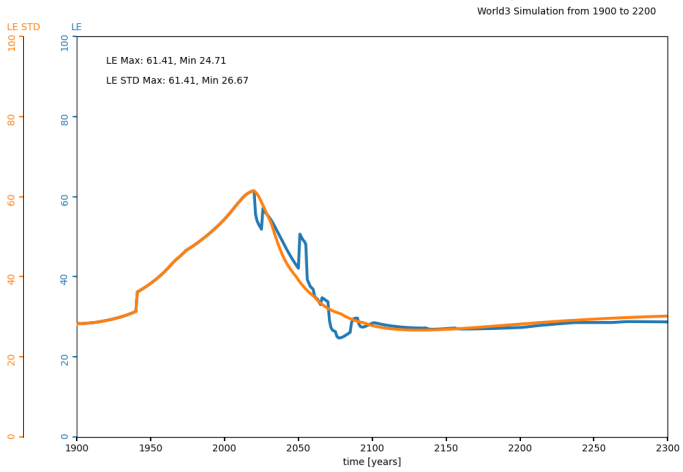

Fig. 14. The controlled Life Expectancy (LE) starting control in 2020, and the Life Expectancy in the standard run (LE STD) with setup 2 III

VII. Discussion

The trend of the rewards in both of the different setups are positive, as can be seen in the graphs. It is clear to see that the reward increases as the agent is trained on more episodes, and that the agent is learning a behaviour aligning with the designed reward system. This shows that the implementation of the agent was successful at learning and interacting with the environment.

However, the graphs showcasing the comparison of life expectancy between the trained model and the standard run, shows that there is a discrepancy between our desired outcome and the behavior that the agent is showing. Even though the agent learns as its reward increases over the episodes, the life expectancy neither stabilizes or increases. This misalignment can probably be attributed to the reward system, where the agent have found ways to accumulate rewards throughout the simulations without aligning with our objective.

This misalignment with our objective and reward system was something that we could see early on. We therefore extensively tested a wide variety of reward systems. The reward system that performed the best, 2, is the one used in the simulations presented in the result section. Because of our extensive testing we believe that the designed reward system used in this project is within reasonable range of the optimal design. It might be possible to get better results with another reward system, and follow up research based on this study could focus on fine-tuning the reward system even more and find a better design.

Since the simulation environment itself is computationally heavy, each episode requires about 10-20 seconds to complete, meaning that running the agent for 2000 episodes, takes about 8-10 hours. Given the complexity of the model and all possible states and actions, 2000 episodes might not be enough for the agent to fully explore the environment. If the agent was able to gain more experience from more training episodes the results might be different. More training episodes would allow the agent to explore more possible strategies, and exploiting more known ones. From the results we can see that the moving average rewards stabilizes around 1700 episodes for both setups, 9 12, this is due to the ϵ and ϵ_{decay} leading the agent to explicitly exploit known values after about 1700 episodes. By running the simulations longer, higher rewards might be achievable.

It is also unclear whether or not the set objective of a stable life expectancy above 60 years is achievable within the World3 model. By letting the agent interact with the environment from different points of time we found a point of time in the simulation where the life expectancy in the agent-controlled simulation does not deviate from the standard run. This is most likely because it is impossible to change the outcome of the simulation when it has gone past a certain point. There is a built in ”dooms day” metric in the model. If the variable *nrfr* (nonrenewable resource fraction remaining) goes under 0.6 the *io* (industrial output) will start to decline, leading to a dramatic drop off in both population and life expectancy [12].

It is important to once again reiterate that the model has been approximated as an MDP. This could be another reason why the model can not be controlled by using RL, as MDPs are a prerequisite for RL.

Even though our agent successfully interacted and learned in the World3 environment, the outcomes of our simulations were not fully satisfactory. The creators of the World3 model wanted to show that the human race will eliminate itself in the future if it keeps on consuming and producing at the rates it is right now. Thus, questions can be raised whether there are underlying biases in the model, with the biases resulting in a model with a world that always collapses. There is a possibility that whatever is done to control the world, it will always end up collapsing.
Further studies in the domain could be based on trying to control a different model than the World3 model. The model *Earth4All* is a newer model, which better reflects the current state of the world and could prove to produce more relevant results compared to the World3 model.

VIII. Conclusion

From our results it is clear that our agent is successfully interacting, learning and accumulating rewards throughout the episodes. Although it does not align with the set objective of stabilizing life expectancy and promote global well-being. This is partly due to faults in the design of the reward system, where the agent finds loopholes to gain rewards without reaching the goal, computational requirements, and partly due to constraints in the World3 model. However, we believe that with more research there is potential to achieve some interesting results, possibly with another simulation model, a more carefully designed reward system, and more

computational power.

IX. Acknowledgment

The authors would like to thank Elisa Bin for her support and advice throughout the project.

References

[1] D. Meadows, J. Randers, and D. Meadows, *The Limits to Growth The 30-Year Update*. London, United Kingdom: EARTHSCAN, 2006.

[2] United Nations Development Programme (2024, Apr), "Sustainable development goals." [Online]. Available: https://www.undp.org/sustainable-development-goals

[3] D. Meadows, W. Behrens, D. Meadows, R. Naill, J. Randers, and E. Zahn, Eds., *Dynamics of Growth in a Finite World*. Cambridge, MA: Wright-Allen Press, 1974.

[4] T. Wolf, "Climate change policy exploration using reinforcement learning," *arXiv preprint arXiv:2211.17013*, 2022. [Online]. Available: https://arxiv.org/abs/2211.17013

[5] C. Vanwynsberghe (2021, Nov), "PyWorld3 - The World3 model revisited in Python." [Online]. Available: https://hal.archives-ouvertes.fr/hal-03414394

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018.

[7] T. T. Hills, P. M. Todd, D. Lazer, A. D. Redish, and I. D. Couzin, "Exploration versus exploitation in space, mind, and society," *Trends in Cognitive Sciences*, vol. 19, no. 1, pp. 46–54, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364661314002332

[8] H. Abdi, D. Valentin, and B. Edelman, *Neural networks*, ser. Sage university papers series. Quantitative applications in the social sciences ; 07-0124. Thousand Oaks, California: SAGE, 1999.

[9] A. Tsantekidis, N. Passalis, and A. Tefas, "Chapter 6 - deep reinforcement learning," in *Deep Learning for Robot Perception and Cognition*, A. Iosifidis and A. Tefas, Eds. Academic Press, 2022, pp. 117–129. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323857871000117

[10] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[12] W. Thissen, "Investigations into the world3 model: Overall model behavior and policy conclusions," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 8, no. 3, pp. 172–182, 1978.