Training of Reinforcement Learning Algorithms for Increased Flexibility in Pumped Storage Power Plants

by Moritz Zebenholzer

A thesis for the degree of
Diplom-Ingenieur (Dipl.-Ing. or DI)

In the
Master program Chemical and Process Engineering

At the
Faculty of Mechanical and Industrial Engineering, TU Wien

Supervised by
DI Dr.techn. Felix Birkelbach, BSc

DI^{in} Carlotta Sophie Freiin von Tubeuf, BSc

Author
Moritz Zebenholzer, BSc
Matr. Nr.: 11802510
moritz.zebenholzer@tuwien.ac.at
TU Wien
Institute for Energy Systems and Thermodynamics
Getreidemarkt 9, A-1060 Wien

Supervisor
DI Dr.techn. Felix Birkelbach, BSc
TU Wien
Institute for Energy Systems and Thermodynamics
Getreidemarkt 9, A-1060 Wien

Co-Supervisor
DI[in] Carlotta Sophie Freiin von Tubeuf, BSc
TU Wien
Institute for Energy Systems and Thermodynamics
Getreidemarkt 9, A-1060 Wien

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume. If text passages from sources are used literally, they are marked as such. I confirm that this work is original and has not been submitted elsewhere for any examination, nor is it currently under consideration for a thesis elsewhere.

I acknowledge that the submitted work will be checked electronically-technically using suitable and state-of-the-art means (plagiarism detection software). On the one hand, this ensures that the submitted work adheres to the high-quality standards of the current rules for ensuring good scientific practice "Code of Conduct" at the Vienna University of Technology. On the other hand, a comparison with other students' theses avoids violations of my copyright.

Vienna, January 14, 2024

Moritz Zebenholzer

Abstract

Climate change requires converting the energy supply to renewables, notably by expanding electricity generation, grid infrastructure, and storage options. Pumped storage power plants are suitable for this and will be investigated in this thesis using methods of *Reinforcement Learning* (RL), a form of machine learning. In particular, blowing out the water in the turbine housing using compressed air will be investigated. This occurs when switching between turbine and pump operation, limiting the starting torque and the electrical power. In addition, blow-out is necessary for phase shift operation.

To generate a control law, the pump-turbine model is embedded in a simulation environment where the RL algorithm learns an optimal control scheme through interaction with it. Modern algorithms are based on neural networks (NN) as universal function approximators, also known as *Deep Reinforcement Learning* (DRL).

The thesis aims to compare RL algorithms with regard to their performance systematically. For this purpose, neural network architectures are designed and compared with each other in a study. In addition, the influence of hyperparameters on robust and fast convergent training behavior is analyzed.

It can be shown that RL is able to control the blow-out process with discrete and continuous actuating variables. Furthermore, it can be seen that these have the same optimal behavior as defined by conventional controllers (hysteresis and PID) and are achieved in a short training process. The designed neural networks can fully represent the problem, requiring a minimum number of neurons per layer or learnable parameters depending on the training parameters. According to the hyperparameter study, it follows that there are (combinations of) learning rates where the optimum is reached or where a predefined value is learned fastest. In addition, dependencies of the influencing variables on training behavior were identified.

It can be concluded that the *Reinforcement Learning* concept is appropriate for handling control problems satisfactorily. This requires a systematic approach to selecting the proper neural network structure and the right choice of hyperparameters for convergent training.

Kurzfassung

Der Klimawandel erfordert einen Umstieg auf erneuerbare Energien, vor allem durch den Ausbau der Stromerzeugung, der Netzinfrastruktur und der Speichermöglichkeiten. Pumpspeicherkraftwerke eignen sich hierfür und sollen in dieser Arbeit mit Methoden des *Reinforcement Learning (RL)*, einer Form des maschinellen Lernens, untersucht werden. Im Speziellen soll der Ausblasvorgang des Wassers im Turbinengehäuse mittels Druckluft untersucht werden. Dies tritt beim Wechsel zwischen Turbinen- und Pumpenbetrieb auf, wobei das Anfahrmoment und die elektrische Leistung begrenzt werden soll. Außerdem muss im Phasenschieberbetrieb ausgeblasen werden.

Um ein Regelgesetz zu erzeugen, wird das Pumpturbinenmodell in eine Simulationsumgebung eingebettet. Der RL Algorithmus erlernt durch oftmalige Interaktion mit einem Modell, Trial and Error, ein optimales Regelschema. Moderne Algorithmen bauen auf neuronalen Netzen (NN) als universiellen Funktionsapproximatoren, auch Deep Reinforcement Learning (DRL) genannt, auf.

Ziel der Arbeit ist es, RL Algorithmen systematisch auf ihre Leistungsfähigkeit zu vergleichen. Zu diesem Zweck werden neuronale Netzarchitekturen zur Problembeschreibung entworfen und in einer Studie miteinander verglichen. Weiters wird der Einfluss von Hyperparametern auf ein robustes und schnell konvergierendes Trainingsverhalten analysiert.

Es kann gezeigt werden, dass RL den Ausblasprozess mit diskreten oder kontinuierlichen Stellgrößen regeln kann. Außerdem wird gezeigt, dass diese das gleiche optimale Verhalten aufweisen, wie durch konventionelle Regler (Hysterese und PID) definiert und in einem kurzen Trainingsprozess erreicht werden. Die entworfenen neuronalen Netze können das Problem vollständig abbilden, wobei eine minimale Anzahl von Neuronen pro Schicht oder erlernbare Parameter in Abhängigkeit von den Trainingsparametern erforderlich sind. Aus der Hyperparameter-Studie folgt, dass es (Kombinationen von) Lernraten gibt, bei denen das Optimum erreicht wird bzw. bei denen ein bestimmter Wert am schnellsten erlernt wird. Weiters konnten Abhängigkeiten der Einflussgrößen auf das Training aufgezeigt werden.

Daraus lässt sich schließen, dass das Konzept des *Reinforcement Learning* anwendbar ist, um Regelungsprobleme zufriedenstellend zu lösen. Dies erfordert einen systematischen Ansatz zur Auswahl der passenden Architekur des neuronalen Netzes und der richtigen Wahl der Hyperparameter für ein konvergentes Training.

Acknowledgments

I would like to express my sincere gratitude to my supervisors, Felix Birkelbach and Carlotta Tubeuf, for their caring support and valuable contributions. Their guidance has been instrumental in both my daily work and the process of writing the thesis. I am particularly thankful for their openness to any questions and the invaluable advice they provided.

During my time at the institute, I was delighted to experience a warm and heartfelt welcome. This supportive atmosphere greatly enhanced my overall experience.

I also want to extend my appreciation to my girlfriend, Lili, and my fellow students. Their presence and support have been crucial, and without them, I would not have completed my studies in such a balanced and relaxed manner.

Contents

Nomenclature

Acronyms

AC	Actor-Critic
BEE	Bellman Expectation Equation
BOE	Bellman Optimality Equation
DQN	Deep-Q-Network
HPA	Hyperparameter Analysis
KL	Kullback Leibler
MDP	Markov Decision Process
ML	Machine Learning
NNA	Neural Net Analysis
PPO	Proximal Policy Optimization
Q	Q-Learning
RL	Reinforcement Learning
SARSA	State Action Reward succesor-State succesor-Action
TD	Temporal Differences
TRPO	Trust Region Policy Optimization
TU	Technical University

Greek symbols

α	Learning rate	-
δ	Threshold, Error	-
ϵ	Exploration	-
γ	Discount factor	-
ϕ	Parameters critic	-

π	Policy	-
θ	Parameters actor	-

Subscripts

i, j, k	Counting indices
max	Maximum
min	Minimum
sim	Simulation

Roman symbols

A	Action space	-
a	Action	-
a	Pressurized air supply	%
c	Clip factor PPO	-
D	Advantage (function)	-
E	Least Mean Square Error	-
$\mathbb{E}$	Expectation probability distribution	-
$\mathcal{F}$	Activation function	-
G	Return, total (discounted) cumulative reward	-
g	Guide vane opening	m
H	Hidden layer neuron	-
h	History of observations	-
h	water height turbine housing	m
I	Input neuron	-
J	Expected return	-
L	Loss function	-
L_2	Generalisation factor NN training	-
M	Mini batch size	-
N	Experience size	-

n	Rotational speed	rpm
N_a	Number of neurons per layer, actor	-
N_c	Number of neurons per layer, critic	-
O	Output neuron	-
P	State transition probability	-
p	Pressure	bar
$\mathbb{P}$	Probability distribution	-
Q	Action-value function space	-
q	Action-value function	-
R	Reward space	-
r	Reward	-
$r(\theta)$	Policy probability ratio	-
S	State space	-
s	State	-
T	Termination time step	s
t	Time step	s
ts	Start time step experience	s
V	State-value function space	-
v	State-value function	-
w	Connection weight neuron	-
w	Entropy loss weight	-
x	Input signal	-
y	Output signal	-

Superscripts

$'$	Successor
$*$	Optimum

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Due to climate change, there is a need to reduce greenhouse gas emissions significantly. In specific, this means converting electricity generation to renewable forms of energy. To this end, Austria has set itself the goal of becoming climate-neutral by 2030. For this reason, the expansion of photovoltaic, solar thermal energy and wind power, in addition to the traditionally widespread hydroelectric and biomass power plants, is to be increased. [1]

As renewable energy generation is accompanied by significant power fluctuations, a structural expansion of the electricity grid and increased storage options for surplus energy are required. The same amount of energy must always be fed into and consumed in the grid to be stable. Pumped storage power plants are the most flexible and economical option for storing electricity. With regard to grid stability, they can add and remove reactive power from the grid and can also perform phase shifts in idle mode. [2, 3]

For this reason, pumped storage power plants are used on a large scale in Austria to store surplus energy by pumping water from deep levels into a higher reservoir so that it can be converted back into electrical power in a turbine when needed [2]. This process can be carried out with a separate pump and turbine or a reversible pump turbine, whereby switching between the two operating modes is necessary. When the pump turbine starts up or when switching between turbine and pump operation, it is necessary to blow out the water in the turbine housing in order to limit the torque and the electrical power. The water is also blown out in phase shift mode to minimize the active power. [4]

Reinforcement Learning (RL) algorithms can be used to control and optimize the blow-out process to enable faster changes between operating states and thus operate the power plant more dynamically [5]. RL is a type of machine learning in which the algorithm learns an optimal control scheme itself through trial and error [6]. Its ability to learn complex behaviors without explicit guidance has attracted a great deal of attention in recent years. In combination with neural networks (NN) as universal function approximators, deep reinforcement learning (DRL) has achieved great success in the research field of robotics, game simulators, and control technology [7].

Imagined figuratively, the RL agent wanders through the unknown state space described by the model or reality. It uses its senses to observe its environment, which changes with every step or action it takes. It gets a reward or punishment from his surroundings as information along the way he chooses. Its goal is to find the optimal path to receive the most rewards at the end. To this end, it proceeds with a certain strategy to improve its view of the world through new insights to ultimately know which path to take in the future. [6]

1.2. Aim

The aim of the diploma thesis is to apply the Reinforcement Learning control concept to an existing simulation model of a reversible pump turbine on a laboratory scale. In particular, the blow-out process that occurs when switching between turbine and pump operation is to be investigated. The main focus of the work is on the systematic analysis of RL algorithms and NN structures for this sub-process.

Reinforcement Learning is a potent and universally applicable tool, but it must be adapted specifically for each problem. This means that different (families of) algorithms are more suitable for certain problems, and it is unclear from the outset which hyperparameters must be set and which neural network must be used.

The motivation described above gives rise to the following research questions to be addressed in this thesis:

- Is RL suitable for controlling the simulation model in a meaningful way?
- How should the deep neural network be constructed?
- How must the hyperparameters be selected to achieve optimal training behavior?
- Which RL algorithm is best suited to solve the problem in terms of performance?

1.3. Methodological Approach

In order to answer these questions, a literature review is conducted on the formal structure of the reinforcement learning concept and its implementation as algorithms, as well as on the architecture and mode of operation of neural networks. A simulation environment is set up in MatLab/Simulink [8, 9] using the Deep Learning Toolbox and Reinforcement Learning Toolbox [10, 11]. Different NN architectures are systematically compared with each other in terms of training performance in a study. Hyperparameters are systematically changed and their influence on the training behavior is analyzed in studies. As an example, the control behavior is compared with conventional controllers such as hysteresis and PID.

1.4. Structure

The structure of the work is divided into the following topics: Chapter 2 describes the theoretical background of the mathematical concepts used, the numerical algorithms and the architecture of the neural networks. Chapter 3 then provides an overview of the simulation environment's structure and the pump turbine's embedded model. The state and action space are built up, and a structure of the reward function is defined. Finally, this part describes the neural network variants used and shows the application of the algorithms to the problem. In Chapter 4, the results are presented in detail in the form of plots and discussed based on performance and convergence. Finally, Chapter 5 concludes and provides an outlook on possible future experiments.

2. Theoretical Background and Methods

The relevant theoretical foundations are divided into the concept of Reinforcement Learning in general, its application in the form of algorithms and the underlying description by neural networks. In the first section, the mathematical formalism and its description methods are developed. The second part presents numerical solution methods and their hyperparameters. The third section deals with the universal function approximation required for the practical application of the algorithms.

2.1. Reinforcement Learning

2.1.1. Origin and Application

Reinforcement Learning (RL) is a form of Machine Learning (ML) where, in contrast to supervised and unsupervised ML, no external information about input and output data is required. In supervised learning, the input and output are labeled by knowledge and the algorithm approximates a relationship between the two. This is used for classification or regression problems, for example. In unsupervised learning, knowledge about the input is used to collect information about the output, such as similarities in clustering. [6, 12]

The combination of modern algorithms and neural networks as universal function approximators enables RL, known as Deep Reinforcement Learning (DRL), to achieve outstanding results in current research areas of robotics, process control and games. [12, 7]

2.1.2. Mathematical Description

The following chapter is based on the textbook "Reinforcement Learning: An Introduction" by R. S. Sutton and A. Barto [6].

Markov Decision Process

Mathematically, the RL problem can be represented in the form of a Markov Decision Process (MDP). This is shown in Figure 1.

The agent interacts with the environment, receiving a state and reward signal and performing an action. The state $s_t \in S_t$, also known as observation, describes the environment and the scalar value $r_t \in R_t$ indicates how well the agent behaves at time t. S and A are finite sets from the space of possible states and actions. Due to

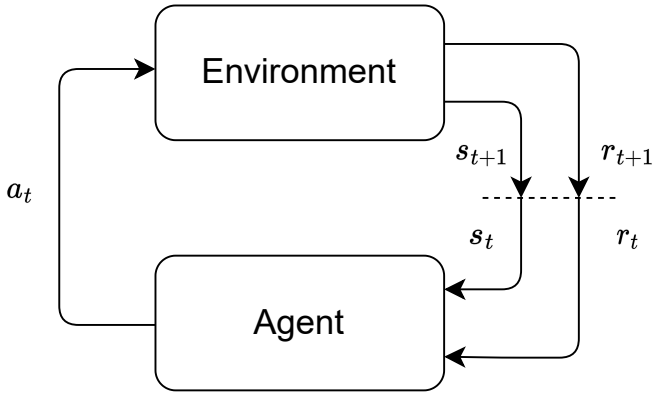


Figure 1.: *Representation of a Markov Decision Process (MDP) adapted from [6] with action a_t, state s_t, reward r_t, successor state s_{t+1} and reward r_{t+1}.*

the action $a_t \in A_t$, which acts as an input to the environment, changes. A successor state s_{t+1} or s' and a new reward r_{t+1} occur. The internal dynamics of the change are described by the probability

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]. \tag{1}$$

The sequence of states, rewards and actions is referred to as a history h, see Eq. (2). The trajectory ends with the terminating state s_T and reward r_T at time T and can be understood as an episode.

$$h = \{s_t, r_t, a_t, s_{t+1}, r_{t+1} \dots s_T, r_T\} \tag{2}$$

If an MDP is present, then its states are fully observable and have the Markov property,

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]. \tag{3}$$

> "The future is independent of the past given the present" - David Silver, Introduction to Reinforcement Learning, p. 21 [13]

This means that only knowledge about the current state is required to describe the system entirely. Previous states and actions from history can be neglected. Most RL problems can be described as MDPs, whereby extensions are made in the MDP for continuous state and action spaces.

Policy and Return

Which action the agent selects based on the current state is described by its control strategy or policy π. This represents a stationary mapping between states and actions. A distinction can also be made between deterministic, Eq. (4), and stochastic policies, Eq. (5).

$$\pi(s) = a \in A_t, \qquad \forall s \in S_t \tag{4}$$

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s], \qquad \forall s \in S_t, a \in A_t \tag{5}$$

The agent's goal is to learn a control strategy that maximizes the total (discounted) cumulative reward G_t along the trajectory. This so-called return can be calculated according to Eq. (6), where R is the set of possible rewards.

$$G_t = \sum_{t=1}^{T} \gamma^{t-1} R(s_t, a_t, s_{t+1}) \tag{6}$$

The discount factor γ takes into account the uncertainty of future rewards. For $\gamma = 0$, only the successive reward is taken into account; with $\gamma = 1$, all future rewards are weighted equally.

The reward generated from the environment is formed by

$$R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]. \tag{7}$$

> Reward Hypothesis: "All goals can be described by the maximisation of expected cumulative reward" - David Silver, Introduction to Reinforcement Learning, p. 13 [13]

Value Function and Bellman Equation

The state-value function $v_\pi(s)$ given in Eq. (8) indicates how good it is to be in a state. It predicts future rewards, the expected return if the policy π is followed. The action-value function $q_\pi(s, a)$ given in Eq. (9) indicates how good it is to be in a state and perform an action and then to follow the policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{8}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{9}$$

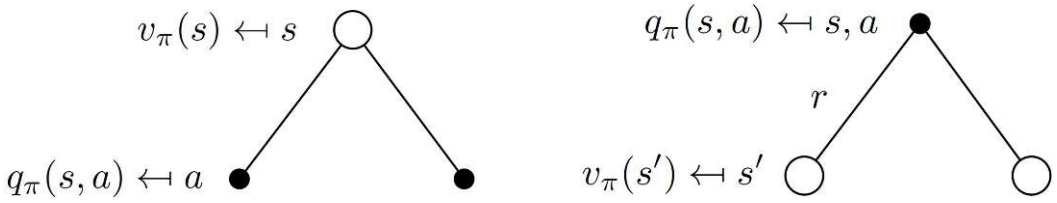
The Bellmann Expectation Equation (BEE) can be used to separate the state-value into an immediate portion R_{t+1} and a discounted state-value function of the succeeding state $\gamma v_\pi(S_{t+1})$,

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. \tag{10}$$

The same can be done for the action-value, where $\gamma q_\pi(S_{t+1}, A_{t+1})$ is the discounted action-value function,

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]. \tag{11}$$

Figures 2a and 2b show the one-step look-ahead for the state-value and action-value functions. In the first case, the possible actions and their probability of being executed by the policy are averaged. In the second case, the internal dynamics of the environment are averaged for an action in a specific state. Mathematically, this relationship can be expressed by Equations (12) and (13), where R_s^a is the reward function, see Eq. (7), and $P_{ss'}^a$ is the probability function of the internal dynamics, see Eq. (1).



(a) *One-step look-ahead for state-value function* $v_\pi(s)$.

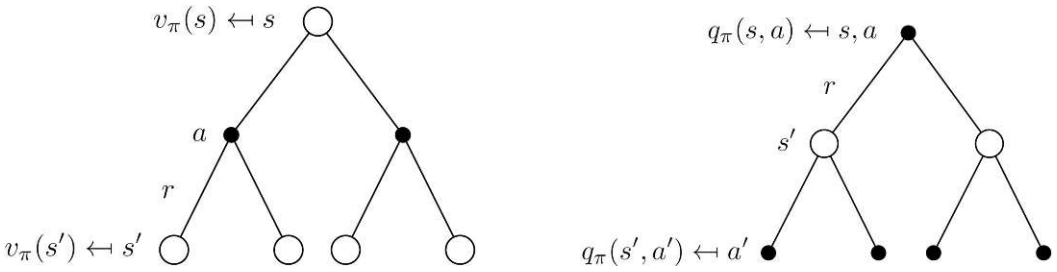
(b) *One-step look-ahead for action-value function* $q_\pi(s,a)$.

Figure 2.: *Bellman Expectation Equation for one-step look-ahead with white circles as states* s*, black circles as actions* a *and successor state* s' *as well as reward* r*, taken from [13] made available under CC-BY-NC 4.0.*

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) q_\pi(s,a) \tag{12}$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \tag{13}$$

If the descriptions 2a, 2b are combined in reverse, the result is a two-step look-ahead, which is shown in Figure 3a and 3b. Equations (14) and (15) are assembled from Equations (12) and (13). They give the recursive procedure for solving the MDP.



(a) *Two-step look-ahead for state-value function* $v_\pi(s)$.

(b) *Two-step look-ahead for action-value function* $q_\pi(s,a)$.

Figure 3.: *Bellman Expectation Equation for two-step look-ahead with white circles as states* s*, black circles as actions* a*, successor state* s' *and action* a' *as well as reward* r*, taken from [13] made available under CC-BY-NC 4.0.*

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right) \tag{14}$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s', a') \tag{15}$$

Exploration and Exploitation

To approach the optimal policy, the agent must consistently execute the best-evaluated actions and exploit the best path. However, it does not know from the outset how good it is to be in a particular state (and to perform an action). The agent must, therefore, explore the state space to determine the best actions for the states. The so-called exploration/exploitation dilemma arises. This is often solved using an ϵ-greedy exploration model, see Eq. (16). Here, a random action is selected with a probability ϵ. The action with the maximum action-value is executed with the complementary probability. To ensure sufficient exploration in the initial training phase and convergence to the optimal strategy at the end, the probability ϵ may be reduced over time. This can be done via an exploration decay rate ϵ_{decay}, whereby the following calculation, Eq. (17), can be used to reduce the exploration every episode until a minimum ϵ_{min} is reached.

$$A = \begin{cases} \underset{a \in A}{\operatorname{argmax}}\, Q(s,a) & \text{for } 1-\epsilon \\ a_{\text{random}} & \text{for } \epsilon \end{cases} \tag{16}$$

$$\epsilon \leftarrow \begin{cases} \epsilon(1-\epsilon_{decay}) & \text{if } \epsilon > \epsilon_{min} \\ \epsilon & \text{otherwise} \end{cases} \tag{17}$$

A policy is optimal, marked with an asterisk ($*$), if the optimal description of the action-value function is maximized, Eq. (18). There is always a deterministic optimal policy for each MDP.

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{for } a = \underset{a \in A}{\operatorname{argmax}}\, q^*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

The Bellman Optimality Equation (BOE) describes the optimal state-value, Eq. (19) and action-value, Eq. (20) functions.

$$v^*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s') \tag{19}$$

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q^*(s', a') \tag{20}$$

Temporal Differences

Temporal Differences (TD) can be used as a solution method for estimating the value functions. Here, the description is changed iteratively by using a so-called TD target, consisting of the reward of the next time step and the discounted value of the successor state. The difference between the TD target and the current estimated value is described as TD error and multiplied by the learning rate α, resulting in

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - V(S_t)}_{\text{TD error } \delta}]. \tag{21}$$

Policy Gradient

Another way to solve MDPs is to optimize the policy directly and not to estimate the value function. For this purpose, a scalar performance value is introduced depending on the parameters θ, the expected return $J(\theta)$ [12]. For the episodic case, the expected return can be expressed as a value function for the policy π as a function of the parameters

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t \mid S_t = s]. \tag{22}$$

To achieve the optimal policy, the expected return should be maximized by moving along the gradient

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t). \tag{23}$$

Stochastic Gradient Ascent (Descent) has proven to be a good calculation method to ensure simultaneous exploration alongside optimization. The loss function, used for the algorithms in section 2.2, can be regarded as a negative return.

2.2. Reinforcement Learning Algorithms

The following chapter is based on the documentation of the Reinforcement Learning Toolbox [11] and the Deep Learning Toolbox [10].

Training begins with a random initialized value function or policy to be successively improved at each time step. To do this, the agent must explore the state space to identify the best actions for the states to exploit them. [6]

In general, RL algorithms can be classified according to whether they have a value function and/or a policy. Table 1 shows a selection of agents implemented in MatLab in the Reinforcement Learning Toolbox, where a critic estimates the value function and an actor emulates the policy. A distinction can be made between discrete and continuous state or action spaces. Representatives of discrete action spaces are *Q* (2.2.1), *SARSA* (2.2.1) and *DQN* (2.2.1); representatives of continuous action spaces are *AC* (2.2.2), *TRPO* (2.2.2) and *PPO* (2.2.2). [12]

Table 1.: *Overview of RL algorithms implemented in MatLab.*

Critic/Actor	none	stochastic	deterministic
none		Policy Gradient	
state-value		Actor Critic Trust Region Policy Optimization Proximal Policy Optimization	
action-value	Q-Learning SARSA Deep-Q-Network	Soft Actor Critic	Deep Deterministic Policy Gradient TD3

If the internal system dynamics $P^a_{ss'}$, see Eq. (1), and the reward function R^a_s, see Eq. (7), are known, it is referred to as model-based RL; if these are unknown to the agent, then it is a model-free RL problem. Model-free RL is present in most cases; the state value and action value function must be estimated. This can be done either online, i.e., directly as a closed loop controller in the system, or offline using a batch of data. Another way of distinguishing is whether the policy used for interaction with the environment and for (policy) optimization is the same. If only one policy π is used, it is an on-policy procedure; if two different Policies μ and π are used, it is off-policy. [6]

State-of-the-art RL methods like Deep Reinforcement Learning use approximative functions, such as neural networks, to describe the state- and action-value and the policy. For this reason, all further relationships are specified for the parameterized form. These are described in detail in Chapter 2.3.

2.2.1. Value-based RL Algorithms

The RL algorithms *Q-Learning (Q)*, *SARSA* and *Deep-Q-Network (DQN)* are online, model-free and value-based. This means they construct the policy using a learned value function by maximizing over it; see Equation (18). With the off-policy representatives Q and DQN, this is done for the "optimal" course of action. With SARSA, this return is estimated for the current policy, as it is an on-policy procedure. These algorithms have a discrete action output.

Q-Learning

The *Q-Learning (Q)* algorithm uses the following temporal difference mapping, Equation (24), for iterative estimation of the critic $Q(S, A; \phi)$. The agent observes its environment and receives a state S. For this state, it performs an action A according to ϵ-greedy exploration, see Eq. (16), whereby it receives a reward R and the new state S'. This represents the behavior policy. For the evaluation of the TD target, however, an alternative action A' is used, which is formed greedy with respect to Q, thus resulting in the maximization over it. The new parameters are calculated using the learning rate α and the gradient of the square of the difference (25).

$$\Delta Q(S, A; \phi) = R + \gamma \max_{A} Q(S', A; \phi) - Q(S, A; \phi) \tag{24}$$

$$\phi \leftarrow \phi + \frac{1}{2}\alpha \nabla_{\phi} (\Delta Q(S, A; \phi))^2 \tag{25}$$

SARSA

The *SARSA* algorithm uses the Equation (26) to iteratively improve the value function $Q(S, A; \phi)$. Here, the agent receives a state S from the environment and executes an action A according to Equation (16). It receives the reward R and the new state S', for which it, in turn, calculates an action A' using the same Equation (16). The name of the algorithm is derived from this quintuple (S, A, R, S', A'). The parameters are calculated similarly to *Q-Learning*.

$$\Delta Q(S, A; \phi) = R + \gamma Q(S', A'; \phi) - Q(S, A; \phi) \tag{26}$$

Deep-Q-Network

The *Deep-Q-Network (DQN)* algorithm, introduced in [14], is a development of *Q-Learning* with two critics: $Q(S, A; \phi)$ and $Q_{\text{target}}(S, A; \phi_{\text{target}})$. The target critic was introduced to achieve more stable optimization. The function approximations map the value function using a neural network with parameters ϕ and ϕ_{target}. Both have the same structure and parameterization. As with the *Q-Learning* algorithm, an action A is performed according to ϵ-greedy exploration, see Eq. (16). The

quadtuples (S, A, R, S') obtained are stored in a buffer memory. A certain number M of uniformly random tuples (S_i, A_i, R_i, S_i') are sampled from this buffer. The Equation (27) specifies the loss function $L(\phi)$ to be minimized over the parameters ϕ. The target parameters are updated with the network parameters periodically.

$$L(\phi) = \frac{1}{2M} \sum_{i=1}^{M} \left(R_i + \gamma \max_{A'} Q_{\text{target}}(S_i', A'; \phi_{\text{target}}) - Q(S_i, A_i; \phi_i) \right)^2 \rightarrow \min. \quad (27)$$

Algorithm 1 shows the implementation of the *DQN* method in MatLab/Reinforcement Learning Toolbox. The target parameters are formed using a proportion τ of the parameters, among the old target parameters.

initialize $Q(S, A; \phi)$, $Q_{target}(S, A; \phi_{target})$ with $\phi = \phi_{target}$ randomly
for *episodes = 1* **to** *termination condition* **do**
  for $t = 1$ **to** T **do**
    $a_t = \begin{cases} \underset{a \in A}{\text{argmax}}\, Q(s_t, a; \phi) & \text{for } 1 - \epsilon \\ a_{\text{random}} & \text{for } \epsilon \end{cases}$
    execute a_t, observe r_t, s_{t+1}
    store (s_t, a_t, r_t, s_{t+1}) in replay buffer
    sample random minibatch (s_i, a_i, r_i, s_{i+1}) of size M
    $y_i = \begin{cases} r_i & \text{if episode terminates} \\ r_i + \gamma \max_{a'} Q_{\text{target}}(s_{i+1}, a', \phi_{\text{target}}) & \text{otherwise} \end{cases}$
    minimize Loss function $L = \frac{1}{2M} \sum_{i=1}^{M} (y_i - Q(s_i, a_i; \phi))^2$
    update parameters $\phi_{\text{target}} = \tau\phi + (1 - \tau)\phi_{\text{target}}$
  end
end

Algorithm 1: DQN

2.2.2. Actor-critic-based RL Algorithms

The *Actor-Critic (AC)*, *Trust Region Policy Optimization (TRPO)*, and *Proximal Policy Optimization (PPO)* algorithms are representatives of the actor-critic-based methods, as they each have a function approximator for both the policy and the value function. They combine the advantages of value-based and policy gradient methods. The actor is represented by $\pi(A \mid S; \theta)$ and the state-value critic via $V(S; \phi)$.

Actor-Critic

To learn the policy and value function, the *AC* agent interacts with the environment for N steps and receives the trajectory,

$$h = (S_{ts}, A_{ts}, R_{ts+1}, \ldots, S_{ts+N}) \tag{28}$$

before the estimates are changed with start time step ts of each trajectory. With this interaction, the N-step return G_t can be calculated, similar to Equation (6). If the estimate of the value function is subtracted from this as a baseline, the so-called advantage function D_t, Eq. (29), is obtained, reducing the variance [13].

$$D_t = G_t - V(S_t; \phi) \tag{29}$$

To calculate the actor's parameters, the gradients are summed over the N experiences along the direction of the policy gradients weighted with the advantage function, Eq. (30). For the parameters of the critic, the mean squared error between the estimated value function and the return is minimized, Eq. (31). The parameter sets are renewed with the corresponding learning rates times the update; see Equations (32), (33).

$$\Delta\theta = \sum_{t=1}^{N} \nabla_\theta \ln \pi(A \mid S_t; \theta) D_t \tag{30}$$

$$\Delta\phi = \sum_{t=1}^{N} \nabla_\phi (G_t - V(S_t; \phi))^2 \rightarrow \min. \tag{31}$$

$$\theta \leftarrow \theta + \alpha_{actor} \Delta\theta \tag{32}$$

$$\phi \leftarrow \phi + \alpha_{critic} \Delta\phi \tag{33}$$

Trust Region Policy Optimization

The *Trust Region Policy Optimization (TRPO)* algorithm, introduced in [15], interacts with the environment for a certain number of steps, similar to the *AC*. It calculates an advantage function D_t from the states and rewards. An object function $J(\theta)$ consisting of a probability ratio

$$r_t(\theta) = \frac{\pi(A_t \mid S_t; \theta)}{\pi(A_t \mid S_t; \theta_{\text{old}})} \tag{34}$$

between the new and old parameterized policy and the advantage function is then maximized

$$J(\theta) = \mathbb{E}[r_t(\theta) D_t] \rightarrow \max. \tag{35}$$

while maintaining the Kullback Leibler (KL) divergence constraint

$$\mathbb{E}[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \tag{36}$$

The KL divergence indicates the statistical distance, i.e., how different two probability distributions are from each other. The parameters of the critic are updated similarly to the *AC* algorithm. δ is a threshold value.

Proximal Policy Optimization

The *Proximal Policy Optimization (PPO)* algorithm, introduced in [16], is a further development of *TRPO* and a simplification. It uses a clipped surrogate objective function. This can be structured according to Eq. (37), where c indicates the clip factor and $J(\theta)$ is the surrogate objective function for the timestep t.

$$J(\theta) = \mathbb{E}\begin{cases}(1-c)D_t & \text{if } r_t(\theta) \leq 1-c \text{ and } D_t < 0\\ (1+c)D_t & \text{if } r_t(\theta) \geq 1+c \text{ and } D_t > 0\\ r_t(\theta)D_t & \text{otherwise}\end{cases} \tag{37}$$

As a first-order approximation, the object functions of *TRPO* and *PPO* are the same for the old parameters π_{old}. As the policy ratio is clipped outside the range $[1-c, 1+c]$, there is no incentive for huge updates.

Algorithm 2 shows the implementation of the *PPO* method in MatLab/Reinforcement Learning Toolbox. The term $w \cdot H_i(\theta, S_i)$ indicates the entropy loss weight w times the entropy H for iteration i.

initialize $\pi(A \mid S;\theta)$ with θ randomly
initialize $V(S;\phi)$ with ϕ randomly
for *episodes = 1* **to** *termination condition* **do**
  for $ts = 1$ **to** T **do**
    generate N experiences following π: $h = (S_{ts}, A_{ts}, R_{ts+1}, \ldots, S_{ts+N})$
    $ts \leftarrow ts + N$
    for $t = h(1)$ **to** $h(N)$ **do**
      calculate advantage $D_t = \sum_{k=t}^{ts+N-1} (\gamma\lambda)^{k-t}\delta_k$
      and return $G_t = D_t + V(S_t;\phi)$
      with $\delta_k = R_{k+1} + b\gamma V(S_{k+1};\phi) - V(S_k;\phi)$
      and $b = \begin{cases} 0 & \text{if } k+1 \text{ is a terminal state} \\ 1 & \text{otherwise} \end{cases}$
    end
    sample M experience over K epochs
    minimize critic Loss function $L_{\text{critic}}(\phi) = \frac{1}{2M}\sum_{i=1}^{M}(G_i - V(S_i;\phi))^2$
    minimize actor Loss function $L_{\text{actor}}(\theta) = \frac{1}{M}\sum_{i=1}^{M}(wH_i(\theta, S_i) - y_i)$
    with $y_i = \begin{cases} (1-c)D_i & \text{if } r_i(\theta) \leq 1-c \text{ and } D_i < 0 \\ (1+c)D_i & \text{if } r_i(\theta) \geq 1+c \text{ and } D_i > 0 \\ r_i(\theta)D_i & \text{otherwise} \end{cases}$
    and $r_i(\theta) = \frac{\pi(A_i|S_i;\theta)}{\pi(A_i|S_i;\theta_{\text{old}})}$
  end
end

Algorithm 2: PPO

2.3. Neural Network

The following chapter is based on the article "An Introduction to Neural Networks" by Ben Krose and Patrick van der Smagt [17].

2.3.1. Origin and Application

Neural networks (NN) are universal function approximators that can theoretically describe any relationship between input and output. They originated in neurology and psychology, with the human brain in particular as a biological model. In the human brain, tens of trillions of neurons connected via synapses form the basis for intelligent life [18]. In technology, neural networks are designed for highly specialized applications. They have very poor extrapolation behavior and are generally considered a black box. For this reason, they can only be used for the range of the training data. NNs are a widly used to form the basis for machine learning applications.

2.3.2. Artificial Neuron

The artificial neuron, also known as a perceptron, is the basic building block of every neural network. Its input comprises a number of weighted signals x_j from $j = 1 \dots n$, which are added up plus a bias b. The output y is computed via an internal (non-linear) relationship, the activation function $\mathcal{F}$. Figure 4 shows a typical neuron structure.

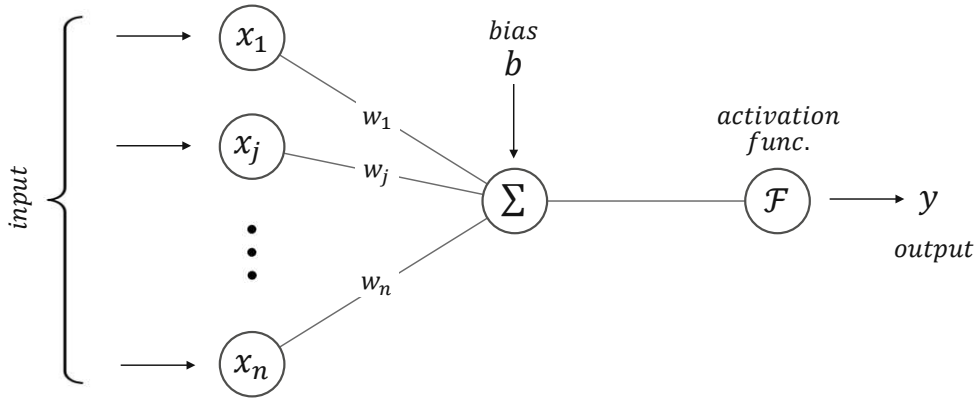


Figure 4.: *Representation of an artificial neuron with inputs x_j, weights w_j, bias b, activation function $\mathcal{F}$ and output y from $j = 1 \dots n$, number of connections, adapted from [18].*

The behavior can be described mathematically using the relationship

$$y = \mathcal{F}(\sum_{j=1}^{n} w_j x_j + b). \tag{38}$$

2.3.3. Activation Function

There are a variety of activation functions that are used depending on the problem. In most cases, it is a monotonically increasing function that provides a threshold. Step-like functions such as binary distributions tend to be used for categorization tasks. A common function for biological processes is the sigmoidal distribution $f(x) = 1/(1+e^{-x})$. Another important activation function is the hyperbolic tangent $f(x) = \tanh(x)$. For Reinforcement Learning problems, the simple rectified linear unit (ReLU) function is often used, which maps the non-linearity by clipping the negative range according to

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{39}$$

2.3.4. Neural Network Architecture

A suitable neural network architecture must be selected depending on the input and output data's type and nature. There is no a priori way to design the network except by trial and error. Each input neuron represents a specific piece of information and can be, for example, a pixel in image processing, a motor voltage in a robot controller or, in general, a state. The inputs can be different, so they are usually normalized to the range [-1,1] to ensure equal influence. The output can, in turn, be normalized to a certain range. In between are the so-called hidden layers responsible for the actual approximation. The topology of the NN consists of the number of neurons per layer, the number of hidden layers, their activation functions and the connections between the neurons. The more neurons per layer and more layers in the body of the NN, the more complex relationships can be represented.

The flow of information in the network can be from input to output, i.e., only in one direction, also known as feedforward, or with a loop-back, i.e., recurrent. This feedback can take place in the same layer, but also to previous network layers. This allows time-dependent, dynamic data to be represented. If the output of the previous neurons acts as a signal on all neurons in the next layer, then it is a fully connected net. Figure5 shows the general structure of a fully connected two-layer neural network.

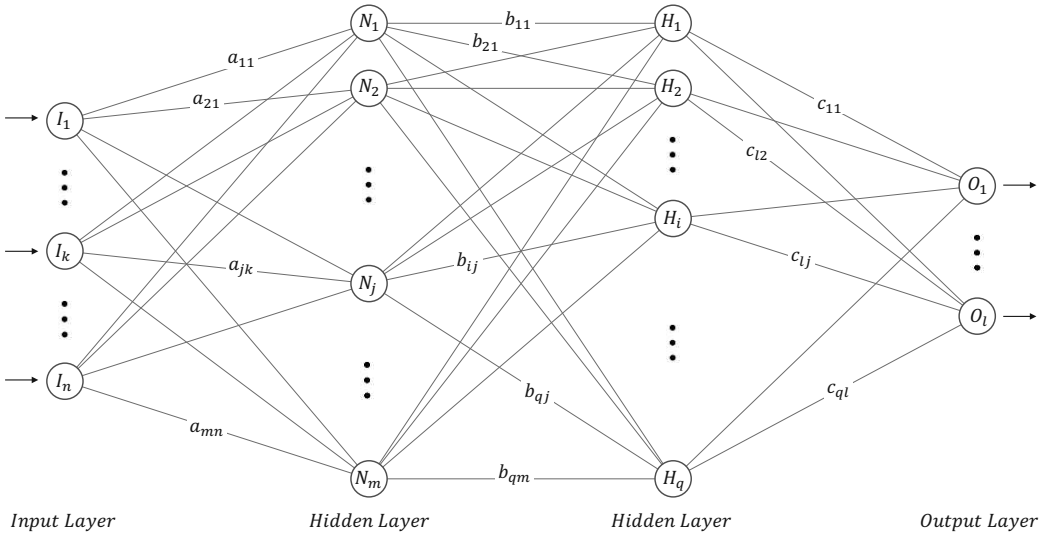


Figure 5.: *A general form of a fully connected two hidden layer neural network with varying neurons per layer. Inputs I, outputs O, hidden neurons N, H and connection weights a, b, c.*

2.3.5. Neural Network Training

The training of the NN consists of a learning rule that systematically adjusts the weights of the connections, i.e., the parameters of the NN. This learning process aims to approximate the predicted output y to the actual one, the target t. The Least Mean Square Error between the two outputs, defined for one neuron (Eq. (4)) in Equation (40), is used as the error criterion. Several data sets N are used to calculate the weights.

$$E = \sum_{k=1}^{N} E_k = \frac{1}{2} \sum_{k=1}^{N} (\underbrace{t_k - y_k}_{e_k})^2 \tag{40}$$

To minimize the error criterion, the weights are changed in the direction of the derivative of the error function with respect to each weight using Gradient Descent. The action rule, also known as Delta Rule, can be taken from Equation 41, where α is the step size and the search direction corresponds to the gradient for one input set k.

$$w_j \leftarrow w_j - \alpha \frac{\partial E_k}{\partial w_j} \tag{41}$$

For a multi-layer net, the back propagation learning rule is applied, which is a generalized gradient (Delta) rule for non-linear activation functions. The parameters are updated starting from the output in backward direction via the hidden layers.

3. Use Case and Numerical Experiments

This chapter deals with the integration of the simulation model of a small-scale pump turbine located at the test facilities of the Institute of Energy Technology and Thermodynamics (IET) at TU Wien into the RL environment, a more detailed description of the reversible pump turbine and its relevant parameters. The state and action spaces for the RL problem are then created, and a suitable reward function is defined. The different variants of the neural networks and their configuration, the application of the algorithms, and the influence of hyperparameters on the training and control behavior are presented.

3.1. Reinforcement Learning Environment

In a previous diploma thesis, a laboratory system of a small-scale pumped storage power plant was modeled, set up at the test facilities of IET at TU Wien [19]. The simulation was generated in the MatLab/Simulink environment [8, 9] using the Simscape toolbox [20]. Based on this work, the model of the pumped storage turbine for this application was integrated into the reinforcement learning environment as a black box (Fig. 6) with the corresponding inputs and outputs, see section 3.3.

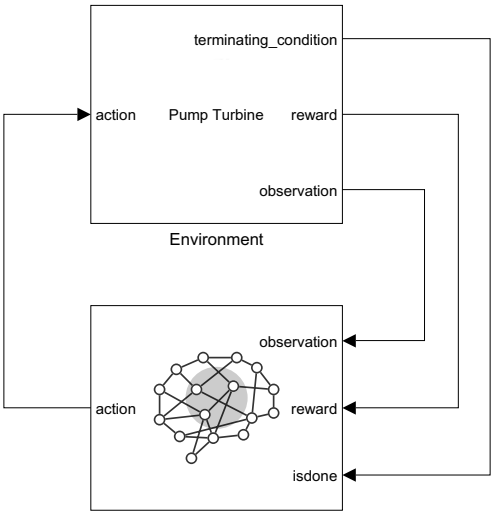


Figure 6.: *Reinforcement Learning Setup in Simulink with agent and environment connected by action, observation, reward and terminating condition.*

In addition to the observations and the reward, the terminating condition when

the last time step is reached, see Eq. (2), is passed. The agent trains its policy and executes an action. Some agents have unbounded actions that must be clipped to be physically meaningful. Constraints on the actions and normalization of the observations are executed at an intermediate level, as shown in Figure 7. In addition, the reward function for the state is evaluated; see Chapter 3.4.

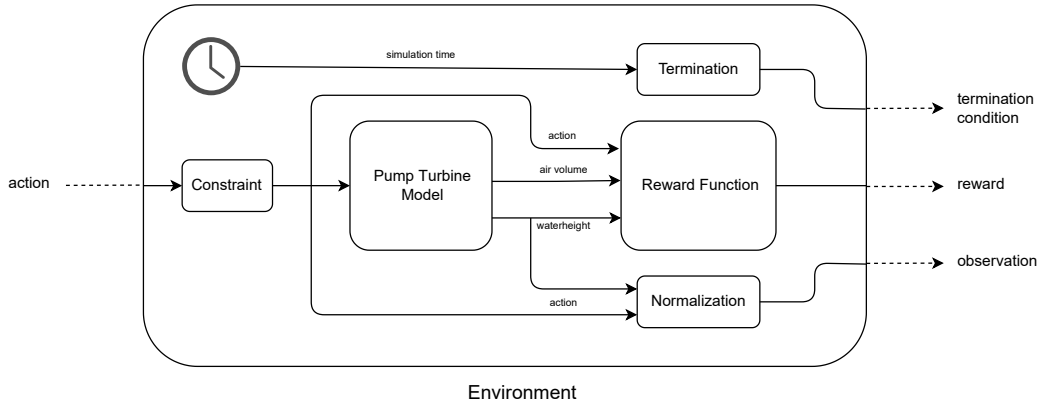


Figure 7.: *Representation of the environment subsystem where constraint, normalization, checking of the termination condition and calculation of the reward, as well as integration of the black-box model (pumped storage) pump-turbine, are encompassed.*

3.2. Hydro Power Plant Model

The centerpiece of the simulation model is the reversible pump turbine, shown in the cross-section in Figure 8. In turbine operation, the water flows through the radial housing, whereby the guide vane apparatus adjusts the mass flow. For pump operation, the direction of rotation of the runner is reversed. To switch between turbine and pump operation, the guide vanes are closed, the water in the housing is blown out by compressed air and the direction of rotation is reversed by using the electric machine as a motor. The guide vanes of the resulting pump system are then opened.

The main focus is on the blow-out process, whereby relevant heights are the maximum height possible, the blow-out height at which the runner is water-free and the critical height at which there is a risk of air entering the tailwater area. Even when the guide vane apparatus is closed, there is an opening and a resulting leakage. The pressure of the injected air is set constant and the runner is assumed to be at a standstill for blow-out. The values of the laboratory system are given in Table 2.

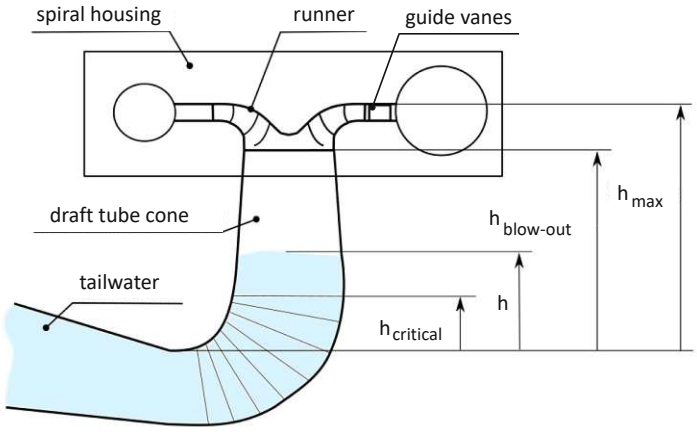


Figure 8.: *Schematic cross-section of the pump-turbine with height indication adapted from [5].*

3.3. State and Action Space Definition

The observations of the RL Agent, outputs from the environment, are the water height in the pump-turbine housing h_t in m and the momentary normalized injection air a_t in % for a timestep t (see Fig. 7). The guide vane opening g in m can also be recorded. The current normalized compressed air a_t in % is used as the actions of the RL agent, i.e., inputs to the model. In addition, the guide vane opening g can also be set in m.

To realize the blow-out process, a distinction must be made between value-based and actor-critic-based algorithms since their action space is set to discrete and continuous, respectively. For this reason, the definition of the state and action

Table 2.: *Overview of relevant parameters in the pump turbine.*

Property	Variable	Value	Unit
maximum height	h_{max}	0.807	m
initial height	h_{init}	0.767	m
blow-out height	$h_{\text{blow-out}}$	0.639	m
critical height	h_{crit}	0.2	m
injected air pressure	p	8	bar
minimal guide vane opening	g_{min}	0.001	m
maximum guide vane opening	g_{max}	0.034	m
rotational speed runner	n	0	rpm

spaces is designed differently. In doing so, the simplest possible state space for a complete problem description is considered.

3.3.1. Discrete Action Space

In the case of the discrete controller the observation space consists of the water level in the turbine housing and the previously executed action, the amount of compressed air, which can be seen in Eq. (42). With these two pieces of information, the problem of the turbine blow-out process is completely defined—the height and whether it rises or falls can be derived. The compressed air is used as the action, which is regulated discretely between the maximum quantity and no air, see Eq. (43).

$$S_D = \begin{cases} \text{water-height } h_t & \in [0, h_{\max}] \text{ m} \\ \text{air supply } a_{t-1} & \in \{0, 100\} \ \% \end{cases} \tag{42}$$

$$A_D = \Big\{ \text{air supply } a_t \quad \in \{0, 100\} \ \% \tag{43}$$

3.3.2. Continuous Action Space

A state space similar to the discrete one can be selected for the continuous case. However, the previously recorded water level can also replace the previously used action. For the continuous control, the water level at the current and previous time is used for observation (Eq. (44)). This results in information about the change in height and the height itself, which fully describes the problem. The compressed air is continuously regulated between zero and maximum volume and forms the action space (Eq. (45)).

$$S_C = \begin{cases} \text{water-height } h_t & \in [0, h_{\max}] \text{ m} \\ \text{water-height } h_{t-1} & \in [0, h_{\max}] \text{ m} \end{cases} \tag{44}$$

$$A_C = \Big\{ \text{air supply } a_t \quad \in [0, 100] \ \% \tag{45}$$

3.3.3. Extended Continuous Action Space

To demonstrate the straightforward extensibility of the problem, the guide vane opening is included based on the control with continuous action space. Now, not only the blow-out process but also the closing process of the guide vanes is examined. The state space can be seen in Eq. (46), where the guide vane opening is added and the action space by the control of the vanes, see Eq. (47). This use case demonstrates how the problem description can be more complex based on the existing control system.

$$S_{CC} = \begin{cases} \text{water-height } h_t & \in [0, h_{\max}] \text{ m} \\ \text{water-height } h_{t-1} & \in [0, h_{\max}] \text{ m} \\ \text{guide vane opening } g_{t-1} & \in [0, g_{\max}] \text{ m} \end{cases} \tag{46}$$

$$A_{CC} = \begin{cases} \text{air supply } a_t & \in [0, 100] \text{ \%} \\ \text{guide vane opening } g_t & \in [0, g_{\max}] \text{ m} \end{cases} \tag{47}$$

3.4. Reward Function Definition

The reward function includes the objective of the control task but must not contain any prior information that could influence the training. This could effect the training towards sub-optimal policies. The blow-out process aims to reach a previously defined water level range (Chapter 3.4) as quickly as possible and to remain in this range in a controlled manner. Furthermore, the amount of compressed air should be minimized. These requirements result in the reward function

$$R_s^a = h_r - |a_t - a_{t-1}| - a_t,\ h_r = \begin{cases} -h_t & \text{for } h_t \geq h_{\text{blow-out}} \\ +1 & \text{for } h_{\text{blow-out}} > h_t \geq h_{\text{crit}} \\ -1 & \text{for } h_{\text{crit}} > h_t \end{cases}. \tag{48}$$

The first term of Equation (48) contains the desired range of water height, which is therefore given a positive value, with non-preferred heights weighted negatively. Implicitly, there is also a time dependency, as the positively weighted range should be reached as quickly as possible to maximize the return in Equation (6). The change and the amount of air consumed are weighted negatively, according to which they should be used minimally.

3.5. Neural Network Architecture

3.5.1. Neural Networks for Value-based Algorithms

For the Reinforcement Learning algorithms *Q*, *SARSA* and *DQN*, which are value-based (2.2.1) and have a discrete action space, a Q-value critic or action-value critic is required, see equation (9). This is described using a neural network, which opens up two possibilities for implementation. Either the Q-value can be calculated for the observations and actions (Q-value critic 1), or a Q-value can be approximated for each possible action for the observations (Q-value critic 2). This results in one neuron that describes the Q-value for the first critic and two neurons that each represent the Q-value for the two different discrete actions for the second critic. The number of neurons per layer is denoted by N_c for the critic and N_a for the actor.

Q-value Critic 1

In this neural network, hereafter referred to as Q-value critic 1, the action (input I_1) and the observation (input I_2 and I_3) are treated separately and then merged. This neural network has three input neurons, one output neuron (O_1), and three hidden layers, as shown in Figure 9.

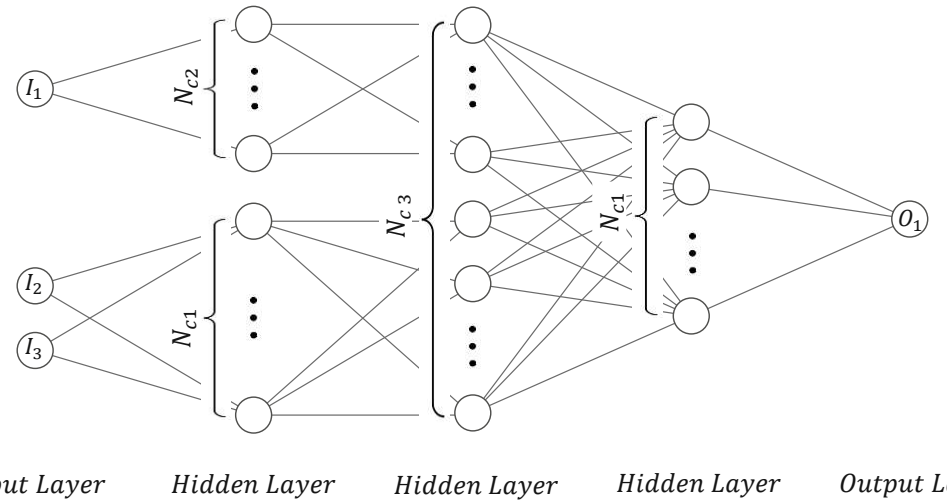


Figure 9.: *Q-value critic with observations (I_2, I_3) and action (I_1) as input and Q-value (O_1) as output as well as number of neurons per layer N_{c1}, N_{c2} and N_{c3}.*

The different versions of the Q-value critic 1 neural network are listed in Table 3. There is always a factor of 2 between the number of neurons per layer for each different version and the path of the actions (N_{c2}) is chosen to be half as wide as that of the observations (N_{c1}). The merged path (N_{c3}) corresponds to the added number of neurons per layer. The number of learnable parameters indicates the degrees of freedom and corresponds to the weights of the connections.

Table 3.: *Neural net parameters for Q-value critic 1.*

Neural net name	Number of neurons per layer N_{c1}	N_{c2}	N_{c3}	Learnable parameters critic
NN1.1	128	64	192	46017
NN1.2	64	32	96	11745
NN1.3	32	16	48	3057
NN1.4	16	8	24	825
NN1.5	8	4	12	237
NN1.6	4	2	6	75
NN1.7	2	1	3	27

Q-value critic 2

In this neural network version, Q-value critic 2, the observation (input I_1 and I_2) is mapped to two outputs (O_1 and O_2). Two hidden layers are used for this fully connected net, as shown in Figure 10.

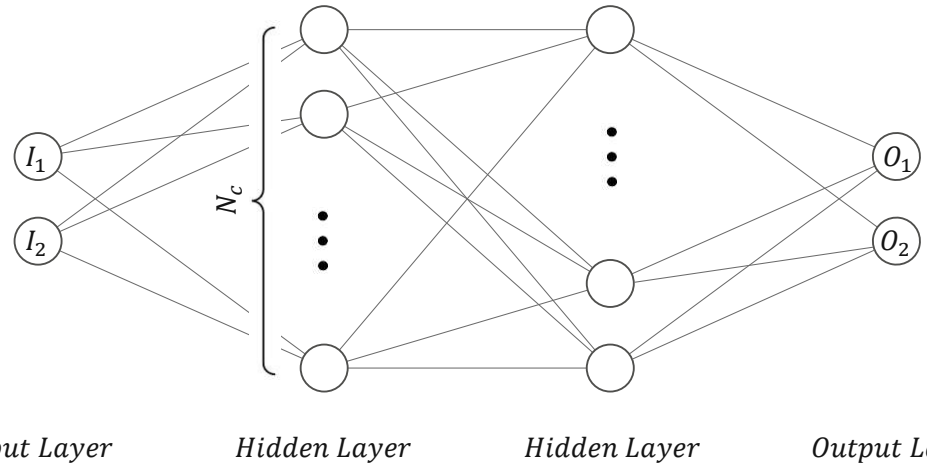


Figure 10.: *Q-value critic with observation (I_1, I_2) as input and Q-values for each action (O_1, O_2) as output as well as number of neurons per layer N_c.*

The different versions of the Q-value critic 2 neural network are listed in Table 4, where there is a factor of 2 between the number of neurons per layer.

Table 4.: *Neural net parameters for Q-value critic 2.*

Neural net name	Number of neurons per layer N_c	Learnable parameters critic
NN2.1	256	67074
NN2.2	128	17154
NN2.3	64	4482
NN2.4	32	1218
NN2.5	16	354
NN2.6	8	114
NN2.7	4	42
NN2.8	2	18

3.5.2. Neural Networks for actor-critic-based Algorithms

For the actor-critic-based algorithms *AC*, *TRPO* and *PPO* (2.2.2), which are used with a continuous action space, a state-value critic, see Eq. (8), and an actor, see

Eq. (5), are used. The state-value critic is structured similarly to the Q-value critic 2, requiring only one output. It takes the observation as input (input I_1 and I_2) and approximates the state-value (output O_1). This can be seen in Figure 11 and the parameters are listed in Table 5.

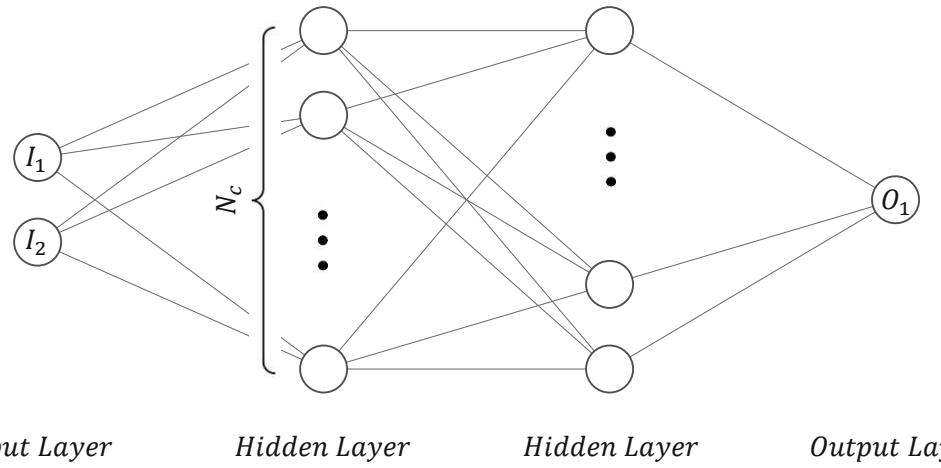


Figure 11.: *State-value critic with observation (I_1, I_2) as input and state-value (O_1) as output as well as number of neurons per layer N_c.*

The actor, which can be seen in Figure 12, uses the observation (input I_1 and I_2) as input and calculates a mean value (output O_2) and a standard deviation (output O_1) for the action according to Eq. (5). For this purpose, a softmax activation layer (normalized exponential function) is used in the neuron prior to the output (O_1). The parameters are in the Table 5.

Table 5.: *Neural network parameters for state-value critic and actor.*

Experiment	Number of neurons per layer		Learnable parameters	
name	N_a	N_c	actor	critic
NN3.1	256	256	67074	66817
NN3.2	128	128	17154	17025
NN3.3	64	64	4482	4417
NN3.4	48	48	2594	2545
NN3.5	32	32	1218	1185
NN3.6	24	24	722	697
NN3.7	16	16	354	337
NN3.8	8	8	114	105
NN3.9	4	4	42	37
NN3.10	2	2	18	15

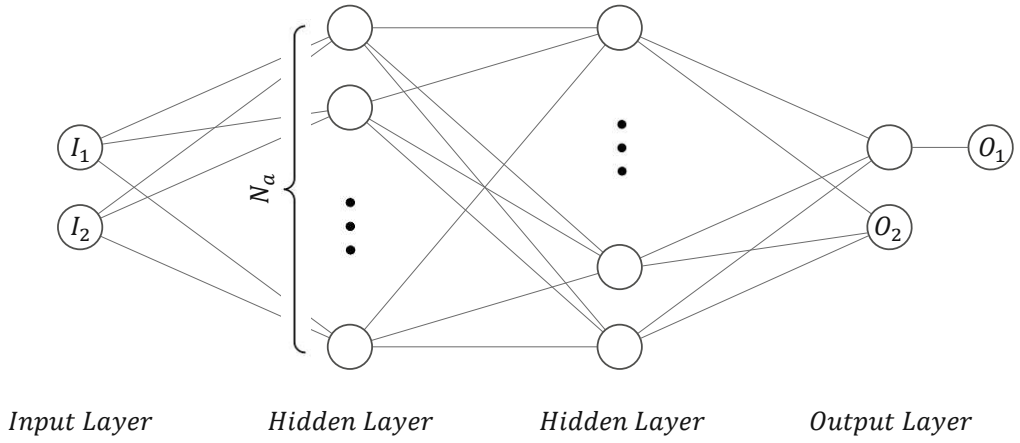


Figure 12.: *Actor with observation (I_1, I_2) as input and mean (O_2) and standard deviation (O_1) as output as well as number of neurons per layer N_a.*

3.6. Reinforcement Learning Algorithms

3.6.1. General Settings

The conventional hysteresis and PID controller for the discrete or continuous action spaces provide the returns defined as optimal that can be achieved with the given reward function, see chapter 3.4. To find a measure of how well the policy behaves in comparison to the optimum, a lower limit is required. About the control behavior, the minimum is defined if no action is selected at all, i.e., the controlled variable compressed air is 0 % for the entire simulation. In between, the required limit returns can be computed for certain thresholds shown in Tab. 6. The values refer to a sample time of 1 s.

Table 6.: *Average rewards and threshold values for discrete and continuous action spaces.*

threshold	average reward	
value	discrete	continous
min.	-97.58	-97.58
max.	93.85	92.88
99 %	91.93	90.98
98 %	90.02	89.07
95 %	84.28	83.36
90 %	74.71	73.83

Relevant parameters in the RL training are given in Tab. 7 and consist of the

simulation and sample time, the discount factor, and the number of episodes averaged for the average reward value. A minimum exploration ϵ_{min} is defined for the value-based algorithms. A mini-batch size and update frequency of the parameters are set for both thethe actor-critic representatives. The L_2 factor, an additional term in the loss function, helps avoid overfitting the NN parameters when it is increased. In order to exclude any starting deviations in the training progress, the neural networks were always initialized in the same way.

Table 7.: *Overview of relevant parameters in the RL training.*

Property	Variable	Value	Unit
simulation time	t_{sim}	120	s
sample time	t_{sample}	1	s
averaging window length		15	-
discount factor	γ	0.99	-
minimal exploration	ϵ_{min}	0.001	-
mini-batch size DQN	M	64	-
mini-batch size PPO	M	128	-
update frequency		1	-
regularization factor	L_2	0.005	-

3.6.2. Preceding Iterative Process

The value-based algorithms *Q*, *SARSA*, *DQN* and the actor-critic-based algorithms *AC*, *TRPO*, *PPO* are compared with each other in terms of performance. This includes the maximum reward achieved and the number of episodes to reach a certain threshold. This requires an adequate neural network and a robust choice of hyperparameters to enable a stable and fast training process that achieves high rewards. These requirements result in an iterative process where a very broad network (256 neurons/layer) is initially used to find robust parameter sets for all six algorithms. This includes a preliminary hyperparameter study to obtain convergent training curves. Based on initial qualitative differences in performance, the most promising algorithm in each category is selected. The assumption is that *DQN* is the most suitable value-based agent. For actor-critic agents, it is assumed that *PPO* has the superior behavior in terms of performance. These assumptions need to be verified.

The actual neural network analysis (NNA) study can then be carried out with a defined set of parameters. The most suitable network is then used for the hyperparameter analysis (HPA) study. The comparison of the algorithms using the default and the chosen network for the best-performing parameters is a way of confirming the previous choice of algorithm for each category.

In the following, unless otherwise stated, the D in the experiment names refers to

discrete action space (value-based) for the *DQN* algorithm. The same applies to *C* in the experiment names, i.e., continuous action space (actor-critic-based) for the *PPO* algorithm.

3.6.3. Neural Net Analysis Study

The neural networks of the value-based representative *DQN* and actor-critic-based variant *PPO* are analyzed in terms of how many neurons per layer are required and how many episodes must be trained to reach a specific threshold value which is defined in Tab. 6. The experimental configurations used for this can be found in Tables 8, 9, and 10.

3.6.4. Hyperparameter Analysis Study

The learning rate α, the exploration ϵ and the exploration decay rate ϵ_{decay} are identified as hyperparameters for the analysis for the value-based algorithms with a discrete action space, see Chapter 2.2.1. The learning rates (actor α_{actor} and critic α_{critic}) and the entropy loss weight w are considered for the actor-critic-based representatives; see Chapter 2.2.2. In the simulations, one parameter is changed at a time, and the others are fixed to identify correlations.

For both variants of the Q-value critic, the learning rate is varied in the range of 4 orders of magnitude between 10^{-1} and 10^{-5}. The experimental parameter sets can be found in Table 11 and 12. Table 13 shows the hyperparameters used for the variation of the learning rates of actor and critic for the *PPO* algorithm. The learning rates are also varied in the same order of magnitude. For each case, the same neural network is used.

For the *DQN* algorithm, the influence of exploration and its rate of decline over time is analyzed. The parameter sets used for this study can be found in Tables 14 and 16. The exploration is analyzed in a range of 1 to 100 %. The exploration decay rate varies between $5 * 10^{-3}$ and 10^{-5}. The influence of entropy loss weight is to be investigated for the *PPO* algorithm. A range between 0 and 90 % is selected for this. Table 15 shows the parameter sets used for this study.

3.6.5. Comparison of Algorithms

The selected neural networks and parameter sets are used for the comparison in the group of value-based (*Q*, *SARSA*, *DQN*) and actor-critic-based (*AC*, *TRPO*, *PPO*) algorithms. In addition, the tests are repeated with a wide NN (256 neurons/layer). The test parameters can be taken from Table 17 for the value-based algorithms and Table 18 for the actor-critic-based representatives. The hyperparameters used differ within the comparison group, as they each require different parameter sets for fast and robust convergence. For the algorithms Q, SARSA, AC and TRPO, this was determined in a previous iterative process, which is not part of this work.

Table 8.: *Neural net analysis parameters for Q-value critic 1.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D1.1	NN1.1			
D1.2	NN1.2			
D1.3	NN1.3			
D1.4	NN1.4	$1 \cdot 10^{-3}$	50 %	$1 \cdot 10^{-4}$
D1.5	NN1.5			
D1.6	NN1.6			
D1.7	NN1.7			

Table 9.: *Neural net analysis parameters for Q-value critic 2.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D1.8	NN2.1			
D1.9	NN2.2			
D1.10	NN2.3			
D1.11	NN2.4	$1 \cdot 10^{-3}$	50 %	$1 \cdot 10^{-4}$
D1.12	NN2.5			
D1.13	NN2.6			
D1.14	NN2.7			
D1.15	NN2.8			

Table 10.: *Neural net analysis parameters for state-value critic and actor.*

Experiment name	Neural network	Learning rate actor α_{actor}	Learning rate critic α_{critic}	Entropy loss weight w
C1.1	NN3.1			
C1.2	NN3.2			
C1.3	NN3.3			
C1.4	NN3.4			
C1.5	NN3.5	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	10 %
C1.6	NN3.6			
C1.7	NN3.7			
C1.8	NN3.8			
C1.9	NN3.9			
C1.10	NN3.10			

Table 11.: *Hyperparameter learning rate analysis for Q-value critic 1.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D2.1		$1 \cdot 10^{-1}$		
D2.2		$5 \cdot 10^{-2}$		
D2.3		$1 \cdot 10^{-2}$		
D2.4		$5 \cdot 10^{-3}$		
D2.5	NN1.4	$1 \cdot 10^{-3}$	50 %	$1 \cdot 10^{-4}$
D2.6		$5 \cdot 10^{-4}$		
D2.7		$1 \cdot 10^{-4}$		
D2.8		$5 \cdot 10^{-5}$		
D2.9		$1 \cdot 10^{-5}$		

Table 12.: *Hyperparameter learning rate analysis for Q-value critic 2.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D2.10		$1.0 \cdot 10^{-1}$		
D2.11		$5.0 \cdot 10^{-2}$		
D2.12		$1.0 \cdot 10^{-2}$		
D2.13		$5.0 \cdot 10^{-3}$		
D2.14		$2.5 \cdot 10^{-3}$		
D2.15	NN2.4	$1.0 \cdot 10^{-3}$	50 %	$1 \cdot 10^{-4}$
D2.16		$7.5 \cdot 10^{-4}$		
D2.17		$5.0 \cdot 10^{-4}$		
D2.18		$1.0 \cdot 10^{-4}$		
D2.19		$5.0 \cdot 10^{-5}$		
D2.20		$1.0 \cdot 10^{-5}$		

3.6.6. Validation

A modified initial state that was not initially used for learning can be specified to validate the trained RL policies. The water level at the start of the blow-out process is varied. In addition, the opening degree of the guide vanes can be changed, and thus, the leakage of the turbine housing. This can be seen in Tables 19 and 20.

3.6.7. Extended Use Case

For the more complex use case, the state and action spaces are changed according to the Chapter 3.3.3 and trained with the following parameter set, see Table 21. This corresponds to that of experiment C1.1.

Table 13.: *Hyperparameter learning rate analysis for state-value critic and actor.*

Experiment name	Neural network	Learning rate actor α_{actor}	Learning rate critic α_{critic}	Entropy loss weight w
C2.1	NN3.5	$1 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	10 %
C2.2			$1 \cdot 10^{-4}$	
C2.3			$1 \cdot 10^{-3}$	
C2.4			$1 \cdot 10^{-2}$	
C2.5			$1 \cdot 10^{-1}$	
C2.6		$1 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	
C2.7			$1 \cdot 10^{-4}$	
C2.8			$1 \cdot 10^{-3}$	
C2.9			$1 \cdot 10^{-2}$	
C2.10			$1 \cdot 10^{-1}$	
C2.11		$1 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	
C2.12			$1 \cdot 10^{-4}$	
C2.13			$1 \cdot 10^{-3}$	
C2.14			$1 \cdot 10^{-2}$	
C2.15			$1 \cdot 10^{-1}$	
C2.16		$1 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	
C2.17			$1 \cdot 10^{-4}$	
C2.18			$1 \cdot 10^{-3}$	
C2.19			$1 \cdot 10^{-2}$	
C2.20			$1 \cdot 10^{-1}$	
C2.21		$1 \cdot 10^{-1}$	$1 \cdot 10^{-5}$	
C2.22			$1 \cdot 10^{-4}$	
C2.23			$1 \cdot 10^{-3}$	
C2.24			$1 \cdot 10^{-2}$	
C2.25			$1 \cdot 10^{-1}$	

Table 14.: *Hyperparameter exploration analysis for Q-value critic 2.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D3.1	NN2.4	$1 \cdot 10^{-3}$	100 %	$1 \cdot 10^{-4}$
D3.2			90 %	
D3.3			70 %	
D3.4			50 %	
D3.5			30 %	
D3.6			10 %	
D3.7			1 %	

Table 15.: *Hyperparameter entropy loss weight analysis for state-value critic and actor.*

Experiment name	Neural network	Learning rate actor α_{actor}	Learning rate critic α_{critic}	Entropy loss weight w
C3.1				0 %
C3.2				1 %
C3.3				5 %
C3.4				10 %
C3.5	NN3.5	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	15 %
C3.6				20 %
C3.7				50 %
C3.8				70 %
C3.9				90 %

Table 16.: *Hyperparameter exploration decay rate analysis for Q-value critic 2.*

Experiment name	Neural network	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D3.8				$5.0 \cdot 10^{-3}$
D3.9				$1.0 \cdot 10^{-3}$
D3.10				$5.0 \cdot 10^{-4}$
D3.11	NN2.4	$1 \cdot 10^{-3}$	50 %	$1.0 \cdot 10^{-4}$
D3.12				$7.5 \cdot 10^{-5}$
D3.13				$5.0 \cdot 10^{-5}$
D3.14				$1.0 \cdot 10^{-5}$

Table 17.: *Comparison of value-based algorithms.*

Experiment name	Neural network	Algorithm	Learning rate α	Exploration ϵ	Exploration decay rate ϵ_{decay}
D4.1	NN1.4	Q	$1 \cdot 10^{-4}$	50 %	
D4.2	NN2.4				
D4.3	NN1.4	SARSA	$1 \cdot 10^{-3}$	1 %	$1 \cdot 10^{-4}$
D4.4	NN2.4				
D4.5	NN1.4	DQN	$1 \cdot 10^{-3}$	50 %	
D4.6	NN2.4				

Table 18.: *Comparison of actor-critic-based algorithms.*

Experiment name	Neural network	Algorithm	Learning rate actor α_{actor}	Learning rate critic α_{critic}	Entropy loss weight w
C4.1 C4.2	NN3.1 NN3.5	AC	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	
C4.3 C4.4	NN3.1 NN3.5	TRPO	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	10 %
C4.5 C4.6	NN3.1 NN3.5	PPO	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	

Table 19.: *Validation of DQN and PPO agent with varied initial heights.*

Initial water height	Percentage of total height
0.16 m	20 %
0.4 m	50 %
0.65 m	80 %

Table 20.: *Validation of DQN and PPO agent with varied guide vane openings.*

Guide vane opening	Percentage of total opening
0.006 m	18 %
0.014 m	41 %
0.034 m	100 %

Table 21.: *Extended use case.*

Experiment name	Neural network	Algorithm	Sample time	Learning rate actor α_{actor}	Learning rate critic α_{critic}	Entropy loss weight w
C5.1	NN3.1	PPO	0.25 s	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	10 %

4. Results and Discussion

This chapter starts by showing the control behavior of the water height in the pump-turbine housing and the underlying policy for the *DQN* and *PPO* algorithms. The training processes that led to the control principle are shown. The different variants of the neural networks are subsequently analyzed in a study. The hyperparameter analysis presents the influence of relevant training parameters on performance. Based on this, the algorithms of the value-based (*Q*, *SARSA*, *DQN*) and actor-critic-based (*AC*, *TRPO*, *PPO*) representatives are compared with each other. The control behavior of the *DQN* and *PPO* algorithms is validated by demonstrating different use cases. Finally, the guide vane opening extends the use case, displaying the resulting control behavior.

4.1. Policy Deployment

The policy or control strategy corresponding to the parameterized neural network is extracted from the fully trained agent. This acts as a controller on the environment, whereby the observation is taken as input, and the calculated output on the environment corresponds to the manipulated variable.

4.1.1. DQN Discrete Output

Figure 13 displays the control behavior of the water level in the turbine housing due to the compressed air blown into the runner. The compressed air volume is either maximum or 0, also called discrete action, as two states are possible. The diagram shows the water level (h_t in m, blue) and the air supply (a_t in %, red) over time (t in s) for the *DQN* RL control with a solid line and the hysteresis controller with a dashed line. The blow-out and critical height (black) are also plotted. For Chapters 4.1 and 4.2, experiment D1.11, see Tab. 9, is shown as an example of the *DQN* agent.

The curves of the different control concepts for discrete outputs are on top of each other; they are identical. For this reason, the trained RL agent reaches the optimum previously defined by the two-point controller. As the observations required to calculate the action according to the policy are not yet available at the start of the simulation, the agent must wait for two time steps. As the reward function, defined in Chapter 3.4, weights the compressed air and the change in air volume with the same factor, it is better to blow in longer and less often and cause a lower water level. This results in a higher average reward.

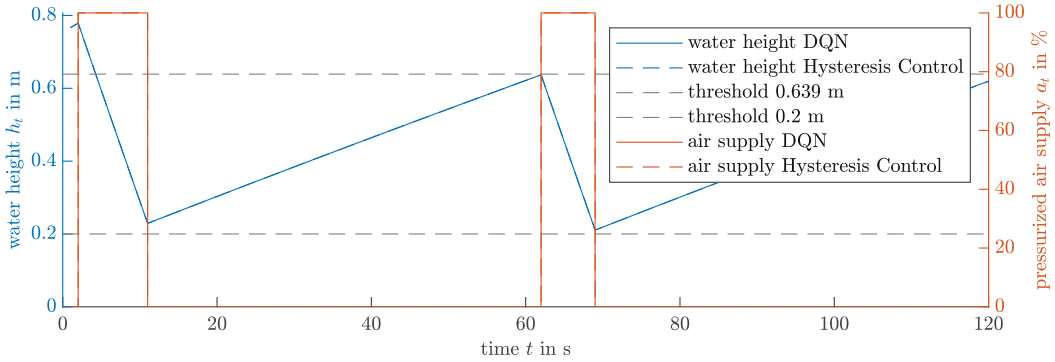


Figure 13.: *Control behavior of DQN policy & hysteresis controller for the water level corresponding to the compressed air blown into the runner over time.*

4.1.2. DQN Policy

Figure 14 illustrates the policy, whereby two curves (red and blue) are shown for the observations of water height and previously executed action, the compressed air. This visual representation corresponds to the policy described by the underlying neural network.

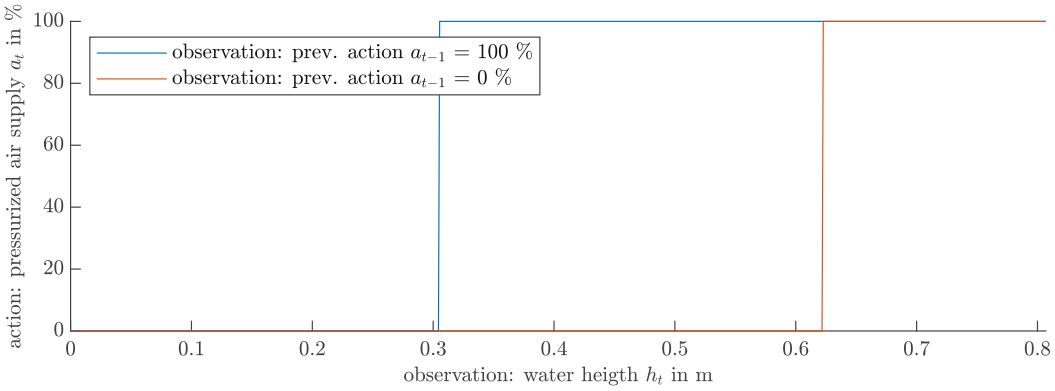


Figure 14.: *DQN RL agent policy: pressurized air supply over water height.*

4.1.3. PPO Continuous Output

Figure 15 displays the control characteristics of the water level in the turbine housing due to a continuously controlled air supply. It shows the water level (h_t in m, blue) and the air supply (a_t in %, red) over time (t in s) for the *PPO* RL control (solid) and the PID controller (dashed). For Chapters 4.1 and 4.2, experiment C1.10, see Tab. 10, is shown as an example of the *PPO* agent with a modified sampling time of $t_{\text{sample}} = 0.25$ s.

In continuous control, the trained PPO agent can reproduce a similar control behavior to the PID controller. The curves are approximately superimposed. The neural network consists of 2 neurons per layer, resulting in 18 degrees of freedom for the actor and 15 for the critic.

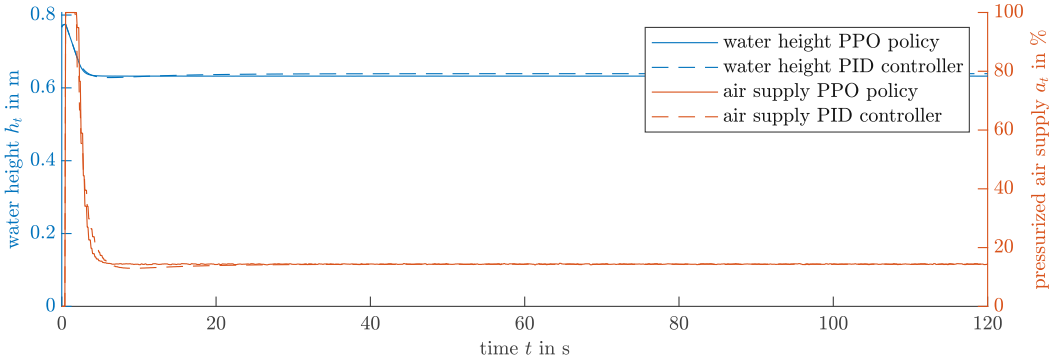


Figure 15.: *Control behavior of PPO policy & PID controller for the water level corresponding to the compressed air blown into the runner over time.*

4.1.4. PPO Policy

The underlying policy can be represented as a surface plot (Figure 16), where the water height at two consecutive points in time is used as the input and the action as the height coordinate. The yellower, the more compressed air is used; the bluer, the less. Three different ranges can be recognized: 100 % compressed air, where the water level is above the level at which the runner is considered to be blown out. 0 %, where the water level is far below the blow-out height. The air supply in between depends on how close the level is below the blow-out height and how high it was one-time step earlier.

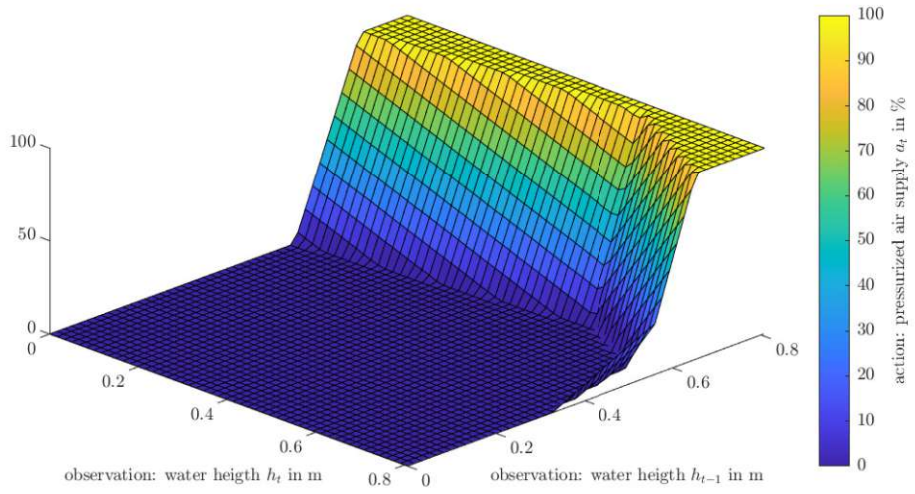


Figure 16.: *PPO RL agent policy: pressurized air supply over water heights.*

4.2. Training Curve

The training progression shows how the episode reward (blue) and the moving average reward over 15 episodes (red), calculated from this, change over episodes. The episode reward varies significantly at the beginning of the training, but the fluctuation becomes smaller and smaller as the training progresses. The average value increases (approximately) steadily.

4.2.1. DQN Training Curve

Figure 17 shows an example of the training curve of the *DQN* agent for experiment D1.11. The threshold value (black dashed) is specified as 99 % of the optimum and is reached after about 500 episodes.

4.2.2. PPO Training Curve

Figure 18 displays an example of the training curve of the *PPO* agent for the modified experiment C1.10. The threshold value (black dashed) is specified as 99 % of the optimum for this configuration and is reached after about 3850 episodes. Due to the changed sample time, the episode and average reward also change accordingly. As this is a finer time resolution for illustrative purposes only, the limits have not been stated.

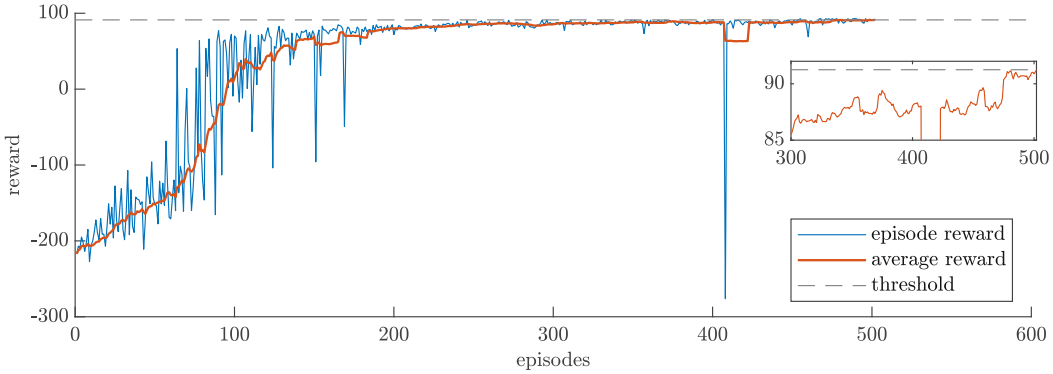


Figure 17.: *Training curve DQN agent, episode and average reward over episodes and threshold value.*

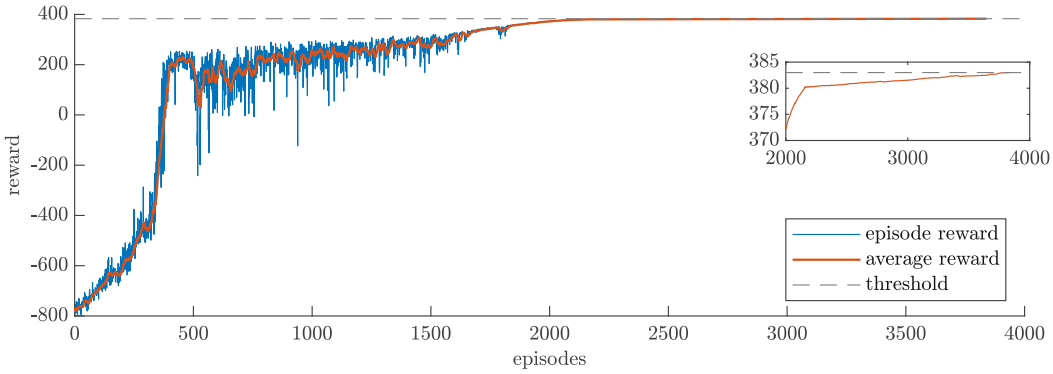


Figure 18.: *Training curve PPO agent, episode and average reward over episodes and threshold value.*

4.3. Neural Network Analysis Study

The neural network analysis (NNA) study investigates which architecture with which number of neurons per layer is best suited to achieve a particular threshold value.

4.3.1. DQN Critic

Figures 19 and 20 show the NNA study for the *DQN* agent and the Q-value critic 1 and 2, respectively, with episodes to reach the thresholds plotted over the learnable parameters of the neural network. The specific limits are given as 90 % (yellow), 95 % (red) and 98 % (blue) of the previously defined optimum.

The experiments used in Figure 19 are defined in Table 8 and relate to the neural network structures defined in Table 3. The number of degrees of freedom equals the number of parameters that can be learned. Experiments D1.1 with neural networks NN1.1 to D1.7 and NN1.7 are plotted from left to right.

The experiments used in Figure 20 are defined in Table 9 and relate to the neural network structures defined in Table 4. Experiments D1.8 with neural networks NN2.1 to D1.15 and NN2.8 are plotted from left to right.

For the studied range of approximately 10^1 to $5 \cdot 10^4$ degrees of freedom, the number of neurons used per layer has almost no influence on the Q-value critic 1 at a low threshold value of 90 % of the optimum. A difference in the number of training episodes can only be seen at higher target values. As narrow NNs only have a few neurons per layer and few parameters, the values must be finely tuned to each other to map the functional relationships. This requires a longer training phase, as progress can be very slow at the beginning of training. This can be seen in Figure 38 in the Appendix. A good compromise for the choice of neural network structure is the smallest number of neurons per layer for good generalization but so many that the functional relationship can be reproduced with the required accuracy. The local minimum number of episodes needed to reach the defined threshold is approximately 800 degrees of freedom. NN1.4 is selected and can be recognized by the black vertical line.

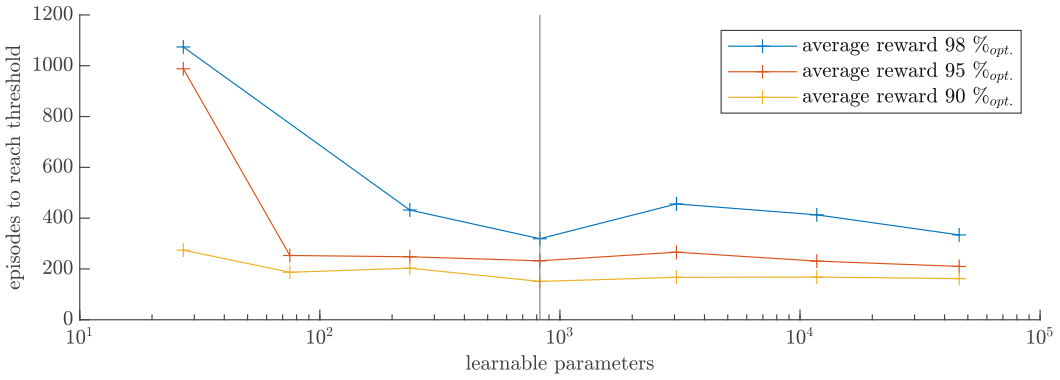


Figure 19.: *Neural network analysis study for DQN agent and Q-value critic 1 with episodes to reach threshold over learnable parameters of the neural network.*

The Q-value critic 2 results in an increase in the number of episodes for narrow neural networks that cannot fully map the degree of complexity. The corresponding training curve can be seen in Figure 39 in the Appendix. Only with more neurons per layer (≥ 8) can the required limit of 98 % be achieved. Very broad NNs can make rapid training progress in the beginning due to the high number of adjustable parameters. However, they may have problems reaching the optimum because there are too many degrees of freedom to be adjusted. More emphasis is placed on achieving the required average reward (98%) for the NN choice. Furthermore, a fast convergence is advantageous and if several structures show similar behavior, then the one with the fewest neurons per layer is chosen for better generalization. The

network NN2.4 is selected with approximately 1200 degrees of freedom and can be recognized by the black line.

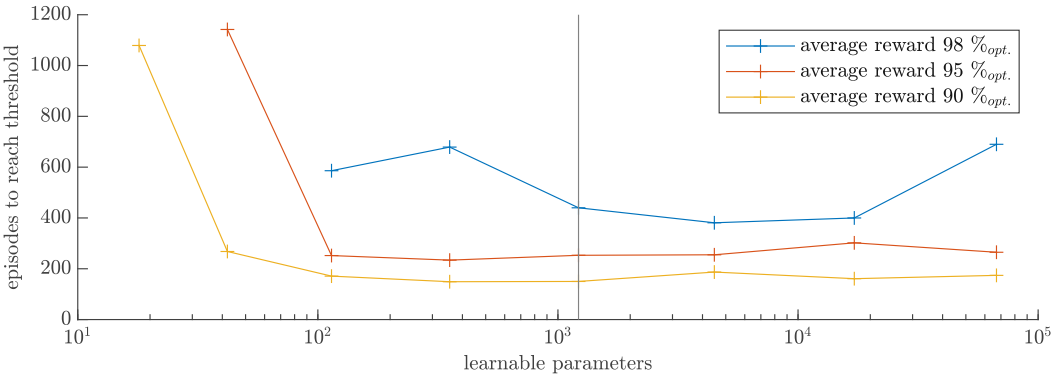


Figure 20.: *Neural network analysis study for DQN agent and Q-value critic 2 with episodes to reach threshold over learnable parameters of the neural network.*

At this point, it is not yet possible to say which critic structure is preferred for this problem, so both are analyzed regarding the learning rate.

4.3.2. PPO Actor-Critic

Figure 21 shows the NNA study for the *PPO* agent and state-value critic and actor with episodes for reaching threshold values plotted over learnable parameters of the actor. The specific limits are given as 90 % (yellow), 95 % (red) and 99 % (blue) of the previously defined optimum.

The experiments used in Figure 21 are defined in Table 10 and refer to the neural network structures for the critic and actor defined in Table 5. They are connected by the number of parameters that can be learned. Experiments C1.1 with neural networks NN3.1 to C1.10 and NN3.10 are plotted from left to right.

The network structure for the actor-critic-based algorithms is varied so that both the critic and actor are changed in width simultaneously. For this reason, the two NNs must be regarded as one training structure. It can be seen that only four variants reach the limit value of 99 % of the optimum. There is a much stronger dependence on the degrees of freedom, which indicates a more complex problem in the continuous case. Broad NN versions show faster convergence, although the required 99 % cannot be achieved with 128 neurons per layer. However, slower convergence is accepted due to better generalization properties with fewer degrees of freedom. It is also assumed that stronger dependencies of the hyperparameters can be shown with less complex network structures.

The neural network, NN3.5, with approximately 1200 degrees of freedom for each actor and critic, is again selected, which offers the best compromise between a low number of neurons (32) and reaching the optimum in a small number of episodes. The training progressions can be seen in Figures 40 and 41 in the Appendix.

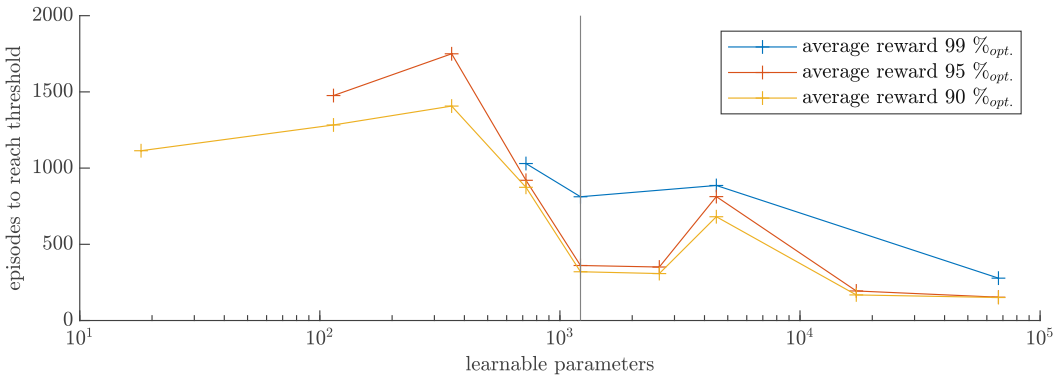


Figure 21.: *Neural network analysis study for PPO agent and state-value critic and actor with episodes to reach threshold over learnable parameters of the neural network.*

4.4. Hyperparameter Analysis Study

The hyperparameter analysis (HPA) study aims to determine the dependencies of parameters on training behavior. To this end, the learning rate(s), the exploration, the exploration decay rate and the entropy loss weight are examined. The hyperparameters were selected so that it is assumed that they influence the learning process and occur across all algorithms (of the group). There are more influencing factors, but the analysis of these is not part of the work.

4.4.1. DQN Learning Rate

Figures 22 and 23 show the HPA study for the *DQN* agent and the Q value critic 1 and 2, respectively, with the episodes for reaching the thresholds plotted against the learning rate α. The specific limits are given as 90 % (yellow), 95 % (red) and 98 % (blue) of the previously defined optimum. The experiments in Figure 22 are defined in Table 11 with the neural network NN1.4. Figure 23 parameter sets can be found in Table 12 with NN2.4.

If the two variants of the learning rate HPA are compared with each other, it can be seen that a noticeable difference only occurs at higher target values (average reward $\geq 95\ \%_{\text{opt}}$). For both, there is a minimum number of episodes to reach the threshold value of 98 $\%_{\text{opt}}$ and is therefore marked as the best learning rate with a

vertical line (black). Since a larger range of learning rates leads to good results for the simple neural network structure used in Q-value critic 2, see Chapter 3.5.1, this is preferred and used for the subsequent HPAs.

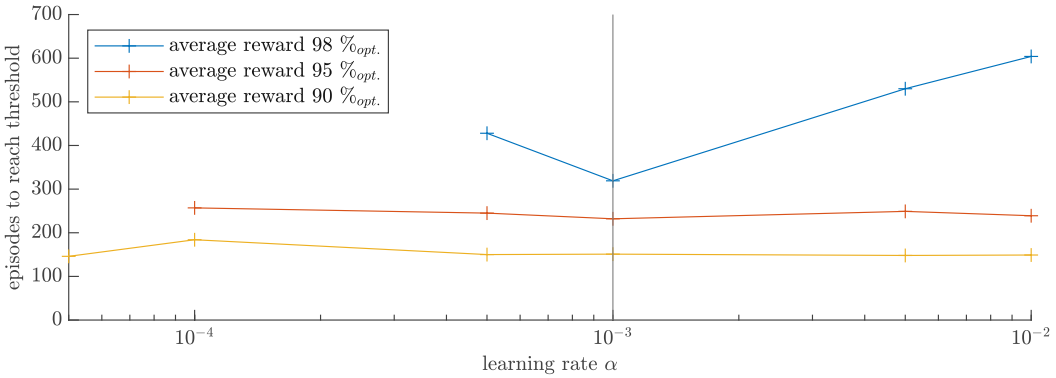


Figure 22.: *Hyperparameter learning rate analysis for DQN agent and Q-value critic 1 with episodes to reach threshold over learning rate.*

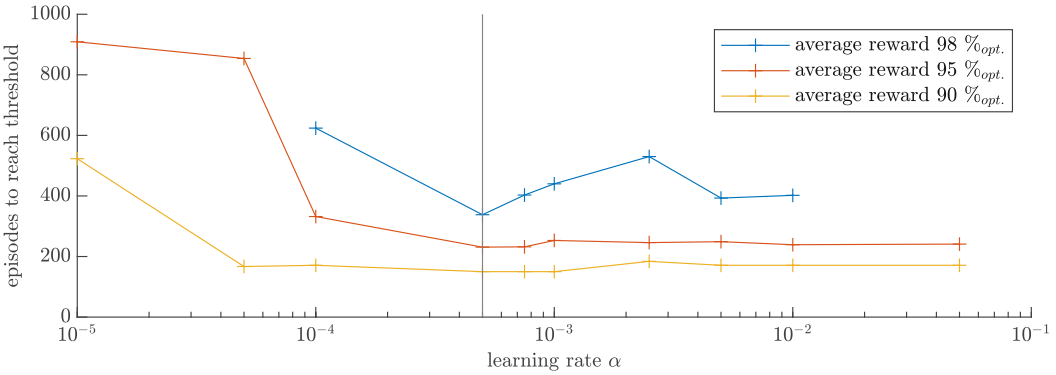


Figure 23.: *Hyperparameter learning rate analysis for DQN agent and Q-value critic 2 with episodes to reach threshold over learning rate.*

4.4.2. DQN Exploration Rate

Figure 24 shows the HPA study for the *DQN* agent and Q-value critic 2 with episodes for reaching threshold values plotted over exploration rate ϵ. The specific limits are shown for 90 % (yellow), 95 % (red) and 98 % (blue) of the previously defined optimum. Table 14 defines the experiments used for this study.

Higher exploration means that a random action is chosen more often than the one where the highest Q-value is assumed. As far as the very simple case of the discrete action space is concerned, in which either air is blown into the runner housing or nothing is done, no statement can be made about the effect of exploration on the

maximum reward achieved or how quickly this is achieved. It is assumed that this must be an error of some kind, as the exploration does not influence the training behavior. No difference could be detected for different initialization of the neural network weights.

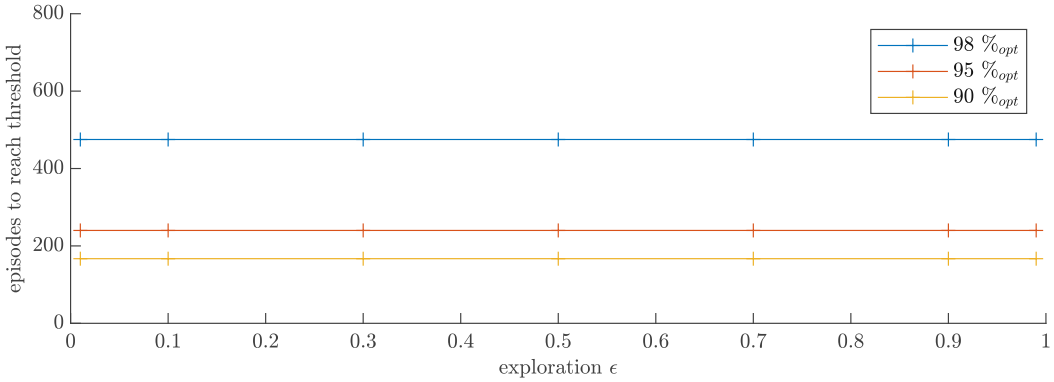


Figure 24.: *Hyperparameter exploration rate analysis for DQN agent and Q-value critic 2 with average reward over episodes.*

4.4.3. DQN Exploration Decay Rate

Figures 26 shows the HPA study for the *DQN* agent and Q-value critic 2 with episodes for reaching threshold values plotted over exploration decay rate ϵ_{decay}. The specific limits are given as 90 % (yellow), 95 % (red) and 98 % (blue) of the previously defined optimum. Table 16 defines the experiments used for this study.

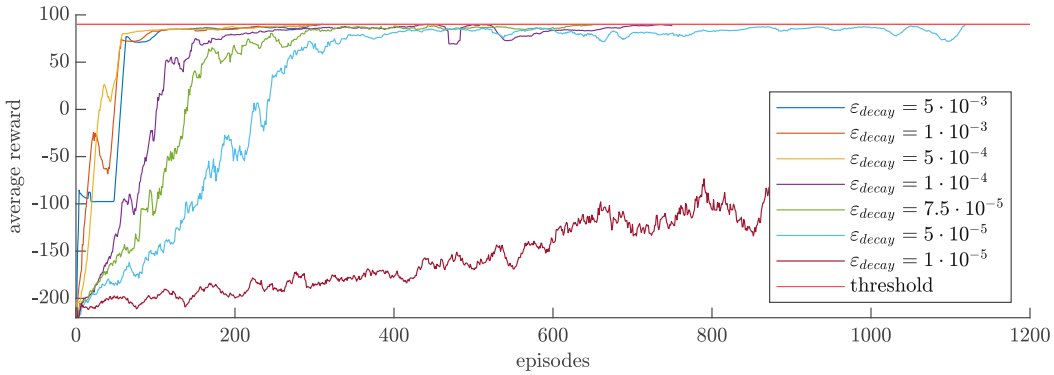


Figure 25.: *Training curves of DQN agent for different exploration decay rate of Q-value critic 2.*

It can be seen that with lower exploration decay rates, more training episodes are needed to reach the threshold. Higher rates result in an almost constant value. The training curves of the experiments can be found in the appendix in Figure 25. It

is clear that at the beginning of the training with a low decay rate, the training curve is very flat and steeper as the decay rate increases. This is the case up to $\epsilon_{\text{decay}} = 10^{-3}$, where the average value drops in the training only to rise sharply again later.

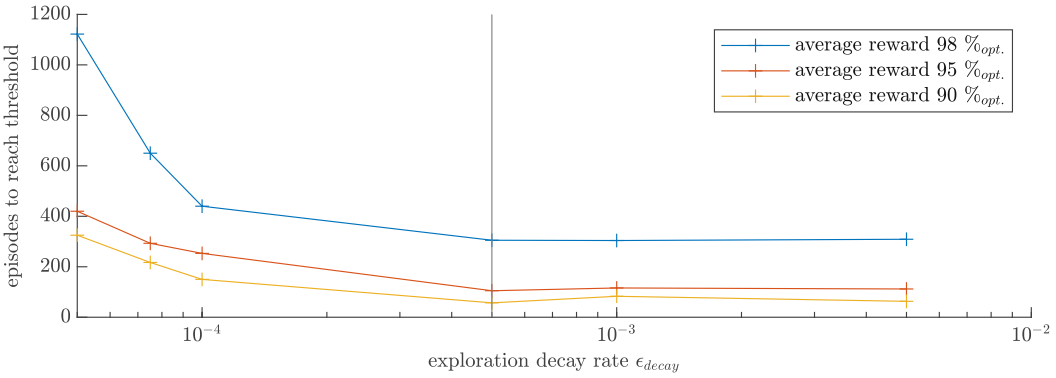


Figure 26.: *Hyperparameter exploration decay rate analysis for DQN agent and Q-value critic 2 with episodes to reach threshold over exploration decay rate.*

4.4.4. PPO Learning Rate

Figure 27 shows the HPA study for the *PPO* agent and state-value critic and actor with average reward values plotted over the learning rates of actor α_{actor} and critic α_{critic}. The yellower the color bar, the higher the average reward, the bluer the lower. The training progressions were simulated for a maximum of 2000 episodes and the highest average reward achieved was used. The line where the learning rates are equal is also shown in red. All the parameter sets can be found in Table 13.

There are two areas (shown here in yellow) where very high values close to the optimum can be achieved. On the one hand, this is with equal learning rates of $\alpha = 10^{-2}$ or with an actor learning rate of $\alpha_{\text{actor}} = 10^{-3}$ and critic learning rate of $\alpha_{\text{critic}} = 10^{-4}$. It can generally be seen that higher average rewards can be achieved in the range where the learning rate of the critic is smaller than that of the actor. Therefore, the actor's neural network should learn and execute faster to achieve stable training progress and a higher reward. The actor's prolonged learning rate of 10^{-5} does not result in any meaningful learning behavior, which is why it was not included in the plot.

4.4.5. PPO Entropy Loss Weight

Figure 28 shows the HPA study for the *PPO* agent, state-value critic and actor with episodes to reach threshold over entropy loss weight w. The maximum average values achieved for different entropy loss weights can be taken from Figure 42 in the

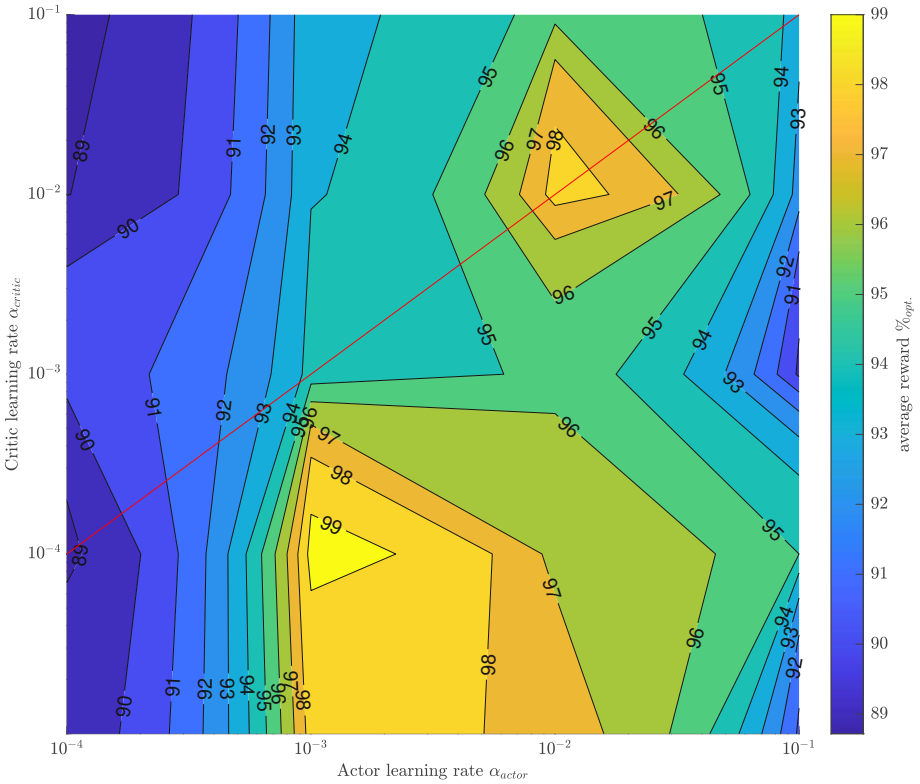


Figure 27.: *Hyperparameter learning rate analysis for PPO agent and state-value critic and actor with average reward over learning rates.*

appendix. The experiments are defined in Table 15.

The additional term in the actor loss function of the PPO agent (Algorithm 2), the entropy loss term $w \cdot H(\theta, S)$, causes the agent to favor exploration, as for higher values, it is more uncertain which action to take. This can help to get out of local optima. If there is no need to ensure excessive exploration due to a very simple use case, which is present, the optimum can also be achieved without an additional term, as shown in Figure 28.

Convergence towards the optimum is only possible with low entropy loss weight values. High values favor more exploration and are, therefore, more of a handicap at the end of the training process, which can be recognized by the fact that from $w > 0.2$, the progressions no longer reach 99 % of the optimum.

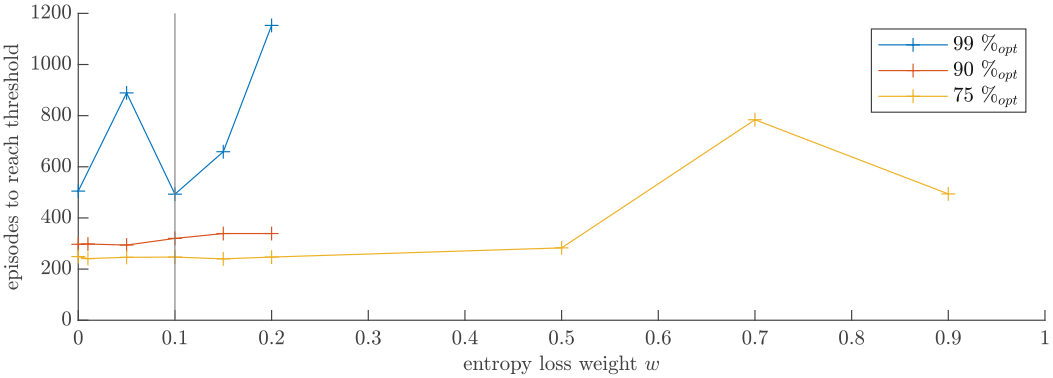


Figure 28.: *Hyperparameter entropy loss weight analysis for PPO agent and state-value critic and actor with episodes to reach threshold over entropy loss weight.*

4.5. Comparison of Algorithms

The value-based and actor-critic-based algorithms are tested for the initial assumption that *DQN* and *PPO* are the best-performing agents in their group.

4.5.1. Comparison of Value-based Algorithms

Figures 29 and 30 show the comparison of the training curves of *Q* (blue), *SARSA* (red) and *DQN* (yellow) agents for Q-value critic 2 with 256 and 32 neurons per layer, respectively. The average reward is plotted over the number of episodes. Reaching the threshold value is indicated by a vertical black line. The parameter sets are in Table 17.

The *DQN* agent is superior for both the broad and the shallow neural network, as it reaches the optimum much faster. Both the *Q* and *SARSA* agents do not reach the threshold value (black dashed). This comparison is consistent with the original assumption that the DQN agent is the most suitable for value-based algorithms.

4.5.2. Comparison of Actor-critic-based Algorithms

Figures 31 and 32 show the comparison of the training curves of *AC* (blue), *TRPO* (red) and *PPO* (yellow) agents for state-value critic and actor with 256 and 32 neurons per layer, respectively. The average reward is plotted over the number of episodes. A vertical black line indicates reaching the threshold value (black dashed). The parameter sets are in Table 18.

The *TRPO* agent trains very quickly and reaches a high value after a few training episodes but then has problems reaching the maximum average reward. This is because it has extensive updates of the actor and critic. In contrast, the *PPO*

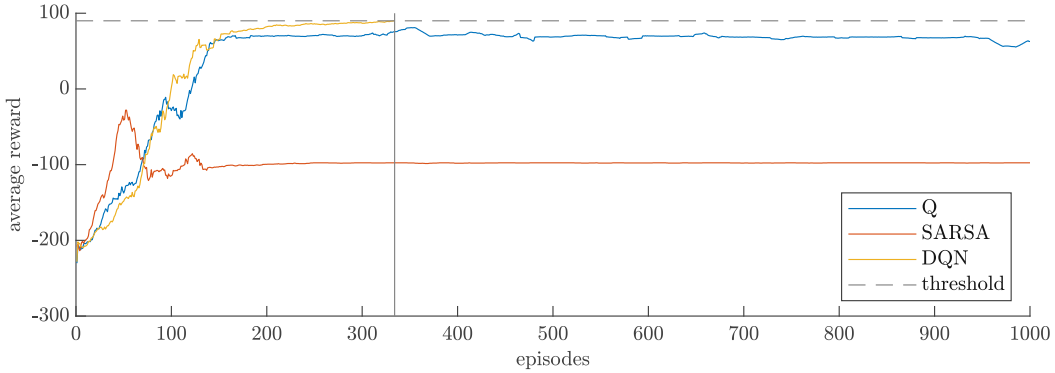


Figure 29.: *Comparison of the training curves of Q, SARSA and DQN agents for Q-value critic 2 with 256 neurons/layer and average reward over episodes.*

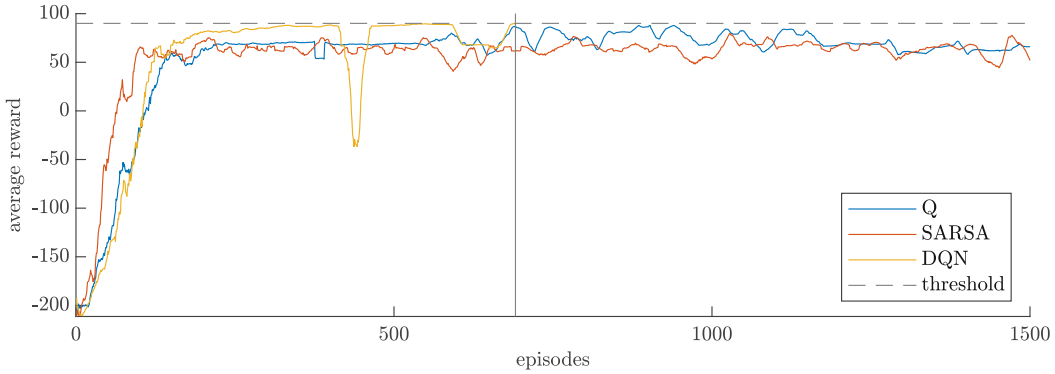


Figure 30.: *Comparison of the training curves of Q, SARSA and DQN agents for Q-value critic 1 with 32 neurons/layer and average reward over episodes.*

is a simplification and has a clipped update, which is characterized by a slower progression at the beginning but convergence to the optimum. The *AC* algorithm does not reach the target value. This comparison supports the original thesis that the *PPO* agent from the actor-critic-based algorithms is the most suitable.

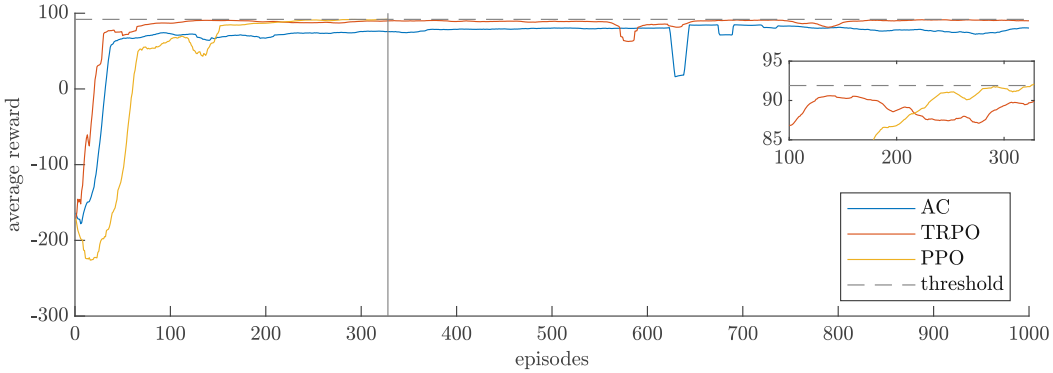


Figure 31.: *Comparison of the training curves of AC, TRPO and PPO agents for state-value critic and actor with 256 neurons/layer and average reward over episodes.*

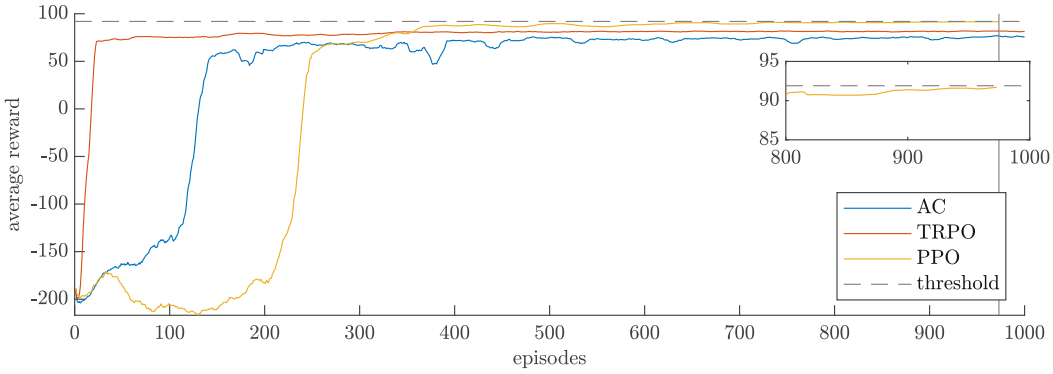


Figure 32.: *Comparison of the training curves of AC, TRPO and PPO agents for state-value critic and actor with 32 neurons/layer and average reward over episodes.*

4.6. Validation

Figures 33 and 34 show the validation plots for the *DQN* and Figures 35 and 36 for the *PPO* agents with varied water heights and guide vane openings, respectively. The control behavior of the water level in the turbine housing due to the compressed air is shown over time. The experimental data used can be found in Tables 19 and 20 for the *DQN* and *PPO* agents, respectively.

It can be seen that although no training was done for these initial conditions, the polices show very good generalization. The *DQN* agent maintains the water level between the boundaries for different initial heights of water in the turbine housing or degrees of opening of the guide vanes. For all three different water heights, the *PPO* policy regulates the water level very quickly to the same value (just below the blow-out height) and keeps it constant. The same statement can be made for

different guide vane openings. It is necessary to add here that it must be assumed that the simulation does not correctly represent the air leakage for larger openings. This is because a much larger opening requires just a minimum more compressed air supply.

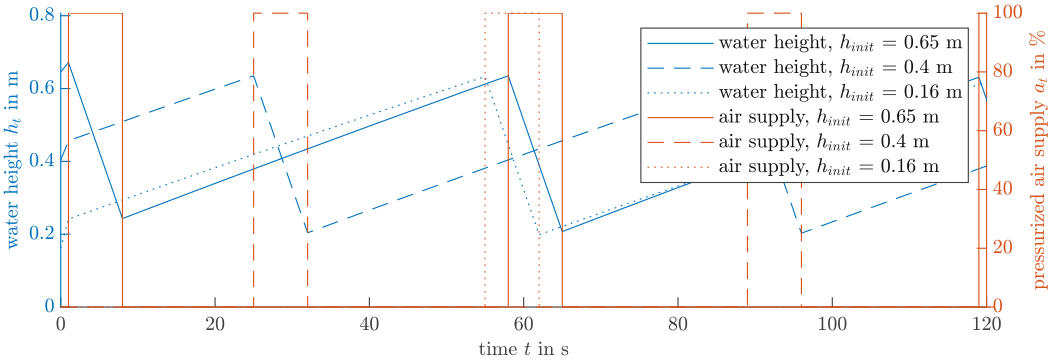


Figure 33.: *Validation DQN agent with varied water heights; control behavior of DQN policy for the water level corresponding to the compressed air blown into the runner over time.*

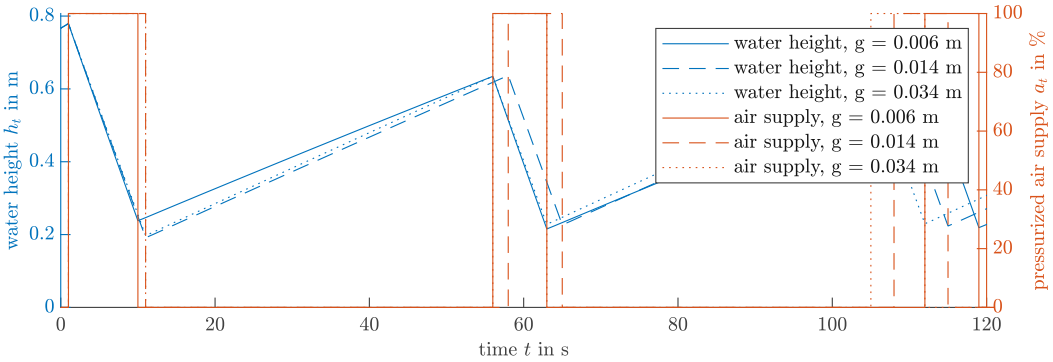


Figure 34.: *Validation DQN agent with varied guide vane openings; control behavior of DQN policy for the water level corresponding to the compressed air blown into the runner over time.*

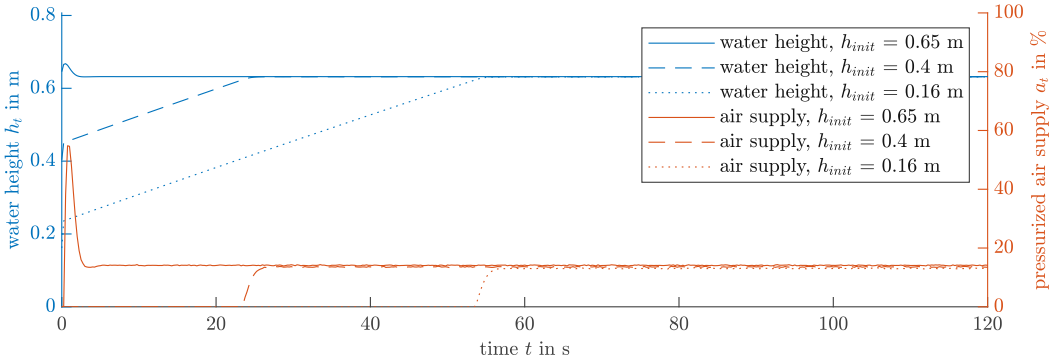


Figure 35.: *Validation PPO agent with varied water heights; control behavior of PPO policy for the water level corresponding to the compressed air blown into the runner over time.*

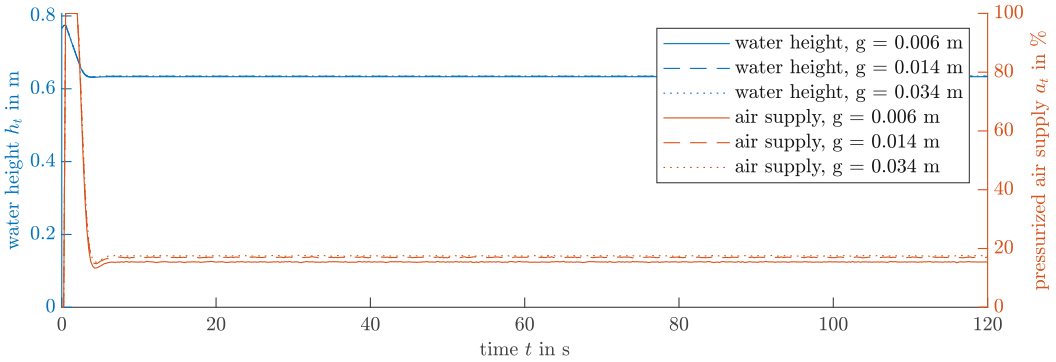


Figure 36.: *Validation PPO agent with varied guide vane openings; control behavior of PPO policy for the water level corresponding to the compressed air blown into the runner over time.*

4.7. Extended Use Case

In the extended use case, where the guide vane opening can also be controlled in addition to the water height of the turbine housing, the resulting control behavior is shown in Figure 37. The opening of the guide vanes (g_t in cm, blue) is plotted in addition to the water level and compressed air. Experiment C5.1, see Tab. 21, is an example of the *PPO* agent with the neural network NN3.1 for extended state and action spaces. In this experiment, the same parameters that were found to be most suitable for the continuous action space were used. These parameters would have to be adapted for the extended use case in order to achieve better control behavior.

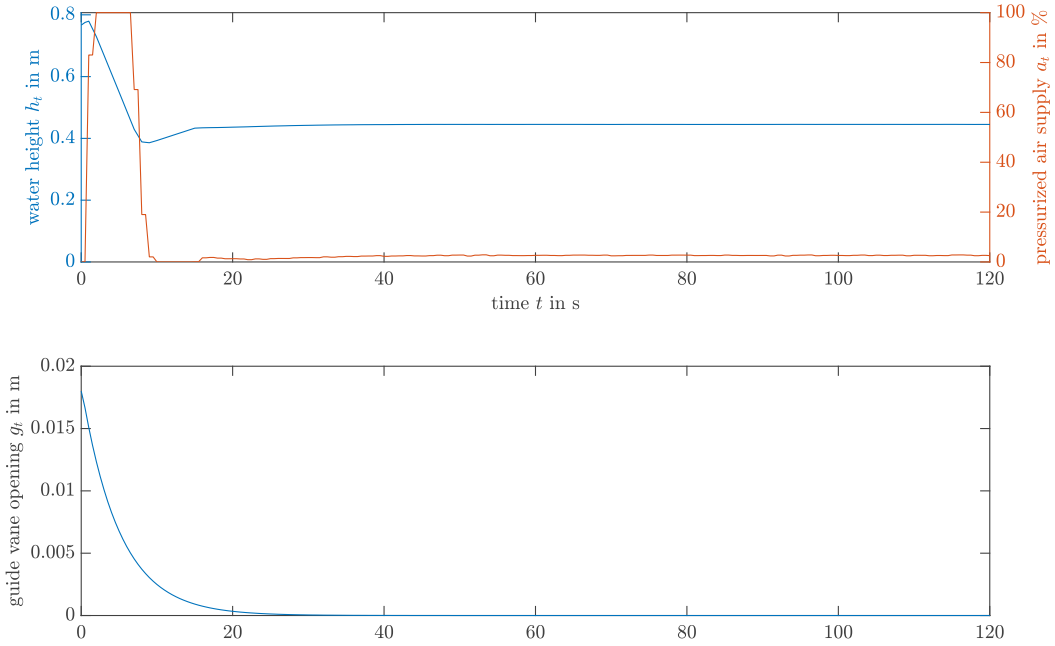


Figure 37.: *Control behavior of the water level in the turbine housing and guide vane opening due to the compressed air and guide vane actuator of the PPO RL agent.*

5. Conclusion and Outlook

This thesis investigated how Reinforcement Learning (RL) methods can be applied to a model of a small-scale pumped storage power plant located at the Institute of Energy Systems and Thermodynamics (IET) test facilities at TU Wien. For this purpose, a sub-process of the transition from turbine to pump operation in a reversible pump turbine, the blow-out process of water from the turbine housing by compressed air, was investigated. This process, which was easy to control using conventional methods (hysteresis and PID controller), determined the maximum return and optimum behavior. A model created in a previous thesis was embedded in an RL simulation environment. Suitable state and action spaces were defined to describe the problem and enable control. To have a proper solution metric, a reward function was established that includes the control objective. Neural network architectures were designed and analyzed in a study. A hyperparameter study was carried out to determine the parameters' dependencies on the training behavior of the agent and the subsequent control. The most suitable algorithms were found, their control behaviors were compared with conventional controllers, and their underlying control policies were analyzed.

Regarding the question of whether RL is suitable for controlling the simulation model in a meaningful way, it was shown that the maximum return according to the reward function and the optimum control behavior could be reproduced for both discrete (with a *DQN* agent) and continuous (with a *PPO* agent) actions. This means that the water height in the turbine housing to be regulated reaches and remains within a predefined height range as quickly as possible with the least amount of compressed air. This is equivalent to correctly approximating the action-value function for value-based and value function and policy for actor-critic-based algorithms. The control behavior was visualized for better clarity for both algorithms. The training process was fast and convergent.

With knowledge of the optimum, the question of which deep neural network is appropriate can be answered. The structures of neural networks were compared with each other in a study. Four different basic architectures were designed for the various function approximators. For a robust set of training parameters retained for the study, it was found that a minimum number of neurons per layer is required to represent the system entirely. With regard to the number of training episodes, a minimum occurs for broader networks, which was defined as the most suitable structure. For both the *DQN* and *PPO* agents, an optimal result for a neural network with only two neurons per layer (order of degrees of freedom $\sim 10^1$) can be

found with specially adapted training parameters.

When asked how the hyperparameters must be selected to achieve optimal training behavior, the following hyperparameters' dependencies on the training performance could be shown with the choices of neural networks. For the *DQN* agent, it can be seen that there is a learning rate at which a maximum reward is achieved or a defined limit is trained the fastest. No clear dependence of the training behavior on exploration could be shown. For the exploration decay rate, it was demonstrated that fewer training episodes are required for larger values, although a limit can be observed. For the *PPO* agent, there is a combination of learning rates at which a maximum reward is achieved or a defined limit is trained the fastest. Also, smaller entropy loss weights favor faster training and higher average rewards.

With the neural networks and hyperparameters found, it could be demonstrated that the *DQN* agent is superior to the *Q* and *SARSA* agent from the value-based algorithms. The same applies to the *PPO* agent compared to the actor-critic-based algorithms *AC* and *TRPO*. In a final step, the *DQN* and *PPO* agents were validated with varying initial water heights and guide vane openings. The system could be satisfactorily controlled for different initial states. Therefore, the question of the most suitable RL algorithm for the problem description can be answered with the *Deep-Q-Network* for discrete actions and the *Proximal Policy Optimization* algorithm for continuous actions.

In summary, it can be said that the RL concept is applicable to control challenges but must be specifically adapted to the problem at hand. Systematically optimized control of a model with RL requires a suitable state space selection that describes the problem as simply as possible but entirely and an action space that enables the desired control. The reward function must be defined to contain the control objective completely. Modern, simplified algorithms have proven to be suitable. The neural network's structure must be designed so that the complexity can be represented. A good compromise of neurons per layer should be determined in a coarse neural network study. In addition, the hyperparameters must be chosen appropriately to achieve convergent and fast training. This requires a coarse hyperparameter study in which combinations of parameters are found. Since it becomes difficult to define an optimal control behavior for more complex use cases, the algorithms' results must be compared with each other.

The use case was chosen so that it can be easily controlled with conventional controllers to obtain a defined optimum with regard to the reward function. Based on this upper bound, all experiments could be compared with each other. However, this meant that some of the neural network and hyperparameter study findings were not completely conclusive. For example, no dependence of the exploration rate on the training behavior could be shown. This may be due to a calculation error or a use case that is too simple. For this reason, it is difficult to determine whether the

results can be generalized to higher-dimensional state and action spaces.

It must be added that the solver often had difficulties solving the given states in the model. The model must be better conditioned so that fewer algebraic loops, which are difficult to resolve, occur.

Future research questions could deal with extending the discussed process. In the first step, the guide vane opening was also included as a controllable variable and successfully trained in addition to the water height. For this purpose, an arbitrary time dependency of the guide vane closing was incorporated. The model of the pump-turbine must be modified in such a way that the dynamics of the guide vane apparatus are physically correctly represented. In addition, the speed of the rotor must be included in the system as an adjustable variable. With these modifications, the entire transition from turbine to pump operation could be analyzed using RL methods.

Bibliography

[1] Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie (BMK). "Energie in Österreich 2021: Zahlen, Daten, Fakten". In: (2022), p. 64.

[2] Deutsche Energie-Agentur GmbH. "Analyse der Notwendigkeit des Ausbaus von Pumpspeicherwerken und anderen Stromspeichern zur Integration der erneuerbaren Energien". In: *Integr. Vlsi J.* (2010), p. 176.

[3] Fraunhofer Institute for Solar Energy Systems. "Windenergie Report Deutschland 2009". In: (2011), p. 87.

[4] A. Maly and C. Bauer. "Experimental investigation of a free surface oscillation in a model pump-turbine". In: *IOP Conference Series: Earth and Environmental Science* 774 (June 2021), p. 012068. DOI: 10.1088/1755-1315/774/1/012068.

[5] C. Tubeuf, F. Birkelbach, A. Maly, M. Krause, and R. Hofmann. "Enabling Reinforcement Learning for Flexible Energy Systems Through Transfer Learning on a Digital Twin Platform". In: Jan. 2023, pp. 3218–3228. DOI: 10.52202/069564-0289.

[6] R. S. Sutton and A. Barto. *Reinforcement learning : an introduction*. eng. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2018. ISBN: 0262039249.

[7] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko. "Reinforcement learning for control: Performance, stability, and deep approximators". In: *Annu. Rev. Control* 46 (2018), pp. 8–28. ISSN: 1367-5788.

[8] The MathWorks Inc. *MATLAB version: 9.13.0 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com (visited on 01/11/2024).

[9] The MathWorks Inc. *Simulink version: 10.6 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com (visited on 01/11/2024).

[10] The MathWorks Inc. *Deep Learning Toolbox version: 14.5 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com/help/deeplearning/ (visited on 01/11/2024).

[11] The MathWorks Inc. *Reinforcement Learning Toolbox version: 2.3 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com/help/reinforcement-learning/ (visited on 01/11/2024).

[12] R. d. R. Faria, B. D. O. Capron, A. R. Secchi, and M. B. de Souza. "Where Reinforcement Learning Meets Process Control: Review and Guidelines". In: *Processes* 10.11 (2022), p. 2311. ISSN: 2227-9717.

[13] D. Silver. *Lectures on Reinforcement Learning*. URL: `https://www.davidsilver.uk/teaching/`. 2015. (Visited on 01/11/2024).

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. "Human-level control through deep reinforcement learning". eng. In: *Nature (London)* 518.7540 (2015), p. 529. ISSN: 1476-4687.

[15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: `1502.05477`.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`.

[17] B. Krose and P. van der Smagt. "An introduction to neural networks". In: *J Comput Sci* 48 (Jan. 1993).

[18] S. Haykin. *Neural networks : a comprehensive foundation*. eng. Upper Saddle River, NJ [u.a.]: Pentice Hall, 1994. ISBN: 0023527617.

[19] M. W. Krause. *Modellierung einer Pumpturbine für flexiblen Betrieb*. 2023. URL: `https://doi.org/10.34726/hss.2023.96365`.

[20] The MathWorks Inc. *Simscape version: 5.4 (R2022b)*. Natick, Massachusetts, United States, 2022. URL: `https://www.mathworks.com/help/simscape/` (visited on 01/11/2024).

A. Appendix

The following figures give a more detailed view of the training progressions.

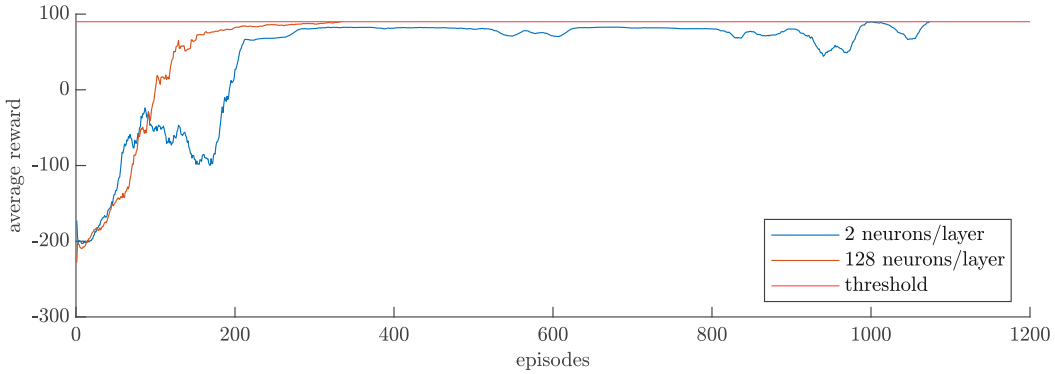


Figure 38.: *Training curves of DQN agent for different neurons per layer of Q-value critic 1 neural network.*

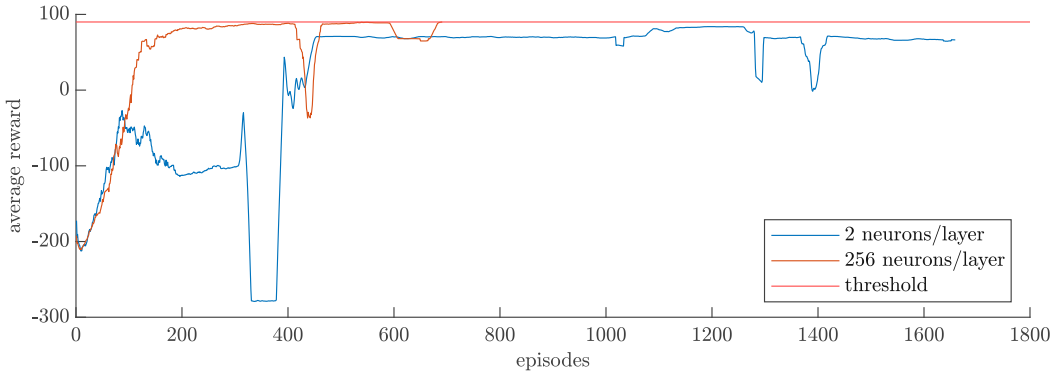


Figure 39.: *Training curves of DQN agent for different neurons per layer of Q-value critic 2 neural network.*

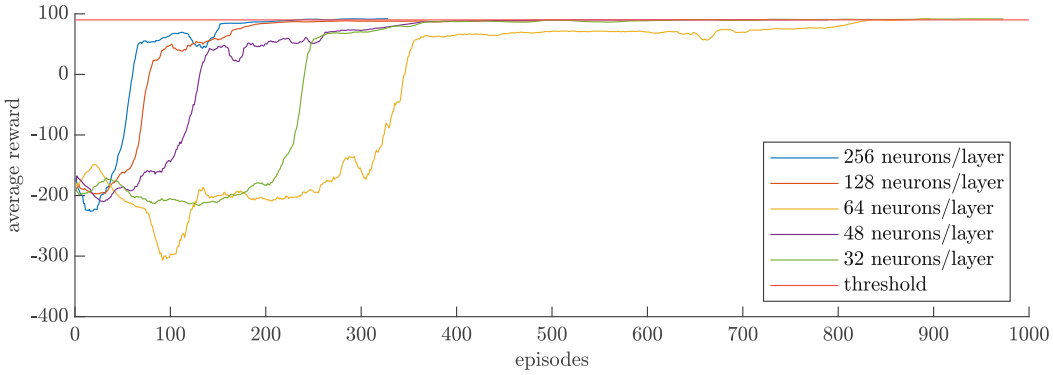


Figure 40.: *Training curves of PPO agent for different neurons per layer of state-value critic and actor.*

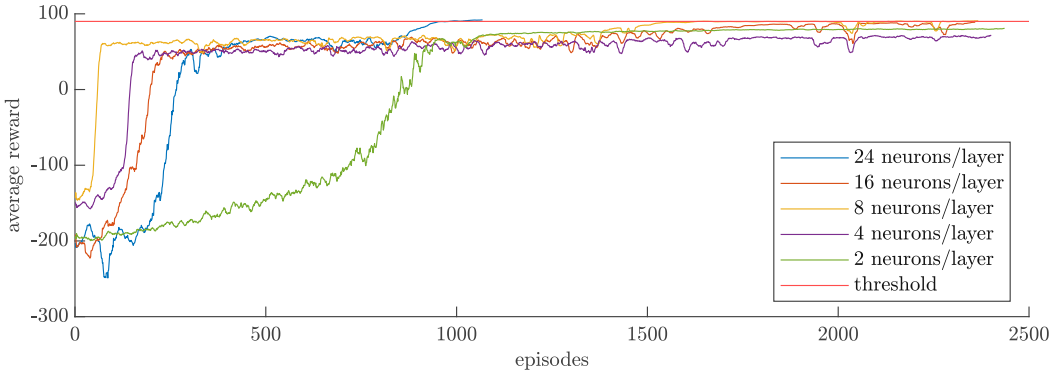


Figure 41.: *Training curves of PPO agent for different neurons per layer of state-value critic and actor.*

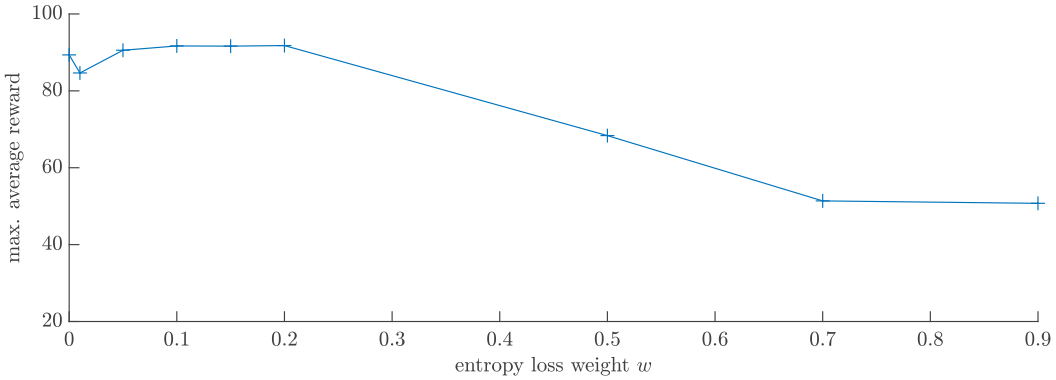


Figure 42.: *Hyper parameter entropy loss weight analysis for PPO agent and state-value critic and actor with maximum average reward over entropy loss weight.*