Carl von Ossietzky Universität Oldenburg
Faculty II – Department of Informatics
Digitalized Energy Systems

Comparison of Reinforcement Learning Algorithms for the Optimal Power Flow Problem

Master Thesis

submitted by

Lorenz Mumm, B. Sc.

born on 04.06.1998 in Aurich

First Examiner:	Prof. Dr.–Ing. Astrid Nieße
Second Examiner:	Thomas Wolgast, M. Sc.
Supervisor:	Thomas Wolgast, M. Sc.

Oldenburg, January 4, 2024

Abstract:
The OPF problem is the most common and fundamental optimization problem regarding power systems. Due to the increased influence of stochastic uncertainties by renewable energy sources, electric vehicles, and heat pumps, new engineering demands are appearing. Using Machine Learning techniques to solve OPF problems, especially with RL, has become a large research field. However, the literature often uses RL algorithms, which are not state–of–the–art, such as DQN or DDPG. This thesis developed a three-phase procedure to compare different RL algorithms, like the DDPG, TD3, SAC, A2C and PPO, onto two different OPF problems with different complexity levels. In the sweep–over–phase, the influence of each RL algorithm HPs was determined to define search spaces. In the next phase, the tuning–phase, the HPs were tuned through population-based training. Afterwards, the tuned HPs were compared with the baseline HPs, which were defined during the sweep–over–phase, ensuring that consistently more performant RL algorithms are compared. Lastly, in the comparison–phase, the overall comparison between the different RL algorithms was performed. In the comparison, the PPO reached top–performances in a smaller action space, but it became unstable after reaching its maximum performance. The SAC reached top–performances in a more complex action spaces and even performed decently in the smaller ones. Therefore, using the SAC in more complex OPF problems and the PPO in simpler ones is advised.

Contents

Abbreviations

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor–Critic
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q–Network
ERM	Episode Reward Mean
GAE	Generalized Advantage Estimation
HP	Hyperparameter
HPT	Hyperparameter Tuning
IEEE	Institute of Electrical and Electronics Engineers
LR	Learning Rate
MLP	Multi Layer Perceptron
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
OPF	Optimal Power Flow
PBT	Population Based Training
PPO	Proximal Policy Optimization
SAC	Soft Actor Critic
SS	Search Space
SGD	Stochastic Gradient Descent
STD	Standard Deviation
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
VSM	Valid Solution Mean

Nomenclature

α	Entropy Temperature Parameter
γ	Discount Factor
ϵ	PPO Clip Parameter
ϵ	Epsilon-Greedy Value
λ	Generalized Advantage Estimation Parameter
τ	Soft Target Network Update

1 Introduction

The Optimal Power Flow (OPF) problem is the most common and fundamental optimisation problem regarding power systems. Due to the increased influence of stochastic uncertainties by renewable energy sources, electric vehicles and heat pumps, new engineering demands are appearing. Furthermore, the complexity is rising, and the time intervals for solving the OPF problem are decreasing. [1, 2]
Conventional methods often try to reduce the problems to solve them with decent procedures iteratively. Another promising approach is Machine Learning, which has become very popular since it achieved good results on the OPF problem. Additionally, Machine Learning algorithms are significantly faster than conventional methods, which addresses the demand for fast solutions. [3]
Especially Reinforcement Learning (RL) has shown impressive results on different tasks, so it has become a large research field regarding the OPF problem [1]. Much research has been done with many different RL algorithms under vast majorities of conditions. This identifies a research gap because papers often use algorithms or derivatives of them without usually making a sufficient empirical and reproducible comparison about the algorithm choice. Furthermore, Hyperparameters (HPs) were usually chosen, with some adjustments to the OPF problem, from base algorithms [2, 3], or they were derived through unsystematic experiments [4]. In some cases, papers cut the algorithm and HP discussion totally and use a predefined RL agent [5].

1.1 Objective

As shown in the chapter 1, authors often use standard algorithms, like Deep Q–Network (DQN), Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO), because some of them achieved good results on OPF problems. Further, some algorithms are more suited to deal with large action and state spaces, comparable to those in OPF environments. Often, the RL algorithm and HP selection were outside the scope of papers.
This simple decision–making process will be addressed by comparing different tuned RL algorithms on the OPF problem. To achieve this objective, at first, standard RL algorithms, like DDPG, PPO or Soft Actor Critic (SAC) will be tuned onto two different OPF envi-

ronments. After determining the optimal HPs, all agents are compared with each other according to specific predefined evaluation criteria. Arising from this approach, the following research question is formulated:

How to develop a procedure to select, tune and compare RL algorithms to determine optimal algorithms for OPF problems?

With the provided answer for the research question, scientists get a recommendation for choosing and optimising RL algorithms for their future work.

1.2 Structure

The master thesis is structured as follows:

chapter 2 The background provides additional information about the Optimal Power Flow (OPF) problem and Reinforcement Learning (RL). The different RL algorithms with their HPs get presented as well as Hyperparameter Tuning (HPT) algorithms.

chapter 3 The related work chapter analyses literature about solving OPF problems with RL algorithms using specific criteria. The main research questions concern the algorithms and HP choice.

chapter 4 The requirements define boundary conditions, procedure settings and evaluation criteria for the tuning and comparing process of this master thesis.

chapter 5 In the fifth chapter, the approach with its sweep–over–, tuning– and comparison–phases is presented, as well as an environment explanation that informs about the state and action spaces and used power grids.

chapter 6 In the implementation chapter, the different framework choices will be reasoned. Further, an impression of how many resources were required to fulfil the comparison with its three phases will be provided.

chapter 7 The sweep–over–phase evaluation chapter will discuss the analysis results of the sweep–over runs, in which the influence of each RL algorithm HP are determined to define HP Search Spaces (SSs) for the following HPT.

chapter 8 The tuned HPs are presented in the tuning–phase evaluation chapter. Further, these are validated with the baseline HPs chosen in the sweep–over–phase to prove if they are more performant than the baseline ones.

chapter 9 The comparison–phase evaluation chapter presents the last and overall comparison results. Afterwards, application recommendations are provided to ensure the correct use of each RL algorithm.

chapter 10 In the procedure evaluation and discussion chapter, the requirements analysis takes place, verifying that the fundamental scientific constraints were not violated. Further, a discussion about the potential procedure optimisations was performed.

chapter 11 In the last section, a discussion about the initial thesis' objective will determine how appropriately the research question has been answered. Additionally, further open questions are presented in the outlook, opening new research possibilities and questions.

2 Background

2.1 Optimal Power Flow Problem

The Optimal Power Flow (OPF) problems are one of the most fundamental optimisation challenges regarding electrical power systems. The OPF problem defines state parameters and control variables to manage a power grid. State parameters deliver information about the current state of the power grid components, like bus voltage magnitudes and phase angles and generator reactive power outputs. Control variables manage the power grid to achieve optimal cost–efficiency and reliability through optimising, for example, the power or voltage output of generators, the load to shed and the phase shifting of transformers. What makes the OPF problem so difficult to solve are constraints, which must be fulfilled to ensure the stability of the power grid. Those constraints are split up into equality and inequality constraints. One equality constraint, for example, is the power balance. The total amount of injected power into the power grid has to be the total power consumed. An inequality constraint are the generator capacity limits. For safety reasons, power generators have operational limits they cannot exceed. [6]

2.2 Reinforcement Learning

Reinforcement Learning (RL) is one of the three parts of Machine Learning, next to Supervised and Unsupervised Learning. Through analysing data, it finds strategies, based on a trial and error approach, to solve problems described by the data.

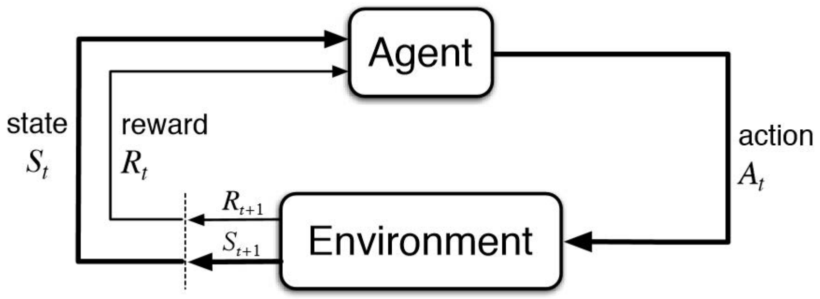


Figure 2.1: The Reinforcement Learning agent–environment interaction [7]

The process and components of RL will be explained in more detail through Figure 2.1. The environment is one of the two components of RL. It contains the problem that should be solved, like a game or an optimisation task and handles interactions with the agent, the second principal component. At first, the environment state will be sent to the agent delivering the problem situation to determine actions. Next, the computed action, which could be either discrete or continuous (see 2.2), gets sent back to the environment, where it gets executed. Then, the environment returns a reward signal indicating whether the action was good or bad. The new state will also be propagated back to keep up the RL–cycle shown in Figure 2.1. Often, a terminal signal will also be delivered to indicate either the environment is in a terminal state and will be reset or the state is not terminal and more actions can be executed. The target of RL agents is to maximise their collected reward. [7]

State The state is the central information transmission construct in RL. It contains information about the environment situation. The state, often called observation, means the same thing: a machine–readable representation of the environment. With this, an agent can determine an action. [8]

Action–Types There are two types of actions: discrete and continuous. Discrete actions are mutually exclusive, like going left or right. Also, the amount of actions in a discrete action space is finite.
Differently, the continuous action space is characterised by a non–finite amount of choosable actions, like the power output of a generator in percent. Usually, those action spaces have lower and upper boundaries, but each possible value can be chosen within these. [8]

2.3 Classification of Reinforcement Learning Algorithms

Many RL algorithms have been invented over the last few years, like the DQN in 2013 [9], DDPG in 2016 [10] or PPO in 2017 [11]. All of these RL algorithms are characterised by how they determine actions, how the data needs to be collected and which environment information they have access to. [8]

2.3.1 Policy–Based and Value–Based Algorithms

One type of classification informs about how the action is determined. Policy–based and value–based are two different variants of RL algorithms.
Policy–based algorithms use a learned policy for action determination, which maps states to probability distributions over all selectable actions in the corresponding state. Therefore,

they determine their actions directly from this policy, which gets trained through increasing or decreasing the probabilities of executed actions directly.
On the contrary, value–based algorithms do not determine their actions directly through a policy. Instead, they choose the action leading to the next state with the highest value. Therefore, only a value network is required. Compared to the policy–based approaches, these procedures are more indirect. [8]

2.3.2 Off–Policy and On–Policy Algorithms

The sampling behaviour is another way to classify RL algorithms. There are two possible types of sampling, called off–policy and on–policy.
Off–Policy RL algorithms can learn from data not sampled under the agent's current policy. Hence, an off–policy agent can learn from data collected by humans or older policies.
On–policy RL algorithms must learn from data sampled under the current policy of the agent because those algorithms try to optimise the policy directly based on the data. [7]

2.3.3 Model–Free and Model–Based Algorithms

The RL algorithms design, regarding the access of environment information, is another possibility to classify RL agents.
Model–free RL algorithms have no model of the environment to plan their next actions. They only have access to the information that environments propagate through the states. With no model, model–free agents are not able to plan the following steps to choose the most promising path in the environment.
In contrast, model–based RL algorithms have an underlying model of the environment. Hence, based on some evaluation criteria, the agent can plan the next most promising steps through the environment. [8]

2.4 Reinforcement Learning Algorithms

In this section, all algorithms to be examined will be introduced. The algorithm structure, functionality and HPs will be explained in more detail.

2.4.1 Deep Deterministic Policy Gradient

The DDPG is one of the numerous derivatives of the DQN algorithm [9]. The main reason for the DDPG development was the incompatibility of the DQN algorithm with continuous action spaces. The DQN cannot correctly determine continuous actions because it relies on finding the action that maximises the action–value function. Workarounds like a one–step optimisation or a simple discretisation lead to not satisfying results and suffer under large

action spaces.
Differently, the DDPG can handle those large action spaces without using any workarounds. Therefore, the algorithm uses an actor critic design. The actor maps states to actions, whereas the critic uses states and actions to determine the Q–values, which provide information about how good or bad actions were. Additionally, target networks, which are copies of the actual actor and critic network, are used to stabilise the learning by determining the critic or value loss for the actual critic network. The stabilisation effect results from the soft update behaviour through the target networks, which gets controlled by the Soft Target Network Update (τ) (see Table 2.1). The DDPG then uses those target networks to compute the loss for the critic network, which results in a slower and more stable gradient update.
Regarding the classification, the DDPG is a model–free algorithm because it has no underlying model to plan the next steps. Furthermore, the DDPG is a policy–based algorithm like the A2C and PPO. It derives its action from a policy network directly. Lastly, this algorithm can learn from data not collected under the current policy, so DDPG can learn off–policy. The Table 2.1 explains the DDPG HPs. [10]

Hyperparameters	**Description**
Actor learning rate	The actor Learning Rate (LR) controls the actor or policy network gradient update. Usually, it is between 1e^{-4} to 1e^{-3}.
Critic learning rate	The critic LR controls the gradient update of the critic or value networks. Usually, it is between 5e^{-4} to 1e^{-3}.
Batch size	The number of samples for a gradient step. Usually, it is between 32 – 4096.
Soft Target Network Update (τ)	Tau τ determines the rate at which the target networks are updated towards the main networks. It is larger than zero but usually very small, between 0.001 to 0.005.
Replay buffer size	Number of collected samples from the environment, which can be used for gradient update.
Exploration strategy	DDPG utilizes exploration noise to encourage exploration and prevent the algorithm from getting stuck in local optima. Noise is added to the actions determined by the actor. The amount of exploration noise applied can be controlled by adjusting its magnitude. Methods of generating noise are the Ornstein–Uhlenbeck noise and Gaussian exploration noise.
Discount Factor (γ)	The discount factor is used to control the time preference. It is between 0 – 1. High values result in far–sighted behaviours, whereas small values result in myopic behaviours.

Policy delay	Twin Delayed Deep Deterministic Policy Gradient (TD3) has a policy delay HP that controls how often the policy networks are updated. Usually it is between one to five. [12]

Table 2.1: DDPG and TD3 hyperparameter explanations

2.4.2 Twin Delayed Deep Deterministic Policy Gradient

The 2018 invented TD3 is one of the more recently developed RL algorithms. It should address weaknesses of the DDPG 2.4.1, like overestimated Q–values and policy overfitting. Therefore, the authors added a second critic network to reduce overestimation biases of the Q–values. They use a clipped double Q–learning technique that chooses both critics' minimum values, reducing overestimation errors. Another new concept of the TD3 algorithm is the target policy smoothing, which reduces the target policy variance and regularizes the learning process. Lastly, the TD3 utilizes delayed updates for the actor networks where the actor network gets updated less frequently than the critic, reducing the correlation between the actor's actions and the critic's evaluations. This results in a more stable and converging learning process. Furthermore, the TD3 algorithm is equal to the already introduced DDPG 2.4.1 because TD3 is a derivative.
Therefore, the TD3 HPs are also explained in Table 2.1. [12]

2.4.3 Soft Actor Critic

Soft Actor Critic (SAC) is a state–of–the–art RL algorithm specifically designed to tackle the complexities of continuous action spaces. SAC has many similarities with the DDPG and TD3 algorithms. In environments without discrete actions, SAC shines with its unique attributes. At its core, SAC introduces the concept of a soft policy. This means that SAC also incorporates an entropy term into its objective function. This entropy term encourages exploration and introduces a stochastic level into the agent's actions, making learning more effective in uncertain and complex environments. Therefore, a Entropy Temperature Parameter (α) is introduced and learned through training. This dynamic parameter allows the agent to adapt its level of exploration and exploitation, providing flexibility in different situations.
Another key component of SAC are the critic networks to estimate the values of states. Importantly, SAC employs two separate value functions through critic networks, from which the minimum value is used for doing gradient steps to address overestimation bias, resulting in more accurate value estimates and greater training stability.
To further increase the training stability, SAC implements target networks, which help to reduce the variance in the value function estimation, leading to an even smoother and more reliable training.

The SAC is a policy–based algorithm because it determines the action directly through a policy network. Further, it learns off–policy from experiences collected under different policies. Finally, SAC uses no underlying model to determine actions. Hence, it is a model–free algorithm. The SAC HPs get explained in Table 2.2. [13]

Hyperparameters	**Description**
Actor learning rate	See 2.1. Usually, the actor and critic LR are the same as well as the entropy LR. Often, they are between $1e^{-4}$ and $1e^{-3}$.
Critic learning rate	See 2.1 and actor LR.
Entropy learning rate	The entropy coefficient will be learned during training. The entropy LR controls the α update speed.
Batch size	See 2.1.
Replay buffer size	See 2.1.
Soft Target Network Update (τ)	See 2.1.
Initial Entropy coefficient / Initial alpha α	The entropy coefficient α is introduced to control the trade–off between exploration and exploitation in the learning process. It adjusts the weight of the entropy term in the overall objective function. The in 2.4.3 mentioned temperature parameter controls the transition from more exploration–based to a more exploitation–based behaviour.
Exploration strategy	Stochastic sampling is based on a normal distribution. Through the network outputs, mean and standard deviation values are generated and then used to create a normal distribution from which the actions get sampled.
Discount Factor (γ)	See 2.1.

Table 2.2: SAC hyperparameter explanation

2.4.4 Advantage Actor Critic

The Advantage Actor Critic (A2C) algorithm is a powerful RL technique that has gained popularity for efficiently training agents in various environments. Its actor critic architecture is at the heart of the A2C algorithm. The actor network is responsible for making decisions. It determines probability distribution from which the actions get sampled. The critic network evaluates the quality of the chosen actions. It estimates the expected discounted cumulative rewards for actions taken in a given state. This value function guides the agent in understanding, which states are desirable. A2C balances exploration and exploitation by combining elements of policy–based and value–based methods. One key innovation that sets it apart is introducing an advantage function. This function quantifies how much better or worse an action is compared to the average action in a given state. It plays a crucial role in reducing the variance of policy gradients, making training more stable and efficient. A2C further benefits from its ability to leverage asynchronous updates (Asynchronous Advantage Actor–Critic (A3C)).

Regarding the classification, the A2C is a model–free algorithm because it has no underlying model to plan the next steps. Further, the A2C is a policy–based algorithm because it directly derives its action from a policy network. Lastly, this algorithm learns on–policy so that after a batch of data was collected and used for training, the A2C needs to collect a new batch of data under the most recent policy. [14]
The A2C HPs get explained in Table 2.3.

Hyperparameters	**Description**
Learning rate	The LR controls the gradient update. Usually, it is between $1e^{-5} - 1e^{-3}$.
Batch size	See 2.1.
Value function loss coefficient	Controls the ratio between value and policy loss in the total A2C loss. Usually, it is between 0.5 – 1.0.
Entropy coefficient	Coefficient of the entropy loss term. Usually between 0.0 – 0.01. High values will reduce the entropy of all determined probability distributions, which leads to less exploration and a more stable training behaviour. Low values lead to more exploration. Furthermore, the terminal states are visited more often.
Exploration strategy	See 2.2.
Discount Factor (γ)	See 2.1.

Table 2.3: A2C hyperparameter explanation

2.4.5 Proximal Policy Optimisation

The PPO is one of the more recently invented RL algorithms. It was built based on the Trust Region Policy Optimization (TRPO) [15] that defines trust regions where the gradient steps are done. This procedure prevents the algorithm from taking large update steps, which ensures more stable learning. However, while the TRPO uses Kullback–Leibler divergence (KL divergence – metric for measuring the difference of probability distributions) to define trust regions, the PPO generally uses a clipping approach, in which losses get clipped. [11]
Like many other policy–driven RL algorithms, the PPO has an actor critic architecture. The actor determines actions by a neural network called the policy or actor network. The states will be propagated through that actor or policy network, resulting in probability distributions from which actions get determined. The critic evaluates these chosen actions

by determining the values through a value or critic network. Those values are the expected and discounted future rewards in the remaining episode. Based on these values, the advantages get calculated, providing information about how good or bad chosen actions were to update the actor and critic network. [11]
Regarding the classifications, the PPO algorithm has no model of the environment. Hence, it is a model–free algorithm. Further, it determines the action by a policy and not based on values, so it uses a policy–based approach. Lastly, it depends on data collected directly under a current policy, so the PPO learns on–policy.
The PPO HPs get explained in Table 2.4.

Hyperparameters	**Description**
Learning rate	See Table 2.3.
Batch size	See Table 2.1.
Minibatch size	The PPO collects a complete batch of experiences, which is then split up into multiple minibatches to train with.
SGD iterations	Number of Stochastic Gradient Descent (SGD) steps for a batch of data.
PPO Clip Parameter (ϵ)	Defines the trust region range. See 2.4.5. Usually, it is between 0.1 – 0.3.
Value function loss coefficient	See 2.3.
Entropy coefficient	See 2.3.
Exploration strategy	See 2.2.
Discount Factor (γ)	See 2.1.
Generalized Advantage Estimation Parameter (λ)	The Generalized Advantage Estimation (GAE) is a technique used to estimate the advantage function, which indicates the difference between the expected cumulative reward of taking an action and the expected cumulative reward of the current policy. GAE smooths the advantage estimates and addresses issues like high variance in reward estimation. [16]

Table 2.4: PPO hyperparameter explanation

2.4.6 Overview

The following Table 2.5 presents an overview of all used algorithms together with their classifications. All algorithms form two groups, which differ only in on– or off–policy sampling.

Algorithms	**policy– or value–based**	**on– or off–policy**	**model–based or model–free**
DDPG	policy–based	off–policy	model–free
TD3	policy–based	off–policy	model–free
SAC	policy–based	off–policy	model–free
A2C	policy–based	on–policy	model–free
PPO	policy–based	on–policy	model–free

Table 2.5: Overview of the RL algorithm classifications

2.5 Hyperparameter Tuning

Hyperparameter Tuning (HPT) is an important procedure for Machine Learning. Often, algorithms depend on multiple HPs, influencing performance, training behaviour and sample efficiency. Mathematically, HPT is an optimisation problem, in which the HPs must be set appropriately to fulfil an optimisation objective, like high performance.
Papers and scientific documentations are one way to get already proven HPs. Another way is to use algorithms and procedures, often without human influences, to determine optimal settings. [17]

2.5.1 Grid Search

Grid search is, until today, a common procedure to determine optimal HPs, especially when there are only a few selectable HPs. The user determines which HPs should be optimised and defines discrete values to try out. Each possible HP combination will then be tested and evaluated. [17]
Grid search is, therefore, one of the most straightforward approaches to optimise HPs. It is easy to run in parallel because the trials do not depend on each other. However, due to the significant number of HP combinations, which increases exponentially through adding new HPs [17] (curse of dimensionality), this HPT technique is computationally costly. It should only be used by experts familiar with the environment and RL algorithms to decrease the complexity. [18]

2.5.2 Random Search

Random search, like grid search 2.5.1, is a common way to optimise HPs. At first, the HPs, which should be optimised, will be chosen, as well as their corresponding SSs. Random values are then sampled and used as HPs for training. [17]
Random search has advantages over grid search, like continuous SSs that enable a more

extensive coverage of the HP value space. With more trials, it is sure that the probability increases to approximate optimal HPs. It also tends to be faster than grid search because there is a more extensive exploration due to the random value sampling [17]. However, random search is also computationally expensive. Therefore, it is recommended to use it, especially in the early stage of HPT. [18]

2.5.3 Population Based Training

The Population Based Training (PBT) is a Machine Learning and optimisation technique that combines elements of evolutionary algorithms and HPT to improve the training of neural networks and other Machine Learning models. It was introduced to address some of the challenges and inefficiencies in traditional HPT methods like grid search 2.5.1 and random search 2.5.1.

The PBT starts by creating a population of model instances. Each model instance represents a neural network or a Machine Learning model with a specific set of HPs (e.g. LR, batch size, architecture). These HPs are randomly or heuristically initialised through a HP SS.

Then, the entire population of models is trained in parallel, typically for a fixed number of iterations or epochs. After an initial training phase, the PBT algorithm evaluates the performance of each individual in the population. Models are then ranked based on their performance. The top–performing individuals can keep their HPs. In this phase, they continue training, possibly with a higher LR or for more epochs, to fine–tune their performance (mutation).

The worst–performing individuals replacing their models and HPs, often by selecting and copying the models and HPs from one of the top–performing individuals. It is also possible to resample new HPs from the initial HP SS. This introduces diversity into the population and potentially allows the discovery of better HP settings.

This training, evaluation, exploitation and exploration cycle will be repeated for multiple iterations to determine the top–performing models. One or more stopping criteria are often defined to end the tuning process.

Like grid search and random search, the PBT is very computationally expensive, but due to its scheduling design, it could evaluate more HPs during one trial. However, the PBT algorithms depend on meta–hyperparameters, which carefully need to be chosen. [19]

3 Related Work

In this chapter, a deeper look into the related work will be done, preparing the formulation of the further chapters. Therefore, the RL algorithms and HP choices, as well as the HPT processes will be examined in more detail regarding the OPF problem. Also the environment information, will be mentioned to obtain an impression about the OPF problem complexity.

3.1 A General Real–time OPF Algorithm Using DDPG with Multiple Simulation Platforms

Nie et al. [1] try to control the active power flow and the bus voltage magnitude with a DDPG agent on multiple platforms, like MATPOWER, a Matlab framework and CloudPSS, a cloud–based framework. The problem consists of three generators and loads, as well as nine buses in an Institute of Electrical and Electronics Engineers (IEEE) nine–bus system. The state, which will be propagated to the agent, is a vector of 21 entries containing the active and reactive power load, the active and reactive power output and the bus voltages. The reward function contains the network power loss and a constraint violation term. The main focus was to develop a general RL solution for solving the OPF problem and achieving comparable results to already available solutions provided by the frameworks.

The authors chose their algorithm by comparing the corresponding predecessor DQN. In an analysis, the authors identified some weaknesses, like the unsatisfying behaviours in large action spaces and the inability to choose continuous actions directly. For these reasons, the DDPG algorithm was chosen because it is capable of dealing with sizeable continuous action spaces.

In the paper, some HPs were discussed, like Discount Factor (γ), which was set to 0.01 because past experiences less influence future actions and states. For exploration reasons, an Epsilon–Greedy method was introduced with an Epsilon-Greedy Value (ϵ) of one at the beginning. During training, epsilon ϵ gets decreased to 0.5 (high values of epsilon ϵ result in more exploration, whereas low values do the opposite). Nie et al. propose that a RL of 1.0 worked well in their experiments because the problem does not depend on previous learning steps. Further HPs are listed in Table 3.1. According to some mentions, Nie et al.

did some experiments to determine single HPs, but an overall HPT was not performed.

Hyperparameters	**Values**
Learning rate	1.0
Batch size	64
Soft Target Network Update (τ)	0.0001
Exploration strategy	Epsilon–greedy
Epsilon-Greedy Value (ϵ)	1.0 decreasing to 0.5 while training
Discount Factor (γ)	0.01

Table 3.1: Hyperparameter selection of "A General Real–time OPF Algorithm Using DDPG with Multiple Simulation Platforms".

3.2 Deep Reinforcement Learning Based Approach for Dynamic Optimal Power Flow in Active Distribution Network

Liu et al. [4] try to solve an OPF problem that is mainly based on dynamic and decentralised power generation methods, like photovoltaic, wind power and energy storage systems. The authors used an improved version of the IEEE 33–bus system with two photovoltaic and energy storage systems for benchmarking.
The environment state contains the reactive and active power flow of the loads and the reactive power flow of the photovoltaic systems. To minimise power losses, the agent controls the active and reactive power outputs of photovoltaic and energy–storing systems. The reward function incorporates the active network loss and a penalty term for violating constraints.

Liu et al. had chosen their algorithm based on analysing the DQN. Problems, like performance issues in high–dimensional continuous state and action spaces, as well as overestimated values, led them to use the DDPG for problem–solving.

The paper provides no discussion about the HP selection. The authors only pointed out that the HPs were determined during various test runs. Table 3.2 shows the HP settings. An overall HPT was not part of this paper.

Hyperparameters	**Values**
Actor learning rate	0.001
Critic learning rate	0.002
Batch size	50

Replay buffer size	480
Soft Target Network Update (τ)	0.001
Exploration strategy	Gaussian noise
Variance of Gaussian noise	0.9
Steps of each episode	96
Discount Factor (γ)	0.8

Table 3.2: Hyperparameter selection of "Deep Reinforcement Learning Based Approach for Dynamic Optimal Power Flow in Active Distribution Network".

3.3 Design and tests of reinforcement–learning–based optimal power flow solution generator

Zhen et al. [20] developed an OPF solution generator based on a TD3 algorithm, which tries to minimise the generation cost of an IEEE 39–bus system by controlling the generator voltages and the active power generation. The corresponding power grid contains about 39 buses, 46 branches, ten generators and 21 loads. The observation contains the reactive and active power flow of the loads only. The complex reward function returns a fixed value when the OPF solution diverges, a variable value based on the number of constraint violations or the active generation loss. The main target of this work was to create an alternative solution method for solving OPF problems using RL. The authors used a supervised–based warm–up training phase to achieve faster training with more promising results to prevent the algorithm from wrong neural network initialization. Further, parallel computing helps to improve the training efficiency.

The authors chose the TD3 mainly because it is an improved derivative of the DDPG algorithm, which had already shown promising results in other papers.

They let out a detailed discussion about the HP selection. They focused more on providing information about the actor and critic network design. Both are three–layer Multi Layer Perceptrons (MLPs), whereas the critic network has 61 neurons in the input layer followed by two layers with both 256 neurons and finally one last layer with one neuron. Similarly, the actor used the same network structure except for the input and output layers, which only contained 42 in the input and 19 neurons in the output layer. Further HPs are listed in Table 3.3. A HPT was not mentioned in the paper.

Hyperparameters	**Values**
Actor learning rate	0.0001

Critic learning rate	0.01
Batch size	256
Soft Target Network Update (τ)	0.05
Exploration strategy	Gaussian noise
Normal distribution variance	1.0 decreasing to 0.01 while training
Policy delay	2

Table 3.3: Hyperparameter selection of "Design and tests of reinforcement–learning–based optimal power flow solution generator".

3.4 Real–Time Optimal Power Flow: A Lagrangian based Deep Reinforcement Learning Approach

Yan et al. [3] use a DDPG algorithm with no critic network. Instead, a Lagrangian computation approach was chosen because the authors argue that a critic network can induce higher approximation errors and learning variances. They tested their approach in an IEEE 118–bus system, where the generator power output and voltage magnitude are controlled. The corresponding state incorporates the load's reactive and active power flow and the last generator's active power flow.
There is no information or discussion provided why the authors used a DDPG algorithm. Further, because of the chosen approach with the changed gradient determination procedure, compared to the standard DDPG algorithm, only a tiny number of HPs were mentioned, like the LR of 0.01 and the memory buffer size of 4000. The authors also did not provide information about a HPT.

3.5 A Data–driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning

Zhou et al. [21] used a PPO algorithm to solve two different OPF problems by minimising the generation cost. They utilized supervised–based imitation learning to pre–train the actor network, improving the learning. To evaluate the robustness of their approach, Zhou et al. used the IEEE 14–bus system with five generators, 20 lines and the Illinois 200–bus system with 38 generators and 245 lines, in which the PPO adjusts the generator's active power and voltage settings. The state consists of each bus' reactive and active power flow and each generator's active power and voltage settings, respectively.
The used complex reward function, for both environments, returns a constant negative value if the solution diverges. If some constraints were violated, a negative reward depending on the violations will be emitted or the active generation loss will be returned.

After a detailed discussion about different algorithms and approaches, the authors paid special attention to robustness, performance, and extensibility. They believe that the PPO algorithm provides these characteristics through the trust regions update procedure (see 2.4.5). Algorithms, like the DDPG, tend to make large gradient updates, which do not satisfy the author's expectations.

The HPs for both OPF problems are presented in Table 3.4. Zhou et al. also provide information about some variable HPs, which change depending on the used OPF environment, for example, the actor and critic network structures. The actor network for the IEEE 14–bus system is a five layer network with [38, 380, 195, 100, 10] neurons, where the hidden layers get activated with a Rectified Linear Unit (ReLU) function and the output layer gets activated with a sigmoid function. The corresponding critic network uses also five layers with [38, 380, 44, 5, 1] neurons. The output layer has no activation because of the value output. Subsequentially, the Illinois 200–bus system network only differ in the number of layers and neurons. The actor network is a ten layer network with [476, 2048, 1024, 1024, 1024, 512, 512, 512, 512, 76] neurons because of the much larger action space. The corresponding critic network is a five–layer network with [476, 4760, 154, 5, 1] neurons. Zhou et al. did not explain how those HPs were derived. Also, they did not mention a HPT procedure, except that they rerun some experiments if the results were unsatisfying.

Hyperparameters	**Values**
Batch size	20
PPO Clip Parameter (ϵ)	0.2
GAE λ 2.4	0.98
Discount Factor (γ)	0.95

Table 3.4: Hyperparameter selection of the paper "A Data–driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning".

3.6 Optimal dispatch of an integrated energy system based on deep reinforcement learning considering new energy uncertainty

In the paper from Zhou et al. [22], the problem of modern random fluctuations in energy systems should be coped with a RL approach. Therefore, they used a PPO algorithm to control dynamic electric and heat power dispatches by minimising the power generation cost.

The authors discussed primarily the DQN, which suffers under the curse of dimensionality regarding the discretisation of the continuous action space. Therefore, the DDPG was evaluated based on literature, which proposed that the DDPG algorithm relies too much on single HPs and tends to unstable convergence. Hence, based on the research, Zhou et al. use the PPO algorithm that incorporates advantages like importance sampling, which is helpful in environments with sparse data amounts.

Although the PPO algorithm was explained extensively, the authors only provided relatively few HPs. They used an actor and a critic network with three ReLU hidden layers with 64 neurons. The batch size is 1000 and the minibatch size is 96. They updated the model 10000 times. Those HPs were derived through some experiments and adjustments during training. They tried out some HP configurations and discussed the results, but a further HPT was not mentioned.
After a sensitivity analysis, Zhou et al. compared their results generated by the PPO with those of a DDPG and traditional algorithms.

3.7 Summary

In this summary, the analysis results are presented. Therefore, it is worth mentioning that each paper has its scope and sometimes, the papers did not meet the analysis' expectations. Regarding the algorithm choice, the analysis showed that three papers (3.1, 3.2, and 3.4) did not use state–of–the–art algorithms. Instead, they used DDPG. Yan et al. 3.4 even developed an improved version of DDPG because the authors guessed that it performs better than a standard DDPG. Only 3.3, 3.5 and 3.6 decided to use the state–of–the–art algorithms TD3 and PPO. However, none of the papers, except for 3.5 and 3.6, provided an algorithm–driven explanation why these specific RL algorithms were chosen. Also worth mentioning is that four papers chose off–policy algorithms, like DDPG or TD3. Only 3.5 and 3.6 have chosen an on–policy RL algorithm. Both papers used the PPO as one of the most efficient on–policy algorithms. Other algorithms like the less efficient A2C were never taken into consideration.
Another essential analysis criterion is the HP selection. Usually, the papers provide the most crucial algorithm HPs, but often, important HPs are not mentioned, like the LR in 3.5 and 3.6, the discount factor in 3.3 or the replay buffer size in 3.1. Sometimes, papers refer to experiments from which they obtained their HPs, like in 3.1 and 3.6. Beyond that, some HP selections seem to be unusual, like a LR of 1.0 in 3.1, a variance of 0.9 for a normal distribution noise or a ten layer actor network with [476, 2048, 1024, 1024, 1024, 512, 512, 512, 512, 76] neurons. Subsequentially, 3.3, 3.4 and 3.5 did not explain how they determined their HPs. Sometimes, even details about the OPF problems are left out, like the observation and reward functions, for example, in 3.6. Moreover, none of the papers optimised their HPs through a HPT.

4 Requirements

In this section, the requirements formulation will be done for the environments, agents and the benchmark procedure, which sets the boundary conditions for this comparison. The in chapter 5 developed approach with all its components must fulfil all the requirements to ensure a scientific procedure.

EFT–01 Literature and Comparison Efforts Restricting the literature research and computation procedures, will retain the time–schedule of this master thesis.

ENV–01 Open–Source Environment To ensure reproducibility of the thesis' results, an open–source environment, which experts developed, should be used.

ENV–02 Environment Diversity The environment must be able to benchmark multiple OPF problems, increasing the reliability of the comparison.

AGT–01 RL Framework To ensure the correct implementation of all RL algorithms and the reproducibility of the comparisons, a standard and open–source RL framework must be used.

AGT–02 RL Algorithm Codebases As proposed in [23], the RL algorithms codebases should be as near as possible to the original algorithm based on the released paper. This guarantees the comparability of results, especially when different RL frameworks will be used. Each deviation from the algorithm standard must be mentioned in chapter 5.

COMP–01 Number of Environments In this thesis, at least two environments will be tested because Henderson et al. [23] pointed out that comparisons of multiple RL algorithms for a single environment can produce misleading results, which are not transferable to different environments.

COMP–02 Number of Trials According to [23], comparisons with only a few number of trials ($N < 5$) can be potentially misleading. Therefore, all results used for the comparison should be repeated at least five times.

COMP–03 Hyperparameter Tuning To elicit the best performance of each RL algorithm, a HPT should be done as mentioned in [23]. Therefore, essential HPs must be determined in a procedure to reduce the computations. Further, all details for the HP determination and the HPT must be documented in chapter 5 to guarantee transparency.

COMP–04 Performance Measurement To measure the performance of RL algorithms, the Episode Reward Mean (ERM) and Valid Solution Mean (VSM) will be used as metrics.

COMP–05 Evaluation Criteria To determine the most appropriate RL algorithm the performance (see 4), reproducibility, training stability, sample efficiency and training time are considered.

5 Concept

This chapter will explain the comparison procedure and settings of agents and environments.

5.1 Approach

The proposed approach of this thesis is divided into three separate phases. The sweep–over–phase will handle the large HP SS through multiple sweep–over runs. Based on these sweep–over results, the HP SS borders for each algorithm will be restricted. These algorithms have been chosen according to section 5.3. Afterwards, in the tuning–phase, the optimal HP configuration will be determined through a HPT, ensuring that only the most performant algorithms with their corresponding HPs are compared. Finally, in the comparison–phase, a comparison between all tuned algorithms will determine the optimal algorithm or algorithms for the proposed OPF problems.
Several sweep–over runs should determine the HP SSs in the sweep–over–phase, which is important to guarantee an efficient HPT. Large SSs could slow down the HPT process significantly and could lead to less performant results.
At first, a baseline set of HPs will be defined. Then, multiple values are tested for one HP in the baseline set. All other HPs in the baseline set stay untouched. This procedure will be done for each environment's HPs.
The performance and reproducibility will be analysed after collecting five trials for each HP configuration. A similar procedure regarding the sweep–over runs was used by Eimer et al. [24].
In the tuning–phase, the best possible HP configurations will be determined through HPT. Therefore, multiple algorithms, like grid or random search (2.5.1 and 2.5.2), can be used. The more recently developed Population Based Training (PBT) (see 2.5.3) has shown promising results, especially for RL algorithms [24]. Hence, the PBT will be used to find the best HP configurations. Further details about the PBT configuration are documented in section 5.6.
Next, a comparison between the baseline and tuned HPs will be done in the tuning–phase, verifying that the HPT was necessary and has found more optimal HPs. Therefore, ten trials on different random seeds (according to [23]) under baseline and tuned HPs are collected and compared regarding the performance, reproducibility and stability. The top–

performing algorithms versions, whether based on baseline or tuned HPs, are then used in the last overall algorithm comparison.
Finally, all top–performing algorithms will be compared with each other. The evaluation metrics are presented, and with it, the final selection of the most suited algorithm or algorithms will be done. All details about these evaluation metrics are shown in section 5.2.

5.2 Performance and Evaluation Criteria

An essential part of this comparison are the metrics, which are measuring the performance and determining the top–performing algorithms. The performance will be measured through the Episode Reward Mean (ERM) and Valid Solution Mean (VSM), whereas the ERM represents the main metric because RL algorithms only try to maximise the collected reward. There is a training and test performance, whereas the training performance is measured on the training data. The test performance is measured on a separate test dataset after the completed training.

The final evaluation compares the algorithms' performance, reproducibility, stability, sample efficiency and training time.
The performance is measured as already mentioned.
The reproducibility is defined as the Standard Deviation (STD) between the collected test trials, providing an impression about the algorithm's fluctuations between single trials.
The stability analyses the training behaviours itself. Are there fluctuations within the training, how smooth was the training procedure and are there significant drops in performance? These questions are thematised in the stability analysis.
The sample efficiency analysis will show how much sampling is required to reach a specific performance.
The training time analysis compares the time required for training.

5.3 Reinforcement Learning Algorithm Choice

One key aspect of this thesis is the reasonable choice of RL algorithms. In this work a DDPG 2.4.1, TD3 2.4.2, SAC 2.4.3, PPO 2.4.5 and an A2C 2.4.4 algorithm are examined. The distribution of algorithms in this work is defined through their classifications in Table 2.5. Hence, two on–policy and three off–policy algorithms are considered.
The promising results of the DDPG, TD3 and PPO in chapter 3 are one reason to examine these algorithms in this work. Another reason for selecting algorithms is their classification to be state–of–the–art. The TD3, PPO and SAC are generally meant to be state–of–the–art.
The last reason for considering an algorithm in this thesis is the published literature in the

context of solving OPF problems. Therefore, the A2C algorithm will also be considered. There is not much research published about solving OPF problems with the A2C because researchers usually use more recently developed algorithms, like PPO (see [21]). Moreover, even if A2C is not the latest state–of–the–art technology, it remains commonly used.

5.4 Environments

In this comparison, two environments are used to measure the performance capabilities of the chosen RL algorithms (see section 5.3). Therefore, both environments have different difficulty levels. The Q–Market environment is simpler, whereas the Economic–Dispatch environment is more challenging. The environments are in the mlopf[1] repository.
Also important is the comparability of the used environments with those of the literature. The analysis in chapter 3 showed that the Q–Market and Economic–Dispatch OPF problems are comparable to those, which are referred in the analysed papers. The number of generators and the observations and actions used are often similar.

One–Step Environments Both environments are one–step environments, which means that an agent can only do a single step in the environment. After this, the episode ends and the following OPF problem situation will be loaded.

Simulation Both environments use the data from SimBench[2] [25], a project for developing power grid benchmark datasets. Further processing, like calculating the next power grid state after an action was processed, will be done with the pandapower[3] library [26].

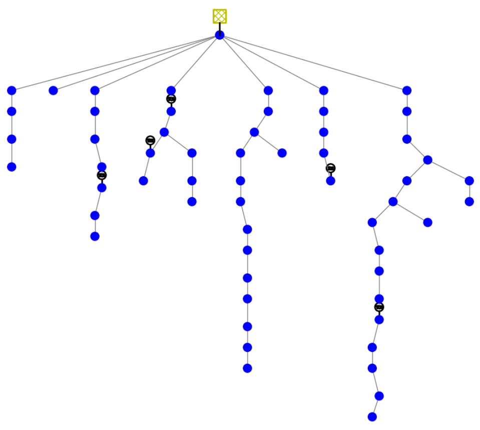
Reward Function Both environments use the same reward function structure based on objectives and penalties. The objective part adds up the active and reactive linear and quadratic poly–costs for all static and normal generators, loads, storages and the external grid. The second part of the reward function adds penalties to the objective reward if constraint violations are committed. Therefore, the agent receives penalties for violating the lower and upper voltage boundaries, overloading lines and transformers, and violating the external grids' minimum and maximum reactive and active power boundaries.

5.4.1 Q–Market Environment

The task in the Q–Market environment is to control the reactive power flow of five static generators, which always output a positive active power flow in a low–voltage urban power

[1] `https://gitlab.com/thomaswolgast/mlopf`
[2] `https://simbench.de/de/`
[3] `https://www.pandapower.org/`

grid. The Simbench power grid dataset can be found under the name: "*1–LV–urban6–0–sw*". The reactive power flow output lies between -100% and +100% for all five static generators. Hence, each action the agent determines contains five values between $[-1.0, 1.0]$.


(a) Static generator locations in the power grid.

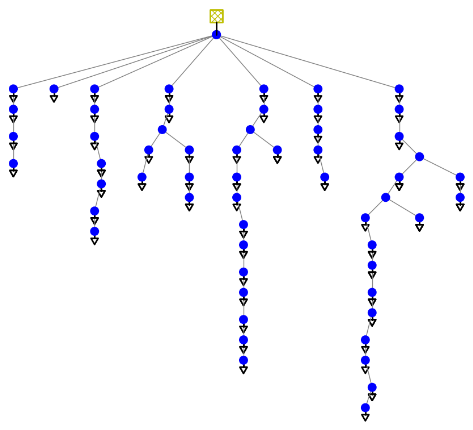

(b) Load locations in the power grid.

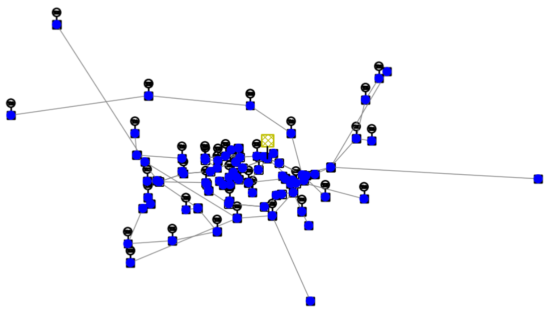
Figure 5.1: Q–Market power grid – A visualization of the power grid, where the yellow rectangle is the connection to the external power grid and the blue circles are the buses connected through the grey lines. The black circles in 5.1a are the generators and the black triangles in 5.1b are the loads.

The observations contain the active feed–in of the generators as well as the reactive and active power flow of the 111 loads, which are connected through 59 buses and 57 lines. The last part of the observation are the prices of the reactive power in euros for all static generators and the one external power grid. Therefore, the observation has 233 inputs, five from static generators, 222 (111×2) from the loads' reactive and active power flow and six from the prices of the reactive power in euro for five static generators and one external grid.

5.4.2 Economic–Dispatch Environment

The task in the Economic–Dispatch environment is to control the active power flow of 42 static generators in a high–voltage urban power grid. The Simbench[2] power grid dataset can be found under the name: "*1–HV–urban–0–sw*". The active power flow output lies between 0% and +100% for all 42 static generators. Hence, each action the agent determines contains 42 values between $[0.0, 1.0]$.
The observation contains the reactive and active power flow of the 79 loads connected through 372 buses and 113 lines. The last part of the observation are the prices for the active power in euro for static generators and the one external power grid. Therefore, the observation has 201 inputs, 158 (79×2) from the loads' reactive and active power flow and 43 from the prices of the active power in euro for the 42 static generators and the one

external grid.


(a) Static generator locations in the power grid.

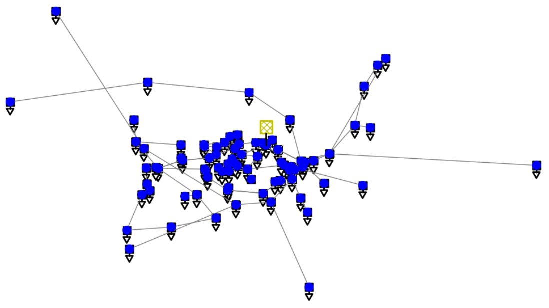

(b) Load locations in the power grid.

Figure 5.2: Economic–Dispatch power grid – A visualization of the power grid, where the yellow rectangle is the connection to the external power grid and the blue rectangles are the buses connected through the grey lines. The black circles in 5.2a are the generators and the black triangles in 5.2b are the loads.

5.5 Sweep–Over–Phase

The different HP settings and corresponding SSs are an essential part of the sweep–over–phase. This section documents all important information about the environments and RL algorithms. The main task of the sweep–over–phase is to obtain the performance influence of each single HP to determine the corresponding SS.
An inspiration for the algorithm's baseline HP section was taken from Ray RLlib[4] [27] and Stable–Baselines–3[5] [28]. Additionally, a series of undocumented experiments were done to get an impression of the algorithm's training behaviours. Another way to find appropriate baseline HPs is the literature in chapter 3, but this is difficult because each paper used a different OPF environment and some HP decisions are questionable as shown in section 3.7. Therefore, the papers have not been taken carefully into consideration.
Each algorithm's baseline and sweep–over values are shown in section 7.3.

Keep in mind that the SAC, A2C and PPO algorithms have twice as many output neurons in the last layer as actions in the environment because one–half determines mean values, and the second half determines STD values. Both are then used to create a normal distribution from which the actions are sampled.
Also important to note is that in this thesis, there are limitations regarding the neuronal

[4]https://docs.ray.io/en/latest/rllib/index.html
[5]https://stable-baselines3.readthedocs.io/en/master/

network designs because of the available computational resources. Therefore, only two or three hidden layers are tested. Furthermore, some undocumented experiments showed only small changes with four and five hidden layers, so these are not considered.
Similarly, the activation function will also not be tuned because the observation contains relatively large values, which may lead to extensive gradient updates. Due to an applied normalization filter, based on the mean and STD of each single observation unit (tensor unit), the observation is centred at a mean of zero with a STD of one. Therefore, the tanh (hyperbolic tangent) function is especially suited for this new observation space.

5.6 Tuning–Phase

After determining the HP SSs during the sweep–over–phase, the HPT begins. The corresponding HP SSs are documented in section 7.3. For the HPT, the PBT algorithm 2.5.3 will be used because of its great successes, especially in RL.
In this thesis, Ray's PBT standard meta–hyperparameters (the hyperparameter of the PBT algorithm) are used to optimise the ERM. They are similar to those in [19], which were used to tune RL algorithms. Therefore, no significant changes were made to the PBT meta–hyperparameters.
Because the optimisation SS is typically quite large due to the number of HPs, a population size of 50 was chosen. To further ensure a fair procedure, each algorithm performs ten perturbation cycles with a burn–in–time (initial training time before PBT procedure starts) of one perturbation cycle. In each cycle, the parameters of the twelve most performant individuals are transferred to the twelve worst performed individuals during the exploitation. Afterwards, all individuals HPs are mutated or resample during the exploration. There is a 25% chance for resampling and a 75% chance for mutation. Mutations are controlled by the perturbation factors, which control the mutation strength by multiplying the HPs with one random perturbation factor. These are set to (1.2, 0.8). Hence, if a mutation occurs to a HP, it gets multiplied by 1.2 or 0.8.

Note that the performance evaluation in the PBT to exploit and explore is usually performed on a separate validation set. However, because only a test dataset is available, these evaluations are done on the training data.
Also worth mentioning is that there are two variants to use the PBT results. The first variant is to rerun the PBT schedule of the best individual in the population. The second variant uses the best individual's last HP configuration. This thesis will use the second variant, the final HP configuration because the handling is more straightforward than with a PBT schedule.

After performing the HPT for each RL algorithm in each environment separately, the optimal HP configurations are presented in chapter 8. Afterwards, a verification will be done to evaluate whether the baseline or tuned HPs are more performant regarding performance, stability and reproducibility. This will be done for each algorithm in each environment separately by collecting ten trials under different random seeds. Hence, at the end, ten baseline and tuned trials were collected. The superior algorithm version (baseline or tuned) are called top–performing algorithm versions.

5.7 Comparison–Phase

Finally, the top–performing HPs are then used to perform the last overall comparison based on the ten collected trials to determine the RL algorithm with the highest performance (test and training performance will be evaluated), reproducibility, stability, sample efficiency and training time (see COMP–05). The used metrics are further explained in section 5.2. Additionally, RL algorithm recommendations regarding both environments will enable a differentiated selection for further OPF problems under certain situations, like production, testing or proof–of–concept. The comparison–phase results are documented in chapter 9.

6 Implementation

This chapter documents the main technical details about the used frameworks and software. Further, the background information about the data acquisition hardware will be presented.

6.1 Software and Frameworks

For the reason of reproducibility, in this master thesis, mainly open–source software and frameworks are used to compare the different RL algorithms with each other. Ray's RLlib[1] [27] and Tune[2] [29] are the RL and Tuning frameworks used in this thesis. These are still actively supported and receive updates. Especially, the ability of distributed learning in clusters guarantees a fast parallel completion of multiple trials. Also, [2] refer to Ray because of the already mentioned abilities and attributes.

Python 3.10 was used as a foundation. This Python version was chosen because it is the latest stable version on which Ray works and receives long–time–update support.

Also important for the functionality of Ray's RLlib is the Machine Learning framework choice. In this case, Facebook's PyTorch[3] was used because it is a commonly used Machine Learning framework in research. However, because Ray is the top layer software, the choice of the Machine Learning framework has less influence on handling and configuration efforts. For the experiments, the version 2.1.1 was used.

The environment depends on the open–source frameworks SimBench[4] [25] and pandapower[5] [26], which were both developed by the Fraunhofer Institute for Energy Economics and Energy System Technology. With SimBench and pandapower, the OPF problem situations are simulated and solutions are verified. Around these, a RL framework was build, enabling the possibility to solve OPF problems through connecting power grid simulations with the

[1]https://docs.ray.io/en/latest/rllib/index.html
[2]https://docs.ray.io/en/latest/tune/index.html
[3]https://pytorch.org/
[4]https://simbench.de/de/
[5]https://www.pandapower.org/

RL procedure. The repository with the frameworks can be found under mlopf[6].

Next to the explicitly mentioned software, further libraries are documented in the repository OPFBenchmarkFramework[7]. It also contains a setup script with the needed library versions. Sometimes, libraries need unique versions of sub–libraries, which are usually not the latest. All Python files for the sweep–over–, tuning– and comparison–phase are documented in the OPFBenchmarkFramework repository.

6.2 Hardware and Time Consumption

To contextualize the computations with their needed resources, this section will give an impression of the used hardware and computation time. The computations have been done on two different slurm high computation clusters. Scripts for using Ray on a slurm[8] cluster can be found in the OPFBenchmarkFramework repository. The sweep–over computations for each algorithm and environment took about two to three days on 100 nodes, with every node containing 24 CPUs (Intel Xeon CPU E5–2650 v4 12C 2.2GHz) and at least 128 GB RAM.
The tuning computations took about six to seven days on four nodes, with every node containing 128 CPUs (2x AMD Genoa EPYC 9554 64 Cores, 3.1 GHz) and 768 GB RAM. The ten comparison trials per each algorithm were computed on the same cluster as the tuning computations but on a single node in mixed–mode (not all node resources are allocated) with 90 CPUs. For all algorithms together, the comparison computations took about 2.5 days.

6.3 Library Adjustments

During the setup of Ray's RLlib one error appeared, which was fixed. It is located at the ray/rllib/algorithms/a3c/a3c.py file in the overwritten training method. There the developers accidentally set the lr_schedule list value as the use_critic boolean value, which leads to an immediate crash of the program when trying to start the A2C training.

[6]https://gitlab.com/thomaswolgast/mlopf
[7]https://github.com/GerTheMessiah/OPFBenchmarkFramework
[8]https://slurm.schedmd.com/documentation.html

7 Sweep–Over–Phase Evaluation

In this chapter, the sweep–over results will be presented containing the selected baseline HPs and their corresponding training result analysis.

7.1 Sweep–Over Parameter Selection

In this section, the initial baseline HPs and corresponding sweep–over values for each algorithm are presented. These are used to perform the sweep–over runs described in section 5.5. Corresponding results are then discussed in section 7.3.

7.1.1 DDPG Hyperparameter Settings

The tables 7.1 and 7.2 contain the baseline HPs and sweep–over values.

Hyperparameters	**Q–Market**	**Economic–Dispatch**
Actor learning rate	0.0001	
Critic learning rate	0.001	
Actor network	[233, 256, 256, 256, 5]	[201, 256, 256, 256, 42]
Critic network	[233, 256, 256, 256, 1]	[201, 256, 256, 256, 1]
Network activation	tanh	
Batch size	1024	
Soft Target Network Update (τ)	0.0025	
Replay buffer size	131072	
Exploration technique	Gaussian noise	
Gaussian noise STD	0.01	
Iterations until training ends	10000	
Episodes per training iteration	8	
Episodes before training	1500	
L2 regularization [30]	$1e^{-6}$	
Discount Factor (γ)	Not relevant, because one–step environment.	
Reward scaling factor	0.02	0.000025

Table 7.1: Baseline hyperparameters of the DDPG algorithm and the environments.

Hyperparameters	**Sweep–over values**
Actor learning rate	0.00005, 0.0001, 0.0002, 0.0003, 0.0004
Critic learning rate	0.0001, 0.0005, 0.001, 0.002
Actor and critic network hidden layers	[256, 256], [512, 512], [256, 256, 256], [256, 512, 256], [512, 256, 512], [512, 512, 512]
Batch size	128, 256, 512, 1024
Soft Target Network Update (τ)	0.001, 0.002, 0.003, 0.004, 0.005
Gaussian noise STD	0.001, 0.005, 0.01, 0.05, 0.1

Table 7.2: Sweep–over values for the DDPG algorithms.

7.1.2 TD3 Hyperparameter Settings

Because the TD3 algorithm is the successor of the DDPG, which makes them act similarly, the baseline HPs and sweep–over values are the same (see Tables 7.1 and 7.2). Additionally, the TD3 uses a policy delay procedure, which is explained in 2.4.2 that can be controlled by a policy delay HP. Both environments use a baseline policy delay value of three. The sweep–over runs will test the policy delay values from one to five.

7.1.3 SAC Hyperparameter Settings

The SAC baseline HPs and corresponding sweep–over values are documented in Table 7.3 and Table 7.4. For the SAC there are no papers analysed in chapter 3. Therefore, more undocumented experiments were done to find a well–behaving starting area in the SS. Note that the first experiments have shown that the SAC algorithm needs action normalisation and action clipping (unallowed actions will be clipped according to the environment's action space) to function correctly. The SAC is the only algorithm, which requires those extra features. According to undocumented tests, the other algorithms might even perform worse when these features were applied.

Hyperparameters	**Q–Market**	**Economic–Dispatch**
Actor learning rate	0.0003	
Critic learning rate	0.0003	
Entropy learning rate	0.0003	
Actor network	[233, 256, 256, 256, 10]	[201, 256, 256, 256, 84]
Critic network	[233, 256, 256, 256, 1]	[201, 256, 256, 256, 1]
Network activation	tanh	
Batch size	1024	

Soft Target Network Update (τ)	0.005	
Replay buffer size	262144	
N–step	1	
Initial alpha α	1.0	
Iterations until training ends	15000	
Episodes per training iteration	8	
Episodes before training	1500	
Exploration technique	Stochastic sampling	
Discount Factor (γ)	Not relevant, because one–step environment.	
Reward scaling factor	0.02	0.000025

Table 7.3: Baseline hyperparameters of the SAC algorithms and environments.

Hyperparameters	**Sweep–over values**
Actor learning rate	0.0001, 0.0003, 0.0007, 0.001
Critic learning rate	0.0001, 0.0003, 0.0007, 0.001
Entropy learning rate	0.00005 0.0001, 0.0003, 0.0007, 0.001
Actor network hidden layers	[256, 256], [512, 512], [256, 256, 256], [256, 512, 256], [512, 256, 512], [512, 512, 512]
Critic network hidden layers	[256, 256], [512, 512], [256, 256, 256], [256, 512, 256], [512, 256, 512], [512, 512, 512]
Soft Target Network Update (τ)	0.0001, 0.005, 0.01, 1.0
Batch size	128, 256, 512, 1024

Table 7.4: Sweep–over values of the SAC algorithms.

7.1.4 A2C Hyperparameter Settings

The baseline HPs and corresponding sweep–over values for the A2C can be found in Table 7.5 and Table 7.6. For the A2C, no papers were analysed in chapter 3. Hence, the HPs were mainly obtained from undocumented experiments, RLlib [27] and Stable Baseline 3 [28].

Hyperparameters	**Q–Market**	**Economic–Dispatch**
Learning rate	0.0001	
Network	[233, 256, 256, 256, 10]	[201, 256, 256, 256, 84]
Activation	tanh	
Batch size	1024	

Value function loss coefficient	0.5	
Entropy coefficient	0.0	
Exploration technique	Stochastic sampling	
Iterations until training ends	600	
Discount Factor (γ)	Not relevant, because one–step environment.	
Reward scaling factor	0.02	0.000025

Table 7.5: Baseline hyperparameters of the A2C algorithms and environments.

Hyperparameters	**Sweep–over values**
Learning rate	0.00005, 0.0001, 0.0002, 0.0003, 0.0004
Network hidden layers	[256, 256], [512, 512], [256, 256, 256], [256, 512, 256], [512, 256, 512], [512, 512, 512]
Batch size	128, 256, 512, 1024
Value function loss coefficient	0.5, 0.7, 0.9, 1.0
Entropy coefficient	0.0001, 0.0005, 0.001, 0.005, 0.01

Table 7.6: Sweep–over values of the A2C algorithms.

7.1.5 PPO Hyperparameter Settings

The baseline HPs for the PPO can be found in Table 7.7 and the corresponding sweep–over values are shown in Table 7.8.

Hyperparameters	**Q–Market**	**Economic–Dispatch**
Learning rate	0.0001	
Network	[233, 256, 256, 256, 10]	[201, 256, 256, 256, 84]
Network activation	tanh	
Batch size	4096	
Minibatch size	1024	
SGD iterations	5	
PPO Clip Parameter (ϵ)	0.2	
Value function loss coefficient	0.5	
Entropy coefficient	0.0	
Iterations until training ends	150	
Discount Factor (γ)	Not relevant, because one–step environment.	
Exploration technique	Stochastic sampling	

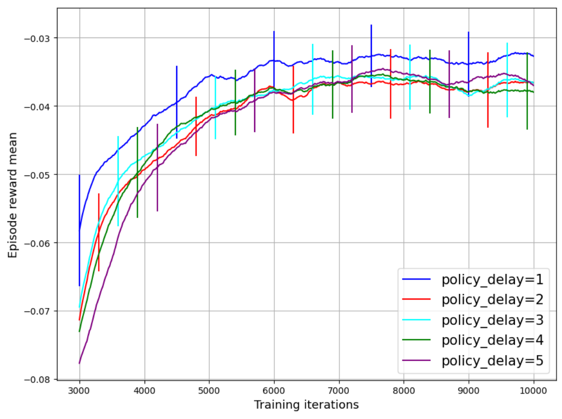
Reward scaling factor	0.02	0.000025

Table 7.7: Baseline hyperparameters for the PPO algorithms and environments.

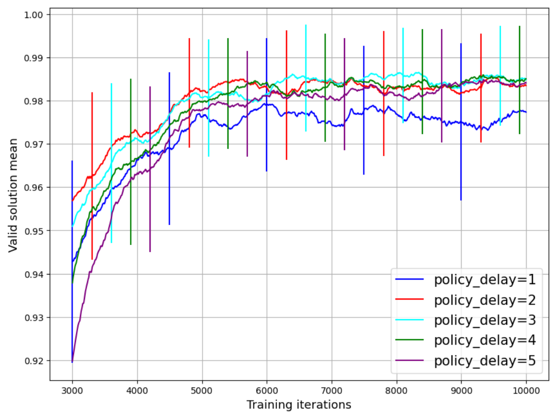
Hyperparameters	**Sweep–over values**
Learning rate	0.00007, 0.0001, 0.0002 0.0003, 0.0005, 0.00075
Network hidden layers	[256, 256], [512, 512], [256, 256, 256], [256, 512, 256], [512, 256, 512], [512, 512, 512]
Batch size	1024, 2048, 4096, 8192
Minibatch size	128, 256, 512, 1024
SGD iterations	3, 5, 7, 10
PPO Clip Parameter (ϵ)	0.10, 0.15, 0.20, 0.25, 0.30
Value function loss coefficient	0.5, 0.7, 0.9, 1.0
Entropy coefficient	0.0001, 0.0005, 0.001, 0.005, 0.01

Table 7.8: Sweep–over values of the PPO algorithms.

7.2 Reward Function Effects



(a) Episode reward mean



(b) Valid solution mean

Figure 7.1: TD3 – Q–Market – Training – Sweep–over policy delay results.

During the analysis of the sweep–over results, different behaviours between ERM and VSM manifested. One example can be seen in Figure 7.1. The policy delay value 1 reaches a large ERM but only a relatively small VSM. This anomaly is caused by the reward function (see 5.4) of the environments. The ERM and VSM should typically correlate strongly, which is not always valid for the used reward function. It seems that the algorithms are motivated to commit more minor constraint violations because they receive less negative reward for violations as for better optimisations. Hence, committing minor violations could be more beneficial than avoiding them. Therefore, the ERM is chosen to be the primary metric in this thesis because each RL algorithm's target is to maximise its received reward. HPT algorithms are also often only designed to tune for the reward. A well–formulated reward function is assumed for the final algorithm recommendation in this thesis.

7.3 Sweep–Over–Phase Analysis

In this section, the visualisation of the results will take place. These results have been obtained according to the specifications in section 5.5. Therefore, trends, abnormalities and particularities are mentioned and explained. Each environment and RL algorithm was treated separately. The ERM are presented graphically with an impression of the STDs, visualised through error bars in the corresponding graphics. According to section 7.2 the VSM will not be considered.
For clarity reasons, not all graphics are shown in the following result presentation. Moreover, a selection based on interesting behaviours was done. The other not shown graphics can be found in Appendix A.
Also note, that each following SS interval, like $[5e^{-5}, 4e^{-4}]$ in Table 7.9, means a uniform distribution, from which the HPs are sampled.

7.3.1 DDPG Q–Market Sweep–Over Results

The presented DDPG sweep–over Q–Market results have been sampled according to the approach documented in section 7.1.1.
The results of the actor LR A.1, actor network hidden layers A.5, critic network hidden layers A.6 and Soft Target Network Update (τ) A.4 showed no significant influence on the ERM. Hence, the standard HPs are used (see Table 7.9). The batch size results shown in Figure A.3 indicate that larger batch sizes result in larger ERMs due to more experiences seen by the networks. Accordingly, the batch size 128 was removed from SS. The critic LR results shown in Figure 7.2 reach higher ERMs with larger LRs of 0.002 and 0.001. Therefore, the SS is set up to $[5e^{-4}, 2.5e^{-3}]$. The Gaussian noise STD results in Figure A.7

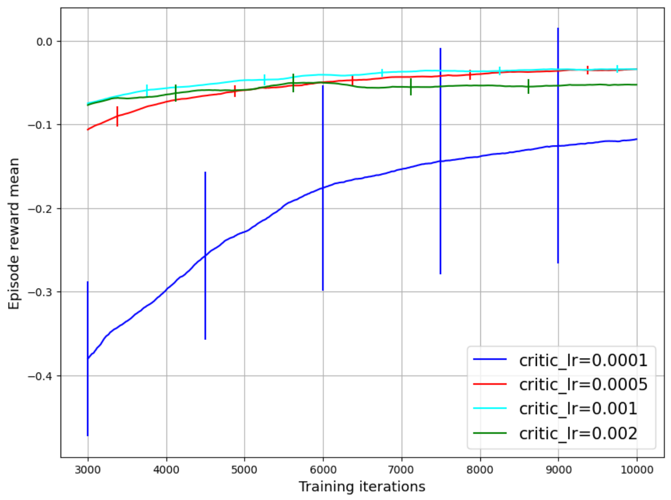


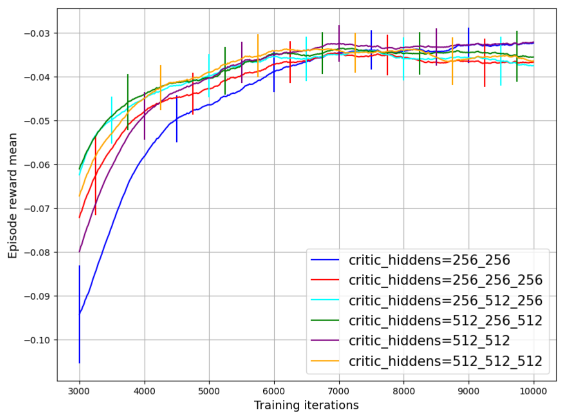
Figure 7.2: DDPG – Q–Market – Training – Sweep-over critic learning rate results.

benefit more from smaller values, which increase performance and reproducibility slightly, whereas larger values inhibit the learning process because of too much exploration. The SS for the Gaussian noise STD is set to $[0.001, 0.05]$. The resample and mutation SSs, which will be used in the PBT, are documented in Table 7.9.

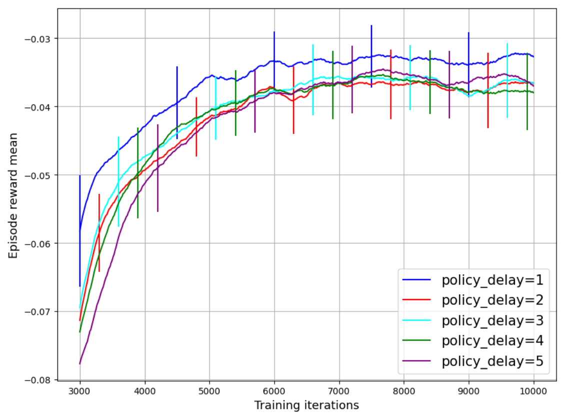
Hyperparameters	**Search Space**
Actor learning rate	$[5e^{-5}, 4e^{-4}]$
Critic learning rate	$[5e^{-4}, 2.5e^{-3}]$
Actor and critic hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Batch size	256, 512, 1024
Soft Target Network Update (τ)	$[0.001, 0.01]$
Gaussian noise STD	$[0.05, 0.001]$

Table 7.9: DDPG Q–Market resample and mutation values.

7.3.2 TD3 Q–Market Sweep–Over Results



(a) Critic network hidden layer results.



(b) Policy delay results.

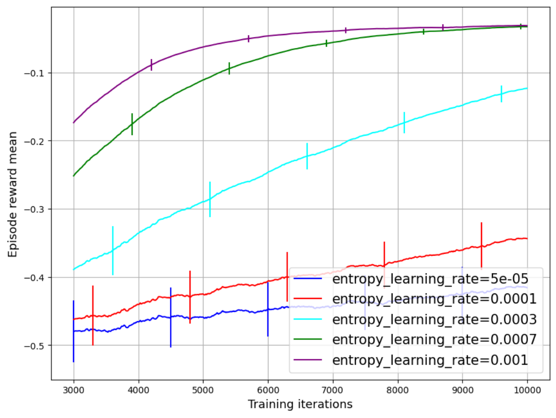
Figure 7.3: TD3 – Q–Market – Training – Sweep–over critic network hidden layer and policy delay results.

The presented TD3 sweep–over Q–Market results have been sampled according to the approach documented in section 7.1.2.
The results of the actor LR A.8, critic LR A.9, batch size A.10, Soft Target Network Update (τ) A.11, actor network hidden layers A.12 and Gaussian noise STD A.14 are equivalent to those of the DDPG presented in section 7.3.1.
Moreover, the critic network hidden layer results in Figure 7.3a shows that two hidden layer networks perform slightly better than three hidden layer networks. However, this

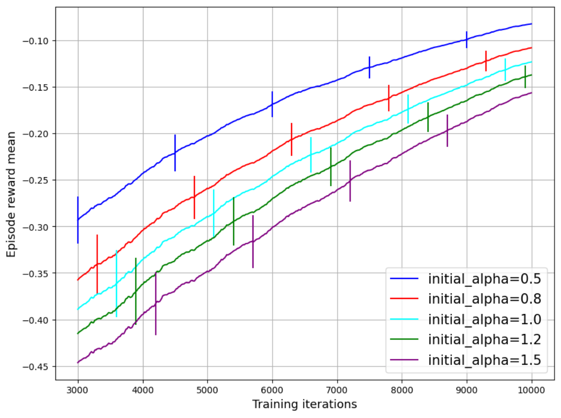
behaviour was probably caused by stochastic uncertainties. Therefore, this SS will be equal to the DDPG critic network hidden layer SS. Also interesting are the policy delay results shown in Figure 7.3b, where value 1 performs better than the remaining. This has been caused by the more frequently updated actor or policy network. However, because the values from two to five are nearly identical and the SS is quite small, all sweep–over policy delay values are kept in the SS. The mentioned similarities between TD3 and DDPG are the reason for using the DDPG resample and mutation values, shown in Table 7.9.

7.3.3 SAC Q–Market Sweep–Over Results

The presented SAC sweep–over Q–Market results have been sampled according to the approach documented in section 7.1.3.
After a series of undocumented experiments it turned out that the Ray RLlib SAC only functions with an action normalization and clipping in the Q–Market environment.
The results of the actor LR A.16, critic LR A.17, batch size A.19, Soft Target Network Update (τ) A.20, actor network hidden layers A.22 and critic network hidden layers A.23 showed no influence on the performance and reproducibility. Therefore, only the actor and critic LR SSs were expanded to find their SS borders. The other HP SSs have most likely already reached their boundaries.



(a) Entropy learning rate results.



(b) Initial alpha results.

Figure 7.4: SAC – Q–Market – Training – Sweep–over entropy learning rate and initial alpha results.

The influential HPs include, on the one hand, the entropy LR (see Figure 7.4a). Its results indicate that larger entropy LRs lead to faster and more performant convergence. The trials of 0.001 and 0.0007 are more reproducible than those of $5e^{-5}$ and 0.0001, which could be explained through the exploration and exploitation balance. Smaller entropy LRs allow a longer exploration, which reduces the convergence speed. The results further indicate that more deterministic sampling leads to better and faster performances.
In this context, also interesting are the initial alpha α value results in Figure 7.4b, which

show better performance with smaller values. According to these results, the initial alpha α SS will be shifted down from $[0.5, 1.5]$ to $[0.3, 1.0]$ and the entropy LR SS will be shifted up to $[5e^{-4}, 1.5e^{-3}]$. The resample and mutation values, which will be used in the PBT, are documented in Table 7.10.

Hyperparameters	**Search Space**
Learning rate actor	$[5e^{-5}, 2.0e^{-3}]$
Learning rate critic	$[5e^{-5}, 2.0e^{-3}]$
Entropy learning rate	$[5e^{-4}, 1.5e^{-3}]$
Initial alpha α	$[0.3, 1.0]$
Actor and critic hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Batch size	128, 256, 512, 1024
Soft Target Network Update (τ)	$[0.001, 1.0]$

Table 7.10: SAC Q–Market resample and mutation values.

7.3.4 A2C Q–Market Sweep–Over Results

The presented A2C sweep–over Q–Market results have been sampled according to the approach documented in section 7.1.4.

The value function loss coefficient A.27 results showed no significant influence on performance and reproducibility. The same applies to the batch size results shown in A.25, except for the convergence speed. Larger batch sizes converge faster than small values. For both HPs the standard SSs shown in Table 7.11 are used.

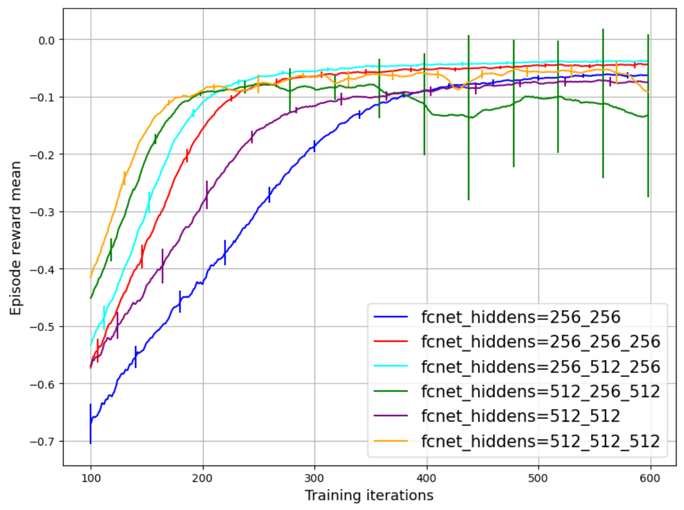


Figure 7.5: A2C – Q–Market – Training – Sweep–over network hidden layer results.

The LR A.24 analysis showed more performant results with smaller LR values of $5e^{-5}$ and $1e^{-4}$, which stabilize the training at the cost of longer convergence time. Therefore, the LR SS will be reduced to $[5e^{-5}, 3e^{-4}]$.

The entropy coefficient has only a tiny influence on the ERM as visualized in A.28. Larger

values of 0.01 and 0.005 decreased the performance slightly because of less exploration. Hence, the entropy coefficient SS will be set to $[0, 0.001]$.
Also interesting are the network hidden layer results, shown in Figure 7.5, indicating that three hidden layer networks are slightly more performant than two hidden layer networks. The actor or critic may require more layers to appropriately approximate the policy and value function. Additionally, the network with (512__256__512) showed fluctuations in stability, possibly caused by the hidden layer layout. This indicates that maybe three–layer networks with 512 neurons in the first hidden layer perform worse than networks with 256 neurons. Hence, all hidden layer combinations with 512 neurons in the first layer are removed from the SS. The resample and mutation values, which will be used in the PBT, are documented in Table 7.11.

Hyperparameters	**Search Space**
Learning rate	$[5e^{-5}, 3e^{-4}]$
Batch size	128, 256, 512, 1024
Network hidden layers	(256, 256), (256, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512)
Value function loss coefficient	$[0.5, 1.0]$
Entropy coefficient	$[0, 0.001]$

Table 7.11: A2C Q–Market resample and mutation values.

7.3.5 PPO Q–Market Sweep–Over Results

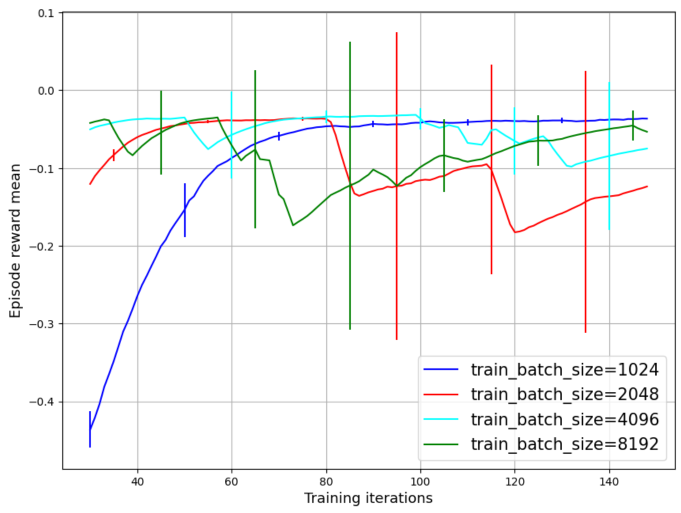


Figure 7.6: PPO – Q–Market – Training – Sweep–over batch size results

The presented PPO sweep–over Q–Market results have been sampled according to the approach documented in section 7.1.5.
No significant influence was going out from the LR A.29. Only the value 0.00075 did not perform as well as the others. Based on the experiences of the A2C the LR SS will be shifted down to guarantee more stability.
The results of the value function loss coefficient A.35, PPO Clip Parameter (ϵ) A.33 and entropy coefficient A.36 have a negligible influence on the performance.
The SGD iteration A.32 results showed that smaller values lead to a more stable learning.

However, all these mentioned HP results, except the LR, show volatilities, which stochastic uncertainties may have caused. Hence, choosing distinct SSs is difficult, which leads to the decision to use the standard SS shown in Table 7.12. The batch size results in Figure 7.6 further indicate that smaller values lead to performant results. Larger batch sizes are struggling more with stability and reproducibility issues. Typically, the opposite behaviour is expected, which leads to the assumption that stochastic uncertainties may cause these results. Therefore, all sweep–over values are used as SS.

The minibatch size results show the opposite behaviour in Figure 7.7. They imply that values like 128 and 256 are less performant and reproducible than larger values of 1024 and 512. One reason could be the number of SGD updates, which, together with the minibatch size, may destroy the policy due to overfitting. However, because the minibatch size 128 has a significantly bad performance, it will be removed from the SS. The network hidden layer results in Figure A.34 showed the same behaviour as those of the A2C in 7.3.4. Therefore, all hidden layer combinations with 512 neurons in the first layer are removed from the SS.

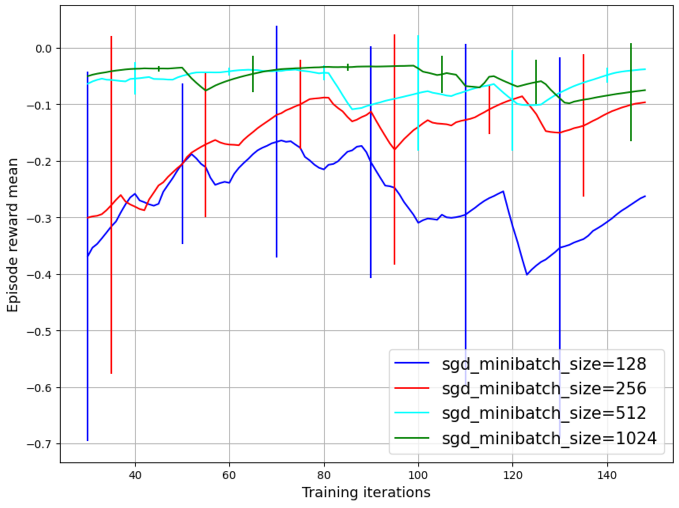


Figure 7.7: PPO – Q–Market – Training – Sweep–over minibatch size results.

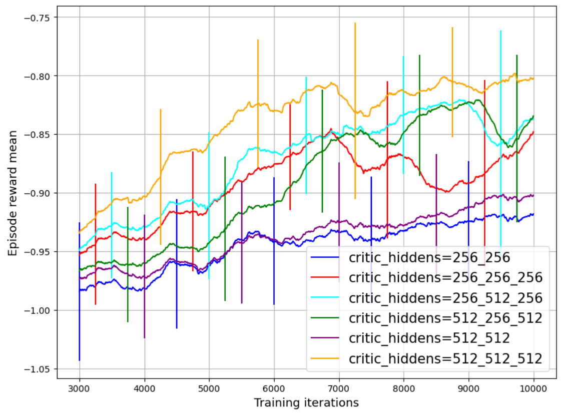
The PPO HP resample and mutation values are documented in Table 7.12.

Hyperparameters	**Search Space**
Learning rate	$[5e^{-5}, 5e^{-4}]$
Batch size	1024, 2048, 4096, 8192
Minibatch size	256, 512, 1024
SGD iterations	3, 4, 5, 6, 7, 8, 9, 10
PPO Clip Parameter (ϵ)	$[0.1, 0.3]$
Network hidden layers	(256, 256), (256, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512)
Value function loss coefficient	$[0.5, 1.0]$
Entropy coefficient	$[0, 0.01]$

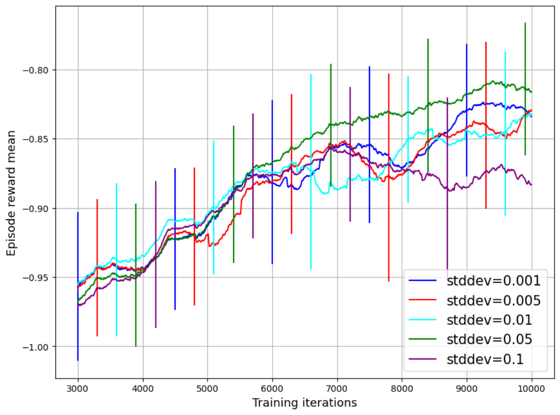
Table 7.12: PPO Q–Market resample and mutation values.

7.3.6 DDPG Economic–Dispatch Sweep–Over Results

The presented DDPG sweep–over Economic–Dispatch results have been sampled according to the approach documented in section 7.1.1.
The actor LR A.37, Soft Target Network Update (τ) A.40 and actor network hidden layer A.41 results showed no significant influence on performance and reproducibility. Therefore, these HP SSs are equivalent to the sweep–over value range. The critic LR A.38 benefits from higher values, like 0.002 and 0.001, similarly like in the Q–Market environment before (see 7.3.1).



(a) Critic network hidden layer results.



(b) Gaussian noise STD results.

Figure 7.8: DDPG – Economic–Dispatch – Training – Sweep–over critic network hidden layer and Gaussian noise STD results.

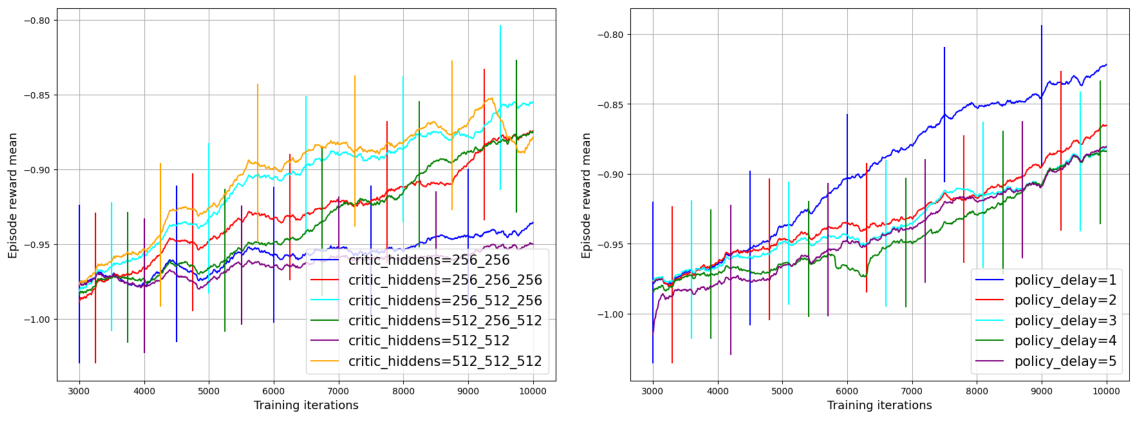
The same applies to the batch size A.39, which obtains larger ERMs with values of 1024 and 512. Hence, the critic LR SS will be shifted up to $[5e^{-4}, 2.5e^{-3}]$ and the batch size value of 128 will be removed. Also interesting are the critic network hidden layer results, which indicate that more neurons and layers are beneficial to achieve higher ERMs. Additionally, two hidden layer networks perform worse than three hidden layer networks, which led to the decision to remove all two hidden layer networks from the SS.
The Gaussian noise STD behaves similarly to the Q–Market results in section 7.3.1. The value 0.1, however, performed worse than all other values. Therefore, 0.05 will be the upper border of the SS. The DDPG HP resample and mutation SSs are presented in Table 7.13.

Hyperparameters	**Search Space**
Learning rate actor	$[5e^{-5}, 4e^{-4}]$
Learning rate critic	$[5e^{-4}, 2.5e^{-3}]$
Actor hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)

Critic hidden layers	(256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Batch size	256, 512, 1024
Soft Target Network Update (τ)	$[0.001, 0.01]$
Gaussian noise STD	$[0.001, 0.05]$

Table 7.13: DDPG Economic–Dispatch resample and mutation values.

7.3.7 TD3 Economic–Dispatch Sweep–Over Results



(a) Critic network hidden layer results.

(b) Policy delay results.

Figure 7.9: TD3 – Economic–Dispatch – Training – Sweep–over critic network hidden layer and policy delay results.

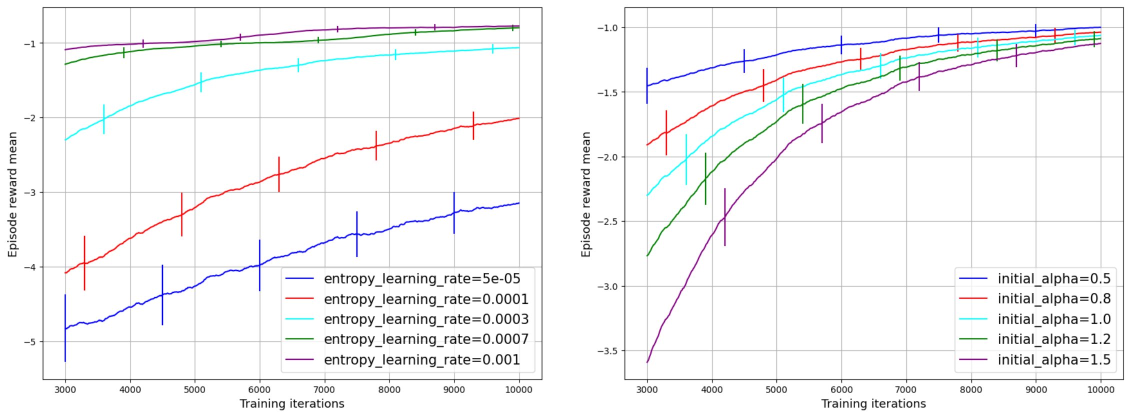
The presented TD3 sweep–over Economic–Dispatch results have been sampled according to the approach documented in section 7.1.2.
The actor LR A.44, Soft Target Network Update (τ) A.47, actor network hidden layer A.48 and Gaussian noise STD A.50 results showed no significant influences. Only the largest value of the Gaussian noise STD was less performant than the others. Hence, it will be removed from the SS. The critic LR A.45, batch size A.46 and critic network hidden layer results are showing similar behaviours as the DDPG Economic-Dispatch results in section 7.3.6. However, the TD3 results are more distinct than those of the DDPG. Hence, the SSs are the same as from the DDPG Economic–Dispatch. Moreover, the TD3 Economic–Dispatch and Q–Market results for the policy delay HP showed significant similarities. The policy delay value 1 has reached higher ERMs than the other values. However, all values from 1 to 5 will be part of the SS because of the small SS size. The TD3 HP resample and mutation value spaces are presented in Table 7.14.

Hyperparameters	**Search Space**
Actor learning rate	$[5e^{-5}, 4e^{-4}]$
Critic learning rate	$[5e^{-4}, 2.5e^{-3}]$
Actor hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Critic hidden layers	(256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Batch size	256, 512, 1024
Soft Target Network Update (τ)	$[0.001, 0.01]$
Gaussian noise STD	$[0.001, 0.05]$
Policy delay	1, 2, 3, 4, 5

Table 7.14: TD3 Economic–Dispatch resample and mutation values.

7.3.8 SAC Economic–Dispatch Sweep–Over Results

The presented SAC sweep–over Economic–Dispatch results have been sampled according to the approach documented in section 7.1.3. The Ray RLlib SAC only functions with a action normalisation and clipping.



(a) Entropy learning rate results.

(b) Initial alpha results.

Figure 7.10: SAC – Economic–Dispatch – Training – Sweep–over entropy LR and initial alpha results.

The results of the actor LR A.52, critic LR A.53, batch size A.55, Soft Target Network Update (τ) A.56, actor network hidden layers A.58 and critic network hidden layers A.59 did

not influence the performance and reproducibility. Therefore, for these HPs the standard SSs will be used, which are identical to those of the Q–Market environment (see section 7.3.3). Like the SAC Q–Market results, only the entropy LR and the initial alpha α value influence the performance and reproducibility. Larger entropy LRs lead to higher ERMs, which can be seen in Figure 7.10a. This behaviour is identical with the SAC Q–Market results shown in section 7.3.3. Similarly, the initial alpha α results in Figure 7.10b show the same characteristics as the corresponding Q–Market results in section 7.3.3. Therefore, these HP SSs will be the same as from the SAC Q–Market environment.
The resample and mutation values, which will be used in the PBT, are documented in Table 7.15.

Hyperparameters	**Search Space**
Actor learning rate	$[5e^{-5}, 2.0e^{-3}]$
Critic learning rate	$[5e^{-5}, 2.0e^{-3}]$
Entropy learning rate	$[3e^{-4}, 1.5e^{-3}]$
Initial alpha α	$[0.3, 1.0]$
Actor and critic hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Batch size	128, 256, 512, 1024
Soft Target Network Update (τ)	$[0.001, 1.0]$

Table 7.15: SAC Economic–Dispatch resample and mutation values.

7.3.9 A2C Economic–Dispatch Sweep–Over Results

The presented A2C sweep–over Economic–Dispatch results have been sampled according to the approach documented in section 7.1.4.
The results of the batch size A.61, value function loss coefficient A.63 and entropy coefficient A.64 showed that these HPs did not influence the performance and reproducibility significantly. Therefore, these SSs will be identical to the sweep–over value range.
Differently, the LR A.60 results influenced the convergence speed. Lower LRs, like $5e^{-5}$ or 0.0001, are at the beginning of the training less performant and reproducible. However, because the results indicate that the lower LRs eventually converge, the corresponding SS will also be identical to the sweep–over value range. Smaller LRs are

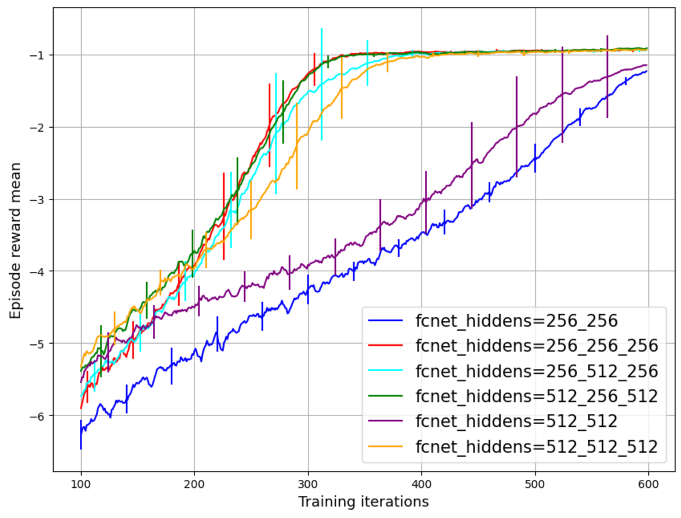


Figure 7.11: A2C – Economic–Dispatch – Training – Sweep–over network hidden layer results.

usually better suited for long training to prevent large gradient updates.

Lastly, the network hidden layer results in Figure 7.11 showed that two hidden layer networks are significantly less performant than three hidden layer networks. Accordingly, all two hidden layer networks are removed from the SS.

The resample and mutation values, which will be used in the PBT, are documented in Table 7.16.

Hyperparameters	**Search Space**
Learning rate	$[5e^{-5}, 3e^{-4}]$
Batch size	128, 256, 512, 1024
Network hidden layers	(256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Value function loss coefficient	$[0.5, 1.0]$
Entropy coefficient	$[0, 0.01]$

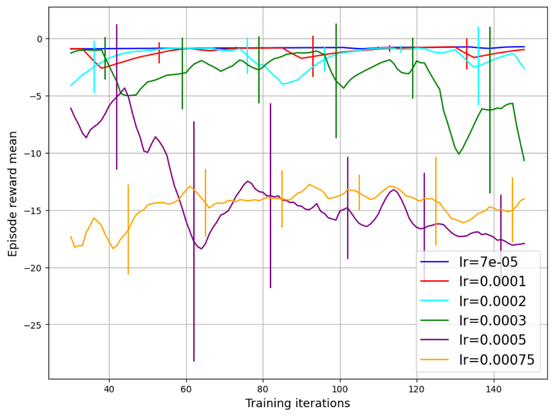
Table 7.16: A2C Economic–Dispatch resample and mutation values.

7.3.10 PPO Economic–Dispatch Sweep–Over Results

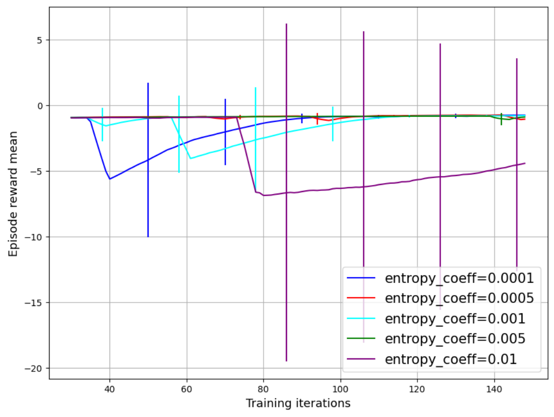
The presented PPO sweep–over Economic–Dispatch results have been sampled according to the approach documented in section 7.1.5. The results of the batch size A.66, minibatch size A.67, SGD iterations A.68, PPO clip parameter ϵ A.69, network hidden layer A.70 and entropy coefficient A.71 showed that these HPs influenced the performance and reproducibility not significantly.

However, it is worth mentioning that the first four mentioned HP results are quite volatile. One explanation could be the fact that all directly influence the gradient update. This leads to the presumption that the baseline HP configuration is suboptimal, which could cause these volatile behaviours. Considering this, the sweep–over value ranges are used.

The LR results shown in Figure 7.12a indicate that large values from 0.0005 to 0.00075 are too large and therefore lead to lower and unstable performances. Only values below 0.0003 reached convergence. Therefore, the LR SS is set to $[5e^{-5}, 0.0003]$.



(a) Learning rate results.



(b) Entropy coefficient results.

Figure 7.12: PPO – Economic–Dispatch – Training – Sweep–over learning rate and entropy coefficient results.

The entropy coefficient results, shown in Figure 7.12b, showed that nearly all values converged. Only the coefficient 0.01 has not reached convergence. Hence, the SS will be set to $[0, 0.005]$.

The resample and mutation values, which will be used in the PBT, are documented in section 7.1.5.

Hyperparameters	**Search Space**
Learning rate	$[5e^{-5}, 3e^{-4}]$
Batch size	1024, 2048, 4096, 8192
Minibatch size	128, 256, 512, 1024
SGD iterations	3, 4, 5, 6, 7, 8, 9, 10
PPO Clip Parameter (ϵ)	$[0.1, 0.3]$
Network hidden layers	(256, 256), (256, 512), (512, 256), (512, 512), (256, 256, 256), (256, 256, 512), (256, 512, 256), (256, 512, 512), (512, 256, 256), (512, 256, 512), (512, 512, 256), (512, 512, 512)
Value function loss coefficient	$[0.5, 1.0]$
Entropy coefficient	$[0, 0.005]$

Table 7.17: PPO Economic–Dispatch resample and mutation values.

7.4 Summery

The task of the sweep–over–phase was to estimate the influence of HPs to estimate the SSs boarders for the following tuning–phase. Some algorithms are surprised with their results, such as the SAC. According to the analysis, its performance mainly depends on

the entropy LR and initial alpha α value. The other SAC HPs did not show significant influences. All other RL algorithms depend on more HPs, what is a significant advantage of SAC.
Further, the results of the DDPG and TD3 are similar, especially for the Q–Market environment, but, usually, the TD3 results were, however, more distinct.
Also worth mentioning is the PPO, which shows in both environments strong fluctuations regarding the batch size, minibatch size, SGD iterations and PPO clip parameter ϵ results. All these HPs more or less directly control the gradient updates, which could be a reason for these observed uncertainties.
Another interesting point is the distribution of hidden layers over the two environments. Two hidden layer networks achieve good performances less often than three hidden layer networks. Especially for the Economic–Dispatch environment, three hidden layer networks were usually more performant. This behaviour is probably caused by the more extensive and complex action space of the Economic-Dispatch environment, which requires more layers.

8 Tuning–Phase Evaluation

After the HP SSs were determined in the sweep–over–phase, the next step, according to the approach in section 5.5, are the HPTs done with the PBT and the following comparison with the baseline HPs.

8.1 Hyperparameter Tuning Result

Every algorithm was tuned in each environment. Further analysis of the RL algorithm behaviours, like a performance, reproducibility and stability discussion, were left out because usually each PBT individual had multiple different HP configurations during training, and so no meaningful interpretation is analysable.

Moreover, as already mentioned in section 2.5.3, one downside of the PBT are the meta–hyperparameter selection. The shown HPs in this chapter are the results of the second PBT. In the first attempt, the meta–hyperparameters have not led to better HPs regarding the baseline HPs. After adjusting the meta–hyperparameters, more performing results have been obtained but not for each algorithm in each environment.
The following tables show the HPs of the second PBT experiment.

DDPG hyperparameters	**Q–Market**	**Economic–Dispatch**
Actor learning rate	0.000162947	0.000116484
Critic learning rate	0.00121511	0.00086942
Actor network	[233, 512, 512, 5]	[201, 256, 256, 256, 42]
Critic network	[233, 256, 256, 1]	[201, 512, 512, 512, 1]
Batch size	1024	1024
Soft Target Network Update (τ)	0.00576962	0.0013558
Gaussian noise STD	0.0319537	0.0416539

Table 8.1: Tuned DDPG hyperparameters.

TD3 hyperparameters	**Q–Market**	**Economic–Dispatch**
Actor learning rate	0.000188169	$6.740772e^{-5}$

Critic learning rate	0.000540143	0.000834612
Actor network	[233, 512, 512, 5]	[201, 512, 512, 512, 42]
Critic network	[233, 512, 256, 512, 1]	[201, 512, 512, 512, 1]
Batch size	1024	1024
Soft Target Network Update (τ)	0.0013521	0.00385487
Gaussian noise STD	0.0241369	0.0021306
Policy delay	3	1

Table 8.2: Tuned TD3 hyperparameters.

SAC hyperparameters	**Q–Market**	**Economic–Dispatch**
Actor learning rate	0.0005687	0.0006326
Critic learning rate	0.00043889	0.0003623
Entropy learning rate	0.00113877	0.0003922
Actor network	[233, 256, 512, 256, 10]	[201, 256, 512, 512, 84]
Critic network	[233, 512, 256, 512, 1]	[201, 512, 512, 512, 1]
Batch size	1024	1024
Soft Target Network Update (τ)	0.0094264	0.730949
Initial alpha α	0.8130879	0.893302

Table 8.3: Tuned SAC hyperparameter.

A2C hyperparameters	**Q–Market**	**Economic–Dispatch**
Learning rate	0.000155195	$1.52517e^{-4}$
Network	[233, 256, 512, 256, 10]	[201, 256, 256, 256, 84]
Batch size	1024	512
Value function loss coefficient	0.418365	0.512158
Entropy coefficient	0.000639831	0.00029668

Table 8.4: Tuned A2C hyperparameters.

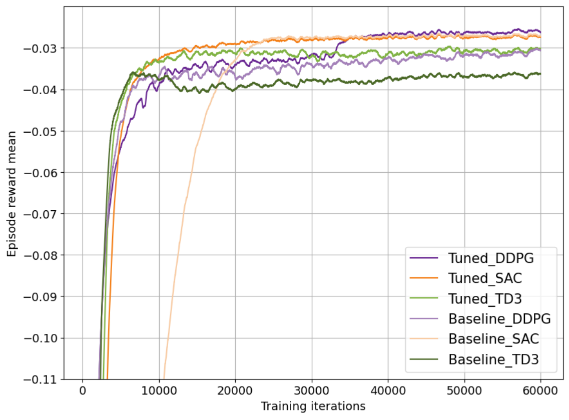
PPO hyperparameters	**Q–Market**	**Economic–Dispatch**
Learning rate	$8.9266e^{-5}$	0.000153459
Network	[233, 256, 256, 256, 10]	[201, 256, 256, 256, 84]
Batch size	8192	8192
Minibatch size	512	1024
SGD iterations	3	4
PPO Clip Parameter (ϵ)	0.148149	0.217686

Value function loss coefficient	0.805692	0.89751
Entropy coefficient	0.00698166	0.000412899

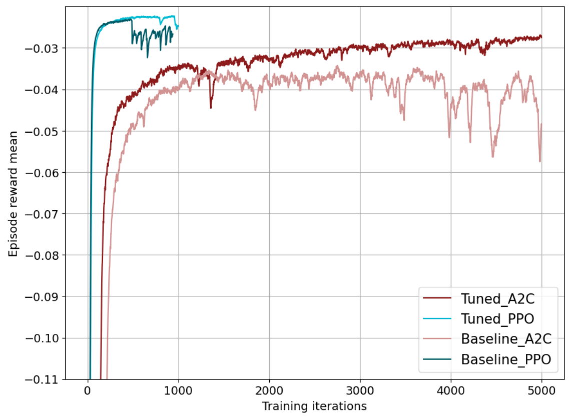
Table 8.5: Tuned PPO hyperparameters.

8.2 Comparison Between Baseline and Tuned Hyperparameters

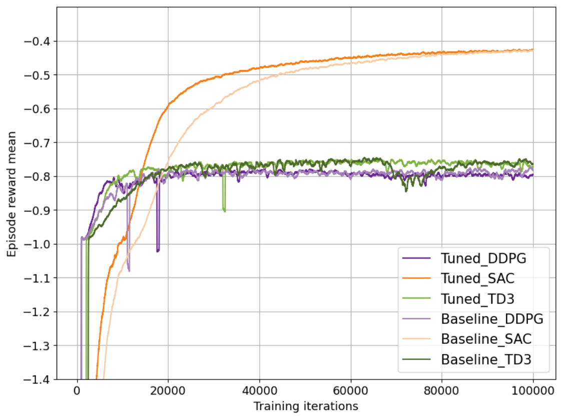
After the tuned HPs for each algorithm in each environment were determined and the baseline and tuned comparison trials are collected, the penultimate comparison follows. According to the approach in section 5.1 and section 5.6, the baseline and tuned results will be compared regarding performance, reproducibility and stability. These metrics are the most important ones.



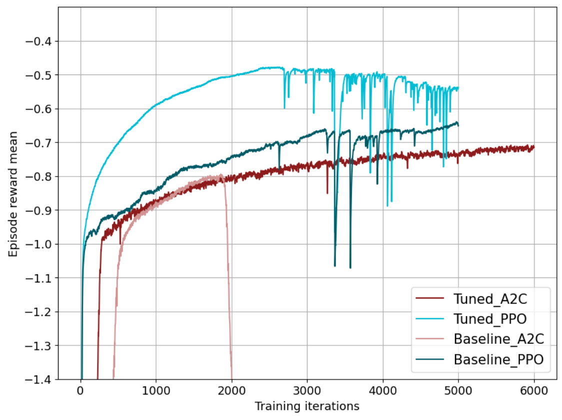
(a) Off–policy Q–Market results.



(b) On–policy Q–Market results.



(c) Off–policy Economic–Dispatch results.



(d) On–policy Economic–Dispatch results.

Figure 8.1: Training comparison results between baseline and tuned hyperparameters.

This procedure ensured that only the best HPs were used for the last comparison, in which only RL algorithms, with optimal HPs, should be compared. Further, this comparison allows seeing the efficiency of the HPT. For each algorithm, the HPT determined better HPs compared to the baseline ones even when the difference between baseline and tuned HPs for the DDPG and TD3 was not large. Next, the identification of the top-performing algorithm versions will be discussed.

Metric	**DDPG**	**TD3**	**SAC**	**A2C**	**PPO**
Baseline Q–Market ERM	−0.0381± 0.0022	−0.0435± 0.0012	−0.0319± 0.0007	−0.0472± 0.016	−0.0358± 0.0047
Tuned Q–Market ERM	−0.0343± 0.0024	−0.0364± 0.0025	−0.0319± 0.0003	−0.0321± 0.001	−0.0318± 0.0031
Baseline Economic–Dispatch ERM	−0.7974± 0.0423	−0.7793± 0.0318	−0.4549± 0.0055	−47.1783± 9.1974	−0.653± 0.0732
Tuned Economic–Dispatch ERM	−0.7893± 0.0278	−0.781± 0.0365	−0.452± 0.0061	−0.7226± 0.0294	−0.5342± 0.0467

Table 8.6: Test comparison results between baseline and tuned hyperparameters.

8.2.1 Q–Market Environment

The tuned DDPG and TD3 are more performant regarding training and testing than their corresponding baseline counterpart, which can be observed in Figure 8.1a and Table 8.6. Both tuned versions of DDPG and TD3 are similarly stable and reproducible to the baseline versions. The tuned DDPG even achieved the best training performance of all algorithms in Figure 8.1a. Therefore, the tuned DDPG and TD3 have the better HPs and will be further used in the comparison–phase.
After a longer convergence time, the SAC showed that the baseline and tuned versions were similarly reproducible and stable. Only negligible differences between both SAC versions could be observed in Figure 8.1a and Table 8.6. The learning velocity is the only difference between both versions, so the tuned SAC will be used in further comparisons.
The A2C and PPO results in Figure 8.1b and Table 8.6 showed that for both algorithms, the tuned HPs have led to more performant, stable and reproducible results. Hence, the tuned versions of the A2C and PPO will participate in the overall last comparison in chapter 9.

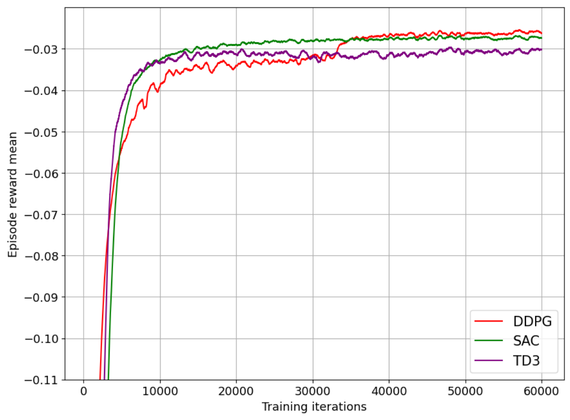
8.2.2 Economic–Dispatch Environment

The tuned DDPG and TD3 in the Economic–Dispatch environment are nearly identical to the baseline versions (see Figure 8.1c). However, the tuned DDPG seems more stable during training and has a slightly better test performance and reproducibility. Hence, the tuned DDPG is used in the comparison–phase.
Differently, determining the top–performing TD3 version is more complex than for the DDPG. The difference between the tuned and baseline TD3 regarding the training and test performance is relatively small. The baseline version reached a slightly better performance and a negligible better reproducibility, as seen in Figure 8.1c and Table 8.6, but it is less stable than the tuned counterpart. During training, the training performance of the baseline version drops for about 10.000 training cycles. The tuned version also had one performance drop, but which was extremely short. Overall, even when the baseline TD3 is slightly more performant and reproducible, the tuned TD3 will participate in the last comparison because of its better stability.
Similarly to the above–mentioned Q–Market results, both SAC versions are highly similar regarding performance, reproducibility and stability. Only the convergence velocity of the tuned SAC was faster, so the tuned SAC version will be used in the comparison–phase.
The tuned A2C is superior to the baseline A2C version in all aspects. The baseline A2C has severe stability problems, which may cause the rapid drop in performance seen in Figure 8.1d. The tuned version did not suffer from these problems and performed a stable training. Therefore, the tuned A2C is the top–performing algorithm version.
Finally, the PPO Economic–Dispatch results show the same picture as those of the SAC. The tuned PPO performs significantly better than the baseline version. It also achieved better test performance and reproducibility. However, both PPOs are having issues with their stability, especially the tuned PPO version drops after reaching the point of maximum performance repeatedly in performance multiple times. Moreover, both PPO versions (baseline and tuned) crashed on some trials due to extensive gradient updates after they had reached their maximum performance. These crashed, which caused uncompleted trials, are not considered because the test results were not obtained. However, this behaviour will influence the decisions in the comparison–phase.

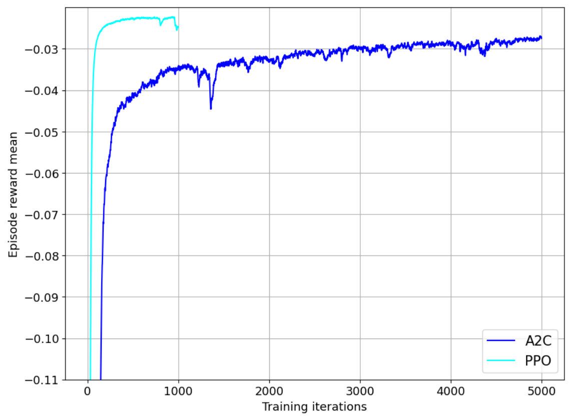
With these top–performing algorithms the last comparison will be performed in the following chapter 9.

9 Comparison–Phase Evaluation

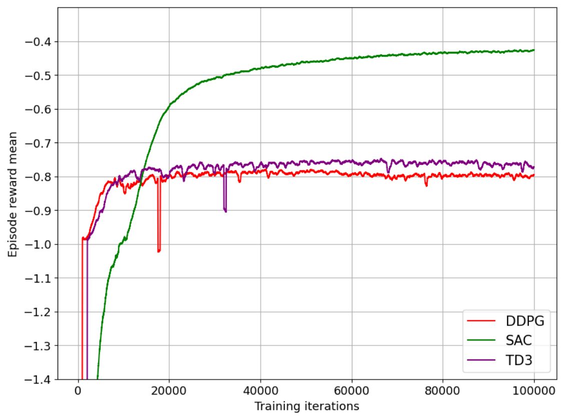
Finally, the results of the comparison–phase are analysed in this chapter. Therefore, the single algorithms are compared regarding performance, reproducibility, stability, sample efficiency and training time. The foundation of this comparison are the visualised training results in Figure 9.1 and test results in Table 9.1. Both are explained in section 5.2.



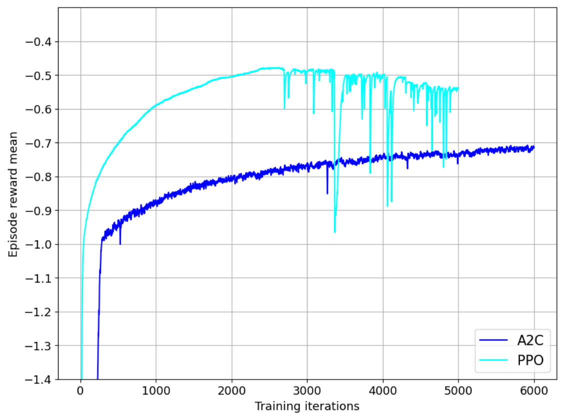
(a) Off–policy Q–Market results.



(b) On–policy Q–Market results.



(c) Off–policy Economic–Dispatch results.



(d) On–policy Economic–Dispatch results.

Figure 9.1: Training episode reward mean comparison results.

Comparing off–policy and on–policy algorithms with each other is often difficult because of the different number of training iterations and sampled episodes from the environment. Therefore, these are separated in Figure 9.1.

Further, because this thesis should provide an overall algorithm recommendation (see section 1.1), the results of both environments are considered together.

At first, some general observations will be discussed. As for the training, some breakthrough points have appeared, which act as barriers. In the Economic–Dispatch environment at an ERM of -0.8, a breakthrough point appeared and the DDPG and TD3 converged at this one. Algorithms with different exploration strategies like the SAC, A2C and PPO were able to pass these boundaries. Therefore, it is possible that the Gaussian noise is maybe not suited to reach optimal performance in larger action spaces.
Moreover, the PPO showed similar training behaviour in both environments. At first, it improves with decreasing speed until the point of maximum performance is reached. Afterwards, the PPO becomes unstable and in further training, it loses performance. Some trials even crashed and stopped after reaching the maximum performance point, due to exploding gradients.
Similarly, the A2C also suffers under these substantial performance drops, which could lead to crashed trial during training. This seems to be a fundamental problem of the on–policy algorithms. Therefore, the training of these should be supervised. According to the results in Figure 9.1, the DDPG, TD3 and SAC, which are off–policy RL algorithms, do not show these uncertainties in their training behaviour.

Next, the evaluation analysis takes place. Therefore, each evaluation criterion is analysed for each RL algorithm.
At first, the training and test performances will be presented in which the SAC and PPO are the top–performers. The results in Table 9.1 show that the PPO was slightly more performant in the Q–Market environment. Due to the PPO's performance fluctuations, less performant test results were obtained because the test data were sampled after the training was finished. However, the PPO is expected to even further outperform the SAC in the Q–Market environment regarding the test performance. On the other hand, the SAC was more performant in the Economic–Dispatch environment according to Table 9.1. Probably, even when the PPO test results were obtained under the maximum training performance, the SAC should outperform the PPO.

The TD3 and A2C underperform in each environment regarding training performance. The same applies to the test results in Table 9.1. The DDPG shows an interesting behaviour because it obtained a higher training performance in the Q–Market environment than the SAC, but not a higher test performance.
The A2C could also convinced with its performance in the Q–Market environment. With more converging time, it may have reached the upper–performance boundary. Unfortunately, the A2C could not perform as well as the PPO in the Economic–Dispatch environment.

The training graph in Figure 9.1d shows the first indications of training slowdown. Maybe with more training time, the A2C would have reached the maximum PPO performance. Nevertheless, due to the slowing down training speed, reaching the maximum PPO performance could probably require much time.

The reproducibility analysis shows interesting results based on the test results in Table 9.1. The SAC had the best reproducibility of all algorithms in each environment. Subsequentially, A2C, DDPG and TD3 showed similar reproducibility values, whereas the A2C is more reproducible in the Q–Market and the DDPG is more reproducible in the Economic–Dispatch environment. However, the DDPG is more stable at lower performance, as seen in Figure 9.1c. The PPO had the worst reproducibility of all algorithms. Further, the PPO crashed on two trials after reaching the maximum performance point in the Economic–Dispatch environment. It seems that the PPO is quite sensible to the network initialization because not all trials failed. The PPO was the only RL algorithm from which trials have crashed in this specific comparison. However, similar behaviours were also observed from the A2C during the first failed PBT and in some undocumented experiments.

Based on the training results in Figure 9.1, the stability analysis shows that the SAC generally learns smoothly and constantly. The PPO acts similar to the SAC as long as the maximum performance point was not exceeded. Afterwards, the performances suffer under solid stability fluctuations. The A2C as well as the DDPG and TD3 are also stable during the training. Only the A2C in the Q–Market environment shows some slight performance fluctuations, which do not influence the overall training process.

Metric	**DDPG**	**TD3**	**SAC**	**A2C**	**PPO**
Q–Market ERM	-0.0343 ± 0.0024	-0.0364 ± 0.0025	-0.0319 ± 0.0003	-0.0321 ± 0.001	-0.0318 ± 0.0031
Economic–Dispatch ERM	-0.7893 ± 0.0278	-0.781 ± 0.0365	-0.452 ± 0.0061	-0.7226 ± 0.0294	-0.5342 ± 0.0467

Table 9.1: Test episode reward mean comparison results.

The following evaluation criterion will be the sample efficiency, which is difficult to measure. Off–policy RL algorithms are usually more sample efficient than on–policy algorithms [8]. Further, a quantitative comparison of sample efficiencies depends on the performance because it usually will be measured in performance (ERM) per sample. Due to the different training lengths caused by the algorithm procedures and environments, a quantitative

winner could not be determined. However, SAC reaches the top–performance with only 480.000 samples (collected samples per training iteration 8 × training iterations 60.000) for the Q–Market and 800.000 samples (8 × 100.000) for the Economic–Dispatch environment and even when the DDPG and TD3 are not as performant as the SAC they also reached considerable performances with less samples.
A2C and PPO are less sample efficient than the already mentioned off–policy algorithms. The PPO needed about 8.028.160 (batch size 8192 × training iterations 980) samples to reach the maximum performance point in the Q–Market environment and 20.480.000 (8192 × 2500) for the Economic–Dispatch. The A2C needed the complete training time to reach the highest performance. Therefore, the A2C took 5.120.000 samples (5000 × 1024) for the Q–Market and 3.072.000 samples (6000 × 512) for the Economic–Dispatch environment.

The last metric to take a look at is the required training time. Likewise the sample efficiency, a quantitative measuring of the algorithm training times lead to meaningless results because both algorithms classes (on– and off–policy) have different advantages and disadvantages. The training itself was the most time–consuming part of the off–policy algorithms because, with each update cycle, only eight new samples are collected from the environment. For the on–policy algorithms, the most time–consuming part was the sampling of new data because, after each update cycle, a new batch needs to be sampled from the environment. Therefore, the required time depends on the available computational resources. The training of off-policy algorithms could benefit from GPUs due to the numerous update cycles involved, which are more efficiently handled by GPU-based computation.
On the other hand, on–policy algorithms do not necessarily require GPUs. More sampler CPUs and rollout workers highly boost the training due to the reduced sampling time. Hence, a meaningful and quantitative analysis cannot be done because only CPUs were available (see section 6.2).
However, a tendency between the algorithms can be observed in Table 9.2. These training times are measured under the available computational resources.

Training time	**Q–Market**	**Economic–Dispatch**
DDPG	3h – 4h	6h – 7h
TD3	4h – 5h	6h – 7h
SAC	6h – 7h	10h – 12h
A2C	5h – 6h	8h – 9h
PPO full	12h – 13h	1d 12h – 1d 14h
PPO max performance	11h – 12h	18h – 19h

Table 9.2: Training times of the different RL algorithms.

Overall, the DDPG and TD3 were the fastest algorithms in this comparison, maybe because these are less complex than, for example, SAC and PPO. A2C, which is in comparison to PPO and SAC, also less complex, took the third place in this competition. Lastly, the slowest algorithms are the SAC followed by the PPO. Even when only taking the time until reaching the point of maximum performance into consideration, the PPO would be the slowest algorithm.

In total, the SAC algorithm is the overall winner of this comparison. It seems to be perfectly suited for achieving top performances, especially for larger environments. Further, the SAC is one of the most sample–efficient algorithms and extremely stable during training. Even under not optimal chosen HPs, the SAC seems to converge with enough training time (see Figure 8.1), which suits it for the use in different OPF problems. The results can be reproduced reliably. The long training time is only a minor downside, which could be handled with more GPUs to increase training time. Based on these characteristics, the SAC can be used in production.

The PPO is not far away in this ranking. It also convinces with good performances during training, especially in the Q–Market environment. However, the training process is only stable until the maximum performance point is reached. Afterwards, the training becomes unstable and unreproducible. Trials may crash in this training section, which is a severe downside. For the PPO, a more complex early–stopping mechanism needs to be used than the usual hardcoded constant training length stopping criterion. A plateau–stopping, which measures the ERM and stops training when the performance decreases, is advised. Further, the training should be actively supervised. Moreover, the PPO results indicate that larger action spaces are probably more challenging for this RL algorithm. The performance reached not the same level as the SAC. Further, the significant sample inefficiency, together with a large amount of needed HPs and the long training time on restricted computational resources, are considerable downsides of the PPO in the context of solving OPF problems. Therefore, the PPO may be a suited choice when many training samples and computational resources are available for environments with a relatively small action space.

The DDPG and TD3 are the least performant algorithms in this comparison, even when the DDPG was able to achieve the best training performance in the Q–Market environment. Especially for larger action spaces, both algorithms could not reach performances over the breakthrough point at an ERM of -0.8, maybe because of the Gaussian noise exploration mechanism. On the other hand, DDPG and TD3 convinced with a good sample efficiency in comparison to the on–policy RL algorithms. Especially, these mentioned aspects together with their fast learning makes them ideal to test new environments or obtain proof–of–concept results. Using these in production is not recommended due to the worse performance regarding the SAC or PPO.

Finally, the A2C takes the last place in this ranking. Even when it performed better than DDPG and TD3, the downsides predominate the advantages. The A2C sample efficiency was not comparable to any other algorithm. Even when the PPO needs more training samples, it reaches at least a good performance. The A2C seem to slowly converge in the Economic–Dispatch environment at a worse performance, which is under the maximum PPO performance.
Further are the training times for the A2C quite large in comparison to the achieved performance.
Another not directly observable reason, why the A2C takes the last place in this ranking, are the already mentioned uncertainties in the training behaviour of the A2C. Many undocumented experiments have shown that the A2C is extremely difficult to control because it sometimes performs disproportionate gradient updates, which destroys the policy permanently (see in Figure 8.1 the baseline A2C). Especially for new environments, it should be challenging to find optimal HPs that lead to a stable performance even when gradient clipping mechanisms are applied. Therefore, the A2C is not recommended for use even when it performs, under certain circumstances, better than DDPG and TD3.

10 | Procedure Evaluation and Discussion

In this chapter, a closer look should be taken at the procedure used in this master thesis. At first, the requirements analysis shows if all self–defined expectations are fulfilled to ensure fundamental scientific correctness. Afterwards, a discussion about the used procedure will reveal optimisation potentials.

10.1 Requirements Analysis

At the beginning of this work, multiple requirements were determined to ensure scientific correctness. In this section, a discussion will show if these requirements have been fulfilled. EFT–01 should ensure that the efforts of the thesis work are appropriate to the available times. According to this, only some selected papers were analysed in chapter 3. Further, restrictions to test only two or three hidden layer networks (see section 5.5) were applied to save computational resources and time.
ENV–01 demands using open–source frameworks developed by experts for the environment. Both environments in this thesis are based on standard and open–source Python libraries and the mlopf[1] repository is accessible to everyone. Further, in the sense of ENV–02, the mlopf repository uses the datasets mentioned in section 5.4, which can be changed at will. With minor adjustments to the environments it is possible to create new OPF situations. Furthermore, the mlopf repository is still being developed to enable new possibilities of performing RL on OPF problems. Further, the commonly used open–source RL framework RLlib (see section 6.1) is used in this thesis, which is still actively maintained and developed, guaranteeing reproducibility as demanded in AGT–01. All further details about the algorithm and environment configuration are presented in chapter 5 (AGT–01 and AGT–02). Moreover, nearly all comparison requirements have been fulfilled. According to COMP–01, two different environments were used for the comparison. Further, all results have been sampled at least five times, which corresponds to COMP–02 (see section 5.1). A HPT, as described in section 5.1, was done through the PBT, for which all details are documented in section 5.6, fulfilling COMP–03. Based on COMP–05, an evaluation was done in the comparison–phase considering the proposed evaluation criteria (see chapter 9). The only requirement, which was not entirely fulfilled was COMP–04 because it demands

[1] `https://gitlab.com/thomaswolgast/mlopf`

measuring the performance with the ERM and VSM. However, during the evaluation, it turned out that the VSM is not suited to measure the performance because the ERM and VSM are showing significant differences, which result from a non–optimal formulated reward function. Therefore, the VSM was not taken into consideration. All details about this anomaly are documented in section 7.2.

10.2 Procedure Discussion

This section's discussion about the procedure used should reveal aspects that could perform better in the future work.
The sweep–over–phase was the first of three phases proposed to prepare the determination of optimal HPs. During the analysis, it became evident that it is helpful to reduce the SS because the HPT is extremely resource–consuming. A larger SS could lead to worse HPs or longer HPTs. However, for the future work, the sweep–over runs should be much larger than in this thesis. Some HP influences have appeared in the later training states and so these were not observable in the sweep–over runs, for example, the maximum performance point of the PPO or the A2C instabilities. Also, is it questionable if the sweep–over runs are necessary for another attempt. Experts who have much experience with the different algorithms may already have expectations of where the best HP spaces are. The SAC analysis showed further that this top–performing algorithm is not very sensitive to HP, changes except for the entropy LR and initial alpha α value. Even the SAC baseline HPs performed quite well compared to the tuned HPs. Hence, without big HP changes, it could be used on different OPF problems. Therefore, another rerun of the sweep–over–phase should not be required, except for OPF problems, which differ significantly from those in this thesis.
In the tuning–phase, optimal HPs should be determined to compare top–performing RL algorithms afterwards. Therefore, the PBT was chosen because of its performant results on other RL problems. Retrospectively, a series of PBT tests would have made sense. The first HPT with the PBT failed due to non–optimally chosen meta–hyperparameters (see chapter 8). In the second, only HPs were determined with a slightly better performance. Probably, because the baseline HPs already lead to top-performances.
Moreover, the PBT HPs seemed to be more suited for the later learning stages than for the earlier ones. The PBT algorithm performs the HP optimisations onto already optimised parameters from early perturbation cycles. Due to less perturbation cycles, a local optima optimisation has occurred, which probably was the reason for the first PBT fail. Therefore, the population size must be large enough and widely spread to be close to an optimum in the HP space. Further, the number of perturbation cycles could be carefully increased to process more HP configurations (see section 2.5.3). Hence, in future work, a larger population size and more perturbation cycles should be used.

11 Conclusion

This chapter's retrospective will show that the proposed procedure has solved the research question. Subsequently, further open questions and interesting aspects will be mentioned in an outlook.

11.1 Research Question Discussion

This section discusses whether the research question has been answered. At the beginning, after a brief introduction with a fundamental motivation 1, the following research question was formulated:

How to develop a procedure to select, tune and compare RL algorithms to determine optimal algorithms for OPF problems?

To answer this question, first the basis of the Optimal Power Flow (OPF) problem 2.1, Reinforcement Learning (RL) 2.2 and Hyperparameter Tuning (HPT) 2.5 were summarised. Afterwards, a literature analysis in chapter 3 has shown that often non state–of–the-art RL algorithms were used to solve OPF problems. Moreover, sometimes, the documentation of the procedure was often left out. These analysis results show the importance of the proposed research question.
To guarantee that this procedure for determining optimal RL algorithms is built upon a scientific foundation, requirements were defined in chapter 4, ensuring the applicability on different OPF problems, reproducibility and measurement correctness.
A procedure inspired by literature was subsequently developed with three phases (see section 5.1). In the first phase, the influence of each HP for each algorithm in each environment was determined, as seen in chapter 7. Afterwards, a HPT with the state–of–the-art PBT algorithm was performed to obtain optimal HPs (see chapter 8). Then, after validating that the tuned HPs are leading to more performant results (see section 8.2), the last comparison was done. All algorithms were compared with each other in chapter 9 under the defined evaluation criteria (see section 5.2). The SAC and PPO went out of the comparison as the most performant RL algorithms, whereas the PPO was more performant in the Q–Market environment with a smaller action space. On the other hand, the SAC was more performant

in the Economic–Dispatch environment with a larger action space.
Through the following procedure evaluation and discussion in chapter 10, the fundamental scientific correctness has been verified (see section 10.1) by a requirements analysis. A further discussion about the proposed procedure in section 10.2 revealed smaller optimisation potentials, which can be applied in further research.

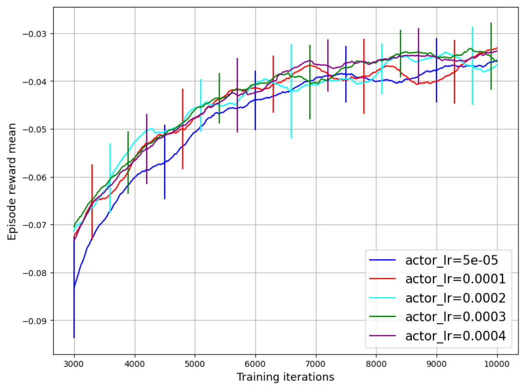
All in all, a procedure was developed, which selected, tuned and compared different RL algorithms to determine the most performant and recommendable algorithms for further research.

11.2 Further Research

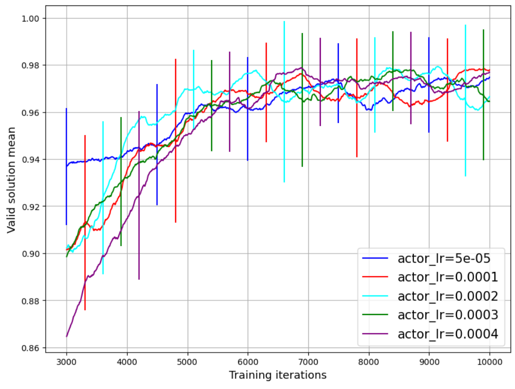
As the last part of the master thesis, an outlook will be provided with further interesting research questions according to the collected experiences during the thesis creation.
One obvious aspect that could be done in further research is to apply the procedure on more and different environments. Only two OPF problem environments have been examined in this work. It would be interesting to see if the procedure with the mentioned enhancements in section 10.2 would maybe lead to different algorithm decisions as in this work. Probably, the PPO results become worse in even larger OPF problems compared to the Economic–Dispatch OPF problem.
During the sweep–over evaluation, it turned out that the reward function leads to different results regarding the ERM and VSM (see section 7.2). After the reward function is optimally formulated, it would be interesting to see if the top–performing algorithms, SAC and PPO, also reach good results regarding the VSM. Probably, the ERM will decrease because committing smaller constraint violations would not result in a larger ERM any more.
Also interesting aspects for further research are the number and kind of layers used in the RL algorithms. Even when some undocumented results implied that these do not influence the performance significantly, it would be worth a look to perform some experiments with more layers and neurons per layer. Additionally, dropout and norm-layer could be applied to see it the test performances can be increased even further.
Another interesting point would be a further comparison of HPT algorithms regarding the OPF problem one–step environments. Maybe a different HPT algorithm, like random search section 2.5.2, would have performed better than PBT.
Also, would it be quite interesting to see if the IMPALA RL algorithm [31], which is based on A2C, may be more performant than its predecessor. Generally, it would be interesting to apply the proposed procedure on other RL algorithms or derivates of already used ones.

A Appendix – Sweep–Over–Phase

A.1 DDPG Q–Market Sweep–Over Results

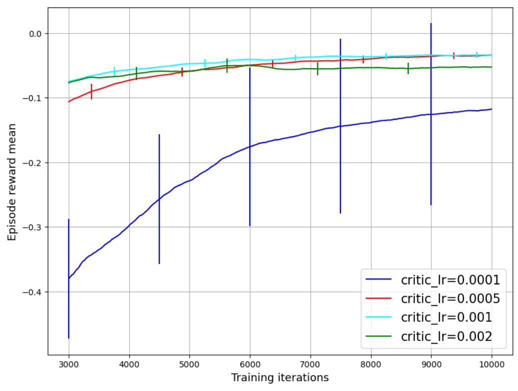


(a) Episode reward mean

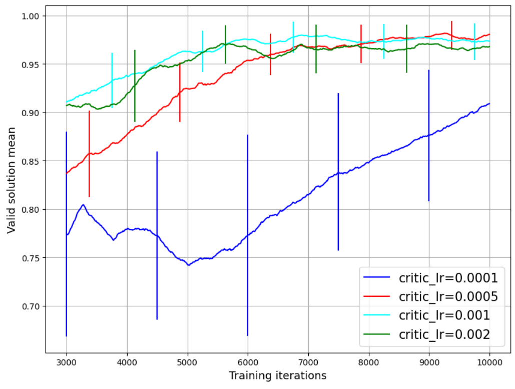


(b) Valid solution mean

Figure A.1: DDPG – Q–Market – Training – Sweep–over actor LR results.

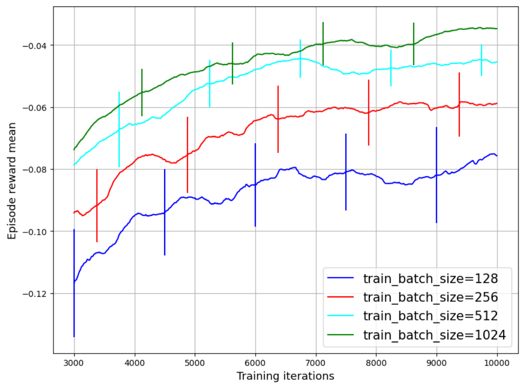


(a) Episode reward mean

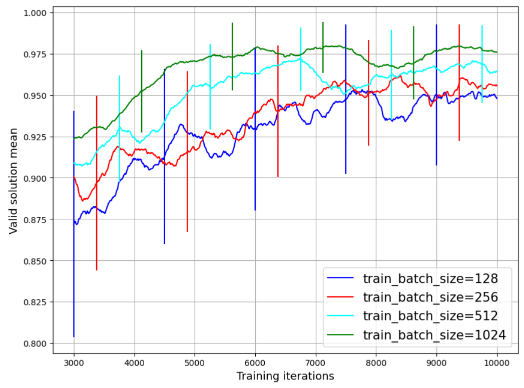


(b) Valid solution mean

Figure A.2: DDPG – Q–Market – Training – Sweep–over critic LR results.

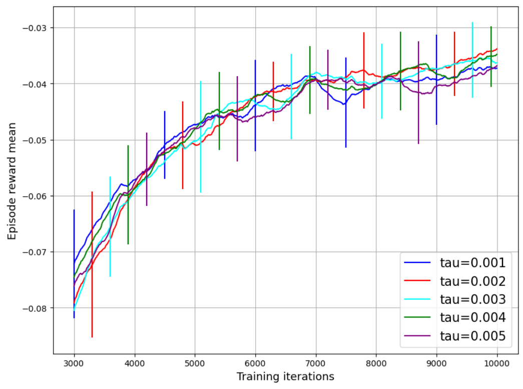


(a) Episode reward mean

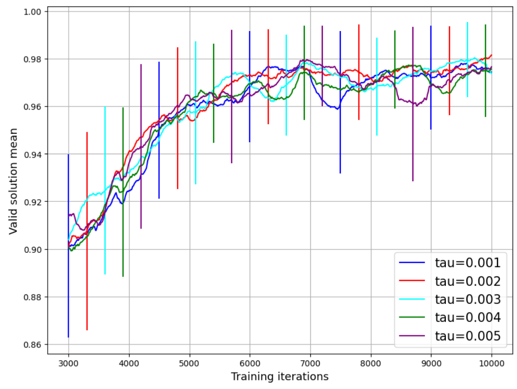


(b) Valid solution mean

Figure A.3: DDPG – Q–Market – Training – Sweep–over batch size results.

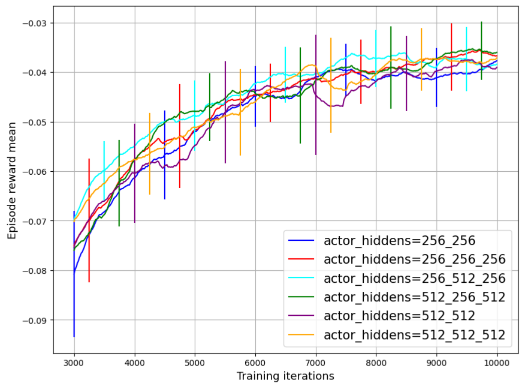


(a) Episode reward mean

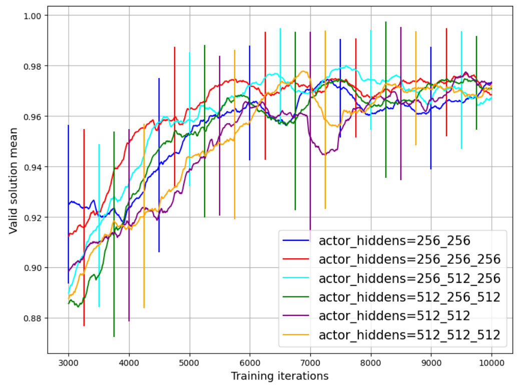


(b) Valid solution mean

Figure A.4: DDPG – Q–Market – Training – Sweep–over tau τ results.

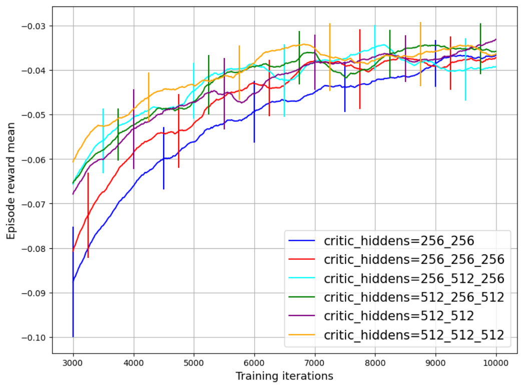


(a) Episode reward mean

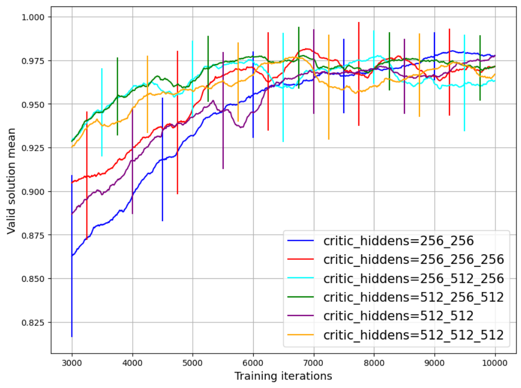


(b) Valid solution mean

Figure A.5: DDPG – Q–Market – Training – Sweep–over actor network hidden layer results.

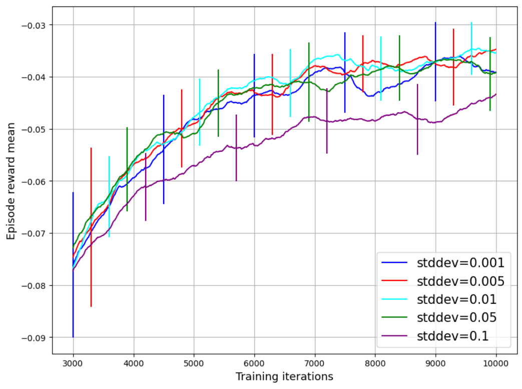


(a) Episode reward mean

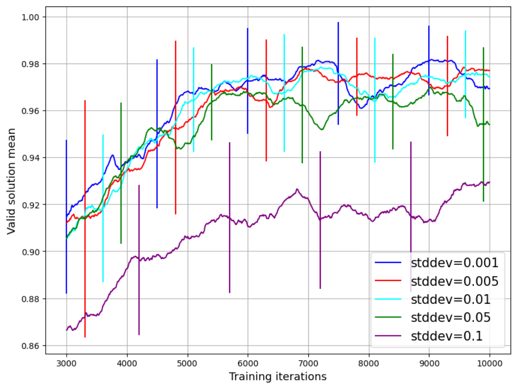


(b) Valid solution mean

Figure A.6: DDPG – Q–Market – Training – Sweep–over critic network hidden layer results.



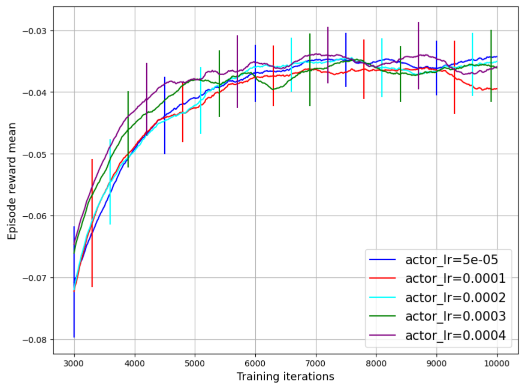
(a) Episode reward mean



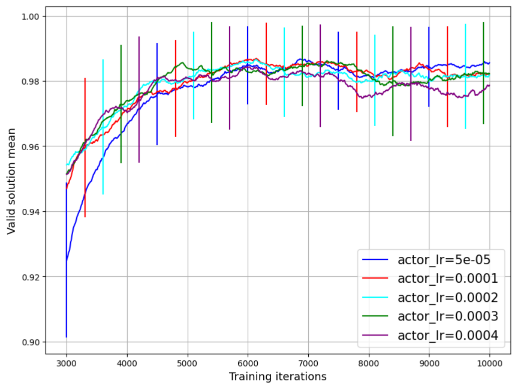
(b) Valid solution mean

Figure A.7: DDPG – Q–Market – Training – Sweep–over Gaussian noise STD results.

A.2 TD3 Q–Market Sweep–Over Results

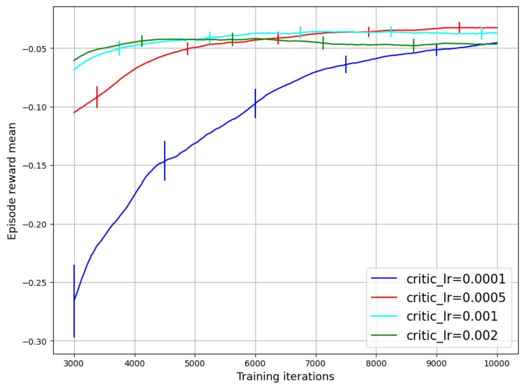


(a) Episode reward mean

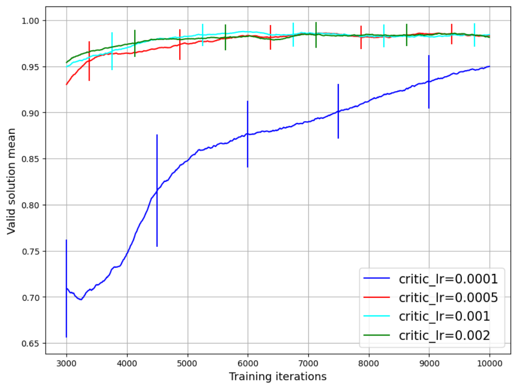


(b) Valid solution mean

Figure A.8: TD3 – Q–Market – Training – Sweep–over actor LR results.

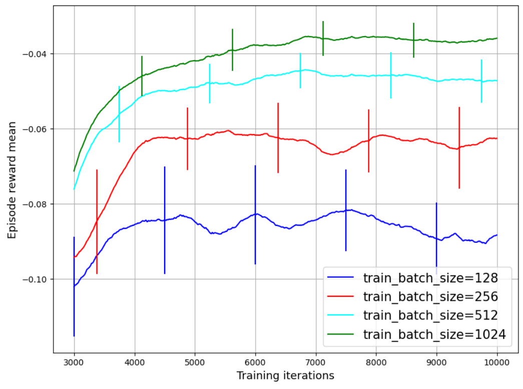


(a) Episode reward mean

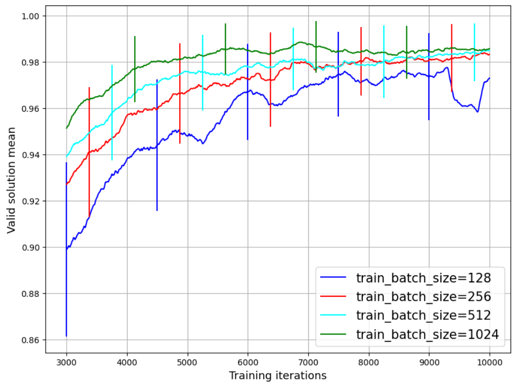


(b) Valid solution mean

Figure A.9: TD3 – Q–Market – Training – Sweep–over critic LR results.

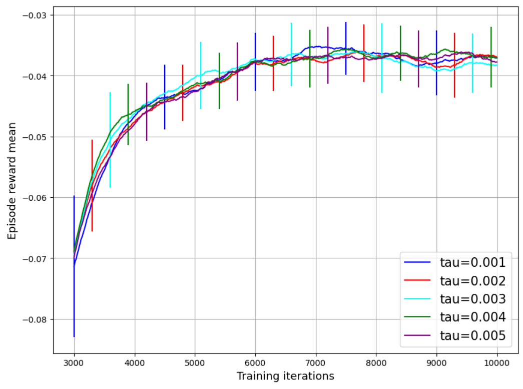


(a) Episode reward mean

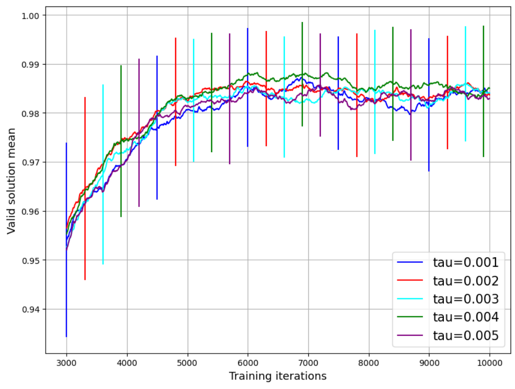


(b) Valid solution mean

Figure A.10: TD3 – Q–Market – Training – Sweep–over batch size results.

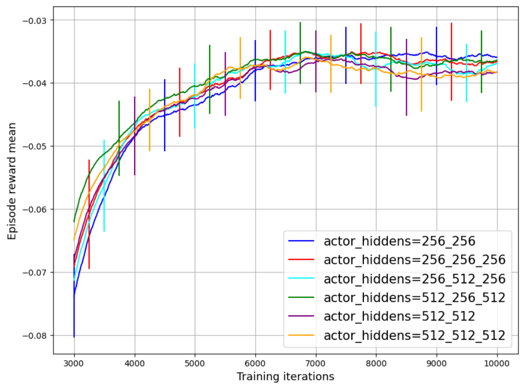


(a) Episode reward mean

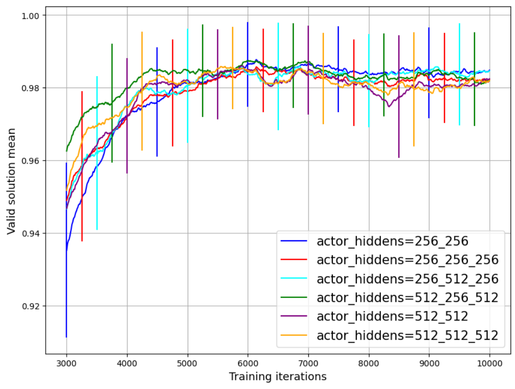


(b) Valid solution mean

Figure A.11: TD3 – Q–Market – Training – Sweep–over tau τ results.

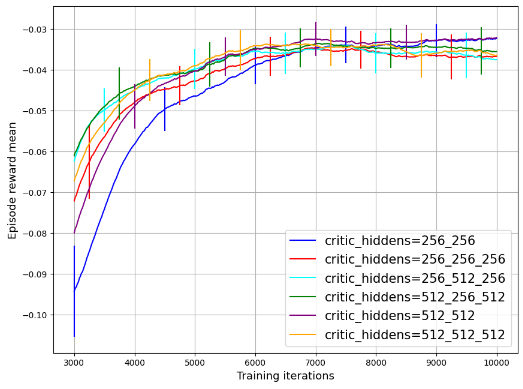


(a) Episode reward mean

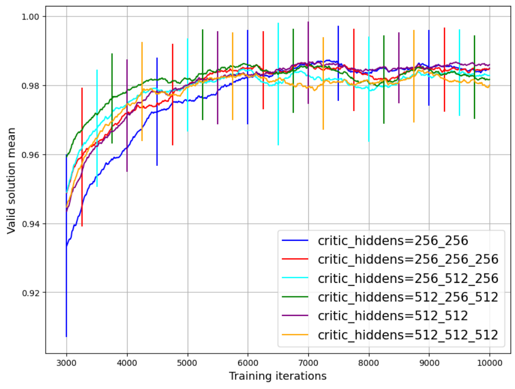


(b) Valid solution mean

Figure A.12: TD3 – Q–Market – Training – Sweep–over actor network hidden layer results.

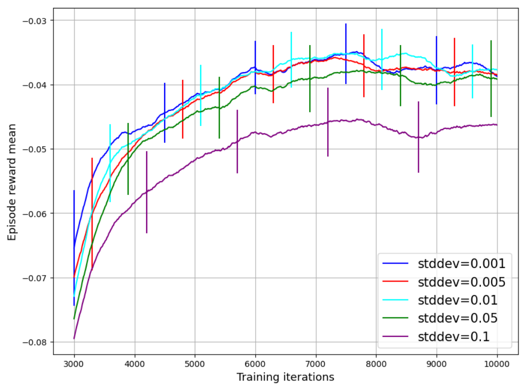


(a) Episode reward mean

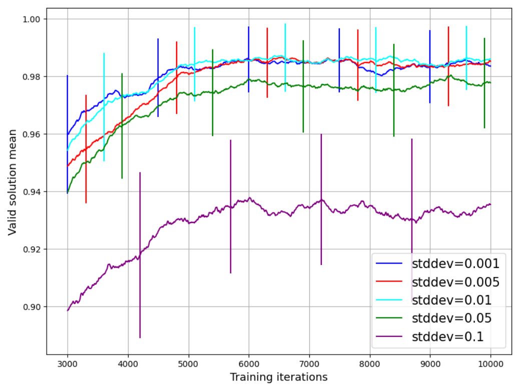


(b) Valid solution mean

Figure A.13: TD3 – Q–Market – Training – Sweep–over critic network hidden layer results.

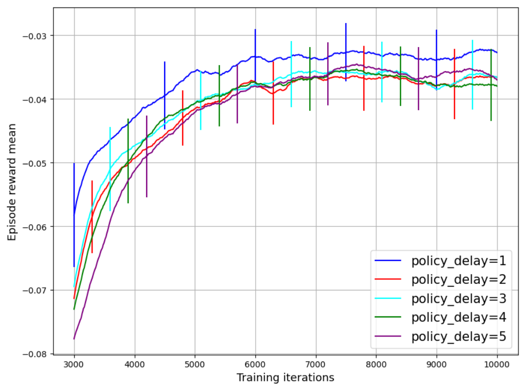


(a) Episode reward mean

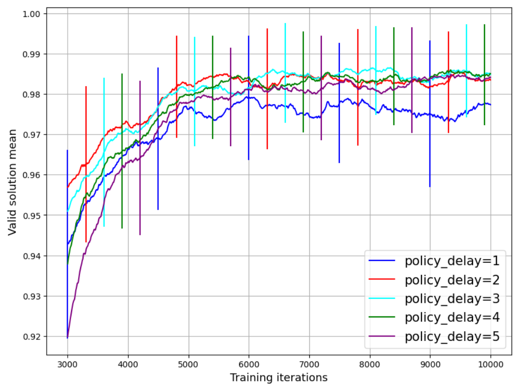


(b) Valid solution mean

Figure A.14: TD3 – Q–Market – Training – Sweep–over Gaussian noise STD results.



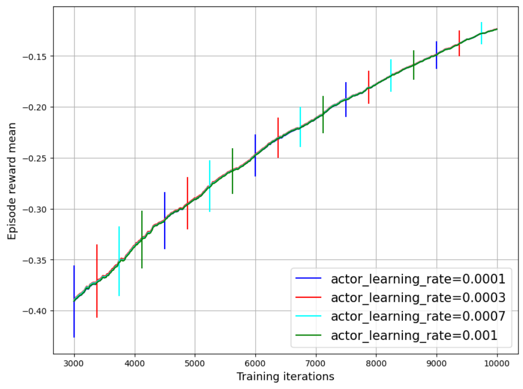
(a) Episode reward mean



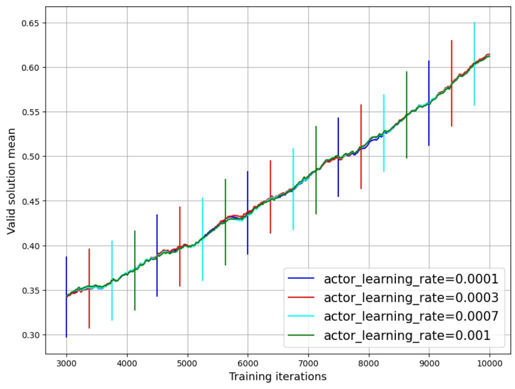
(b) Valid solution mean

Figure A.15: TD3 – Q–Market – Training – Sweep–over policy delay results.

A.3 SAC Q–Market Sweep–Over Results

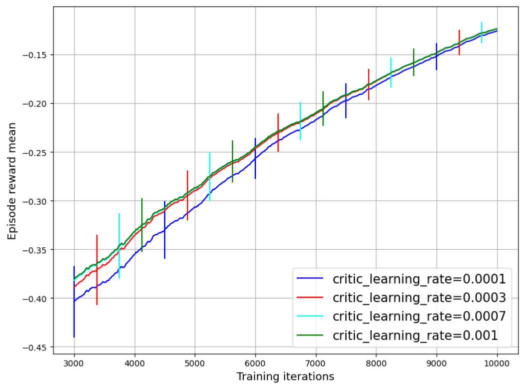


(a) Episode reward mean

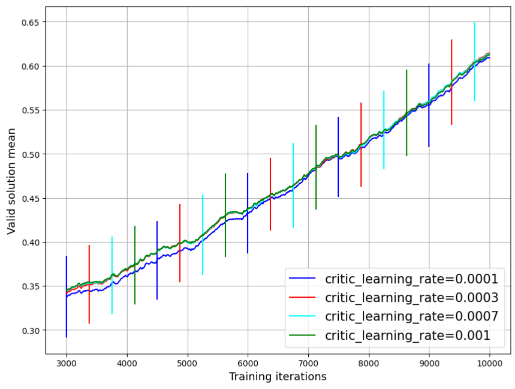


(b) Valid solution mean

Figure A.16: SAC – Q–Market – Training – Sweep–over actor LR results.

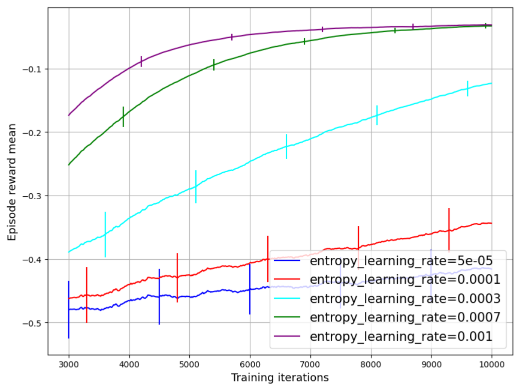


(a) Episode reward mean

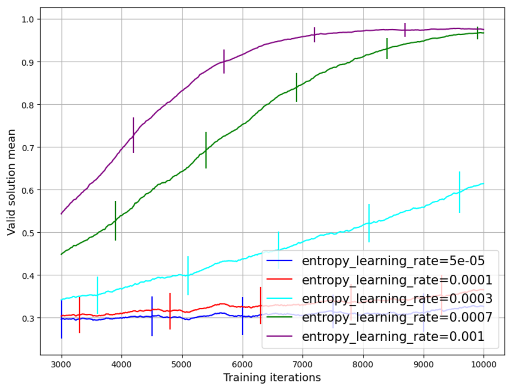


(b) Valid solution mean

Figure A.17: SAC – Q–Market – Training – Sweep–over critic LR results.

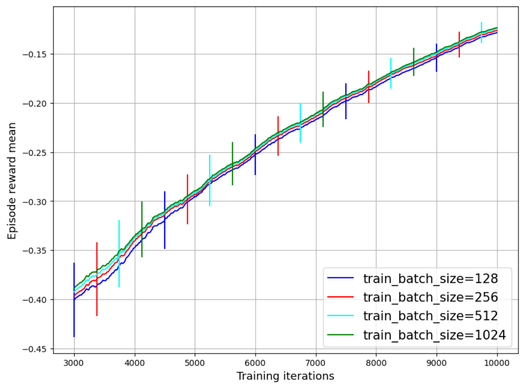


(a) Episode reward mean

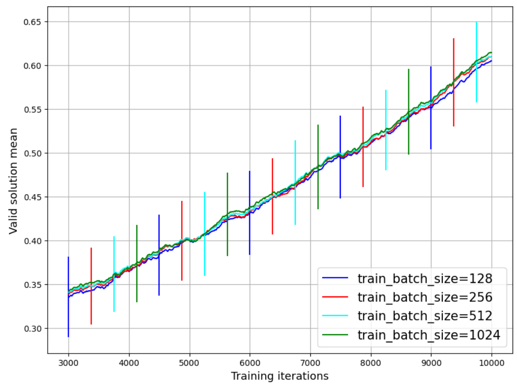


(b) Valid solution mean

Figure A.18: SAC – Q–Market – Training – Sweep–over entropy LR results.

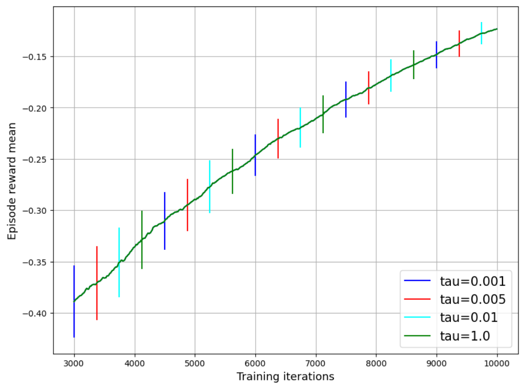


(a) Episode reward mean

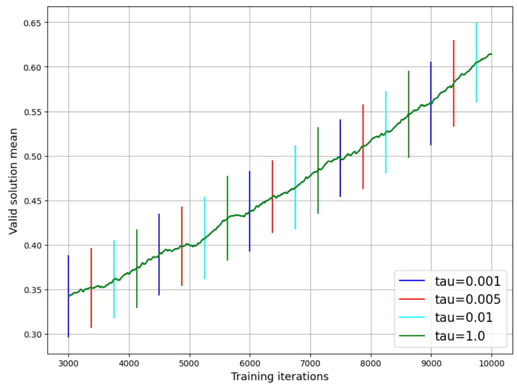


(b) Valid solution mean

Figure A.19: SAC – Q–Market – Training – Sweep–over batch size results.

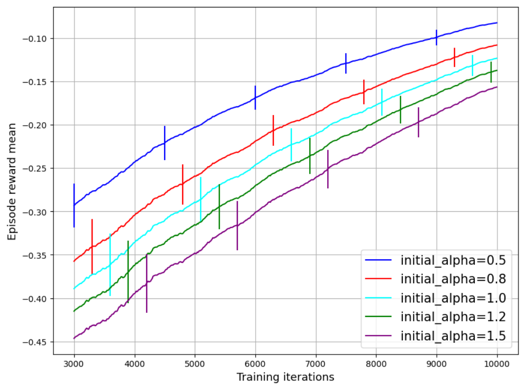


(a) Episode reward mean

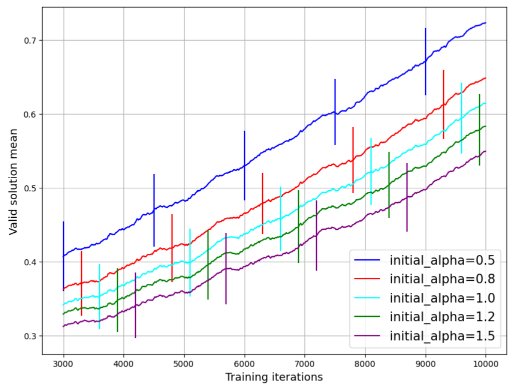


(b) Valid solution mean

Figure A.20: SAC – Q–Market – Training – Sweep–over tau τ results.

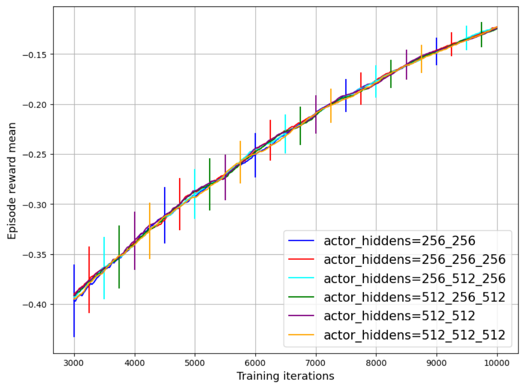


(a) Episode reward mean

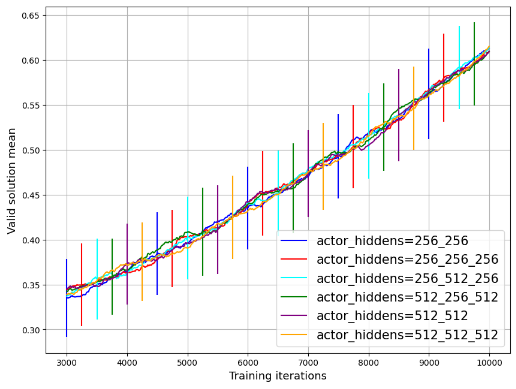


(b) Valid solution mean

Figure A.21: SAC – Q–Market – Training – Sweep–over initial alpha value results.

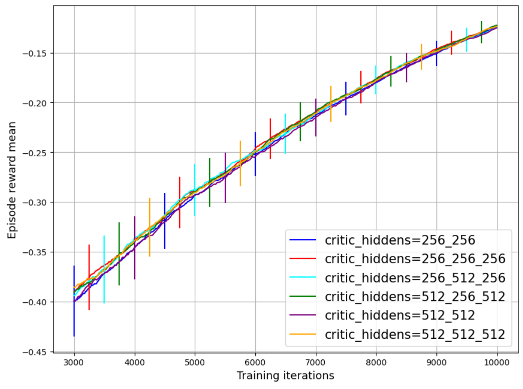


(a) Episode reward mean

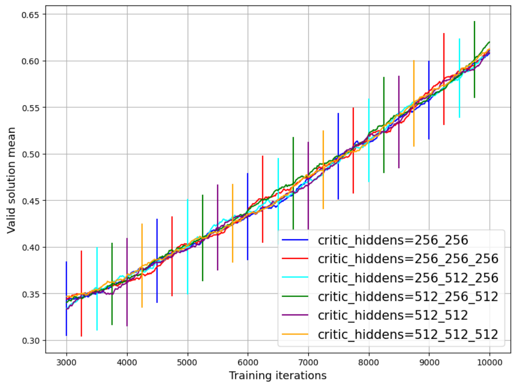


(b) Valid solution mean

Figure A.22: SAC – Q–Market – Training – Sweep–over actor network hidden layer results.



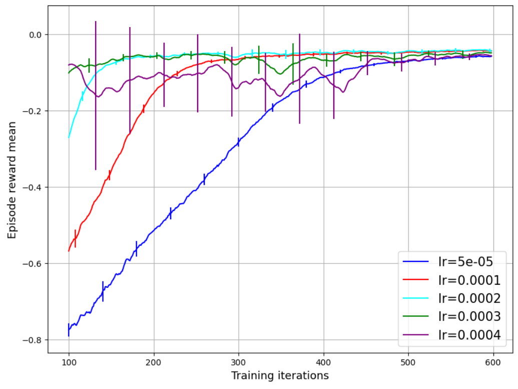
(a) Episode reward mean



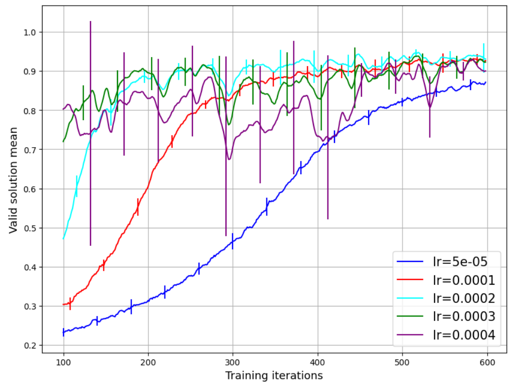
(b) Valid solution mean

Figure A.23: SAC – Q–Market – Training – Sweep–over critic network hidden layer results.

A.4 A2C Q–Market Sweep–Over Results

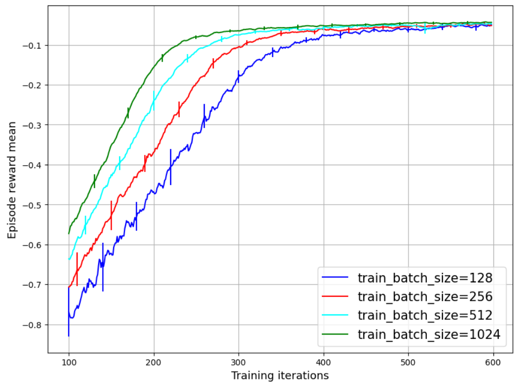


(a) Episode reward mean

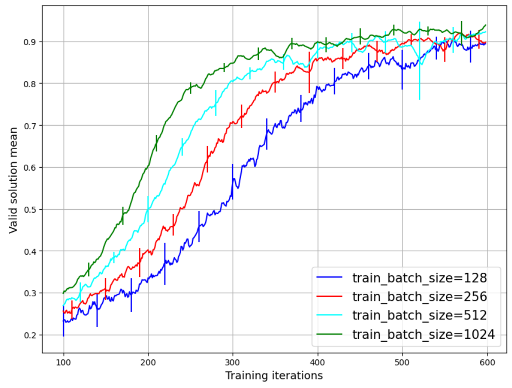


(b) Valid solution mean

Figure A.24: A2C – Q–Market – Training – Sweep–over LR results.

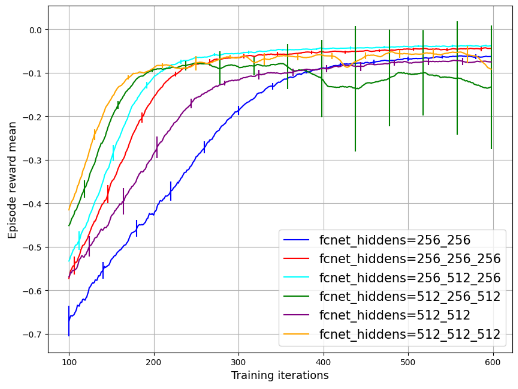


(a) Episode reward mean

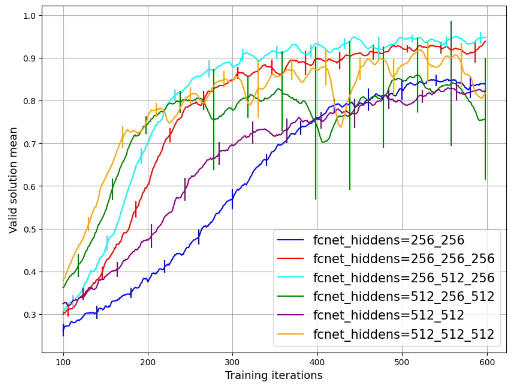


(b) Valid solution mean

Figure A.25: A2C – Q–Market – Training – Sweep–over batch size results.

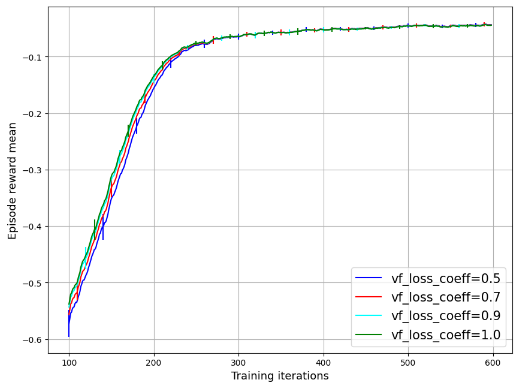


(a) Episode reward mean

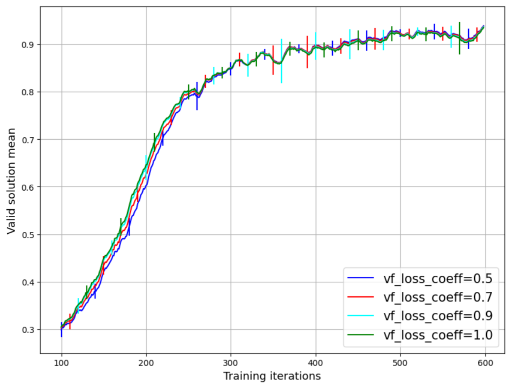


(b) Valid solution mean

Figure A.26: A2C – Q–Market – Training – Sweep–over network hidden layer results.

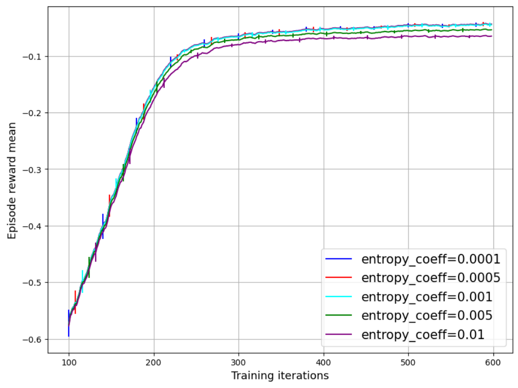


(a) Episode reward mean

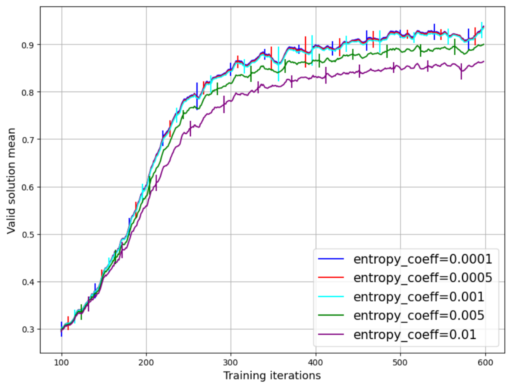


(b) Valid solution mean

Figure A.27: A2C – Q–Market – Training – Sweep–over value function loss coefficient results.



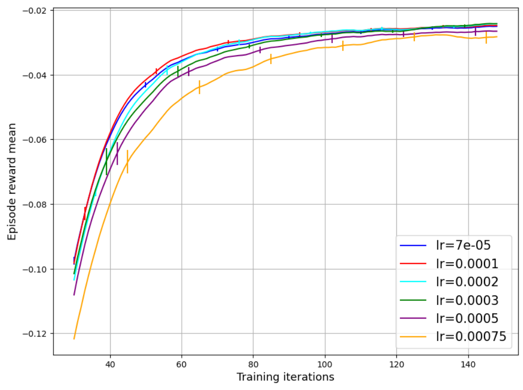
(a) Episode reward mean



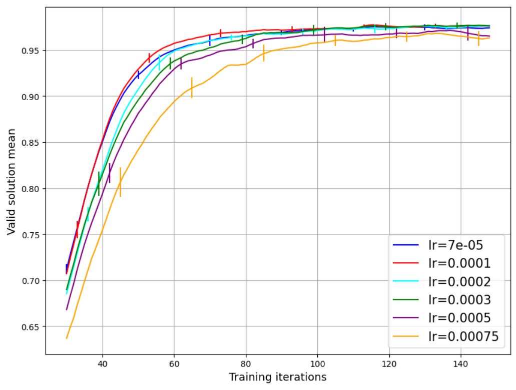
(b) Valid solution mean

Figure A.28: A2C – Q–Market – Training – Sweep–over entropy coefficient results.

A.5 PPO Q–Market Sweep–Over Results

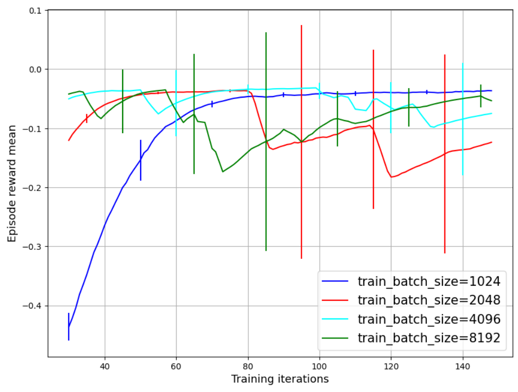


(a) Episode reward mean

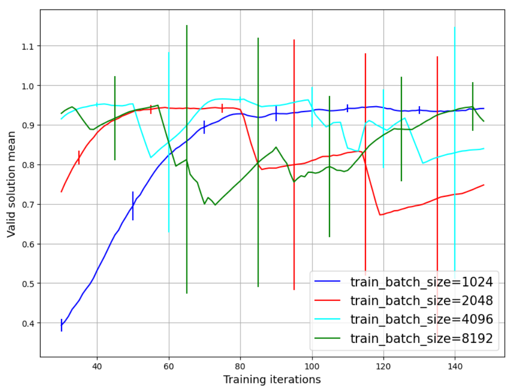


(b) Valid solution mean

Figure A.29: PPO – Q–Market – Training – Sweep–over LR results.

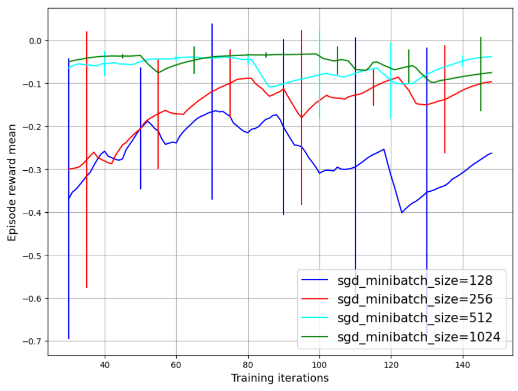


(a) Episode reward mean

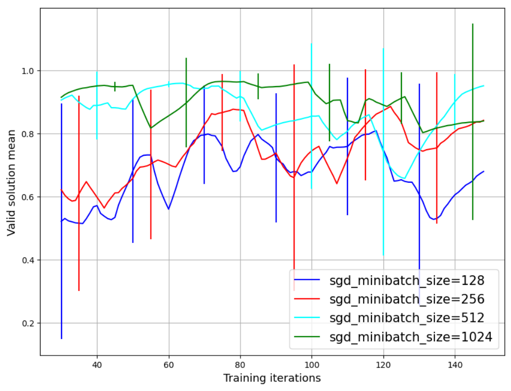


(b) Valid solution mean

Figure A.30: PPO – Q–Market – Training – Sweep–over batch size results.

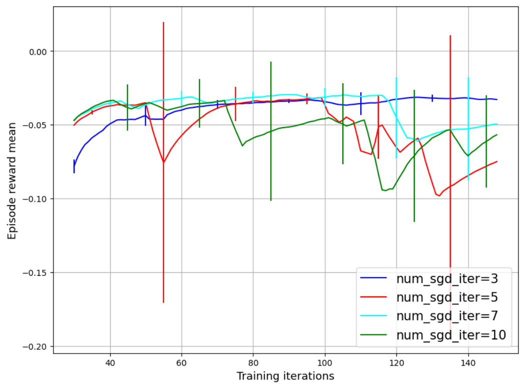


(a) Episode reward mean

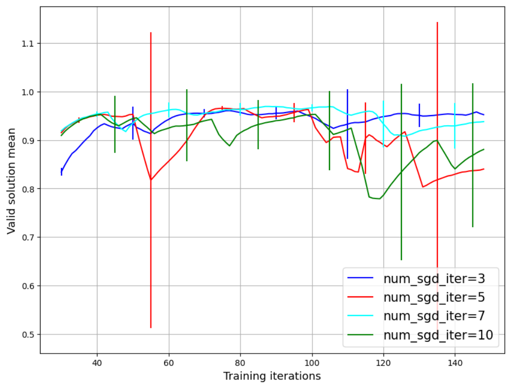


(b) Valid solution mean

Figure A.31: PPO – Q–Market – Training – Sweep–over minibatch size results.

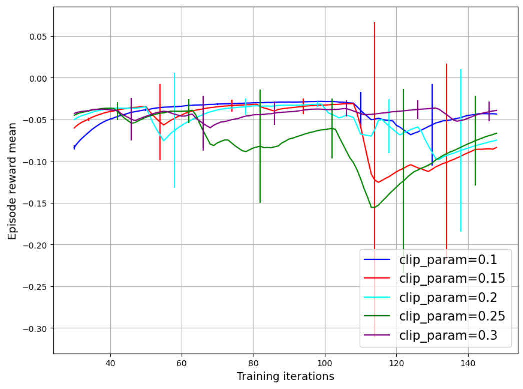


(a) Episode reward mean

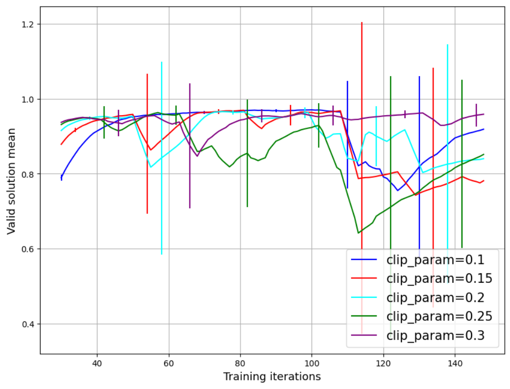


(b) Valid solution mean

Figure A.32: PPO – Q–Market – Training – Sweep–over SGD iteration results.

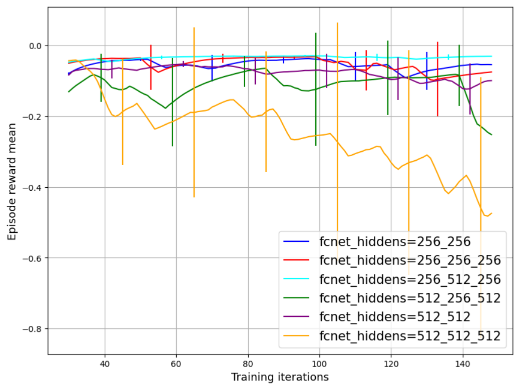


(a) Episode reward mean

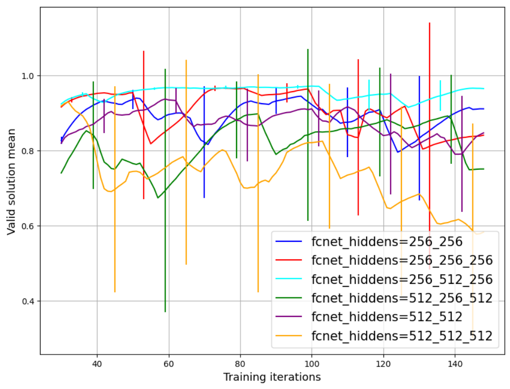


(b) Valid solution mean

Figure A.33: PPO – Q–Market – Training – Sweep–over PPO clip results.

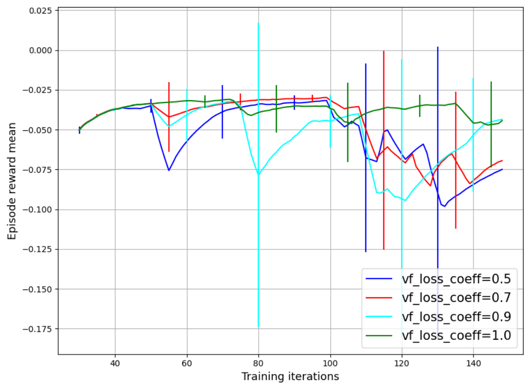


(a) Episode reward mean

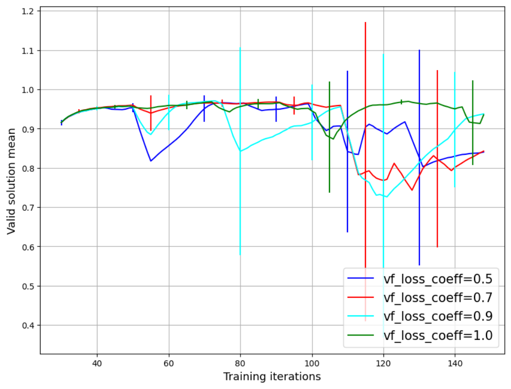


(b) Valid solution mean

Figure A.34: PPO – Q–Market – Training – Sweep–over network hidden layers results.

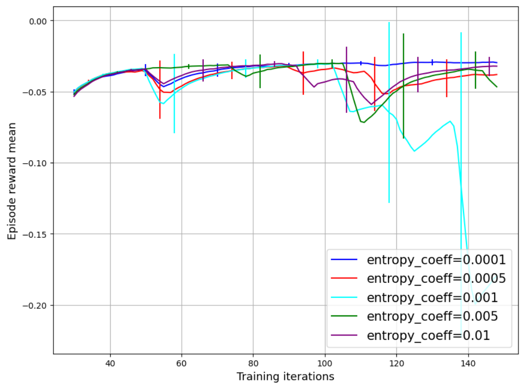


(a) Episode reward mean

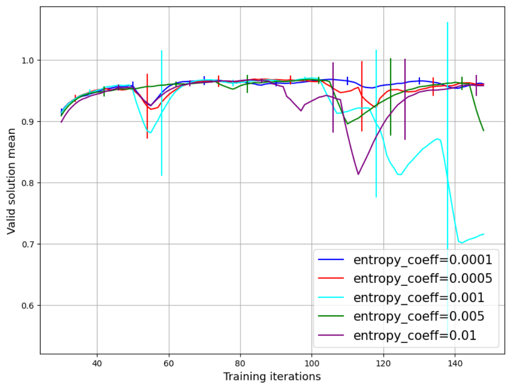


(b) Valid solution mean

Figure A.35: PPO – Q–Market – Training – Sweep–over value function loss coefficient results.



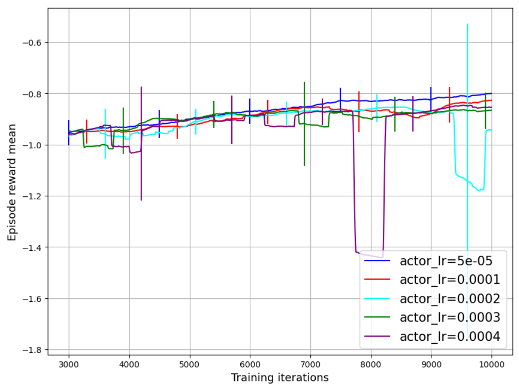
(a) Episode reward mean



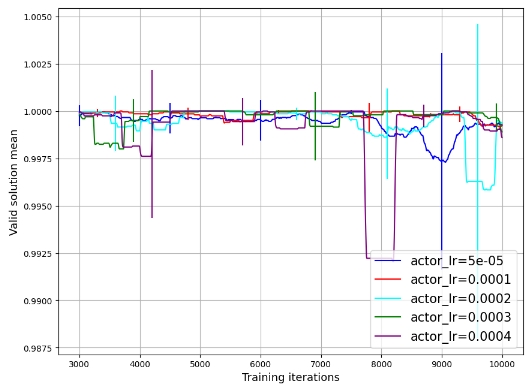
(b) Valid solution mean

Figure A.36: PPO – Q–Market – Training – Sweep–over entropy coefficient results.

A.6 DDPG Economic–Dispatch Sweep–Over Results

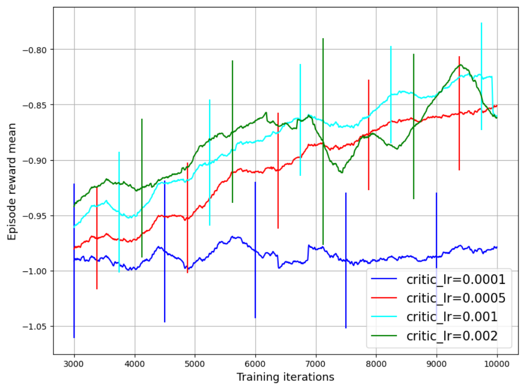


(a) Episode reward mean

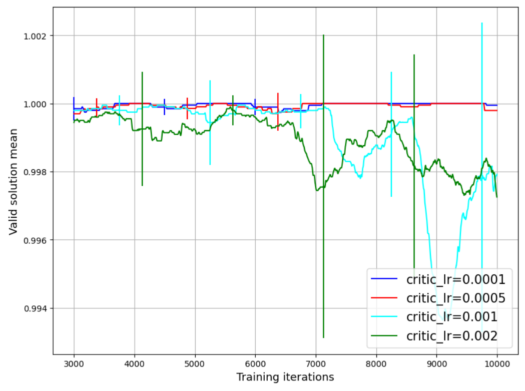


(b) Valid solution mean

Figure A.37: DDPG – Economic–Dispatch – Training – Sweep–over actor LR results.

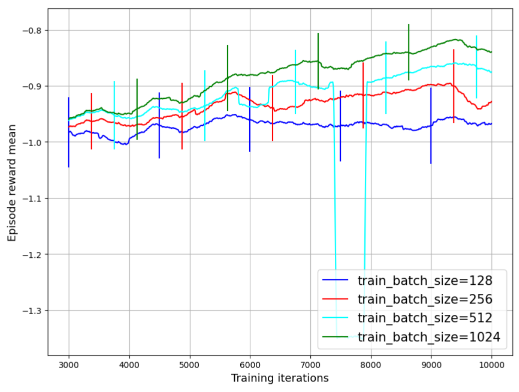


(a) Episode reward mean

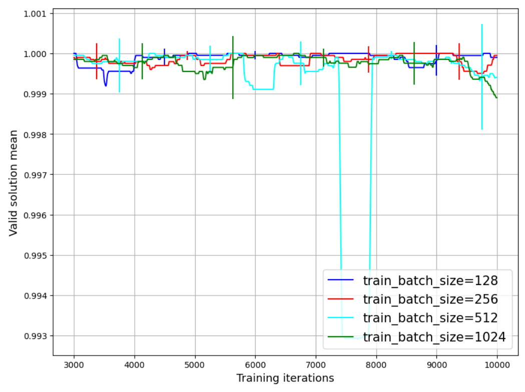


(b) Valid solution mean

Figure A.38: DDPG – Economic–Dispatch – Training – Sweep–over critic LR results.

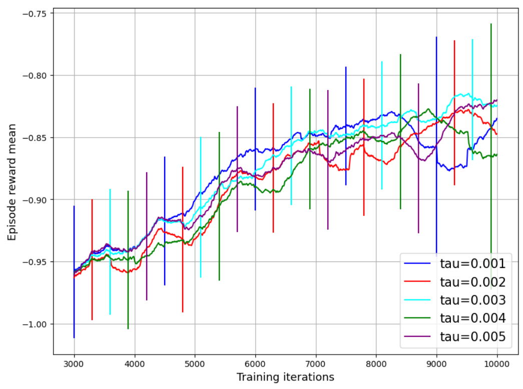


(a) Episode reward mean

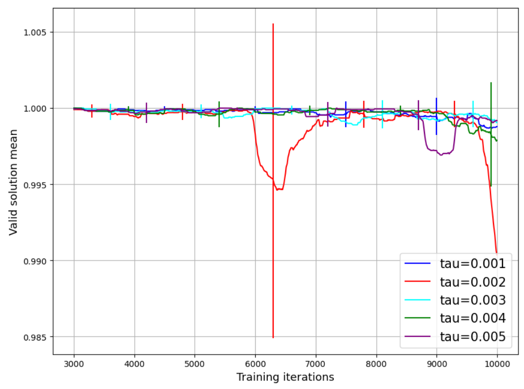


(b) Valid solution mean

Figure A.39: DDPG – Economic–Dispatch – Training – Sweep–over batch size results.

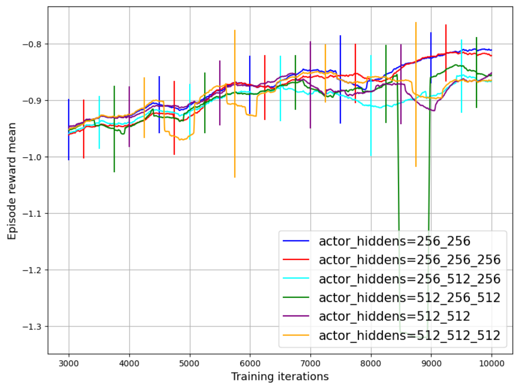


(a) Episode reward mean

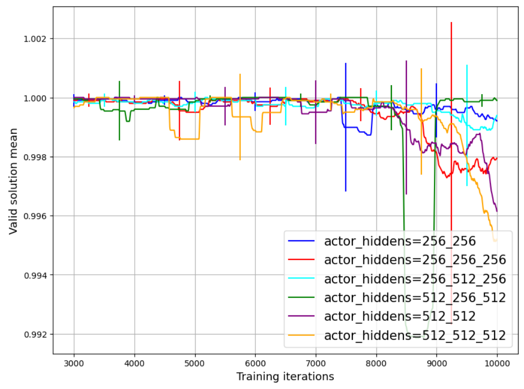


(b) Valid solution mean

Figure A.40: DDPG – Economic–Dispatch – Training – Sweep–over tau τ results.

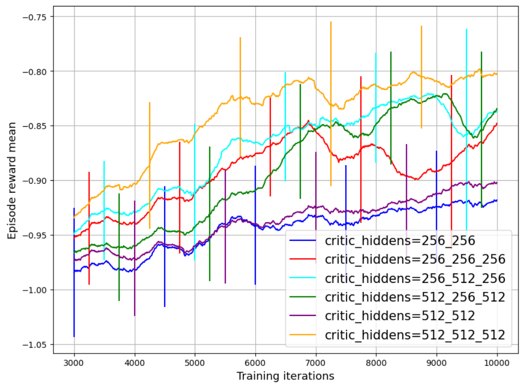


(a) Episode reward mean

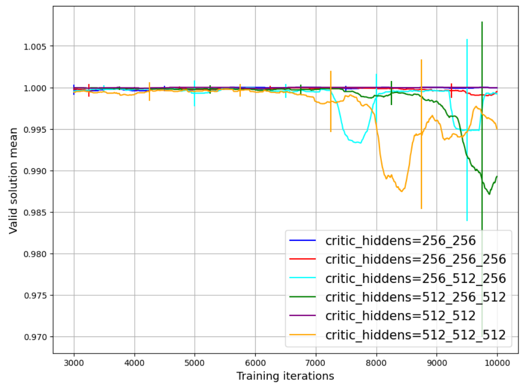


(b) Valid solution mean

Figure A.41: DDPG – Economic–Dispatch – Training – Sweep–over actor network hidden layer results.

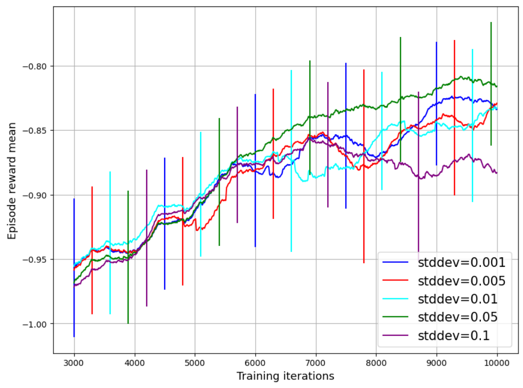


(a) Episode reward mean

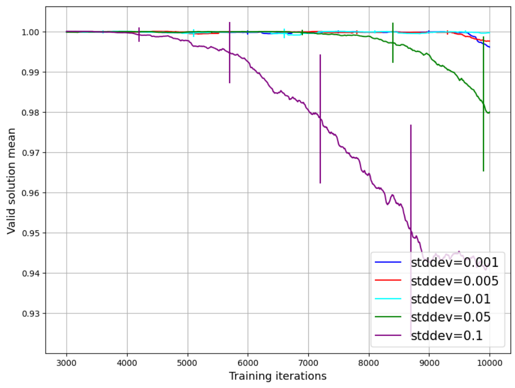


(b) Valid solution mean

Figure A.42: DDPG – Economic–Dispatch – Training – Sweep–over critic network hidden layer results.



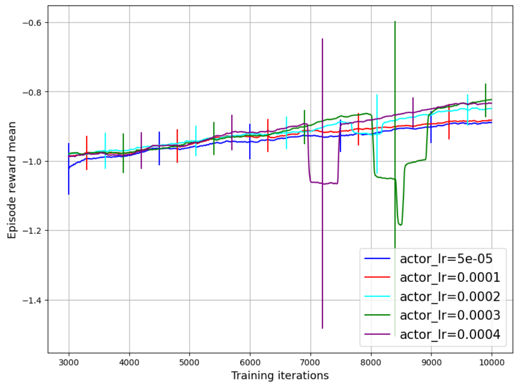
(a) Episode reward mean



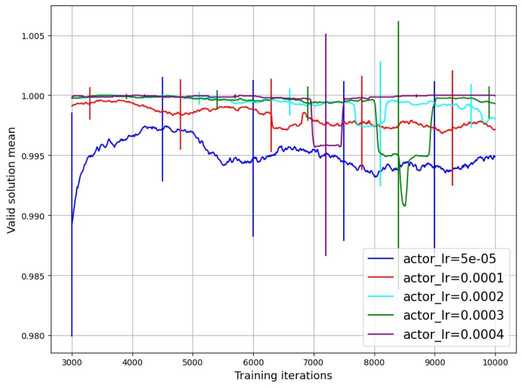
(b) Valid solution mean

Figure A.43: DDPG – Economic–Dispatch – Training – Sweep–over Gaussian noise STD results.

A.7 TD3 Q–Market Sweep–Over Results

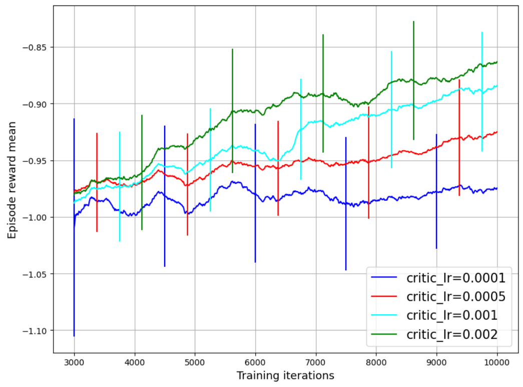


(a) Episode reward mean

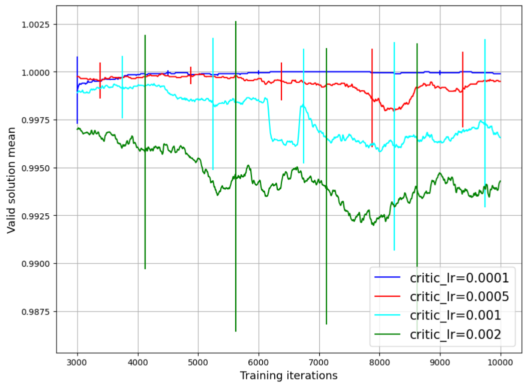


(b) Valid solution mean

Figure A.44: TD3 – Economic–Dispatch – Training – Sweep–over actor LR results.

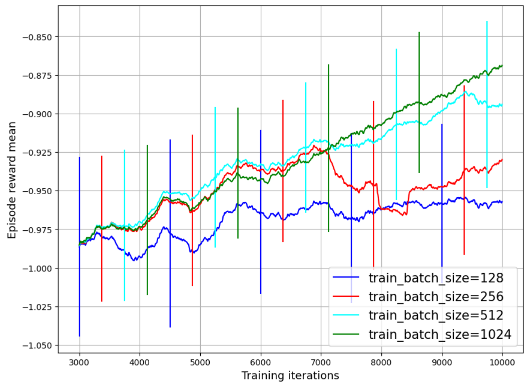


(a) Episode reward mean

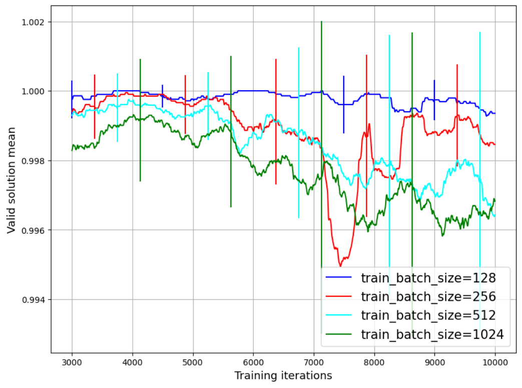


(b) Valid solution mean

Figure A.45: TD3 – Economic–Dispatch – Training – Sweep–over critic LR results.

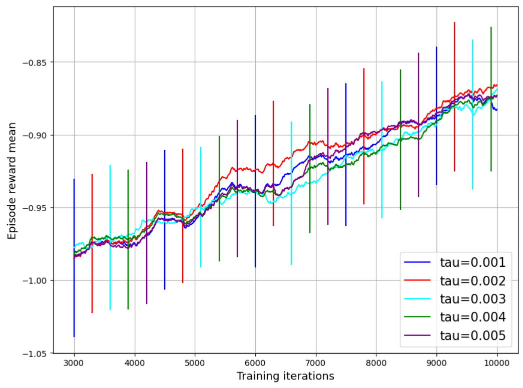


(a) Episode reward mean

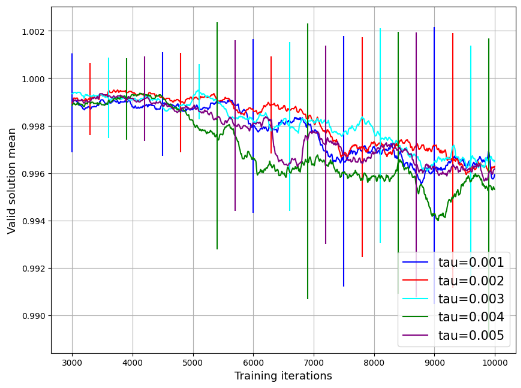


(b) Valid solution mean

Figure A.46: TD3 – Economic–Dispatch – Training – Sweep–over batch size results.

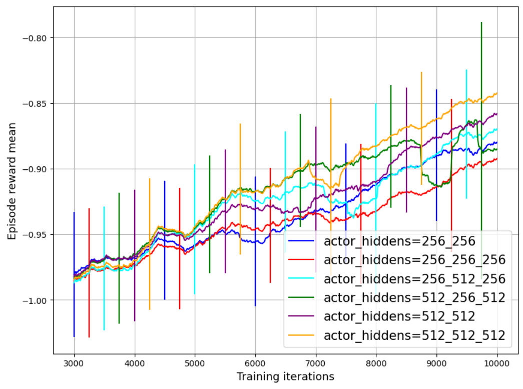


(a) Episode reward mean

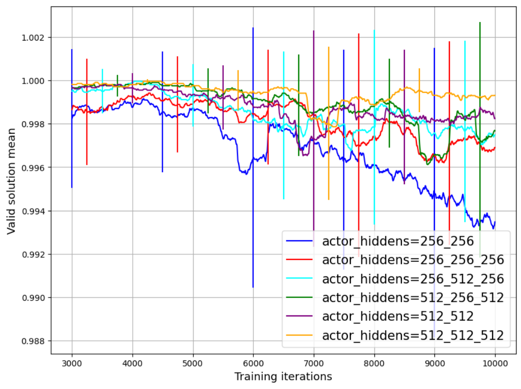


(b) Valid solution mean

Figure A.47: TD3 – Economic–Dispatch – Training – Sweep–over tau τ results.

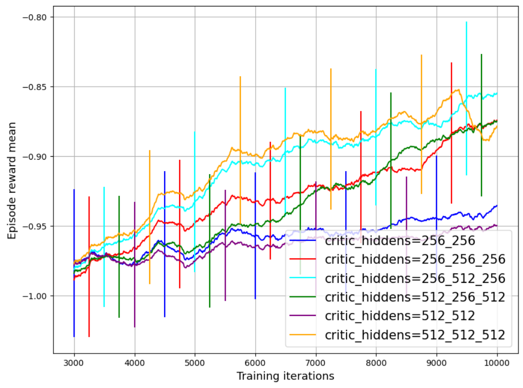


(a) Episode reward mean

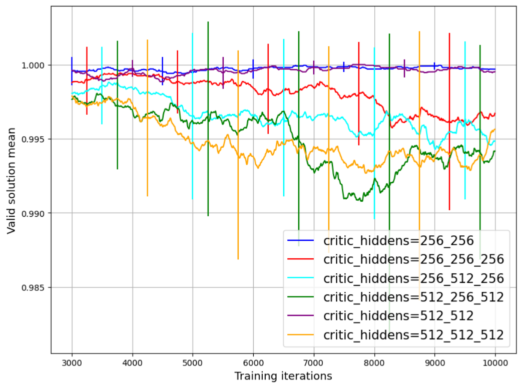


(b) Valid solution mean

Figure A.48: TD3 – Economic–Dispatch – Training – Sweep–over actor network hidden layer results.

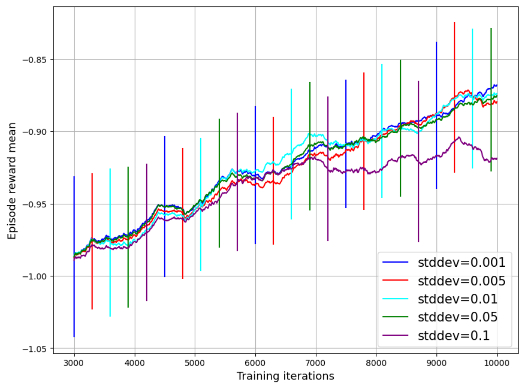


(a) Episode reward mean

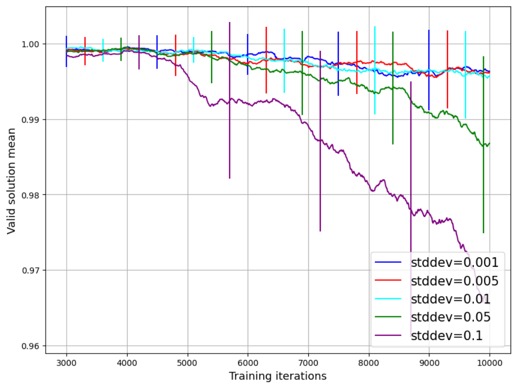


(b) Valid solution mean

Figure A.49: TD3 – Economic–Dispatch – Training – Sweep–over critic network hidden layer results.

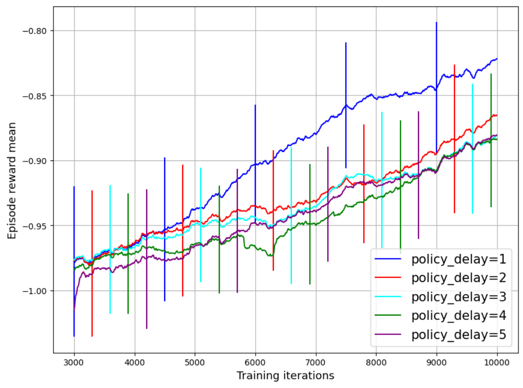


(a) Episode reward mean

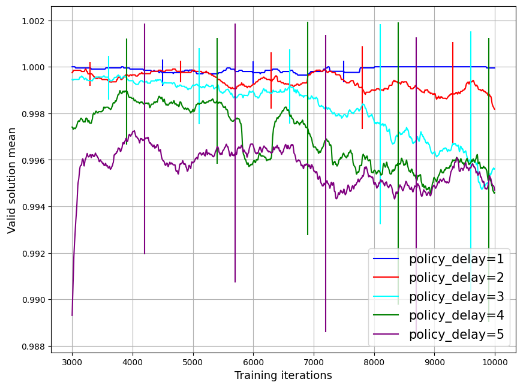


(b) Valid solution mean

Figure A.50: TD3 – Economic–Dispatch – Training – Sweep–over Gaussian noise STD results.



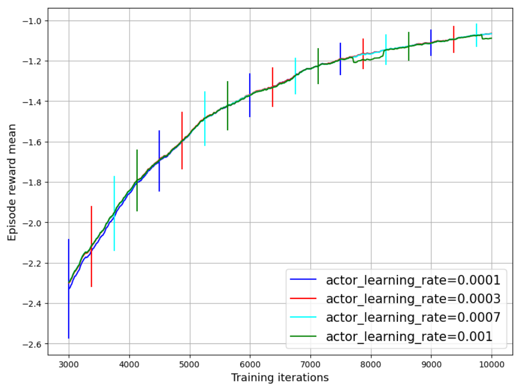
(a) Episode reward mean



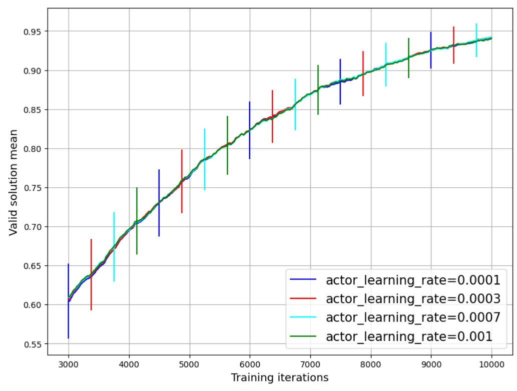
(b) Valid solution mean

Figure A.51: TD3 – Economic–Dispatch – Training – Sweep–over policy delay results.

A.8 SAC Economic–Dispatch Sweep–Over Results

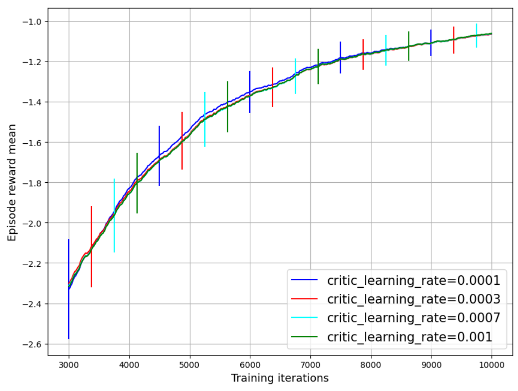


(a) Episode reward mean

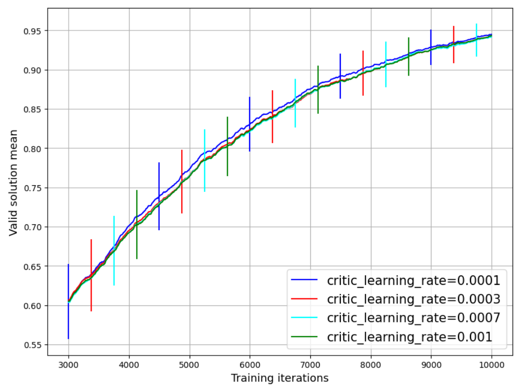


(b) Valid Solution mean

Figure A.52: SAC – Economic–Dispatch – Training – Sweep–over actor LR results.

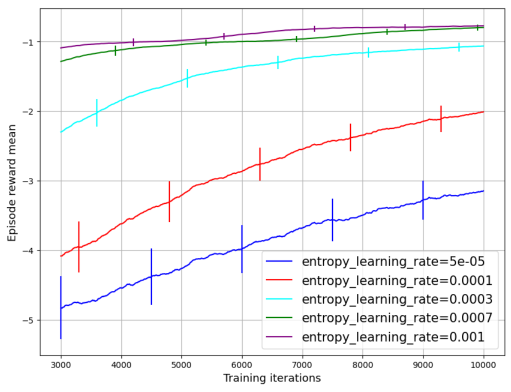


(a) Episode reward mean

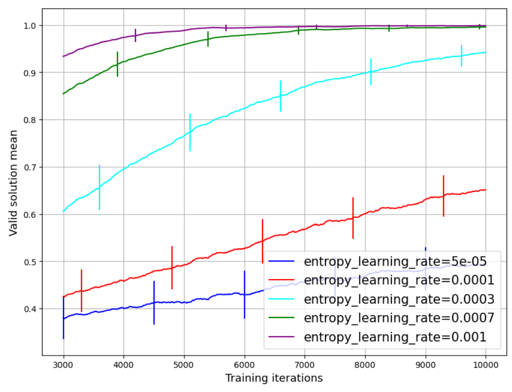


(b) Valid Solution mean

Figure A.53: SAC – Economic–Dispatch – Training – Sweep–over critic LR results.

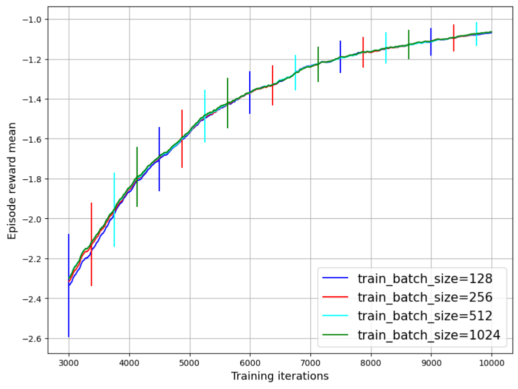


(a) Episode reward mean

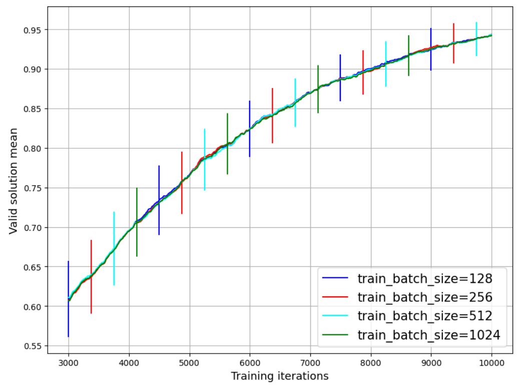


(b) Valid Solution mean

Figure A.54: SAC – Economic–Dispatch – Training – Sweep–over entropy LR results.

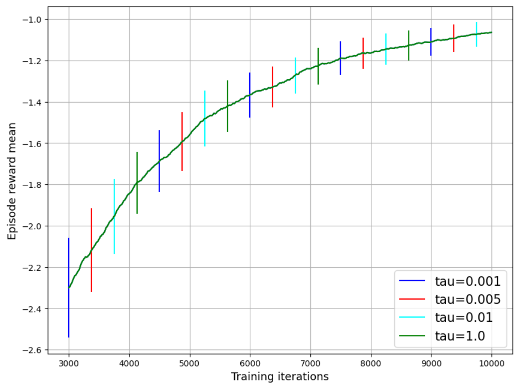


(a) Episode reward mean

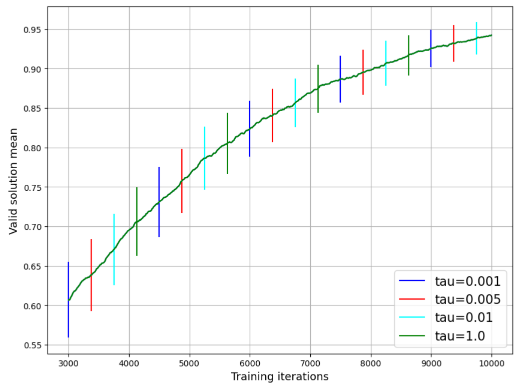


(b) Valid Solution mean

Figure A.55: SAC – Economic–Dispatch – Training – Sweep–over batch size results.

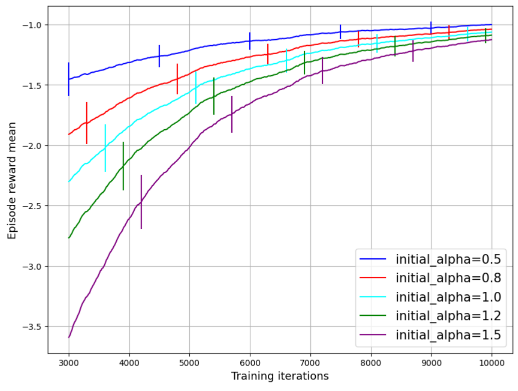


(a) Episode reward mean

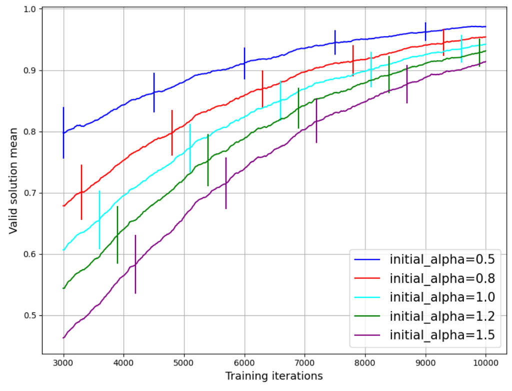


(b) Valid Solution mean

Figure A.56: SAC – Economic–Dispatch – Training – Sweep–over tau τ results.

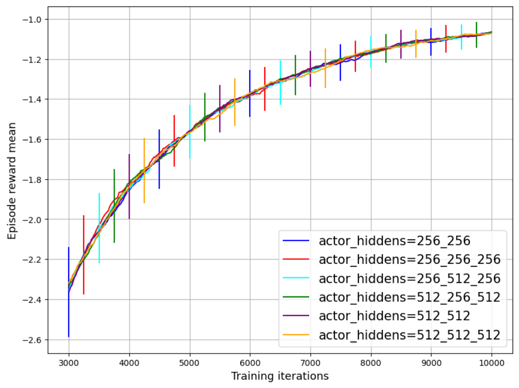


(a) Episode reward mean

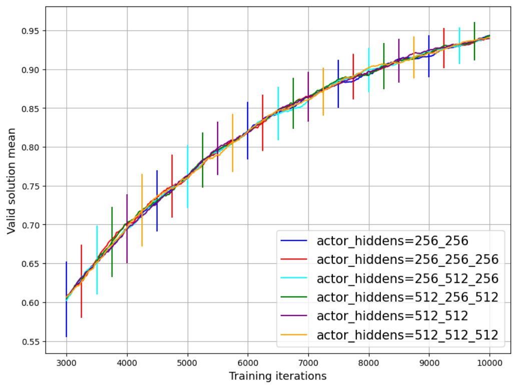


(b) Valid Solution mean

Figure A.57: SAC – Economic–Dispatch – Training – Sweep–over initial alpha value results.

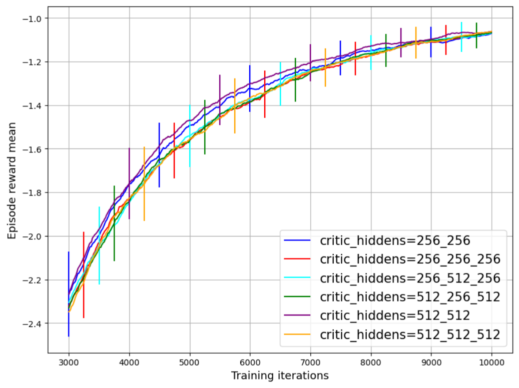


(a) Episode reward mean

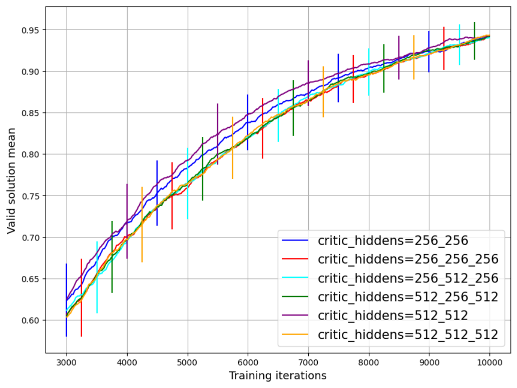


(b) Valid Solution mean

Figure A.58: SAC – Economic–Dispatch – Training – Sweep–over actor network hidden layer results.



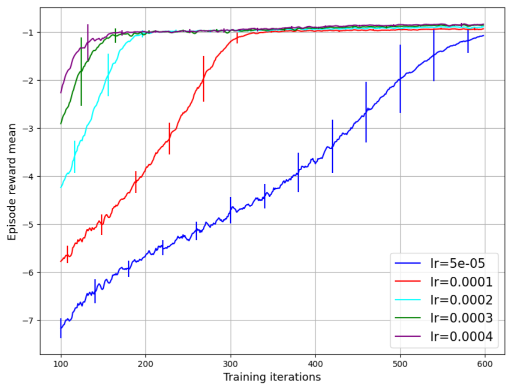
(a) Episode reward mean



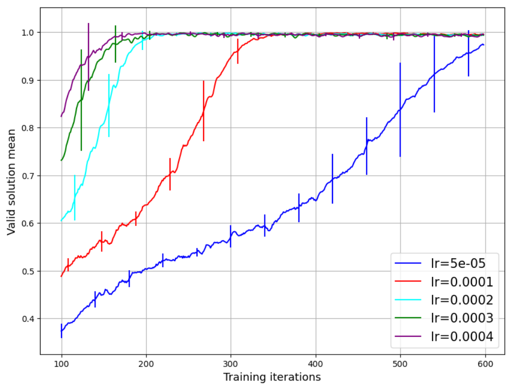
(b) Valid Solution mean

Figure A.59: SAC – Economic–Dispatch – Training – Sweep–over critic network hidden layer results.

A.9 A2C Economic–Dispatch Sweep–Over Results

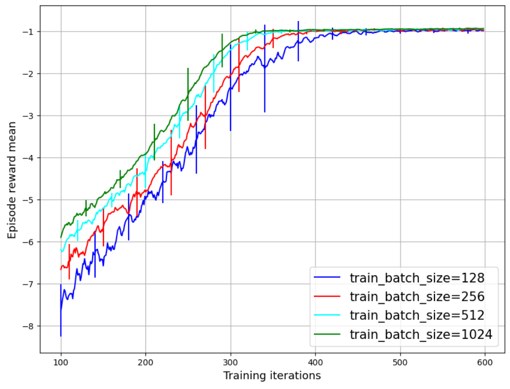


(a) Episode reward mean

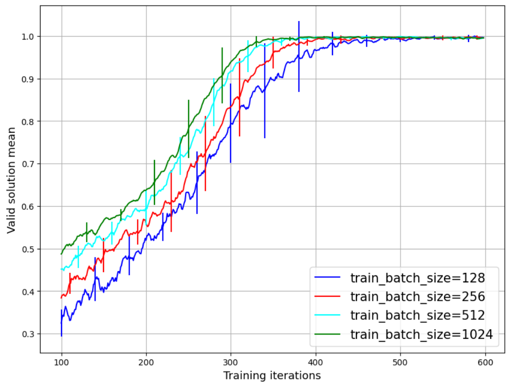


(b) Valid Solution mean

Figure A.60: A2C – Economic–Dispatch – Training – Sweep–over LR results.

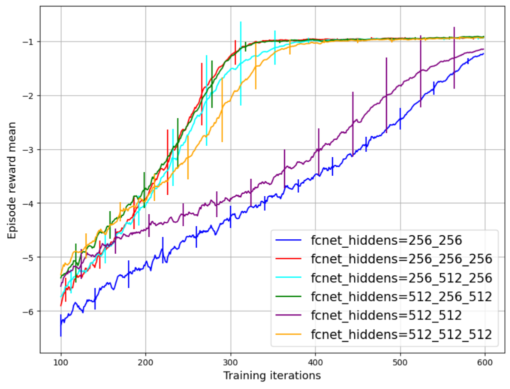


(a) Episode reward mean

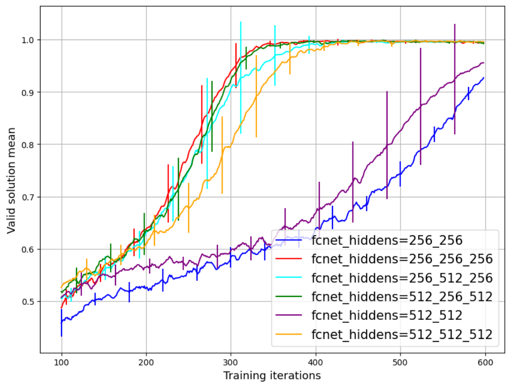


(b) Valid Solution mean

Figure A.61: A2C – Economic–Dispatch – Training – Sweep–over batch size results.

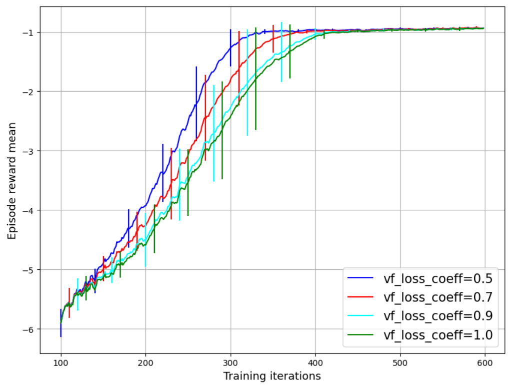


(a) Episode reward mean

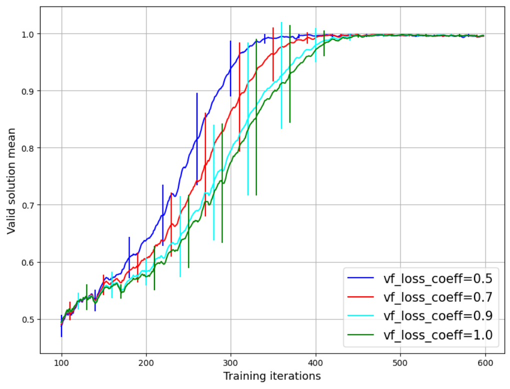


(b) Valid Solution mean

Figure A.62: A2C – Economic–Dispatch – Training – Sweep–over network hidden layer results.

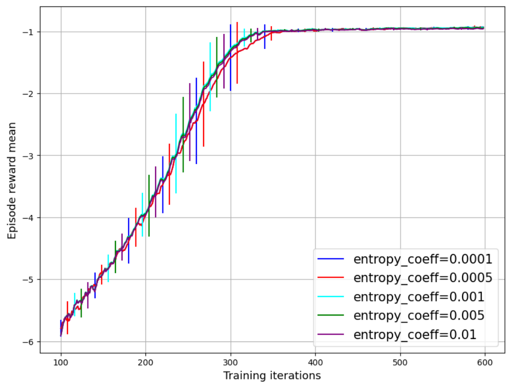


(a) Episode reward mean

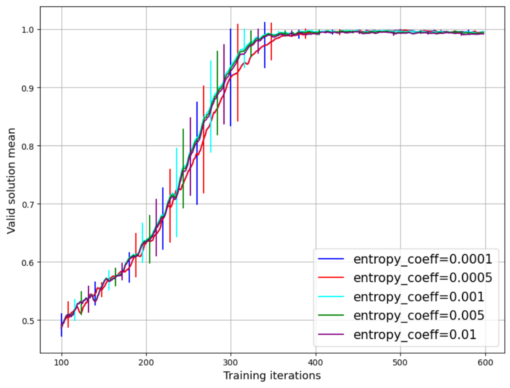


(b) Valid Solution mean

Figure A.63: A2C – Economic–Dispatch – Training – Sweep–over value function loss coefficient results.



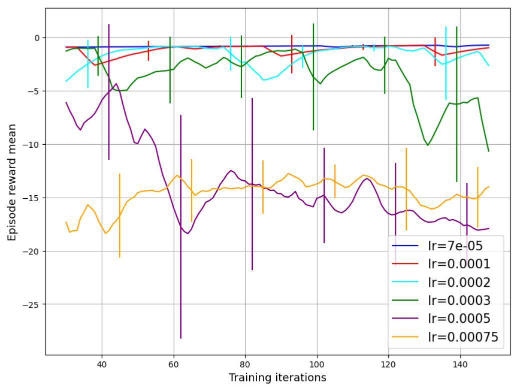
(a) Episode reward mean



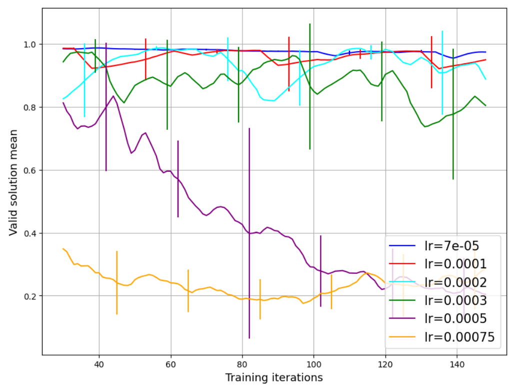
(b) Valid Solution mean

Figure A.64: A2C – Economic–Dispatch – Training – Sweep–over entropy coefficient results.

A.10 PPO Economic–Dispatch Sweep–Over Results

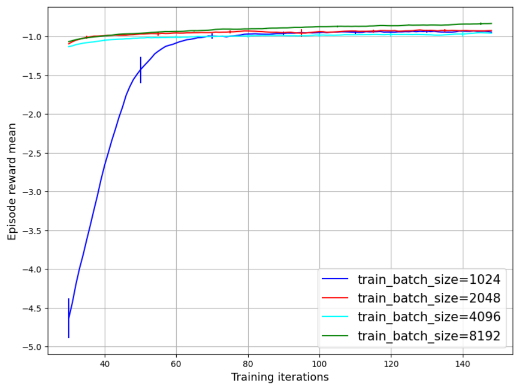


(a) Episode reward mean

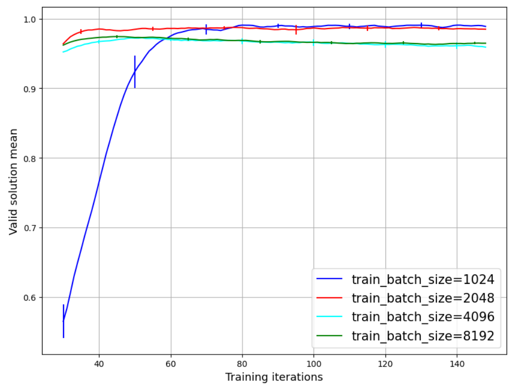


(b) Valid Solution mean

Figure A.65: PPO – Economic–Dispatch – Training – Sweep–over LR results.

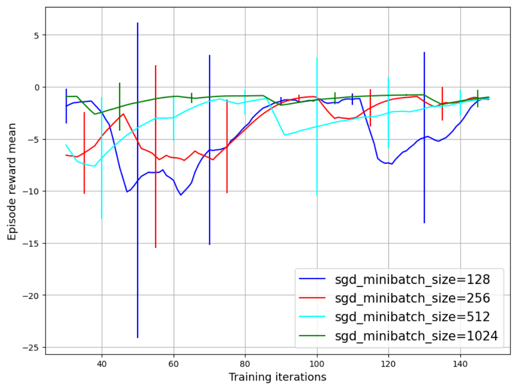


(a) Episode reward mean

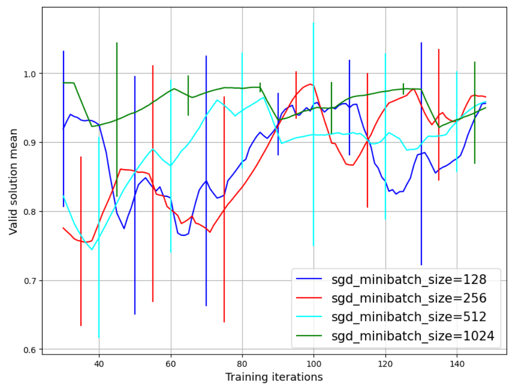


(b) Valid Solution mean

Figure A.66: PPO – Economic–Dispatch – Training – Sweep–over batch size results.

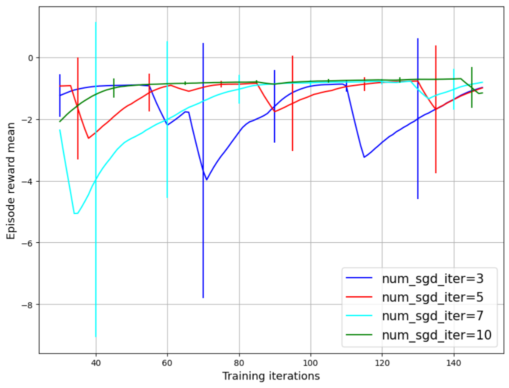


(a) Episode reward mean

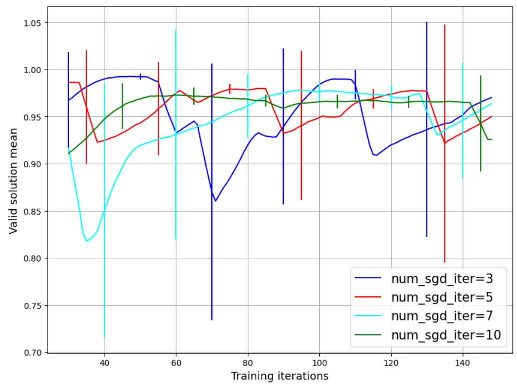


(b) Valid Solution mean

Figure A.67: PPO – Economic–Dispatch – Training – Sweep–over minibatch size results.

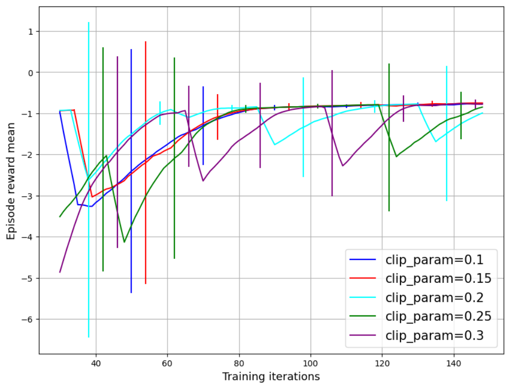


(a) Episode reward mean

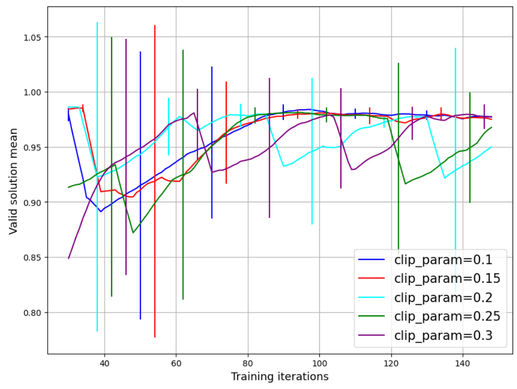


(b) Valid Solution mean

Figure A.68: PPO – Economic–Dispatch – Training – Sweep–over SGD iteration results.

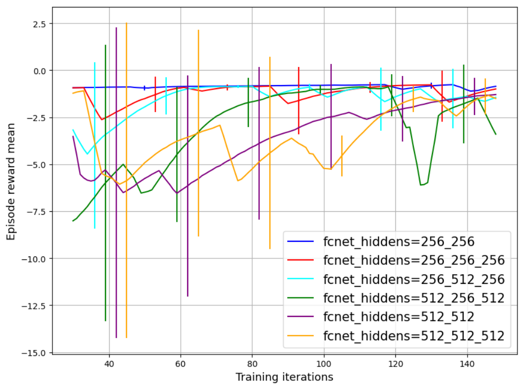


(a) Episode reward mean

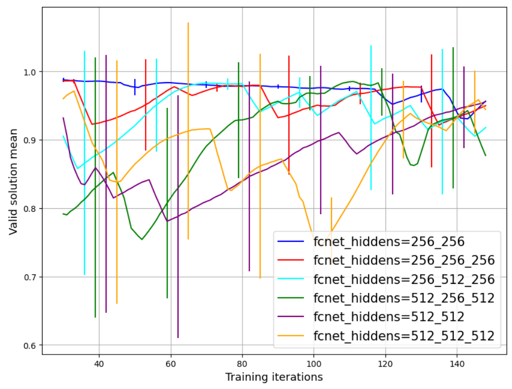


(b) Valid Solution mean

Figure A.69: PPO – Economic–Dispatch – Training – Sweep–over PPO clip hyperparameter results.

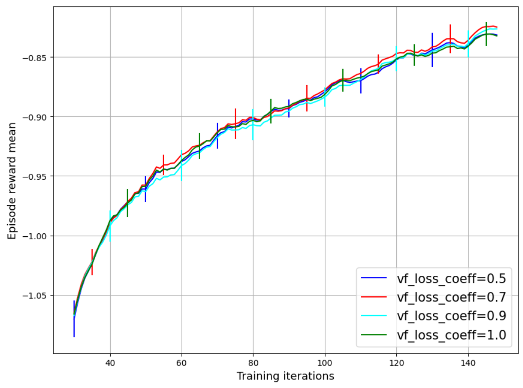


(a) Episode reward mean

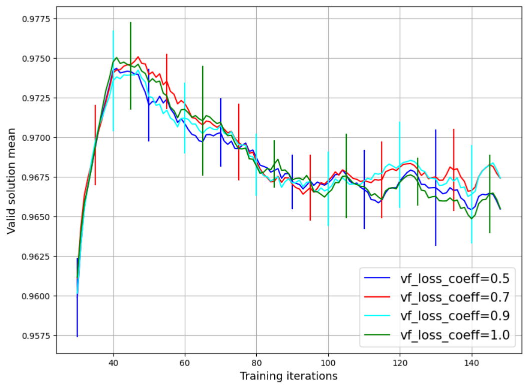


(b) Valid Solution mean

Figure A.70: PPO – Economic–Dispatch – Training – Sweep–over network hidden layer results.

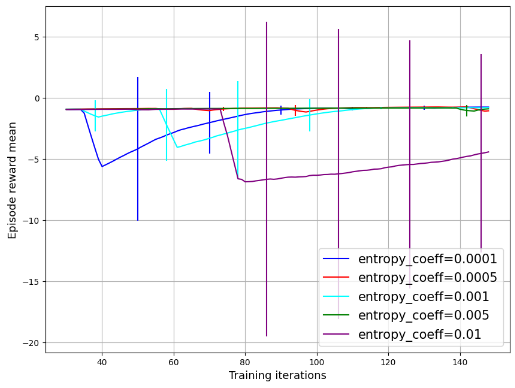


(a) Episode reward mean

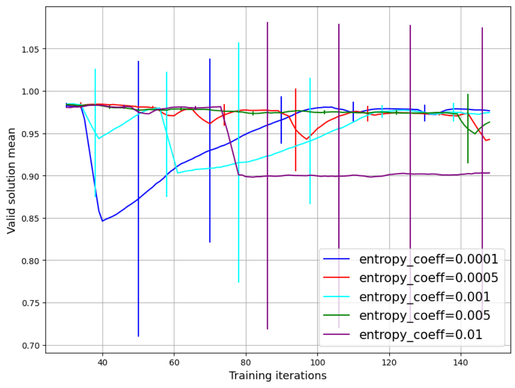


(b) Valid Solution mean

Figure A.71: PPO – Economic–Dispatch – Training – Sweep–over value function loss coefficient results.



(a) Episode reward mean



(b) Valid Solution mean

Figure A.72: PPO – Economic–Dispatch – Training – Sweep–over entropy coefficient results.

List of Figures

List of Tables

Bibliography

[1] Nie, H., Chen, Y., Song, Y., Huang, S.: A General Real-time OPF Algorithm Using DDPG with Multiple Simulation Platforms. In: 2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia), pp. 3713–3718 (2019). doi:10.1109/ISGT-Asia.2019.8881174

[2] Damjanović, I., Pavić, I., Puljiz, M., Brcic, M.: Deep Reinforcement Learning-Based Approach for Autonomous Power Flow Control Using Only Topology Changes. Energies **15**(19) (2022). doi:10.3390/en15196920

[3] Yan, Z., Xu, Y.: Real-Time Optimal Power Flow: A Lagrangian Based Deep Reinforcement Learning Approach. IEEE Transactions on Power Systems **35**(4), 3270–3273 (2020). doi:10.1109/TPWRS.2020.2987292

[4] Liu, X., Fan, B., Tian, J.: Deep Reinforcement Learning Based Approach for Dynamic Optimal Power Flow in Active Distribution Network. In: 2022 41st Chinese Control Conference (CCC), pp. 1951–1956 (2022). doi:10.23919/CCC55666.2022.9902611

[5] Zeng, S., Kody, A., Kim, Y., Kim, K., Molzahn, D.: A reinforcement learning approach to parameter selection for distributed optimal power flow. Electric Power Systems Research **212**, 108546 (2022). doi:10.1016/j.epsr.2022.108546

[6] Frank, S., Steponavičė, I., Rebennack, S.: Optimal power flow: a bibliographic survey I. Energy Systems **3** (2012). doi:10.1007/s12667-012-0056-y

[7] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, (2018). http://incompleteideas.net/book/the-book-2nd.html

[8] Lapan, M.: Deep Reinforcement Learning Hands-On. Packt Publishing, Birmingham, UK (2018)

[9] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing Atari with Deep Reinforcement Learning. CoRR **abs/1312.5602** (2013). doi:10.48550/arXiv.1312.5602

[10] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR

2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016). http://arxiv.org/abs/1509.02971

[11] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. CoRR **abs/1707.06347** (2017). doi:10.48550/arXiv.1707.06347

[12] Fujimoto, S., van Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1587–1596. PMLR, (2018). https://proceedings.mlr.press/v80/fujimoto18a.html

[13] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep reinforcement learning with a stochastic actor. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1861–1870. PMLR, (2018). https://proceedings.mlr.press/v80/haarnoja18b.html

[14] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (2016). https://proceedings.mlr.press/v48/mniha16.html

[15] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1889–1897. PMLR, Lille, France (2015). https://proceedings.mlr.press/v37/schulman15.html

[16] Schulman, J., Moritz, P., Levine, S., Jordan, M.I., Abbeel, P.: High-Dimensional Continuous Control Using Generalized Advantage Estimation. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016). http://arxiv.org/abs/1506.02438

[17] Goodfellow, I.J., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA, USA (2016). `http://www.deeplearningbook.org`

[18] Yu, T., Zhu, H.: Hyper-Parameter Optimization: A Review of Algorithms and Applications. CoRR **abs/2003.05689** (2020). doi:10.48550/arXiv.2003.05689. 2003.05689

[19] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W.M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., Kavukcuoglu,

K.: Population Based Training of Neural Networks. CoRR **abs/1711.09846** (2017). doi:10.48550/arXiv.1711.09846

[20] Zhen, H., Zhai, H., Ma, W., Zhao, L., Weng, Y., Xu, Y., Shi, J., He, X.: Design and tests of reinforcement-learning-based optimal power flow solution generator. Energy Reports **8**, 43–50 (2022). doi:10.1016/j.egyr.2021.11.126. 2021 The 8th International Conference on Power and Energy Systems Engineering

[21] Zhou, Y., Zhang, B., Xu, C., Lan, T., Diao, R., Shi, D., Wang, Z., Lee, W.-J.: A Data-driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning. Journal of Modern Power Systems and Clean Energy **8**(6), 1128–1139 (2020). doi:10.35833/MPCE.2020.000522

[22] Zhou, Y., Jia, L., Zhao, Y., Zhan, Z.: Optimal dispatch of an integrated energy system based on deep reinforcement learning considering new energy uncertainty. In: 2023 IEEE 12th Data Driven Control and Learning Systems Conference (DDCLS), pp. 804–809 (2023). doi:10.1109/DDCLS58216.2023.10166885

[23] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep Reinforcement Learning that Matters. CoRR **abs/1709.06560** (2017). doi:10.48550/arXiv.1709.06560

[24] Eimer, T., Lindauer, M., Raileanu, R.: Hyperparameters in Reinforcement Learning and How To Tune Them. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 9104–9149. PMLR, (2023). https://proceedings.mlr.press/v202/eimer23a.html

[25] Meinecke, S., Sarajlić, D., Drauz, S.R., Klettke, A., Lauven, L.-P., Rehtanz, C., Moser, A., Braun, M.: SimBench — A Benchmark Dataset of Electric Power Systems to Compare Innovative Solutions Based on Power Flow Analysis. Energies **13**(12) (2020). doi:10.3390/en13123290

[26] Thurner, L., Scheidler, A., Schäfer, F., Menke, J.-H., Dollichon, J., Meier, F., Meinecke, S., Braun, M.: Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. IEEE Transactions on Power Systems **33**(6), 6510–6521 (2018). doi:10.1109/TPWRS.2018.2829021

[27] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., Stoica, I.: Ray RLLib: A Composable and Scalable Reinforcement Learning Library. CoRR **abs/1712.09381** (2017). doi:10.48550/arXiv.1712.09381

[28] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)

[29] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: A Research Platform for Distributed Model Selection and Training. CoRR **abs/1807.05118** (2018). doi:10.48550/arXiv.1807.05118

[30] Cortes, C., Mohri, M., Rostamizadeh, A.: L2 Regularization for Learning Kernels. CoRR **abs/1205.2653** (2012). doi:10.48550/arXiv.1205.2653

[31] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., Kavukcuoglu, K.: IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1407–1416. PMLR, (2018). https://proceedings.mlr.press/v80/espeholt18a.html

Statement of Originality

I hereby confirm in lieu of oath, that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgments. I certify, that I have followed the general principles of scientific work and publications as written in the guidelines of good research of the Carl von Ossietzky University of Oldenburg. This work has not yet been submitted to any examination office in the same or similar form.

Lorenz Mumm

Oldenburg, January 4, 2024 Lorenz Mumm, B. Sc.