UNIVERSITÄT AUGSBURG

DISSERTATION

Interpolation Assisted Deep Reinforcement Learning

Author:
Wenzel Baron PILAR VON PILCHAU

Supervisor:
Prof. Dr. Jörg HÄHNER

Universität
Augsburg
University

A thesis submitted in fulfillment of the requirements for the degree of Doktor rer. nat.

in the

Fakultät für Angewandte Informatik
Organic Computing Group

2024

Erstgutachter:	Prof. Dr. Jörg Hähner
Zweitgutachterin:	Prof. Dr. Elisabeth André
Tag der mündlichen Prüfung	15. Februar 2024

Declaration of Authorship

I, Wenzel Baron PILAR VON PILCHAU, declare that this thesis titled, "Interpolation Assisted Deep Reinforcement Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"If wars can be started by lies, they can be stopped by truth."

Julian Assange

UNIVERSITÄT AUGSBURG

Abstract

Fakultät für Angewandte Informatik
Organic Computing Group

Doktor rer. nat.

Interpolation Assisted Deep Reinforcement Learning

by Wenzel Baron PILAR VON PILCHAU

Reinforcement Learning is a field of Machine Learning that, in contrast to the other prominent representatives, generates its training data during runtime in direct interaction with an environment. A sample of this training data is called experience and represents a state-transition that is caused by an action together with the corresponding reward signal. Experiences can be seen as a form of knowledge about the underlying dynamics of the environment and a common technique in the field of Reinforcement Learning is the so-called Experience Replay which stores and replays experiences that have been observed at some point in training. By doing so, sample efficiency can be increased as experiences are used many times for training instead of throwing them away after one update. As experiences are generated during runtime, the learner has to explore the state-space and it is only able to learn the dynamics of specific areas when it has been there sometime in the past. As mentioned earlier, experiences can be seen as knowledge about the underlying problem and it is possible to generate synthetic experiences of states that have not been visited yet based on stored real experiences of neighbouring states. Such synthetic experiences can be generated by means of interpolation and can further on be used to assist the learner with exploration. Also, sample efficiency can be increased even further as real experiences are used to generate synthetic ones. In this work, two different techniques are presented that make use of synthetic experiences to assist the learner. The first approach stores generated synthetic experiences in the buffer alongside real experiences and during training real, as well as synthetic experiences are drawn at random from the buffer. This mechanism is called *Interpolated Experience Replay*. The second approach leverages on the architectural design of the Deep Q-Network and uses synthetic experiences to enable training updates that take the full action-space into account. This second algorithm is called *Full-Update DQN*. As methods that combine interpolation with a replay buffer and model-free learning algorithms fit neither the definition of model-free, nor model-based, the new class *Semi-Model-Based* is introduced to cover them.

Acknowledgements

I would like to express my sincere gratitude and appreciation to the following individuals whose support and guidance have been instrumental in the completion of my PhD thesis.

First and foremost, I express my heartfelt thanks to my supervisor, Prof. Dr. Jörg Hähner. It is through his guidance and mentorship that I entered the realm of academia and had the privilege to pursue a PhD. I am particularly grateful for the support he has provided, both in terms of the subject matter and in other aspects related to my research. I consider myself incredibly fortunate to have had the opportunity to learn from someone as esteemed as him and benefit from his expertise.

I also want to express my gratitude towards my former college Prof. Dr. Anthony Stein. It was him who came up with the idea that resulted into the topic of this thesis. His invaluable insights, constructive feedback, and constant encouragement have been pivotal in shaping the direction of my work. I am proud to call him a friend today and am very thankful for his ongoing support and supervision during this phase of my life.

I am also indebted to my parents for their unwavering love, endless encouragement, and continuous belief in my abilities. Their support, sacrifices, and faith in my capabilities have been a constant source of inspiration throughout this challenging journey. I am truly grateful for the opportunities and resources they have provided me, which have enabled me to pursue my academic aspirations with confidence.

Furthermore, I would like to express my deepest appreciation to my partner, Maggy, for her unwavering support, understanding, and patience throughout this demanding phase of my academic journey. Her presence, encouragement, and belief in me have been the pillars of strength that have kept me motivated during the difficult times. Her unconditional love and unwavering support have meant the world to me, and I am profoundly grateful for her presence in my life.

Finally, I would like to extend my gratitude to the academic community, research institutions, and funding agencies whose contributions, either directly or indirectly, have made this research possible. The resources, facilities, and opportunities provided by these organizations have been vital in shaping my research endeavors. Specifically, I would like to express my thank to the funding agency BMBF (Federal Ministry of Education and Research) which funded my research with the following two projects: FMLA (01IS17074) and FAMOUS (01IS18078E).

Completing this PhD thesis has been a challenging and rewarding journey, and I am sincerely grateful to all those who have played a part, however big or small, in making this achievement possible. Thank you for your unwavering support, guidance, and encouragement.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1

1.1 Scientific Contribution 4
1.2 Relation to Organic Computing 5
1.3 Structure 6
1.4 Integration of the Authors Published Works 6

2 Fundamentals 9

2.1 Finite Markov Decision Processes 9
2.1.1 The Agent–Environment Interface 9
2.1.2 Rewards, Returns and Episodes 11
2.1.3 Policies and Value Functions 13
2.1.4 Optimal Policies and Optimal Value Functions 15
2.2 Temporal-Difference Learning 18
2.2.1 Reminder: Dynamic Programming and Monte Carlo 18
Dynamic Programming 18
Monte Carlo Methods 18
2.2.2 TD Prediction 19
2.2.3 Exploration vs. Exploitation 21
2.2.4 Q-Learning: Off-policy TD Control 22
2.2.5 Double Q-Learning 23
2.2.6 Experience Replay 23
2.3 Types of Reinforcement Learning 24
2.3.1 The Model 24
2.3.2 Model-Free Reinforcement Learning 25
2.3.3 Model-Based Reinforcement Learning 25
Type of model 26
Estimation method 27
Region in which the Model is Valid 27
2.4 Deep Q-Network 28
2.4.1 Limitations of Original Q-Learning 28

2.4.2 Deep Q-Network Algorithm . . . 28
2.4.3 Extensions of the Deep Q-Network . . . 31
Double Deep Q-Network . . . 31
Dueling Deep Q-Network . . . 32
2.5 Interpolation . . . 34
2.5.1 Interpolation vs. Approximation . . . 34
2.5.2 Inverse Distance Weighting . . . 34
2.6 Nearest Neighbor Search . . . 35
2.6.1 K-D Tree . . . 35
2.6.2 Ball Tree . . . 36

3 Related Work **39**
3.1 Experience Replay . . . 39
3.1.1 Prioritized Experience Replay . . . 39
3.1.2 Hindsight Experience Replay . . . 40
3.1.3 Experience Composition . . . 40
3.1.4 Episodic Backward Update . . . 40
3.2 Interpolation in Experience Replay . . . 41
3.2.1 Mixup Sampling in RL . . . 41
3.2.2 Interpolated Experience Replay for Continuous Action-Spaces . 41
3.3 Interpolation in Reinforcement Learning . . . 42
3.3.1 Interpolation in Learning Classifier Systems . . . 42
3.3.2 Interpolated Policy Gradient . . . 43

4 Linear Reward Interpolation in Discrete and Non-Deterministic Environments **45**
4.1 Interpolation of Synthetic Experiences . . . 45
4.1.1 Experience Analysis with Respect to Interpolation Possibility . 46
4.1.2 Discrete and Non-deterministic Environments . . . 47
4.1.3 Linear Interpolation of Rewards . . . 50
4.2 Interpolated Experience Replay . . . 51
4.2.1 Interpolation Component . . . 52
4.2.2 Modifications and Query Functions . . . 55
4.3 Evaluation . . . 57
4.3.1 FrozenLake Environment . . . 57
4.3.2 Experiments . . . 58
4.3.3 Hyperparameters . . . 60
4.3.4 Statistical Analysis . . . 61
Statistical model for the solution rates . . . 62
Statistical model for the TTS units . . . 63
Practical equivalence . . . 64
4.3.5 Results . . . 64
Analysis of the results for the VE state-encoding . . . 65

Analysis of the results for the CE state-encoding 67
Analysis of the results for the LKE state-encoding 68
4.4 Discussion 70

5 Interpolated Experience Replay 73
5.1 Interpolation of the Follow-up State 73
5.1.1 Problem Description 73
5.1.2 Transition-Delta Interpolation 75
5.2 Interpolated Experience Replay for Continuous Environments 76
5.2.1 Nearest Neighbor Search 76
5.2.2 Query Functions 77
5.2.3 IER algorithm 78
5.3 Evaluation 78
5.3.1 MountainCar Environment 78
5.3.2 Experiments 80
5.3.3 Hyperparameters 80
5.3.4 Statistical Analysis 82
5.3.5 Results 82
5.4 Discussion 84

6 Full-Update Deep Q-Network 87
6.1 Full-Update DQN 87
6.1.1 The Unused Potential of a DQN Update 87
6.1.2 Updates for the Whole Action-Space 89
6.1.3 Generation Function *I* 90
6.1.4 Handle Interpolation Noise 90
6.1.5 Expected Benefits 91
6.2 Evaluation 92
6.2.1 Experiments 92
6.2.2 Hyperparameters 94
6.2.3 Statistical Analysis 95
6.2.4 Results 96
Analysis of the CHAIN-N tasks 96
Analysis of the results for the FROZENLAKE task 98
Analysis of the results for the MOUNTAINCAR task 100
Analysis of the results for the UMAZE task 100
6.3 Discussion 100

7 Semi-Model-Based Reinforcement Learning 103
7.1 Semi-Model-Based Reinforcement Learning 103
7.2 Interpolation-Based RL in Organic Computing 105

8 Conclusion **109**
8.1 Conclusion 109
8.2 Summary 112
8.3 Outlook 113

A Evaluation Results from Linear Reward Interpolation **115**
A.1 Solution Rates 115
A.2 Probabilities 120
A.3 Mean TTS 128
A.4 Probability Distributions 133
A.4.1 VE state-encoding 134
A.4.2 CE state-encoding 137
A.4.3 LKE state-encoding 140

B Evaluation Results from IER for Continuous Environments **143**
B.1 Solution Rates, Probabilities and TTS 143
B.2 Probability Distribution 147

Bibliography **149**

Publications of the Author

Journal Articles

[PSH21] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Synthetic Experiences for Accelerating DQN Performance in Discrete Non-Deterministic Environments". In: *Algorithms* 14.8 (2021). ISSN: 1999-4893. DOI: 10.3390/a14080226.

Conference Papers

[Pil+20] Wenzel Pilar von Pilchau, Varun Gowtham, Maximilian Gruber, et al. "An Architectural Design for Measurement Uncertainty Evaluation in Cyber-Physical Systems". In: *Annals of Computer Science and Information Systems, Volume 22: Position Papers of the 2020 Federated Conference on Computer Science and Information Systems*. Ed. by M. Ganzha, L. Maciaszek, and M. Paprzycki. Vol. 22. PTI - Polish Information Processing Society, 2020, pp. 53–57. ISBN: 978-83-959183-0-8. DOI: 10.15439/2020f203.

[PSH20] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Bootstrapping a DQN Replay Memory with Synthetic Experiences". In: *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020), November 2-4, 2020*. Ed. by Juan Julian Merelo, Jonathan Garibaldi, Christian Wagner, et al. 2020. ISBN: 978-989-758-475-6. DOI: 10.5220/0010107904040411.

[Gow+21] V. Gowtham, A. Willner, W. Pilar von Pilchau, et al. "A Reference Architecture enabling Sensor Networks based on homogeneous AASs". In: *Automation 2021: Navigating towards resilient Production*. Ed. by VDI Wissensforum GmbH. 1st ed. Düsseldorf: VDI Verlag, 2021, pp. 5–16. ISBN: 978-3-18-102392-1. DOI: 10.51202/9783181023921-5.

[Gru+21] Maximilian Gruber, S Eichstädt, Wenzel Pilar von Pilchau, et al. "Uncertainty-Aware Sensor Fusion in Sensor Networks". In: *SMSI 2021-Measurement Science* (2021), pp. 246–247. DOI: 10.5162/SMSI2021/D2.2.

[Gru+22] Maximilian Gruber, Wenzel Pilar von Pilchau, Varun Gowtham, et al. "Application of Uncertainty-Aware Sensor Fusion in Physical Sensor Networks". In: *2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2022, pp. 1–6. DOI: 10.1109/I2MTC48687.2022.9806580.

[Kou+22] Nikolaos-Stefanos Koutrakis, Varun Gowtham, Wenzel Pilar von Pilchau, et al. "Harmonization of Heterogeneous Asset Administration Shells". In: *Procedia CIRP* 107 (2022), pp. 95–100. ISSN: 2212-8271. DOI: 10.1016/j.procir.2022.04.016.

[PSH22a] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Semi-Model-Based Reinforcement Learning in Organic Computing Systems". In: *Architecture of Computing Systems*. Ed. by Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck. Cham: Springer International Publishing, 2022, pp. 241–255. ISBN: 978-3-031-21867-5. DOI: 10.1007/978-3-031-21867-5_16.

[PSH22b] Wenzel Pilar von Pilchau, Anthony Stein., and Jörg Hähner. "Interpolated Experience Replay for Continuous Environments". In: *Proceedings of the 14th International Joint Conference on Computational Intelligence (IJCCI 2022) - NCTA*. SciTePress / INSTICC, 2022, pp. 237–248. ISBN: 978-989-758-611-8. DOI: 10.5220/0011326900003332.

[Pil+23] Wenzel Pilar von Pilchau, David Pätzel, Anthony Stein, and Jörg Hähner. "Deep Q-Network Updates for the Full Action-Space Utilizing Synthetic Experiences". In: *2023 International Joint Conference on Neural Networks (IJCNN), to Appear*. 2023.

Doctoral Symposia

[Pil18] Wenzel Pilar von Pilchau. "Combining Machine Learning with Blockchain: Benefits, Approaches and Challenges". In: *Organic Computing - Doctoral Dissertation Colloquium (OC-DDC) 2018*. Ed. by Sven Tomforde and Bernhard Sick. Kassel University Press, 2018, pp. 89–98. ISBN: 978-3-7376-0696-7.

[Pil19] Wenzel Pilar von Pilchau. "Averaging Rewards as a First Approach towards Interpolated Experience Replay". In: *INFORMATIK 2019: 50 Jahre Gesellschaft Für Informatik Informatik Für Gesellschaft (Workshop-Beiträge)*. Ed. by Claude Draude, Martin Lange, and Bernhard Sick. Bonn: Gesellschaft für Informatik e.V., 2019, pp. 493–506. DOI: 10.18420/inf2019_ws53.

[Pil20] Wenzel Pilar von Pilchau. "Interpolated Experience Replay - A Roadmap". In: *Organic Computing - Doctoral Dissertation Colloquium (OC-DDC) 2020*. Ed. by Sven Tomforde and Christian Krupitzer. Universität Kassel, 2020, pp. 65–77.

[Pil22] Wenzel Pilar von Pilchau. "From Interpolated Experience Replay to Full Update DQN". In: *Organic Computing - Doctoral Dissertation Colloquium (OC-DDC) 2022*. Ed. by Sven Tomforde and Christian Krupitzer. 2022.

[Pil23] Wenzel Pilar von Pilchau. "SCHNITZEL: SCHematizing Next-level Interpolation Targeting Small siZe Exploration Learning". In: *Organic Computing - Doctoral Dissertation Colloquium (OC-DDC) 2023*. Ed. by Sven Tomforde and Christian Krupitzer. 2023.

List of Figures

1.1 The DIKW hierarchy 3

2.1 The MDP Cycle 10
2.2 Backup diagrams for value functions 15
2.3 Backup diagrams for optimal value functions 17
2.4 Backup diagram for TD(0) 20
2.5 Backup diagram for Q-Learning 22
2.6 Architecture of a deep Q-network 31
2.7 Architecture of a dueling deep Q-network 33
2.8 Intuition of approximation vs. interpolation 35
2.9 Intuition of a balltree 37

4.1 The FROZENLAKE8x8 environment 49
4.2 Schematic of the Interpolation Component 53
4.3 Intuition of Interpolated Experience Replay memory. 54
4.4 Graphical illustration of FROZENLAKE state-encodings. 59
4.5 Probability distributions of VE state-encoding 67
4.6 Probability distributions of CE state-encoding 69
4.7 Probability distributions of LKE state-encoding 71

5.1 Follow-up state dilemma 74
5.2 State-transition-delta interpolation 75
5.3 MOUNTAINCAR environment. 79
5.4 Probability distributions of IER on MOUNTAINCAR 83

6.1 Graphical illustration of a DQN loss. 88
6.2 Intuition of a DQN update. 88
6.3 Graphical intuition of the generation function 90
6.4 Four different environments for the FU-DQN evaluation 93
6.5 Probability distribution of FU-DQN on CHAIN-N 98
6.6 Probability distribution for FU-DQN on three tasks 99

List of Tables

4.1 Different types of tiles in LKE state-encoding. 58
4.2 Shared hyperparameters for FROZENLAKE 61
4.3 Shared IER hyperparameters for FROZENLAKE 61
4.4 Network architectures for FROZENLAKE 61
4.5 Parameters of FROZENLAKE configurations 61
4.6 E-Q2 results of VE state-encoding 65
4.7 E-Q1 results of VE state-encoding 66
4.8 E-Q3 results of VE state-encoding 66
4.9 E-Q2 results of CE state-encoding 68
4.10 E-Q1 results of CE state-encoding 68
4.11 E-Q3 results of CE state-encoding 68
4.12 E-Q2 results of LKE state-encoding 70
4.13 E-Q1 results of LKE state-encoding 70
4.14 E-Q3 results of LKE state-encoding 70

5.1 Shared hyperparameters for MOUNTAINCAR 81
5.2 Shared IER hyperparameters for MOUNTAINCAR 81
5.3 Specific IER parameters for MOUNTAINCAR 81
5.4 E-Q2 results of IER on MOUNTAINCAR 82
5.5 E-Q1 results of IER on MOUNTAINCAR 83
5.6 E-Q3 results of IER on MOUNTAINCAR 84

6.1 Shared hyperparameters of the CHAIN-N and FROZENLAKE task. 95
6.2 Shared hyperparameters of the MOUNTAINCAR and UMAZE task. . . . 95
6.3 Specific hyperparameters of FU-DQN for each specific task. 95
6.4 E-Q1 results of FU-DQN 97
6.5 E-Q2 results for FU-DQN 97
6.6 E-Q3 results for FU-DQN 99

A.1 E-Q1 results of all configurations on VE state-encoding 115
A.1 E-Q1 results of all configurations on VE state-encoding 116
A.2 E-Q1 results of all configurations on CE state-encoding 117
A.2 E-Q1 results of all configurations on CE state-encoding 118
A.3 E-Q1 results of all configurations on LKE state-encoding 118
A.3 E-Q1 results of all configurations on LKE state-encoding 119
A.4 E-Q2 results of all configurations on VE state-encoding 120

A.4 E-Q2 results of all configurations on VE state-encoding 121
A.4 E-Q2 results of all configurations on VE state-encoding 122
A.5 E-Q2 results of all configurations on CE state-encoding 123
A.5 E-Q2 results of all configurations on CE state-encoding 124
A.5 E-Q2 results of all configurations on CE state-encoding 125
A.6 E-Q2 results of all configurations on LKE state-encoding 126
A.6 E-Q2 results of all configurations on LKE state-encoding 127
A.6 E-Q2 results of all configurations on LKE state-encoding 128
A.7 E-Q3 results of all configurations on VE state-encoding 129
A.7 E-Q3 results of all configurations on VE state-encoding 130
A.8 E-Q3 results of all configurations on CE state-encoding 130
A.8 E-Q3 results of all configurations on CE state-encoding 131
A.8 E-Q3 results of all configurations on CE state-encoding 132
A.9 E-Q3 results of all configurations on LKE state-encoding 132
A.9 E-Q3 results of all configurations on LKE state-encoding 133

B.1 E-Q1 results of all IER configurations on MountainCar 143
B.1 E-Q1 results of all IER configurations on MountainCar 144
B.2 E-Q2 results of all IER configurations on MountainCar 144
B.2 E-Q2 results of all IER configurations on MountainCar 145
B.3 E-Q3 results of all IER configurations on MountainCar 146

List of Abbreviations

CE	**C**oordinates **E**ncoding
DNN	**D**eep **N**eural **N**etwork
DIKW	**D**ata–**I**nformation–**K**nowledge–**W**isdom hierarchy
DP	**D**ynamic **P**rogramming
DQN	**D**eep **Q**-**N**etwork
ER	**E**xperience **R**eplay
E-Q	**E**valuation-**Q**uestion
FU-DQN	**F**ull-**U**pdate **D**eep **Q**-**N**etwork
GA	**G**enetic **A**lgorithm
IC	**I**nterpolation **C**omponent
IDW	**I**nverse **D**istance **W**eighting
IER	**I**nterpolated **E**xperience **R**eplay
LCS	**L**earning **C**lassifier **S**ystem
LKE	**L**ocal **K**nowledge **E**ncoding
LS	**L**ast **S**tate
MAR	**M**inimum **A**verage **R**eward
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MDP	**M**arkov **D**ecision **P**rocess
ML	**M**achine **L**earning
MLI	**M**achine **L**earning **I**nterface
MLOC	**M**ulti-**L**ayer **O**bserver/**C**ontroler-architecture
NN	**N**earest **N**eighbor
NSE	**N**on-**S**tationary-**E**nvironment
ONS	**O**ne **N**ext **S**tate action selection
OnPol	**On**-**Pol**icy action selection
OnPolONS	**On**-**Pol**icy & **O**ne **N**ext **S**tate action selection
OTC	**O**rganic **T**raffic **C**ontrol
PD	Policy **Dist**ribution
RL	**R**einforcement **L**earning
ROPE	**R**egion **O**f **P**ractical **E**quivalence
SP	**S**ampling **P**oints
SuOC	**S**ystem **u**nder **O**bservation and **C**ontrol
TD	**T**emporal-**D**ifference
TTS	**T**ime **T**o **S**olution
VE	**V**ector **E**ncoding

Summary of Notation

s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all states (state-space)
$\mathcal{A}(s)$	set of all actions available in state s (action-space)
$\mathcal{R}$	set of all possible rewards, a finite subset of $\mathbb{R}$ (reward-space)
t	discrete time step
A_t	action at time t
S_t	state at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
R_t	reward at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
π	policy (decision-making rule)
$\pi(s)$	action taken in state s under *deterministic* policy π
$\pi(a\|s)$	probability of taking action a in state s under *stochastic* policy π
G_t	return following time t
$p(s', r\|s, a)$	probability of transition to state s' with reward r, from state s and action a
$p(s'\|s, a)$	probability of transition to state s', from state s taking action a
$r(s, a)$	expected immediate reward from a state s after an action a
$r(s, a, s')$	expected immediate reward on transition from s to s' under the action a
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or q_*
δ_t	temporal-difference (TD) error at t (a random variable)
α	step-size parameter $\alpha \in (0, 1]$; also known as learning rate
e_t	an experience at time t, of the form $(S_t, A_t, R_{t+1}, S_{t+1}, d)$
D_t	the replay buffer at time t, data set of recent experiences including e_t
$\theta, \theta_i, \theta_t$	vector of weights—parameters of a DQN
$Q(s, a; \theta)$	approximate value of state-action pair s, a given parameter θ of the Q-network
$\hat{Q}(s, a; \theta^-)$	approximate value of state-action pair s, a given parameter θ^- of the target network
$V(s, ; \theta)$	approximate value of state s given parameter θ

y_t	target values of a DQN loss at time t—typically $r + \gamma \max_{a'} Q(s, a')$
y^{DQN}	target values of a DQN loss under consideration of $\hat{Q}$
y^{DDQN}	target values of a double DQN
B	minibatch (multiset) of experiences
C	parameter for hard target update
τ	parameter for soft target updates
$L_i(\theta_i)$	loss function for optimizing parameters θ at iteration i
$\mathbb{V}(x)$	variance of x
∇_{θ_i}	gradient of the loss function with respect to weights θ
$\nabla_{\theta_i} Q(s, a; \theta_i)$	partial derivatives of $Q(s, a, ; \theta_i)$ with respect to θ at iteration i
$adv_\pi(s, a)$	advantage of taking action a in state s under policy π
Adv	array estimates of advantage adv_π
$Adv(s, a, ; \theta)$	approximate advantage of choosing action a in state s given parameter θ
P_i	a sampling point, defined as $P_i = (x_i, y_i) = (x_i, f(x_i))$
$f(x), g(x)$	functions
$w_i(x)$	weight
p_{idw}	power parameter for IDW
S_q	a query state for interpolation
A_q	a query action for interpolation
R_{syn}	synthetic generated reward
S_{syn}	synthetic generated state
d_{syn}	synthetic generated terminal tag
x_q	a query point for interpolation, defined as $x_q = (S_q, A_q)$
e_{syn}	an interpolated experience, defined as $e_{syn} = (S_q, A_q, R_{syn}, S_{syn}, d_{syn})$
NN_q	a set of nearest neighbours of x_q
F	a query function that generates x_q
$F_*(D_t)$	a concrete query function that generates x_q given the replay buffer D_t
$T(x_q)$	a transformation function that shifts the position of a given query point x_q
c_{slip}	slippery factor
$R_t^{\text{right}}, R_t^{\text{left}}, R_t^{\text{int}}$	rewards that correspond with c_{slip}
$R_t^{\text{exp}}, R_q^{\text{exp}}$	true expected reward
$e_t^{\text{exp}}, e_q^{\text{exp}}$	expected experience that includes R_t^{exp}
e_q^{avg}	estimation of e_q^{exp}
$\mathcal{R}_q$	set of rewards that correspond to x_q
R_q^{avg}	average reward that correspond to x_q
D_{match}	set of experiences that match $S_t == S_q$
D^a_{match}	set of experiences that match $S_t == S_q \land A_t == a$
$\mathcal{S}_{t+1}$	set of distinct follow-up states in D^a_{match}
D^{inter}	replay buffer for interpolated experiences
$s_{\text{start\_inter}}$	min length of real valued replay buffer before interpolation

s_{ier}	length of IER buffer
s_{er}	length of real valued buffer
$s_{\text{er\_max}}$	maximum length of real valued buffer
s_{syn}	length of interpolated buffer
$s_{\text{syn\_min}}$	minimum length of interpolated buffer at initialization
$s_{\text{syn\_min},i}$	minimum length of interpolated buffer at iteration i
ζ, ζ_i	parameter for fine tuning of $s_{\text{syn\_min},i}$
ζ_{init}	start value for ζ
M	episode when ζ_i reaches zero
$\mathcal{T}$	a learning tasks
p_i	solution rate on a task $\mathcal{T}$
$n_{\text{solved},i}$	number of times that $\mathcal{T}$ was solved by method i
n	number of performed runs
TTS_i	random variables of set of TTS units
α_i^γ	shape parameter
β_i	gamma distribution rate parameter
μ_i	mean parameter
$m_{\mathcal{T}}$	median performance of $\mathcal{T}$
l, u	lower/upper bound of prior on β_i
n	number of trials parameter
L	rebuild interval of a tree-structure
nn_{thresh}	distance threshold for NN_q
r_{sq}	radius for x_q
min_{spoints}	minimum required sampling points in order to perform an interpolation
L_B	loss of a minibatch B
$\tilde{B}_I$	synthetic batch
$I(D;\cdot,\cdot)$	generation function with replay buffer as input
$\mathcal{B}_I$	union of real and synthetic batch $\mathcal{B}_I = B \cup \widetilde{B}_I$
$\mathcal{L}_B$	loss of $\mathcal{B}_I$
r_*	synthetic reward
s'_*	syntehtic follow-up state

To my dear Maggy.

Chapter 1

Introduction

Artificial Intelligence has attracted great attention recently as ChatGPT [Ope23] (a chatbot that is able to conduct a conversation on a human level) showed great success even beyond the scientific world. ChatGPT is a mixture of several known concepts from the domain of *Machine Learning* (ML) [Mit97] that includes *Supervised Learning* [CCD08], which is training with huge amounts of labeled data, and *Reinforcement Learning* (RL) [SB18], which is learning based on a feedback signal.

A general definition of ML was provided by Mitchel, and states:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [Mit97]

Following this definition, a key component of ML is some form of experience. In Supervised Learning, this experience is manifested through labeled data provided to the learner. RL on the other hand, takes another route and generates experience based on interactions with an environment. While the generation of labeled data is an elaborate process, this is not necessary for RL as required data is generated during the learning process. Unfortunately, there are other problems like the design of meaningful and supportive reward functions for instance. However, RL represents a powerful way of learning that is inspired by nature, to be precise, by the way mammals learn when they interact with the world and adapt behaviour based on feedback signals such as pain or pleasure [BO10]. Moreover, RL has achieved remarkable success in playing (video) games such as Go [Sil+17], StarCraft II [Vin+19] and Dota 2 [Ber+19] at a level surpassing human capabilities. These examples highlight the considerable relevance of RL, which still holds vast untapped potential.

RL has been applied to real world scenarios as well, and examples range from the energy sector over healthcare to robotics [Li19]. Outside of simulations, aspects become important that have been negligible before, and one such thing is the amount of performed exploration. In RL, a so-called *agent* learns by interacting with the world it is situated in, but to gather knowledge about the dynamics of that world it needs to explore it first. By doing so it builds up a behaviour policy that maps concrete actions or a probability distribution over all possible actions to concrete situations. If the agent would only explore the environment by trying random actions for the whole time, that behaviour policy would not learn anything useful. Thus, the agent also exploits its knowledge base to intensify it. As both is required to evolve a good behaviour policy, RL agents typically implement some solution for the explore-exploit-dilemma. In other words, a trade-off needs to be found between exploring the environment with random actions and performing meaningful actions based on the behaviour policy. In real world scenarios, exploration can be costly of course, as it might break a robot if it decides to try out what happens when it runs full speed into a wall. Another important thing that can be of relevance is the execution time. A simulation can perform thousands of operations in no time, while a heavy robot is only able to conduct a handful of operations per minute. Both of the aforementioned points are related to the amount of generated experiences and would benefit from a low required number. One possible approach to reduce that number is to exploit the potential of already generated experiences, which is, in other words, the increase of the sample efficiency.

Experience Replay (ER) [Lin92; Lin93] describes a method that stores observed experiences in a buffer and replays them to the agent to increase sample efficiency and speed up learning. It has been extended in several ways (cf. section 3.1), and, for some approaches (like DQN [Mni+15; Mni+16] cf. section 2.4) it is even mandatory to learn at all. The classical method and most of its extensions are restricted to the usage of real experiences that have been generated by interaction with the environment, but there also exist some advancements that use synthetic data in the one or the other way. Experiences are stored as they are used for training, and it might be beneficial to make use of this stored information even beyond the replay mechanic. Experiences e_t are defined as a quin-tuple of state S_t, action A_t, reward R_t, next state S_{t+1} and a terminal tag d. Such an experience describes a transition of one state into another when a concrete action was executed and stores the received reward together with the information if the state S_{t+1} was a terminal one.

To look at an experience from another perspective, a concept that describes the relation of data, information, knowledge and wisdom is consulted and can be found in Figure 1.1. It shows the well known concept of the *data–information–knowledge–wisdom hierarchy* (DIKW) [Row07]. One key assumption of this hierarchy is that higher elements can be explained in terms of the lower elements by identifying an appropriate transformation. Starting at the bottom, data is described in [Row07] as something that has no meaning/value because of its missing context, or respectively as unorganized/unprocessed discrete objective facts or observations. The main thing defining data is the lack of meaning or value and that is exactly what the definition of information adds. Information is often defined in terms of data and some definitions claim it to be formatted data, data which adds value to the understanding of a subject or data that has been shaped into a meaningful form. The key concept of information is the transformation of data into a contextualized form that has value. In other words, information can be described as organized or structured data. Definitions of knowledge are much more complex, but to keep it simple, they can be reduced to describing knowledge as the combination of data and information to which expert opinion, skills and experience is added, such that it can be used to aid decision making, and/or convey understanding, experience, accumulated learning and expertise if applied to a problem or activity. At the top of the DIKW hierarchy stands wisdom, which is defined as accumulated knowledge that helps with understanding of how to apply concepts from one domain to new situations or problems. [Row07]

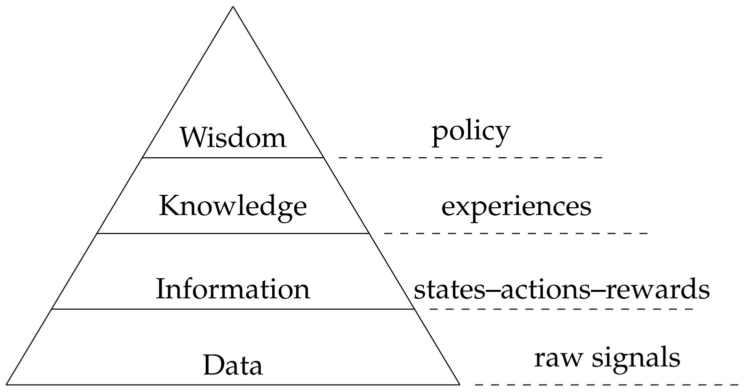


FIGURE 1.1: The DIKW hierarchy, adapted from [Row07]

If a RL method is analyzed with respect to the DIKW hierarchy, then, first of all, raw measurements of signals can be described as data. In the moment when these signals are combined with context, they form a state observation, a reward and an action and emerge from data to information. Just these contextualized data on its own can not be used further on, because it lacks an meaningful combination that takes the temporal dependencies of them into account. By doing so, an experience is generated and consequently, a pool of experiences can be described as knowledge about the underlying problem. Experiences are used to train a policy that is able to solve this problem and, depending of the utilized estimation method (e.g. a neural network), can generalize the learned behaviour to new situations. Thus, a policy fits the definition of wisdom.

As mentioned above, stored experiences resemble gathered knowledge of the underlying problem which is to be solved. The assumption here is that this knowledge, that is stored either way, can be exploited to resemble knowledge (experiences) in yet unexplored areas of the problem space. Further on, it is assumed that the problem's dynamics are at least similar in local neighbourhoods and, therefore, experiences hold knowledge of neighbouring dynamics. A technique that uses data points in a local neighbourhood to generate estimates of points that are unknown is interpolation. Consequently, the assumption is that stored experiences can be used as sampling points to generate synthetic experiences by means of interpolation.

A concrete overview of the investigated problem and the implemented solutions alongside a definition of several research questions is given in the subsequent section.

1.1 Scientific Contribution

Based on the presented thoughts in the former section, the following hypothesis can be formulated that will be in the main focus of the rest of the work:

Hypothesis 1. *Stored samples in the Experience Replay buffer can be used as sampling points for the interpolation of synthetic experiences. These synthetic experiences can be used to assist the learner to speed up learning and increase sample efficiency.*

In order to establish the validity of hypothesis 1, it becomes crucial to identify pertinent research questions that can be explored to examine various aspects of the hypothesis. The subsequent research questions are highly specific and will serve as a guide for scientific contributions that have attempted to address one or more of these questions. By addressing all of these questions, it becomes possible to provide a well-founded and scientific response to the inquiry regarding the truthfulness of hypothesis 1.

Q 1. *How can gathered knowledge in form of experiences be exploited to generate meaningful synthetic experiences?*

Q 2. *Which parts of an experience $e_t = (S_t, A_t, R_{t+1}, S_{t+1}, d_{t+1})$ can be interpolated, and what needs to be considered by doing so?*

Q 3. *How can synthetic experiences be used to assist a Deep Q-Network with learning?*

Q 4. *Does the usage of synthetic experiences in combination with a Deep Q-Network result in faster convergence and consequently in an increased sample efficiency?*

Q 5. *Is the combination of model-free Reinforcement Learning methods that make use of Experience Replay with synthetic experiences that have been generated by means of interpolation classifiable as model-free or model-based Reinforcement Learning?*

Two main concepts will be presented in this work that have been designed with the above questions in mind. The first one is called *Interpolated Experience Replay* (IER) and represents an advancement of the original ER mechanic in combination with an interpolation component that generates synthetic experiences based on stored samples that have been generated by interaction with the environment. As an initial proof of concept and with **Q** 2 in mind, the first version of the IER only interpolates rewards and uses observed follow-up states of discrete environments.

The next approach interpolates complete synthetic experiences and makes the IER ready for continuous problems with continuous state spaces.

The last presented method changes the focus from the ER to the update mechanics of a DQN. While for standard DQNs a loss is computed only for that actions that have been chosen in corresponding experiences, it is actually possible to compute a loss for the whole action space. This is not done in standard DQN because the required values of the rewards and the follow-up states are simply not available at the time of the update. *Full-Update DQN* (FU-DQN) generates these missing values by means of interpolation based on stored experiences as sampling points and is able to perform updates that cover the whole action space.

Q 1 to **Q** 4 are all part of IER and FU-DQN and will be answered accordingly. **Q** 5 is addressed separately with an in-depth analysis of methods that combine interpolation with ER and how they fit the definition of model-based respectively model-free RL. As a result the new class *Semi-Model-Based* is introduced.

1.2 Relation to Organic Computing

Organic Computing [MT17; TSM17] describes the design of "life-like" technical systems with so-called *self-* properties*. An OC system therefore has the ability to act based on its own decisions. A related topic is Autonomic Computing, which uses the biological principle of the autonomic nervous system as paradigm. Each OC system is equipped with sensors and actuators and is located above a system that should be controlled. Such a supervised system is typically called System under Observation and Control (SuOC) in the OC domain [MSU11]. Sensors deliver an observation of the actual system state and actuators can be used to change this state. The system can adapt autonomously to the received observation, and the reaction has to be in a way that the system remains functional. To fulfill this requirement, even in, yet unseen states and unanticipated conditions, an OC system is typically based on (machine) learning.

Another key concept of OC systems is that they are robust to changing world dynamics. Environments that do so are called *non-stationary* environments and real world scenarios are typically such. OC systems are designed to be applied to real world use cases and beyond the possibility of (constantly) changing conditions and dynamics such environments come with several other problems. While measurements can be trusted in a simulation, they suffer from at least some uncertainty in the real world and this uncertainty may even increase over time when sensors are not recalibrated. RL as a learning technique is able to learn online which describes the process of learning constantly, even after deployment. While Supervised Learning is also capable of learning new things after the system was deployed, this is typically realized by retraining at fixed intervals. As RL generates its experiences on the fly it is much better prepared for lifelong learning. Consequently, RL is a good choice for OC systems. The fact that measurements come with uncertainty in real world scenarios results in less reliable experiences that are generated by interaction with the environment. These *real* experiences are typically the most reliable ones in a simulation that can be trusted. However, this changes of course when they suffer from uncertainty. Synthetic experiences that are generated by interpolation can be more reliable as they take all the noisy real samples as input. By doing so, such synthetic experiences are capable of reducing the noise that is induced by the measurement uncertainty. Following this argumentation, synthetic experiences that are generated by interpolation can be of great benefit for OC systems applied to the real world. As discussed above, RL requires at least some exploration and this can be costly in real world scenarios. An increased sample efficiency and corresponding reduced exploration would be a good thing to have in OC systems operating out there.

In summary, it can be stated that interpolation assisted deep RL shows an obvious relation to OC and OC systems. Furthermore, it can be said that it even is of high relevance for this domain.

1.3 Structure

The structure of the following thesis is as follows: First, there will be an extensive overview of relevant fundamental knowledge such as finite Markov Decision Processes and the basics of Reinforcement Learning. This is covered in chapter 2, together with the presentation of model-free RL algorithms like Q-Learning and DQN and basics about interpolation and nearest neighbour searches. In chapter 3, a summary of some related work is provided. Afterwards, the main part of the thesis covers two different ways of how to make use of synthetic experiences that have been generated by means of interpolation in combination with a DQN. The first approach is the Interpolated Experience Replay that is split into a version that is designed towards discrete and non-deterministic environments (chapter 4) and a version that is ready for continuous environments (chapter 5). The second approach is the Full-Update DQN that enables updates for the whole action space which is presented in chapter 6. The class semi-model-based is introduced in chapter 7 to classify approaches that combine model-free RL algorithms with synthetic experiences and replay buffers. The thesis is concluded in chapter 8 with a final evaluation of the identified research questions and and answer to hypothesis 1.

1.4 Integration of the Authors Published Works

The doctoral thesis incorporates previously published scientific contributions, where the author of the thesis played a leading role. The subsequent paragraphs briefly outline the integration of these publications into specific chapters of the thesis.

In chapter 4 an initial proof of concept implementation of Interpolated Experience Replay, limited to discrete and non-deterministic environments is introduced. First ideas and a small evaluation study were presented at a doctoral symposium [Pil19]. Subsequently, a conference contribution [PSH20] provided a more in-depth evaluation using a more difficult version of the investigated problem, employing small neural networks and linear regression. A journal paper [PSH21] concluded the proof of concept evaluation, encompassing a comprehensive assessment of various state-encodings and deep neural networks.

In chapter 5, the focus is set on extending the Interpolated Experience Replay for continuous state-spaces. It introduces a solution involving the interpolation of the state-transition delta instead of raw values. The final version of Interpolated Experience Replay is presented in this chapter. An evaluation of this technique, along with other minor extensions, took place in the context of the MOUNTAINCAR environment, and the findings were published as a conference paper [PSH22b].

While the aforementioned publications concentrate on the usage of interpolated experiences in the replay buffer, an alternative approach leverages the architectural design of Deep Q-Networks. This approach employs synthetic experiences to facilitate training updates that consider the entire action space. This algorithm is introduced in chapter 6, referred to as Full Update-DQN, and its details were presented in a conference paper [Pil+23].

A theoretical discussion on approaches combining synthetic experiences generated through interpolation with a replay buffer, and how they align with the definitions of model-based and model-free Reinforcement Learning is provided in chapter 7. It becomes apparent that these methods do not fit strictly into either category, but rather occupy a space in between. As a result, the new class Semi-Model-Based is introduced, and this classification was presented in a conference paper [PSH22a].

Chapter 2

Fundamentals

In this section, some fundamental knowledge is provided. At first, Markov Decision Processes are introduced and put into relation to Reinforcement Learning. Next, temporal-difference learning is discussed in detail alongside Q-Learning as a prominent algorithm from that field. Afterwards the definitions of model-based and model-free RL is provided and followed by the introduction of the deep Q-network algorithm which is the main used algorithm of this thesis. In the end, a short presentation of the term interpolation is provided as well as an overview of some nearest-neighbor search algorithms.

2.1 Finite Markov Decision Processes

In this section, that is mainly based on Sutton and Bartos standard literature for Reinforcement Learning [SB18], Finite Markov Decision Processes (MDPs) are introduced. MDPs can be used to formalize sequential decision making problems and most of the problems solved by Reinforcement Learning (RL) follow this structure. MDPs are mathematically idealized and can be used to make precise theoretical statements.

The following sections will introduce key elements like returns, value functions, and Bellman equations.

2.1.1 The Agent–Environment Interface

In general, MDPs can be seen as a straightforward framing of the problem of learning from interaction to achieve a goal. The entity that learns and is responsible for the decision making process is called *agent*. It is situated in the so-called *environment* which concludes everything outside of it. In a continually way, both elements interact with each other. The agent selects actions and the environment responds to them by presenting new situations. An important metric that is also returned from the environment in reaction to a selected action is the so-called *reward*. The reward can be described as a special numerical value that the agent seeks to maximize over time through its choice of actions. The agent–environment interaction cycle, also known as the classic RL cycle can be observed in Figure 2.1.

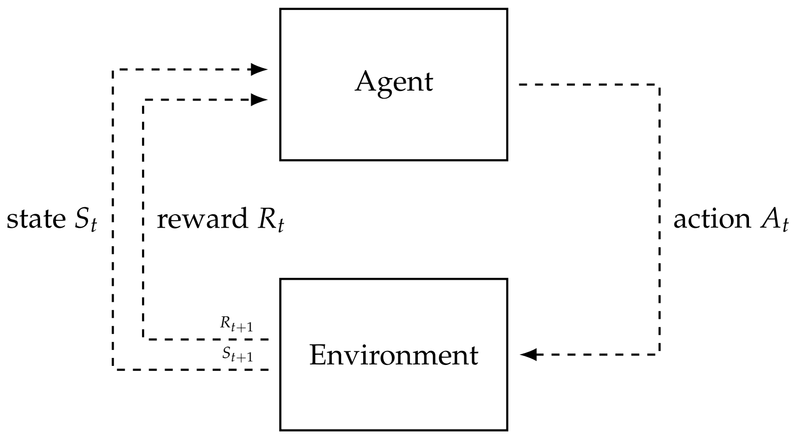


FIGURE 2.1: The agent–environment interaction in a MDP.

In more detail, it can be said that agent and environment interact with each other at discrete time steps $t = 0, 1, 2, 3, \dots$ of a (finite) sequence. At each time step t, the agent receives a representation of the environment's *state*, $S_t \in \mathcal{S}$, and selects an *action* $A_t \in \mathcal{A}(s)$ as reaction. The action-space $\mathcal{A}(s)$ can be shortened to $\mathcal{A}$ when it is the same for every state. One iteration after, in $t = t + 1$, as a consequence to the action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and is relocated to a new state, S_{t+1}. Repeatedly execution of this cycle generates a so-called *trajectory* that is of the following form:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots \tag{2.1}$$

A *finite* MDP is defined by a finite number of elements assigned to the state-, action- and reward-space ($\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$). The random variables R_t and S_t have well defined discrete probability distributions in this case, that only depend on the preceding state and action. For particular values of these random variables $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ a probability exists that they occur at time t, given particular values of the preceding state and action:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{2.2}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$. The function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ defines the *dynamics* of the MDP and is an ordinary deterministic function of four arguments. It specifies a probability distribution for each choice of s and a that sums to 1:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{2.3}$$

The dynamics of a *Markov* Decision Process are completely characterized by the probabilities given by p. Therefore, the probability of each possible value for S_t and R_t depends only on the immediate preceding state and action, S_{t-1} and A_{t-1}. This can be seen as a restriction on the state, and it must include all relevant information of the past agent–environment interaction that make a difference for the future. If this is fulfilled, then the state is said to have the *Markov Property*.

With the four-argument dynamics function p as a basis it is possible to compute any important information of the environment. Those are:

1. The *state-transition-probabilities*, denoted as a three-argument function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$p(s'|s,a) \doteq \Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a), \tag{2.4}$$

2. the expected rewards for state-action pairs, denoted as a two-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r|s, a), \tag{2.5}$$

3. and the expected rewards for state-action-next-state triples, denoted as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s,a,s') \doteq \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}. \tag{2.6}$$

2.1.2 Rewards, Returns and Episodes

As already briefly introduced above, the goal of an agent is formalized in terms of the reward signal. The reward is a scalar value received in each time step $R_t \in \mathbb{R}$. Informally speaking, it can be stated that the agent's goal is to maximize the total amount of receiving rewards. This means that it needs to maximize cumulative rewards in the long run. Sutton and Barto state this informal idea as the *reward hypothesis*:

> "That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)." [SB18, p. 53]

In a more formal way, the agent seeks to maximize the *expected return*, where the return, denoted as G_t, is defined as some specific function of the reward sequence, and is described as the sum of the rewards in the simplest case:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots R_T, \tag{2.7}$$

where T is a final step. A natural notion of the final time step is required for this approach, and that is the case when the agent–environment interaction breaks naturally into subsequences called *episodes*. The final state of such an episode is called *terminal state* which is followed by a reset of the environment to a starting state or to a sample from a standard distribution of starting states. A new episode starts independently of the outcome of the last episode. Even if the rewards of the final states may be different, all episodes can be considered to end in the same terminal state. Tasks with episodes of this kind are called *episodic tasks*. The time of termination T, is considered to be a random variable that normally varies from episode to episode.

However, many problems are of continuous nature, with the agent-environment interaction not breaking naturally into identifiable episodes. These tasks are called *continuing tasks*. The formulation of the return from above (i.e., Equation 2.7) is problematic for such tasks, as the final time step would be $T = \infty$, and the return itself (which is the optimization target) could easily be infinite. To counteract this, a slightly more complex conceptually but much simpler mathematically definition of the return is used instead. Therefore, the concept of *discounting* is introduced. Instead of the sum of rewards, the sum of discounted rewards is maximized. In particular, the agent chooses A_t such, that the expected *discounted return* is maximized:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.8}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

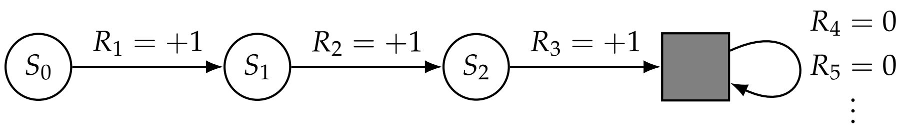
The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. As long as $\{R_k\}$ is bounded, the infinite sum of Equation 2.8 has a finite value for $\gamma < 1$. Whereas, the agent is "myopic" in being concerned only with maximizing immediate rewards for $\gamma = 0$. In this case, the agent's objective is to choose A_t in a way that maximizes only R_{t+1}. A myopic agent could maximize Equation 2.8 by separately maximizing each immediate reward, but in general, this approach can reduce access to future rewards so that the return is reduced. Overall, the agent becomes more farsighted as γ approaches 1, as the return objective takes future rewards into account more strongly.

Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of RL:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.9}$$

If the return for the final state is defined as $G_T = 0$, then this works for all time steps $t < T$, even if termination occurs at $t + 1$.

To obtain a notation that covers both, episodic as well as continuing tasks, the following convention is introduced. Episode termination is considered as the entering of a special *absorbing state* that transitions only to itself and generates rewards of zero. The following state transition diagram sketches the idea:



The special absorbing state is represented by the solid square to the end of an episode. A reward sequence of the following form is produced by the depicted trajectory: $+1, +1, +1, 0, 0, 0, \ldots$, and the sum remains the same for the first T rewards (here $T = 3$) or the full infinite sequence. This holds true, even if discounting is introduced. Thus, the return is defined in general according to Equation 2.8. Alternatively it can be written

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{2.10}$$

including the possibility that $T = \infty$ or $\gamma = 1$ (but not both).

2.1.3 Policies and Value Functions

An important component of almost all RL algorithms are so called *value functions* – functions of states (or of state-action pairs) that estimate *how good* it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The quality thereby, is measured in terms of expected future rewards, or, to be precise, in terms of expected return. As the rewards that can be expected from an agent in the future directly depend on the chosen actions, value functions are defined with respect to particular ways of acting, called policies.

A *policy* can formally be described as a mapping from states to probabilities of selecting each possible action. Therefore, $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$, if the agent is following policy π at time t. RL methods specify how the agent's policy is changed as a result of its experience.

The *value function* of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, v_π is formally defined by

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \text{ for all} s \in \mathcal{S}, \tag{2.11}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. The function v_π is called the *state-value-function for policy* π.

Another, more concrete, form of value function is known as the *action-value function for policy* π. This function is defined as $q_\pi(s, a)$ and assigns a value to the execution of an action a in a state s under a policy π. The assigned value is defined as the expected return starting from s, taking the action a, and thereafter following policy

π.

$$q_\pi(s,a) \doteq \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{K=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \tag{2.12}$$

The value functions v_π and q_π can be estimated from experience. Therefore, an agent traverses the state-space according to a policy π and simultaneously, for each state, stores the encountered immediate rewards that follow that state and computes the average of that rewards. By doing so, these averages will converge to the states values $v_\pi(s)$, as the number of times that the states are encountered approaches infinity. The same thing holds for the action-values $q_\pi(s,a)$, if separate averages are kept for each action taken in each state. Estimation methods of this kind are called *Monte Carlo methods* because they involve averaging over many random samples of actual returns. In fact, it may not be practical to keep separate averages for each state individually, if there are very many states. Alternatively, the agent could maintain v_π and q_π as parameterized functions (with fewer parameters than states) and adjust the parameters to better match observed returns. This approach can be used to produce accurate estimates, although much depends on the nature of the parameterized function approximator.

Similar to the return in Equation 2.9, a fundamental property of value functions used in the context of RL is that they satisfy recursive relationships. For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

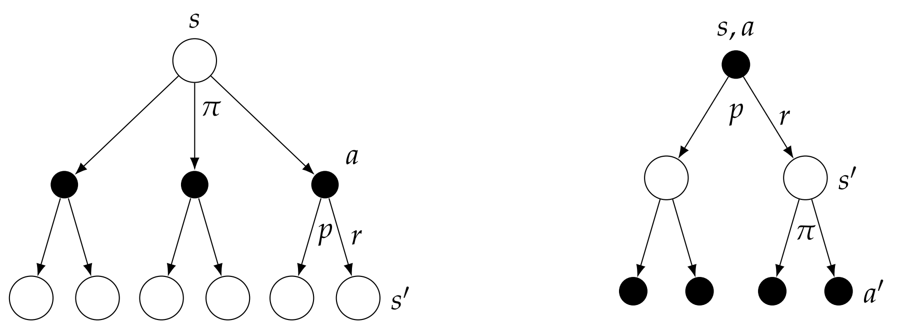
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \middle| S_{t+1} = s']\right] \\ &= \sum_a \pi(s|a) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_\pi(s')\right], \text{ for all } s \in \mathcal{S}, \end{aligned} \tag{2.13}$$

where it is implicit that the actions, a, are taken from the set $\mathcal{A}(s)$, that the next states, s', are taken from the set $\mathcal{S}$, and that the rewards, r, are taken from the set $\mathcal{R}$. The final expression can be read easily as an expected value, as it is really a sum over all values of the three variables, a, s' and r. For each triple, its probability $\pi(a|s)p(s',r|s,a)$ is computed, the quantity in brackets is weighted by that probability, and the sum over all possibilities is calculated to get an expected value.

Equation 2.13 is known as the *Bellman equation* for v_π, and expresses a relationship between the value of a state and the values of its successor states. A graphical illustration of this relationship is depicted in Figure 2.2a. States are encoded in open circles and state-action pairs in solid ones. In general, its about the concept of looking ahead from a state to its possible successor states. Starting at the top, at the root node which is the state s, the agent has the possibility to choose one of the actions from the set $\mathcal{A}(s)$—three are shown in the diagram—based on the policy π. Each action would trigger the environment to respond with one of several next states, s' (two are shown in the figure), along with a reward, r, depending on its dynamics given by the function p. The Bellman equation (Equation 2.13) averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

The value function v_π is the unique solution to its Bellman equation, and this equation forms the basis of a number of ways to compute, approximate, and learn v_π. Diagrams like that in Figure 2.2 are called *backup diagrams*, and that is because they diagram relationships that form the basis of the update or *backup* operations that are at the heart of RL methods. Operations of such kind transfer value information *back* to a state (or a state-action pair) from its successor states (or state-action pairs).

All of the above deduced behavior for v_π also holds for the action-value function q_π, whereas the corresponding backup diagram is shown in Figure 2.2b.



(A) Backup diagram for v_π

(B) Backup diagram for q_π

FIGURE 2.2: Backup diagrams for value functions

2.1.4 Optimal Policies and Optimal Value Functions

Roughly speaking, solving a RL problem can be described as finding a policy that achieves a lot of reward over the long run. To be more precise, for finite MDPs, an optimal policy can be defined as follows. First of all, value functions define a partial ordering over policies. Therefore, a policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. Formally, this can be expressed with $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is better than or equal to all other policies. This policy is called an *optimal policy*. Even if there might be more than one of these policies, all of them are denoted by π_*. They share the same state-value function, called the *optimal state-value function*, denoted v_*, and defined as

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{2.14}$$

for all $s \in \mathcal{S}$. And similarly, they share the same *optimal action-value function*, denoted q_*, and defined as

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \tag{2.15}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. As the optimal action-value function is defined to be the expected return for taking action a in state s and thereafter following an optimal policy, q_* can be written in terms of v_*:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]. \tag{2.16}$$

As v_* is the value function for a policy, the self-consistency condition given by the Bellman equation for state values (Equation 2.13) still holds. However, because it is the optimal value function, it's consistency condition can be written in a special form without reference to any specific policy. This Bellman equation for v_* is known as the *Bellman optimality equation*. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

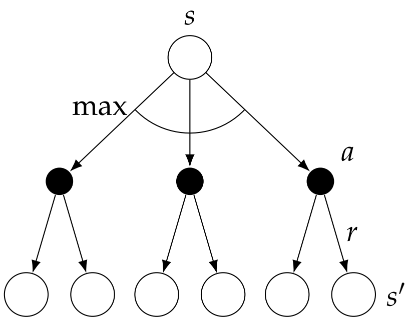
$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]. \end{aligned} \tag{2.17}$$

The equations in the last two lines are two forms of the Bellman optimality equation for v_*. Similar, the Bellman optimality equation for q_* is

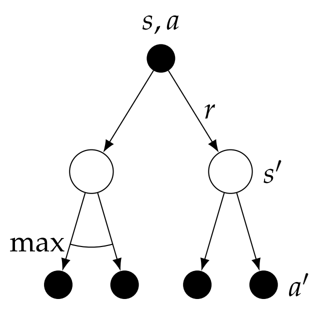
$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \,\middle|\, S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]. \end{aligned} \tag{2.18}$$

Figure 2.3 shows the spans of future states and actions considered in the Bellman optimality equation for v_* and q_*. Arcs have been added at the agent's choice points to indicate that the maximum over that choice is taken rather than the expected value given by some policy. Thus, Figure 2.3a represents Equation 2.17 and Figure 2.3b represents Equation 2.18.

For finite MDPs, the Bellman optimality equation for v_* (Equation 2.17) has a unique solution independent of the policy. Actually, it is a system of equations, one for each state, so, for n states, there are n equations in n unknowns. In principle, given the dynamics p of the environment, it is possible to solve this system of equations for v_* using anyone of a variety of methods for solving systems of nonlinear equations. This also holds for a system of equations for q_*.



(A) Backup diagram for v_*



(B) Backup diagram for q_*

FIGURE 2.3: Backup diagrams for optimal value functions

In fact, it is relatively easy to determine an optimal policy if v_* is known. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Therefore, any policy that assigns nonzero probability only to those actions is an optimal one. This can be imagined as a one-step search. So, given v_*, the best actions after a one-step search will be the optimal ones. In other words, any policy that is *greedy* with respect to v_* is an optimal policy. The term greedy is used in the context of computer science to describe any search or decision procedure that selects alternatives based only on local or immediate considerations, without considering the possibility that such a selection may prevent future access to even better alternatives. In terms of policies, a policy is then greedy, if it selects actions based only on their short-term consequences. However, if v_* is used to evaluate the short-term consequences of actions—specifically, the one-step consequences—then a greedy policy is actually optimal in the long-term sense, because v_* already takes the reward consequences of all possible future behavior into account. This means that the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step ahead search yields the long-term optimal actions.

Choosing optimal actions based on q_* is even easier. The agent can skip the one-step ahead search, as for any state s, it can simply find any action that maximizes $q_*(s, a)$. In fact, the results of all one-step-ahead searches are cached in the action-value function, which provides them as the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

Even if solving the Bellman optimality equation provides a route to finding an optimal policy, and therefore a solution to the RL problem, it is rarely directly useful. The reason for that is the nature of this solution which is akin to an exhaustive search. Consequently, RL problems are typically solved by approximate solutions, and many RL methods can be clearly understood as approximately solving the Bellman optimality equation, using actual experienced transitions in place of knowledge of the expected transitions. Moreover, the online nature of RL makes it possible to approximate frequently encountered states, at the expense of less effort for infrequently encountered states, and this is one key property that distinguished RL from other approaches to approximately solving MDPs.

2.2 Temporal-Difference Learning

An idea that can be described as central and novel to RL, is the so called *temporal-difference* (TD) learning. It combines ideas from the domain of Monte Carlo and dynamic programming (DP). Similar to Monte Carlo methods, TD approaches can learn directly from raw experience without the need of a model of the environment's dynamics. Similar to DP, they bootstrap, which describes the process of updating estimates based in part on other learned estimates, without waiting for a final outcome. The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of RL, and these ideas and methods blend into each other and can be combined in may ways.

The following sections, that are mainly based on [SB18], will give a theoretical overview of the policy evaluation or *prediction* problem—how to estimate v_π for a given policy π—and the *control* problem—how to find an optimal policy.

2.2.1 Reminder: Dynamic Programming and Monte Carlo

As the terms dynamic programming and Monte Carlo are of some relevance for the following sections, here is a small overview of the corresponding ideas.

Dynamic Programming

The term *dynamic programming* refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP. In general, such methods are of limited utility in RL, but still important theoretically. As stated above, a key concept is the usage of a perfect model. With such a model at hand (as discussed in subsection 2.1.4) it is possible to develop optimal value functions and based on them an optimal policy. DP algorithms are obtained by turning Bellman equations into assignments, that is, into update rules for improving approximations of the desired value functions.

Monte Carlo Methods

In contrast to DP, *Monte Carlo* methods do not assume complete knowledge of the environment. The term Monte Carlo is strongly correlated with the idea of sampling. In the context of RL, this means that sequences of states, actions, and rewards from actual or simulated interaction with an environment are sampled. In other words, Monte Carlo methods learn from *experience*. Learning from experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behavior. As mentioned above, the key concept that differs from DP is that no perfect model is required anymore. In terms of methods, the main concepts stay the same, and therefore, ideas from DP can be extended to the Monte Carlo case with only experience at hand.

In a bit more detail, Monte Carlo methods are ways of solving the RL problem based on averaging sample returns. Therefore, experience is divided into episodes that end at some point in time. The actual learning (the update of value functions and policies) happens when an episode has come to its end. Generally, the term Monte Carlo is often used more broadly for any estimation method whose operation involves a significant random component. In terms of RL, it describes methods that are based on averaging complete returns.

2.2.2 TD Prediction

As mentioned before, TD learning combines ideas from DP and Monte Carlo, and similar to the latter, TD methods use experience to solve prediction problems. Both, TD, as well as Monte Carlo methods, update their estimate V of v_π for the nonterminal state S_t occurring in some given experience following a policy π. As Monte Carlo methods rely on sampled sequences of transitions, they have to wait until the return that follows the visit is known, until they can use that return as a target for $V(S_t)$. A simple every-visit Monte Carlo method suitable for nonstationary environments is

$$V(S_t) \leftarrow V(S_t) + \alpha\big[G_t - V(S_t)\big], \tag{2.19}$$

where G_t is the actual return following time t, and α is a constant step-size parameter (also called *learning rate*). As mentioned above, Monte Carlo methods have to wait until the episode is terminated, and that is because only then the value of G_t is known. In contrast, TD methods only need to wait for exactly one time step. At time $t+1$ they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha\big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big] \tag{2.20}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. Thus, the target from the Monte Carlo approach is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$. This TD method is known as *TD(0)*, and because its update is based in part on an existing estimate, its said that it is a *bootstrapping* method, like DP. Following Equation 2.9 v_π is defined as follows:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\big[G_t \big| S_t = s\big] \\ &= \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1} \big| S_t = s\big] \\ &= \mathbb{E}_\pi\big[R_{t+1} + \gamma v_\pi(S_{t+1} \big| S_t = s)\big]. \end{aligned} \tag{2.21}$$

Roughly speaking, Monte Carlo methods use an estimate of the first line in Equation 2.21 as a target, whereas DP methods use an estimate of the last line in Equation 2.21 as a target. The target of the Monte Carlo methods is considered an estimate because the expected value is not known, and a sample return is used in place of the expected return. The target of the DP is considered an estimate because $v_\pi(S_{t+1})$ is not known and the current estimate , $V(S_{t+1})$, is used instead. Consequently, the TD target is an estimate because of both reasons: it samples the expected values *and* it uses the current estimate of V instead of the true v_π. It can be seen, that TD methods combine the sampling of Monte Carlo with the bootstrapping of DP.

Figure 2.4 illustrates the backup diagram for tabular TD(0). Based on one sample transition from the state node at the top to the immediate following state, the value estimate of that state is updated. TD (and Monte Carlo) updates are referred to as *sample updates*, because they update the value of the original state (or state-action pair) after looking ahead to a sample successor state (or state-action pair), and using the value of the successor and the reward along the way to compute a backed-up value. *Sample* updates differ from the *expected* updates of DP methods in that they are based on a single sample successor rather than on a complete distribution of all possible successors.

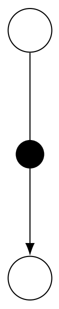


FIGURE 2.4: Backup diagram for TD(0)

The quantity in brackets in Equation 2.20 is some sort of error that measures the difference between the estimate of the state-value of S_t, in form of $R_{t+1} + \gamma V(S_{t+1})$ and the actual estimate of this value $V(S_t)$. This quantity is known as *TD error*, and is an important concept in RL:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{2.22}$$

The TD error at each time t is the error in the estimate *made at that time*. Moreover, because it depends on the next state and the next reward, the TD error is not actually available until one time step later. Thus, δ_t is the error in $V(S_t)$, available at $t+1$. If the array V does not change during the episode (which is the case for Monte Carlo methods), then the Monte Carlo error can be written as a sum of TD errors:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) \qquad (2.23) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0-0) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k. \end{aligned}$$

In TD(0), V is updated during the episode and the above identity therefore is not exact. However, if the step size is small then it may still hold approximately.

2.2.3 Exploration vs. Exploitation

When it comes to learning from experience instead of the utilization of a perfect model of the environment's dynamics, then a crucial question to solve is how this experience is generated. As mentioned in subsection 2.1.4, an advantage of (online) RL is the focus on frequently encountered states, at the expense of less effort for infrequently encountered states. In the best case, it should be ensured that all relevant or important states are under the more frequently encountered ones.

If an estimate of the action-values is maintained, then at any time step there is at least one action whose estimated value is greatest, and such actions are called *greedy* actions. By selecting one of these actions, it is said that the current knowledge of the values of the actions is *exploited*. In contrast, the selection of one of the nongreedy actions is called *exploring*, and that is because this enables the improvement of the estimation of a nongreedy action-value. On the one hand, exploitation is the right thing to do to maximize the expected reward based on actual knowledge, but on the other hand, exploration may produce the greater total reward in the long run. In general, reward is lower in the short run, during exploration, but higher in the long run because after better actions have been discovered, they can be exploited. In multi-step problems, this mechanism is also crucial to overcome local optima. Because it is not possible to do both, explore and exploit, at the same time, this is generally known as the *explore-exploit-dilemma*.

The question of how to balance the two parts, and consequently find an answer to the explore-exploit-dilemma is a distinctive challenge in RL and dozens of approaches exist that try to answer this question (e.g. random exploration, Boltzmann exploration [Sut90] and noisy nets [For+17] to name a few). The present work relies on a very simple approach called *ϵ-greedy*. This exploration strategy introduces the parameter ϵ which is a probability, and in every step the agent explores with that probability or exploits with the probability $1 - \epsilon$. The parameter ϵ generally starts with a high value $0 \leq \epsilon \leq 1$ and decreases over time to either zero or a very small amount that ensures at least small exploration during the whole training.

2.2.4 Q-Learning: Off-policy TD Control

TD(0) offers a solution for the problem of how to estimate v_π for a given policy, but does not tackle the control problem (find an optimal policy). However, TD prediction methods provide a solid basis that is necessary for TD control methods. As motivated in subsection 2.2.3, exploration and exploitation need to be balanced. Furthermore, TD control approaches can be distinguished into two main classes: *on-policy* and *off-policy*. An example for off-policy TD control is SARSA [SB18].

An important breakthrough in RL was the development of an off-policy TD control algorithm which is known as Q-Learning [WD92]. This algorithm makes use of the concepts introduced above and is defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big], \tag{2.24}$$

with α being a constant step-size parameter $\alpha \in (0, 1]$ that is also widely known as learning rate. The special feature of this algorithm is that the learned action-value function Q, directly approximates q_*, which is the optimal action-value function, independent of the policy being followed. This characteristic dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy is still important, as it determines the distribution of the visited state-action pairs that are updated. However, the one thing that is required for correct convergence is that all pairs continue to be updated, and this is a minimal requirement in the sense that any method guaranteed to find optimal behavior in the general case must require it. Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-use parameters, Q has been shown to converge with a probability 1 to q_*.

Figure 2.5 shows the backup diagram for the Q-Learning algorithm. The update for the state-action pair at the top is computed from the maximum of all actions that are possible in the corresponding next state.

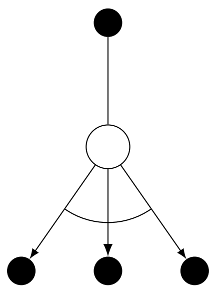


FIGURE 2.5: Backup diagram for Q-Learning

In its simplest form, the action-value function of the Q-learning algorithm is approximated by a table which is then called the Q-table. During training, entries in that table, that represent action-values of concrete state-action pairs, are updated one per time step.

2.2.5 Double Q-Learning

An important concept of the Q-Learning algorithm is that it uses maximization in the construction of its target policy. This target policy is the greedy policy given the current action values, which is defined with a max operator. A maximum over estimated action-values is used implicitly as an estimate of the maximal action-value, and this procedure can lead to significant overestimation (positive bias). In a simple example a single state s has many actions a whose true values $q(s,a)$ are all zero. The estimated values $Q(s,a)$ however are uncertain and thus distributed some above and some below zero. The maximum of the true values is zero, but the maximum of the estimates is positive, a positive bias, which is called *maximization bias*.

The problem can be viewed in the way that it is due to using the same samples both to determine the maximizing action and to estimate its value. A possible solution is to divide the samples (transitions) in two sets and use them to learn two individual estimates. The two estimates are called $Q_1(s,a)$ and $Q_2(s,a)$ and are both an estimate of the true value $q(s,a)$. One estimate, say Q_1, can be used to determine the maximizing action $A^* = \arg\max_a Q_1(s,a)$, and the other, Q_2, to provide the estimate of its value, $Q_2(s, A^*) = Q_2(\arg\max_a Q_1(s,a))$. It is possible to receive an unbiased estimate that way, so that $\mathbb{E}\left[Q_2(s, A^*)\right] = q(s, A^*)$. If this process is alternating both estimates, then two unbiased estimates can be obtained. This idea is called *double learning* and the double Q-Learning algorithm is defined such that in every step there is a 50 % possibility of the following update:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2(S_{t+1}, \arg\max_a Q_1(S_t, a)) - Q_1(S_t, A_t) \right], \tag{2.25}$$

and the same probability for the estimates being swapped.

2.2.6 Experience Replay

TD control methods, especially Q-Learning, make use of experience, and the way they generate this experience is, due to the explore-exploit-dilemma, a trial-and-error approach. As mentioned in subsection 2.2.4, transitions (in the rest of the work referred to as *experiences*) are generated by traversing the problem space and then used once for a training update after which they are thrown away. This procedure is wasteful, since some experiences may be rare and some (such as those involving damages) costly to obtain. A biological inspired [MMO95; ONe+10] technique that stores and reuses experiences to counteract this issue is the so called *Experience Replay* (ER) [Lin92; Lin93].

An experience e_t of the transition at time step t is defined as the following quintuple:

$$e_t = (S_t, A_t, R_{t+1}, S_{t+1}, d), \tag{2.26}$$

where S_t denotes the starting state, A_t the chosen action, R_{t+1} the reward that was received in the next state S_{t+1} and d denoting if S_{t+1} was a terminal state. In contrast to using experiences once and then throw them away, generated experiences are stored in the so called replay buffer, that is denoted as the data set $D_t = \{e_1, \ldots, e_t\}$. Those stored experiences are repeatedly presented to the learning algorithm, mimicking over and over experiencing the same transitions. This results in a speedup of the credit/blame propagation and consequently in a faster convergence. One restriction of the ER technique is that the dynamics of the environment must stay the same (or at least not change rapidly). That is because past experiences may become irrelevant or even harmful otherwise. Overall, only *recent* experiences need to be stored and replayed, and as a matter of fact, it is not only unnecessary but also harmful to replay an experience as many times as possible, because the learner would overfit that experience, which usually harms generalization. Moreover, in a non-episodic/infinite environment (and also in an episodic one after enough time has gone by) the storage represents a limitation. To prevent overfitting and running into storage limitations, the ER buffer typically is realized in a first in, first out (FIFO) manner.

Stored experiences are represented for training in a later point in time, and then, typically, the policy has evolved and differs from the policy that generated that transition which is stored in the experience. Because of this effect, using ER always makes the corresponding algorithm off-policy.

Stored experiences can be utilized for training either online or in a specific training phase. The technique is easy to implement, and the cost of using it is mainly determined by the required storage space.

2.3 Types of Reinforcement Learning

RL methods can generally be distinguished into *model-based* and *model-free* methods. While model-based approaches make use of a model of the environment's dynamics in some form and rely on *planning* as their primary component, model-free methods primary rely on *learning*.

Parts of the following text were published in a similar but extensive form in [PSH22a], and another important reference is [SB18].

2.3.1 The Model

To understand the concepts of model-free and model-based RL, it is necessary to define the term model. The *model* in RL is defined as a model of the environment (or, to be precise, of the dynamics of the environment) and therefore should be able to fulfil the tasks of the environment. These are the calculation of a follow-up state and a reward based on a given state-action pair.

Based on the formalization in section 2.1, the model can be described as the combination of the state-transition function (Equation 2.4) and the reward function (Equation 2.6).

Generally speaking, the model is anything that an agent can use to predict how the environment will respond to its actions. A stochastic model (several possible states and actions, coupled with a probability of occurring) can either produce a description of all possibilities and their probabilities (so-called distribution model) or return just one of them, sampled by its probability (so-called sample model).

2.3.2 Model-Free Reinforcement Learning

The term model-free RL is generally describing all RL techniques that learn with the absent of a model of the environment. *Learning* in this context describes the estimation of a value function like introduced in section 2.2. Instead of computing the value functions based on the actual distribution $p(s', r|s, a)$ (cf. Equation 2.2), in model-free RL this distribution is approximated by traversing the state space.

To determine how the agent should behave in a given state, it can simply choose that action that maximizes the value. A basic RL approach that does so is Q-Learning [WD92]. But there are of course more sophisticated approaches like DQN [Mni+13; Mni+15] and its extensions [vGS16; Wan+16; Hes+17]. The category of these methods, that learn a value function and derive a policy from it is called *value-based*. Another category that takes a different route is called *policy-based*, and methods from this field directly parametrize the policy and try to optimize it according to the expected return. An example is the REINFORCE algorithm [Sut+99]. A third category of RL methods aims to bring the former mentioned approaches together and such algorithms are known as *actor-critic* methods. These approaches parametrize both, a value function and a policy, and try to optimize them together. Examples of actor-critic algorithms are A3C [Mni+16], DDPG [Lil+16] and PPO [Sch+17].

2.3.3 Model-Based Reinforcement Learning

Model-based RL was defined by Moerland et al. [Moe+20] as:

Definition 2.3.1. Model-based reinforcement learning is a class of MDP algorithms that (1) use a model, and (2) store a global solution.

Thereby, a global solution is either a value function or a policy. Furthermore, they found two subcategories:

1. *Model-based RL with a learned model.* Here, both, a model and a global solution is learned. An example is Dyna [Sut91].

2. *Model-based RL with a known model.* A model of the environment is known up front and used to learn the global solution. An example is AlphaZero [Sil+18].

The idea behind having a model is to use it for *planning*. If a model is at hand, it can be used to simulate experience, this describes the process of feeding an input (state-action-pair) to the model and receiving an output (reward and follow-up state). Given a starting state and a policy, it is possible to receive a whole episode by a sample model and all possible episodes from a distribution model. In both cases the environment is *simulated* with the model and creates *simulated experiences*. The term planning is used for every computational process that produces or improves a policy for interacting with the modeled environment, given a model as input:

$$\begin{tikzcd}\text{model} \arrow[r, "\text{planning}"] & \text{policy}\end{tikzcd}$$

Two distinct planning approaches can be determined: (1) *State-space planning* searches the state space for an optimal policy or path to the goal and (2) *plan-space planning* searches the space of plans. While in (1) actions are used to traverse the state space and value functions are computed over states, in (2) plans are transformed in other plans and value functions, if present, are defined over the space of plans.

As state-space planning is more common, the present work will focus on these approaches from now on. State-space planning methods share a common structure that is based on two basic ideas: (1) they involve computing value functions as a key intermediate step towards improving the policy or learn the policy directly, and (2) they compute value functions/policies by updates or backup operations applied to simulated experiences:

$$\begin{tikzcd}\text{model} \arrow[r] & \begin{array}{c}\text{simulated}\\ \text{experience}\end{array} \arrow[r, "\text{backups}"] \arrow[rr, bend right, "\text{backups}"'] & \text{value} \arrow[r] & \text{policy}\end{tikzcd}$$

Referring to the learning process mentioned in subsection 2.3.2 it can be seen that both, learning and planning focus on the estimation of value functions or policies by backing-up update operations. They do in fact differ in the utilized experiences. Planning uses simulated experiences that are generated by a model whereas learning methods make use of real experiences generated by the environment. This means nevertheless, that they share a common structure and algorithms can be transferred between them.

According to Moerland et al. [Moe+20] model-based methods can be distinguished into the following categories addressing the three main considerations: what *type of model* is learnt, what *type of estimation method* is used and in what *region* should the model be valid?

Type of model

Moerland et al. set their focus here on dynamics models (learning the state-transition function). As mentioned above, the reward function is also part of the model, but as it is usually easier to learn, it can be ignored for the classification into the following three categories:

- *Forward model*: $(S_t, A_t) \rightarrow S_{t+1}$. Predicting the follow-up state given a current state and chosen action. Most of the (modern) model-based RL techniques follows this path, and an example is Deep Dyna-Q [Pen+18].
- *Backward/reverse model*: $S_{t+1} \rightarrow (S_t, A_t)$. Predicting the start state and chosen action to end up in a given state. Prioritized sweeping [MA93] uses this approach.
- *Inverse model*: $(S_t, S_{t+1}) \rightarrow A_t$. Predicting the action that needs to be executed to traverse from a given state into another. This technique is for example used in RRT planning [LaV+98].

Estimation method

The method of approximating a model is often a form of supervised learning and can be distinguished into the following categories:

- *Parametric*: this is the most common approach for model approximation. There are two main subgroups:
  - *Exact*: This concludes tabular methods, containing a separate entry for every possible transition.
  - *Approximate*: Function approximation can reduce the required number of parameters and enable generalization. Nowadays, most of the time neural networks are used for this purpose [GBC16].
- *Non-parametric*: Methods from this category directly store and use data to represent the model.
  - *Exact*: Replay buffers [Lin93] could be considered non-parametric versions of tabular transition models, but even if the line between model-based RL and replay buffer methods is very thin, this is a discussable statement [VS15] and for the course of this work they are not considered as model-based RL.
  - *Approximate*: Non-parametric methods (e.g. Gaussian processes [DR11]) could be used to generalize information to similar states.

Region in which the Model is Valid

The following categories specify the area of the state space where the model is valid:

- *Global*: This is the main approach for most of the methods and means the approximation of the dynamics of the entire state space.

- *Local*: In contrast to the global approach, local methods only approximate local dynamics and discard the local model afterwards. This approach is often used in the control community [LA14].

2.4 Deep Q-Network

Even if the original Q-Learning method has been an important breakthrough in RL at its time, it is limited to rather simple problems. An advancement of the original algorithm that replaces the tabular parameterization of the action-value function with a deep neural network (DNN) is known as *deep Q-network* (DQN).

The following sections are mainly based on [Mni+13; Mni+15] and provide a detailed overview of the DQN algorithm, its functionalities and some extensions.

2.4.1 Limitations of Original Q-Learning

The original Q-Learning algorithm introduced in subsection 2.2.4 parameterizes the action-value function estimate $Q(s, a)$ in form of a table that holds entries for the action-values of all state-action pairs for $s \in \mathcal{S}$ and $a \in \mathcal{A}$. This representation method faces some limitations when confronted with more complex problems that have large state- and/or action-spaces. The most obvious one is the need of storage that grows (in the worst case) exponentially with increasing state- and/or action-spaces. At some point, the table (or tables, when double learning is used) is just too big to be stored. As touched in subsection 2.2.4, during training, one entry is updated per step, but all entries (or at least the relevant ones) need to be updated a lot of times to ensure the approximates of the corresponding action-values to be good enough, and consequently for the algorithm to converge. Larger tables require more training, and at some point it just gets inefficient. That is because the tabular representation in combination with this update mechanism does not generalize well. Besides the limitation of the size of the state- and action-space, the tabular representation is also not able to handle continuous state- and/or action-spaces. Such states and/or actions need to be discretised, and by doing so information and precision get lost.

2.4.2 Deep Q-Network Algorithm

Similar to original Q-Learning being an important breakthrough in RL for providing a TD method that enabled convergence proofs, DQN marks an important breakthrough in RL as well, this time for combining RL with deep neural networks. In general, an artificial neural network [MRG+87] is considered a DNN when it has more than two hidden layers.

As DQN is an advancement of Q-Learning, it shares the same functionality in terms of approximating the optimal action-value function $q_*(s, a)$ (cf. Equation 2.16) with an estimate $Q(s, a)$. This estimate is chosen as function approximator with parameters θ such that $Q(s, a; \theta) \approx q_*(s, a)$. However, a DNN represents a nonlinear function approximator, and Tsitsiklis et al. [TR97] showed that such approximators, when used to represent the action-value function, can lead to unstable learning or even divergence. This is because of the following three reasons:

1. There are a strong correlations present in the sequence of observations.
2. Small updates to Q may significantly change the policy and therefore change the data distribution.
3. The action-values (Q) and the target values $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ are strongly correlated as well.

In a bit more detail, the first two points refer to the fact that current parameters determine the next data sample that these parameters are trained on. In a simple example that means, if the maximizing action is "move to the left", then the training samples will be dominated by samples from the left-hand side. It can be seen that consecutive experiences are strongly correlated and even conditioned by each other. If, at some point, the maximizing action switches to "move to the right", then the training distribution will switch as well, which might have been triggered by a small increase of the action-value of that action. This behaviour can induce unwanted feedback loops and consequently, parameters could get stuck in a poor local minimum, or even diverge catastrophically. Moreover, if a DNN is presented training data that is focused on a specific area of the problem space (which is the case in the simple example above), then it will learn the dynamics of this region but is likely to forget them when the data distribution will shift to a new region. This effect is widely known as catastrophic forgetting [Kir+17].

The last of the three points is reasoned in the TD error (Equation 2.20) that is used for the loss function of the DNN and that takes advantage of the recursive relationship of the Bellman equation. For supervised learning [Mit97], the targets of the loss function are fixed before the learning begins, which is a condition that can not be satisfied in RL. As already mentioned above, the target values for the DQN loss are defined by

$$y_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a), \tag{2.27}$$

and, as can be noticed, an estimate of the action-value function—which is parameterized by the DNN—is part of the target. An update that increases $Q(S_t, A_t)$ often also increases $Q(S_{t+1}, a)$ for all a hence also increases the target y_t, and this effect possibly leads to oscillations or divergence of the policy.

Two concrete techniques have been proposed to counteract the problems from above. The first one is Experience Replay (cf. subsection 2.2.6), but with some extended behaviour. Instead of replaying one experience at a time, learning updates are applied to the samples of a whole *minibatch* of experiences $(s, a, r, s') \sim \mathcal{U}(D)$ drawn uniformly at random from the buffer. A minibatch B is defined as a multiset of experiences:

$$B = \{e\} = \{(s, a, r, s')\} \tag{2.28}$$

This approach comes with several advantages. The randomization of the presented training samples breaks the correlations between them and reduces the variance of the updates. Because each experience has the chance to be used in many weight updates the sample efficiency is increased. And at last, ER averages the behaviour distribution over many of its previous states and thereby smooths out learning and avoids oscillations or divergence in the parameters. Moreover, as mentioned in subsection 2.2.6, when using ER, the algorithm will become off-policy and as Q-Learning already is an off-policy method the two fit well together.

To break the correlations present in the TD error and improve stability, a second network that is responsible for the generation of the target values y_t was introduced. This network, which is called *target network* $\hat{Q}$, is a clone of the network Q. In the original version $\hat{Q}$ is updated (by cloning the parameters θ from Q) every C training steps and used in this form to generate y_t for the next C updates to Q. The parameters of $\hat{Q}$ are denoted θ^-, and following this notation Equation 2.27 can be concretized to

$$y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_{a'} \hat{Q}(S_{t+1}, a; \theta_t^-). \tag{2.29}$$

Generating the targets using an older set of parameters adds a delay between the time an update to Q is made and the time the update affects the targets, making divergence or oscillations much more unlikely. When the target network is updated in fixed intervals by completely cloning Q, then this is called *hard target update*. A more sophisticated approach updates the weights in every update to Q, but instead of directly copying the parameters, they are smoothly adjusted: $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$ with $\tau \ll 1$. Generating θ^- in this way is called *soft target update*, and makes the target Q-values change slowly, greatly improving the stability of learning.

The training of a DQN is performed by adjusting the parameters θ_i at iteration i to reduce the mean-squared error of the TD error, where the optimal target values $r + \gamma \max_{a'} Q_*(s', a')$ are substituted with approximate target values $y^{\text{DQN}} = r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-)$ using parameters θ^- from $\hat{Q}$. Therefore, a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i can be defined as

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r}\left[\left(\mathbb{E}_{s'}[y^{\text{DQN}}|s, a] - Q(s, a; \theta_i)\right)^2\right] \\ &= \mathbb{E}_{s,a,r,s'}\left[\left(y^{\text{DQN}} - Q(s, a; \theta_i)\right)^2\right] + \mathbb{E}_{s,a,r}\left[\mathbb{V}_{s'}\left[y^{\text{DQN}}\right]\right], \end{aligned} \tag{2.30}$$

with $\mathbb{E}_{s,a,r}\left[\mathbb{V}_{s'}\left[y\right]\right]$ denoting the variance of the targets. As this variance is independent of the currently optimized parameters θ_i, it may be ignored. The usage of fixed parameters θ^- from previous iterations for the optimization of the ith loss function $L_i(\theta_i)$ results in a sequence of well defined optimization problems. Differentiation of the loss function with respect to the weights results in the following gradient:

$$\nabla_{\theta_i} = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]. \tag{2.31}$$

Similar to original Q-Learning, the behaviour policy is derived from the action-values by greedily choosing that action that maximizes the expected action-value for the current state. Thus, for the action selection, the action-values for all possible actions are required. Motivated by this fact, a DQN follows the architecture depicted in Figure 2.6. The input layer takes a current state as input and is followed by a block of hidden layers defined by the type of DNN and tuned as hyperpatameter for the corresponding problem. The output layer has one node for every action in $\mathcal{A}$, which represents the action-value for the current state and one concrete action. Following this architecture, only one forward pass is required to compute all action-values for a given state and, based on them, choose an action.

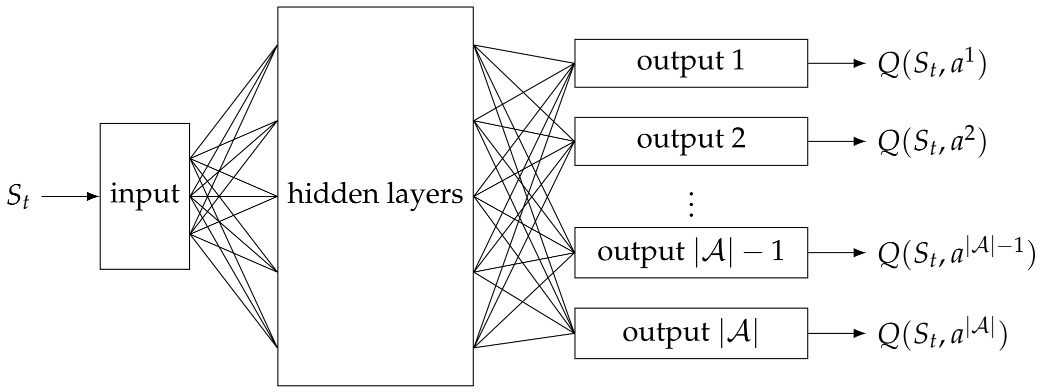


FIGURE 2.6: Architecture of a deep Q-network

2.4.3 Extensions of the Deep Q-Network

The DQN concept as a whole has been modified and extended may times, this section introduces two prominent extensions.

Double Deep Q-Network

The double Q-Learning algorithm (cf. subsection 2.2.5) makes use of a second estimate Q_2 and decomposes the max operation in the target into action selection and action evaluation. As van Hasselt et al. pointed out in [vGS16], a DQN already has two estimates of the action-value function and therefore, the target network provides a natural candidate for the second estimate. A major difference to the original approach is of course the fact that the target network is somewhat coupled to the main network. But as a trade-off for not being forced to introduce and train another network $\hat{Q}$ seems fine to use. In consequence, the update process of the target network remains the same (hard or soft target update), but, according to original double learning, the target is replaced by:

$$y_t^{\text{DDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a \hat{Q}(S_{t+1}, a; \theta_t^-); \theta_t). \tag{2.32}$$

This approach is called *double DQN*, and, in comparison with the original approach, Q_2 is replaced with $\hat{Q}$ to evaluate the current greedy policy. To get most of the benefit from double Q-Learning, while keeping the remaining parts of the DQN algorithm intact, the presented version of double DQN represents the minimal possible change.

Van Hasselt et al. were able to show that (similar to double Q-Learning) overestimation can be reduced, and, additionally, performance can be increased in some cases.

Dueling Deep Q-Network

Wang et al. [Wan+16] proposed an architecture for neural networks that is specifically designed for model-free RL. This architecture in combination with DQN is known as *dueling DQN*.

To express the relationship of the state-value and the action-value beyond Equation 2.17, a new quantity, the so called *advantage*, is defined as:

$$adv_\pi(s,a) = q_\pi(s,a) - v_\pi(s), \tag{2.33}$$

so that $\sum_a adv_\pi(s,a) = 0$. As discussed in section 2.1, the state-value gives an intuition of how good it is to be in a particular state s, and the action-value measures the value of choosing a concrete action a in this state. As the state-value is subtracted from the action-value, the advantage computes a relative measure of the importance of each action. This differentiation of the action-value into two components can be exploited, as for many states, it is of no interest to estimate the value of each action choice. Moreover, in some states, it is of paramount importance to know which action to choose, but in many other states this choice has no impact at all. The estimation of the state-value, however, is of great importance for every state in bootstrapping based methods. An architecture that is able to implement the insights from above in a beneficial way is shown in Figure 2.7. This, so called *dueling architecture*, inserts a new layer behind the hidden layer block from the original DQN. The new layer splits into a node for each advantage (colored in blue) and one node for the state-value (colored in red). The advantage nodes are combined with the value node to compute the action-value in the end.

The combination of the advantages with the value to produce the action-values, however, is a crucial part in the architecture and there are some things that need to be considered here. As mentioned above, $\sum_a adv_\pi(s,a) = 0$, and, following Equation 2.17, for a deterministic policy, $a^* = \arg\max_{a'} q_\pi(s,a')$, it holds that $q_\pi(s,a^*) = v_\pi(s)$ and hence $adv_\pi(s,a^*) = 0$.

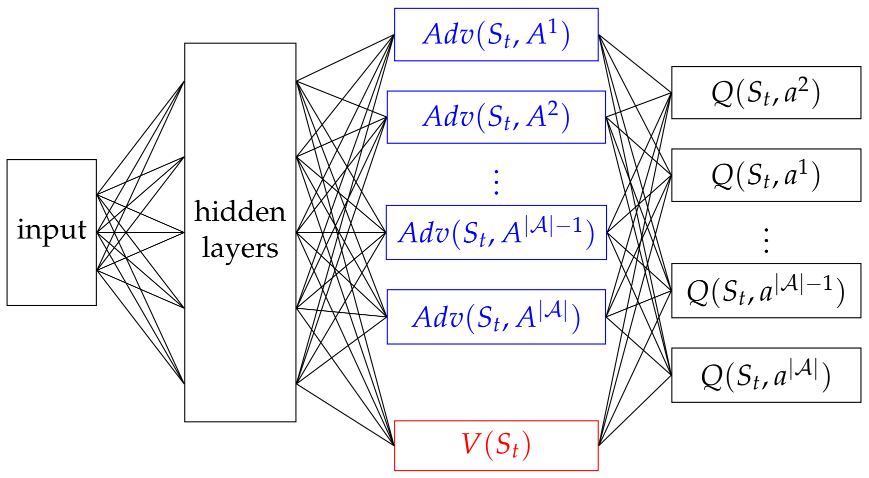

input
hidden layers
$Adv(S_t, A^1)$
$Adv(S_t, A^2)$
⋮
$Adv(S_t, A^{|\mathcal{A}|-1})$
$Adv(S_t, A^{|\mathcal{A}|})$
$V(S_t)$
$Q(S_t, a^2)$
$Q(S_t, a^1)$
⋮
$Q(S_t, a^{|\mathcal{A}|-1})$
$Q(S_t, a^{|\mathcal{A}|})$

FIGURE 2.7: Architecture of a dueling deep Q-network. Nodes that represent advantages are colored in blue and the node that represents the state-value is colored in red.

The naive approach would be simply adding the state-value to an advantage to compute the corresponding action-value. Thus, the aggregation in the network would look like this:

$$Q(s, a; \theta) = V(s; \theta) + Adv(s, a, ; \theta), \tag{2.34}$$

with the scalar, $V(s;\theta)$, been replicated $|\mathcal{A}|$ times to express the equation in matrix form and been appliable to all (s, a). However, it is not possible to recover V and Adv uniquely from a given Q. Equation 2.34 is therefore unidentifiable in this sense. To overcome this issue, the estimate for the advantage function can be forced to have zero advantage at the chosen action. The corresponding aggregation of the advantages with the state-value is then defined as:

$$Q(s, a, ; \theta) = V(s; \theta) + \Big(Adv(s, a; \theta) - \max_{a' \in \mathcal{A}} Adv(s, a', ; \theta)\Big). \tag{2.35}$$

Equation 2.35 fulfills the conditions from above. Alternatively, the max operator could be replaced with an average:

$$Q(s, a, ; \theta) = V(s; \theta) + \Big(Adv(s, a; \theta) - \frac{1}{|\mathcal{A}|} Adv(s, a, ; \theta)\Big). \tag{2.36}$$

Even if the second approach is not able to map the original semantics of $v_\pi(s)$ and $adv_\pi(s, a)$ because they are off-target by a constant, it increases stability. That is because the advantages only need to change as fast as the mean, instead of having to compensate any change to the optimal action's advantage.

2.5 Interpolation

An overview of interpolation and approximation and the relationship of both is presented in the following section. One concrete interpolation technique is introduced afterwards.

Parts of the following text have already been published in a similar form in [PSH22b; PSH22a].

2.5.1 Interpolation vs. Approximation

The task of finding a function $g(x)$ that fits either a real valued function $f(x)$ of the real valued variable x or a given table of data (x_i, y_i) can be solved by *interpolation* or *approximation*. [SK11]

Approximation is thereby defined as finding a function $g(x)$ that minimizes a norm of the difference of the values for the searched function $g(x_i)$ and the given table $||g - y||$ with vectors $g = \{g(x_i)\}$ and $y = \{y_i\}$. Or rather, in case of $f(x)$, minimizes a norm of the difference of the values for the searched and the given function $||g(x) - f(x)||$. Interpolation on the other hand tries to find a function $g(x)$ that fits the given function or data in specific points, such that holds: $g(x_i) = f(x_i)$ or $g(x_i) = y_i$. [SK11]

Both approaches are in fact quite similar and only differ in the estimated values for the given points. Given data points are also called sampling points P_i, and for them applies: $P_i = (x_i, y_i) = (x_i, f(x_i))$. Nevertheless, there are some benefits that determine the cases when to use which method. Given a table of data $(x_i, y_i), i = 0, 1, \dots, n$ there are two main differences:

1. If the amount of given data is (very) high—n is (very) high—interpolation turns out to be not practical. This effect even grows with the amount of noise in the data. The given data should be approximated with a smooth line through the "datacloud" like in Figure 2.8 on the left.

2. If there is only a small amount of data available and it is reasonable or even necessary that $g(x)$ fits the values y_i at x_i or $f(x_i)$ perfectly, then interpolation turns out to be better than approximation. This can be observed in Figure 2.8 on the right. [SK11]

2.5.2 Inverse Distance Weighting

A concrete way of interpolation that is still rather simple and utilizes a weighted average is known as *Inverse Distance Weighting* (IDW) [She68]. IDW strives to create an interpolation using a weighted average, where sampling points that are located in closer distance to the query point have bigger impact than sampling points located farther away. IDW is defined as follows:

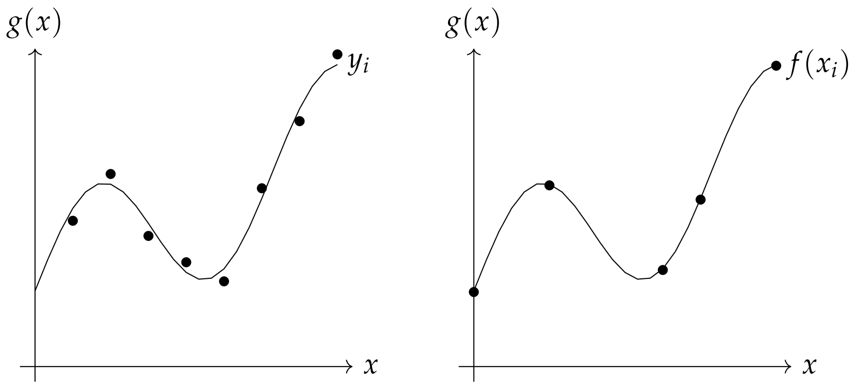


FIGURE 2.8: Intuition of approximation vs. interpolation

$$g(x) = \begin{cases} \frac{\sum_{i=1}^{n} w_i(x) y_i}{\sum_{i=1}^{n} w_i(x)}, & \text{if } d(x, x_i) \neq 0 \text{ for all } i, \\ P_i & \text{if } d(x, x_i) = 0 \text{ for some } i, \end{cases} \tag{2.37}$$

with

$$w_i(x) = \frac{1}{d(x, x_i)^{p_{\text{idw}}}}. \tag{2.38}$$

The interpolation function tries to find an interpolated value y at a given point x based on the value of sampling points $y_i = g(x_i)$ for $i = 1, 2, \ldots, n$. To increase the impact of closer values a weight w_i is used that includes a distance metric d together with a variable p_{idw} called the power parameter controlling the impact of y_i.

2.6 Nearest Neighbor Search

In order to collect relevant sampling points for interpolation a form of *nearest neighbor* (NN) search is required. The approaches presented in this work use the replay buffer as pool of possible sampling points and given a query point $x_q = (S_q, A_q)$ all points P_i that lie within a given distance are to be returned. This section gives an overview of possible solutions to that task while taking required computation time into account.

A naive approach would take x_q and iterate through the whole buffer while computing a distance metric for each entry. Such an exhaustive search has a runtime complexity of $O(n)$ and, therefore, is not practical for large sized buffers.

2.6.1 K-D Tree

One technique that offers reduced computational overhead is the so called *k-d tree* [FBF77]. This approach is a generalization of the simple binary tree in which each node represents a k-dimensional point. Every non-leaf splits the space into two parts by generating a splitting hyperplane. The two resulting spaces are known as half-spaces. Points to the one side of the hyperplane are represented by one subtree and nodes on the other side by the other subtree. Every node is associated with one of the k dimensions and the hyperplane is created perpendicular to that dimension's axis.

A search in the tree aims for the point that has the smallest distance to a given query point x_q. Thereby, the properties of the tree can be used to quickly eliminate large portions of the search space. Starting at the root node, the algorithm traverses the tree downwards recursively, choosing a subtree based on the comparison of the node's position and the value of x_q in the split dimension. Upon reaching a leaf node, that node is checked against the current best solution and replaces it when then distance is closer. Afterwards, a check is performed if any points on the other side of the splitting plane could be closer to x_q as the current best solution. This is done by intersecting the splitting hyperplane with a hypersphere around x_q that has a radius equal to the current nearest distance. Taking advantage of the fact that all hyperplanes are axis-aligned this can be achieved by a simple comparison of the distance of the splitting coordinate of x_q and the current node against the overall distance from x_q to the current best. The possibility of nearer points is indicated by the hypersphere crossing the plane, and, if so, the other branch needs to be traversed from the current node. If the hypersphere does not intersect the splitting plane, the entire other branch can be excluded and the algorithm moves upwards. By completing this process for the root node the algorithm terminates. Since the range of a domain is divided in half at each level, k-d tree structures performs well in searches for all points that lie within a given range (range search).

A search in such a k-d tree architecture has a runtime complexity of $O(k \log n)$. Moreover, a range search has a worst case runtime complexity of $O(k * n^{1-\frac{1}{k}})$. However, as this structure is inflexible and can not be altered during runtime, for online settings it needs to be rebuild in a constant interval L with a runtime complexity of $O(n \log n)$.

2.6.2 Ball Tree

Another approach is the so called *balltree* [Omo89], which is able to tune itself to the structure of the represented data, has good average-case efficiency and deals well with high-dimensional entities. A region bounded by a hyper-sphere in an k-dimensional Euclidean space is referred to as ball, and balls are represented by the $k + 1$ floating point values that specify the coordinates of its center and the length of its radius. A balltree however, is a complete binary tree of balls in the sense that a node is associated with a ball such that a non-leaf node's ball is the smallest which contains the balls of its children. Leaf nodes hold relevant information for the application and non-leaf nodes are used only to guide efficiently through the leaf structure. In comparison to k-d trees, the space does not need to be partitioned and intersections are allowed. A two-dimensional balltree is shown in Figure 2.9.

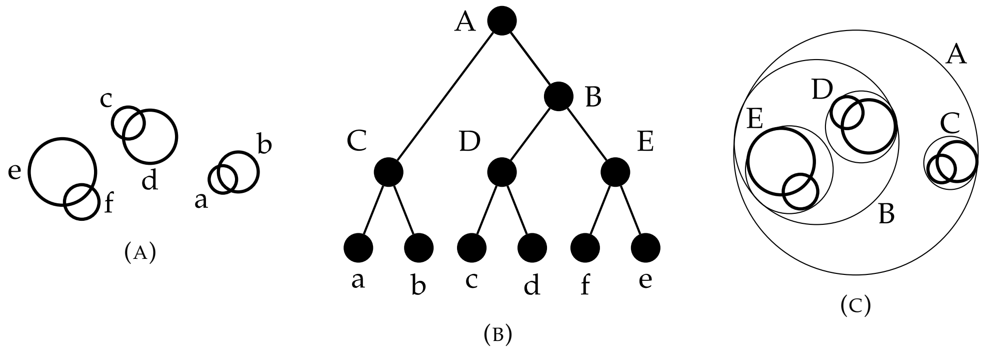


FIGURE 2.9: (A) A set of balls in the plane. (B) A binary tree over these balls. (C) The balls in the resulting balltree. Reprinted from [Omo89].

A search for a query ball is expected to return a list of all leaf balls that contain that ball. Similar to the k-d tree, a recursive search starting at the root node is executed that cancels out branches (balls) that do not contain the query ball. As branches are balls here, it is impossible that a query ball lies within any leaf ball of a non leaf ball that does not contain that query ball.

More complex queries search for concrete points (m nearest neighbours of x_q). The search starts in the root node and traverses the balltree downwards while canceling out irrelevant branches. Therefore, the smallest ball, centered at x_q, that contains the m nearest points that have been seen so far, is remembered, and non leaf balls that do not intersect that ball are canceled out. To do this most efficiently, at any non leaf ball, that child that is closer to x_q is searched before the other one.

A search in a balltree has a runtime complexity of $O(k \log n)$, and the construction of such an architecture has a runtime complexity of $O(n \log n)$.

Chapter 3

Related Work

The following section provides an overview of scientific work that has been published in the domain of (deep) RL and has some similarities with the approaches presented in this work and/or share at least some related ideas.

Parts of the following text have already been published in a similar form in [PSH21].

3.1 Experience Replay

The concept of the ER has been subject to a lot of research and the simple basic concept was extended in many ways. As this work presents an advancement of ER as well, this has some relevance.

3.1.1 Prioritized Experience Replay

Probably the most famous extension to ER is known as *Prioritized Experience Replay* [Sch+15]. The original ER samples experiences uniformly at random from the buffer and trains on minibatches that are generated this way. On average, every experience is presented to the learner equally often. However, some experiences hold more valuable information than the rest and it would be beneficial to use them for training more frequently. A metric that provides an estimate of the potential gain in learning is the TD error. That is because it expresses how good the actual estimate of the action-value function is at a given state-action pair. Therefore, a learning update from an experience with a big TD error results in a reduced TD error and consequently in a better estimate at this point. The Prioritized ER extends the stored information of an experience with the last observed TD error and replaces the uniform sampling with a weighted sampling in favour of experiences with a high TD error. To overcome the problem of never sampling an experience that once produced a TD error of zero, every experience in the buffer has a probability to get sampled greater than zero. This approach modifies the data distribution that is generated by following a policy, and therefore induces bias. To account for that, importance sampling has to be used. In early learning phases, the impact of importance sampling is reduced as the increased sampling of experiences with high TD errors is intended, but as the learner comes near convergence importance sampling is used to compensate this effect. Even if the Prioritized ER is an extension to ER, all stored and used experiences are generated by interaction with the environment and no synthetic experiences are in use. This clearly differs from our approach that is focused on synthetic experiences.

3.1.2 Hindsight Experience Replay

An extension to ER that makes use of synthetic experiences in some form is the *Hindsight Experience Replay* [And+17]. This approach is designed for multi-objective RL and addresses the problem of understanding and interpreting a given goal. In such a setting, an agent is provided with a goal that it is intended to reach. In complex environments it is highly unlikely that the agent reaches such a goal overall and, therefore, it is not able to understand what it should do with that information. The Hindsight ER method uses whole trajectories as experiences that are stored together with the current goal. As mentioned above, an agent might never reach one of the goals and, by doing so, it can not understand what it should do with that provided information. To account for that and help the learner to understand the concept behind a presented goal, synthetic experiences are stored alongside the real ones. These synthetic experiences replace the given goal with the last state of the trajectory. By replaying experiences that were actually able to reach a goal, the learner can understand what this concept is and utilize that knowledge to reach the current goal state. In contrast to the presented approaches in this work, the Hindsight ER treats whole trajectories as experiences instead of transitions. Moreover, the method is designed for multi-objective RL settings and does not use interpolation whatsoever.

3.1.3 Experience Composition

The authors of [De +15] investigated the composition of experience samples in the ER. They discovered that for some tasks it is important, that transitions, made in an early phase, when exploration is high, are important to prevent overfitting. Therefore they split the ER in two parts, one with samples from the beginning and one with actual samples. They also show that the composition of the data in an ER is vital for the stability of the learning process and at all times diverse samples should be included. The insides of de Bruin et al. dealt as an inspiration for some design choices later on.

3.1.4 Episodic Backward Update

Episodic Backward Update [LSC19] is based on sampling whole episodes from the ER buffer instead of individual experiences. Minibatches in the Episodic Backward Update therefore have different lengths (the length of the episode). The main contribution is the recursive target generation that traverses the sampled episode backwards and computes TD targets such that reward propagation is accelerated. The authors are able to show that sample efficiency improves as less episodes are required to solve the investigated problems. Even if the targeted outcome (an increase in sample efficiency) equals the one from FU-DQN, no synthetic experiences are used in the process.

3.2 Interpolation in Experience Replay

Storing observed experiences in a buffer gives access to data of the problem space by design and to use this data for interpolation seems like an attractive thing to do.

3.2.1 Mixup Sampling in RL

Mixup Sampling was originally designed for supervised and self-supervised ML, but has also been applied successfully to RL. In general, Mixup Sampling is a data-augmentation strategy that uses pairs of training samples and their modeling targets to generate synthetic samples by linear interpolation. In the context of RL, synthetic experiences are generated by Mixup Sampling. The authors of [Lin+20] generate so called *Continuous Transitions* based on pairs of consecutive discrete transitions in a temporal fashion. Whereas, current and corresponding followup states are used as sampling points to interpolate synthetic experiences in [SMG21]. A main difference to the approaches presented in this work is that exactly two sampling points are used for an interpolation. Moreover, proximity between points is calculated based on their temporal relations instead of local distance.

3.2.2 Interpolated Experience Replay for Continuous Action-Spaces

An approach that uses interpolation in combination with ER was presented by Sander [San21]. In contrast to original ER that replays stored experiences, synthetic experiences are generated by means of interpolation and presented to the learner. Batches of experiences are drawn uniformly at random from the buffer and for every sample a new synthetic sample is interpolated. Based on the start state and the action a NN search in the batch is used to find the K nearest neighbours. A set of one-dimensional Gaussian Process Regression models are fit for each output dimension given by the state-transition delta and the reward. Afterwards, one experience is sampled uniformly from the set of neighbours and in combination with the original experience a state-action-pair is generated by means of Mixup Sampling. This state-action-pair is fed to all of the Gaussian Process Regression models to predict the corresponding mean and variance of the state-transition delta and reward. The state-action-pair in combination with the predicted means form a synthetic experience that is added to the training batch. In the end, a likelihood weight based on the variances is computed and stored in a separate buffer. This procedure is repeated for all experiences and the training batch is presented to the learner in combination with the calculated likelihood weights.

This approach has a lot of similarities to the presented methods in this work, but does in fact also differ in some relevant points. First of all, interpolated experiences are generated and used for learning immediately and thrown away afterwards, while the Interpolated Experience Replay presented in this work stores them in the ER buffer where they can be presented to the learner many times. Subsequent to the last point, only interpolated experiences are used for training, while a mix of both is used in the approaches of this work. Another difference is located in the utilized interpolation techniques, Sander uses Gaussian Process Regression in combination with Mixup Sampling while the focus on this work is set on IDW. As interpolation can be very costly, the choice for the interpolation technique in the approaches later on fell to a rather cheap in computation alternative. However, Sander performs NN searches only on a small set of sampled experiences while in this work the whole buffer needs to be searched. As a NN search in the whole buffer is quite costly with increasing length, the bottleneck of the approaches in this work is the NN search which motivates the usage of less computational expensive interpolation techniques. A different focus on components that can require high computational effort becomes apparent.

3.3 Interpolation in Reinforcement Learning

Interpolation has not only been combined with ER, as demonstrated earlier, but there has also been research exploring the use of interpolation within the context of RL.

3.3.1 Interpolation in Learning Classifier Systems

Learning Classifier Systems [Hei+23] are a family of rule-based learning systems that construct a finite collection of if-then rules. The functionality can be split in two parts, the individual conditions (the if-part) is optimized by a metaheuristic (typically an evolutionary method), whereas the conclusions (the then-part) of each rule is based on a problem-dependent local submodel (e.g. linear regression). Stein [Ste19] investigated how interpolation can be used to counter knowledge gaps in an XCS Classifier System [BW01]. To do so, raw experiences are incorporated in a transductive manner to improve the exploitation of already acquired knowledge elements. Thereby, transductive describes the process of immediately leveraging already made and remembered experiences instead of inducing a model first and deducing to new situations afterwards. Already existing but not directly matching knowledge elements in a currently queried problem space niche are used for interpolation. XCS has also been combined with ER [Ste+20]. The focus however was set on single-step problems instead of truly sequential decision problems.

3.3.2 Interpolated Policy Gradient

Gu et al. [Gu+17] presented an interpolation of on-policy and off-policy model-free deep RL techniques. In their publication an approach of interpolation between on- and off-policy gradient mixes likelihood ratio gradient with Q-Learning which provides unbiased but high-variance gradient estimations. This approach does not use ER and therefore differs from our work.

Chapter 4

Linear Reward Interpolation in Discrete and Non-Deterministic Environments

The following chapter is focused on the question of how synthetic experiences can be generated by means of interpolation. To find an answer to this question, an analysis of all components of an experience is conducted with respect to their individual suitability for interpolation. With the results of this analysis at hand, an initial proof of concept of the Interpolated Experience Replay is presented that performs linear interpolation for rewards only and is designed for discrete and non-deterministic environments. An in-depth evaluation on different versions of the FROZENLAKE problem concludes the chapter.

Parts of the following text have already been published in a similar form in [Pil19; PSH20; PSH21; PSH22b; Pil+23].

4.1 Interpolation of Synthetic Experiences

Experiences represent knowledge of the environment that produced them (cf. chapter 1). As already discussed before, the idea is to exploit this knowledge in a way that can be used to assist the agent with learning. Experiences include a state-transition and a received reward that correspond to a chosen action in a concrete state. As introduced in subsection 2.3.3, the model of an environment is defined by its state-transition function and its reward function, and this also describes the dynamics of the environment. Consequently, an experience encodes knowledge about the dynamics of the environment. Dependent of the form of the two functions the dynamics of neighboring state-action-pairs can be similar, while closer pairs should be more similar than pairs that are located further away. Based on this intuition, and with **Q** 1 in mind, a technique that could be capable of exploiting knowledge to generate meaningful synthetic experiences is sketched in Algorithm 1. This basic algorithm describes the process of finding a query state-action-pair (S_q, A_q) based on a not further defined query function F and using it to search for a set of nearest neighbors NN_q in the replay buffer D_t. With that set at hand, a synthetic experience

Algorithm 1 Generation of Synthetic Experiences

```
1: while Training do
2:     Find State-Action-Pair $(S_q, A_q)$ based on a Query Function $F$
3:     Find Nearest Neighbors $NN_q$ of $(S_q, A_q)$ in $D_t$
4:     Generate Synthetic Experience $e_{syn}$ based on $NN_q$
```

e_{syn} is then generated. This algorithm serves as a formalization of the general idea with no concrete proposals for the different steps so far. Also no concrete usage of these synthetic experiences is proposed.

A naive solution for finding (S_q, A_q), which is also denoted as the query point x_q, would be to sample it from the state-/action-space and nearest neighbors could be searched either by a radius around that point or a method that finds the k-nearest points. Following the main line of this work, e_{syn} could be generated by means of interpolation.

Algorithm 1 provides an initial response to **Q** 1 by outlining a method for generating synthetic experiences using stored samples in the ER buffer. However, the meaningfulness of these experiences cannot be determined at present. It necessitates a concrete implementation coupled with a scientific evaluation to address this question.

4.1.1 Experience Analysis with Respect to Interpolation Possibility

To develop a method for the interpolation of e_{syn} based on NN_q it is required that the single parts of an experience are analyzed with respect to their possibility of being interpolated first. This section is targeted to **Q** 2 and tries to provide an initial answer to the question of what parts of an experience can be interpolated and what needs to be considered by doing so. As introduced in subsection 2.2.6 an experience is defined as a quintuple of state, action, reward, follow-up state and terminal tag. All of these components will be investigated for their role in e_{syn} and their suitability for interpolation. First of all, the state and the action are generated by the query function and therefore S_q and A_q can be used as they are as the state and the action of e_{syn}. The rest of the components need to be generated such that they reflect the dynamics of the environment with an error as small as possible. To make further assumptions and continue with the analysis, an investigation of the role of the different components in a learning update is required. As a reminder, an update is based on the TD-error (cf. Equation 2.20), which is the the part in brackets of the update rule for Q-Learning (cf. Equation 2.24):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big]. \tag{2.24}$$

It can be seen that the reward has direct impact, whereas the follow-up state is used to estimate the maximal action-value of that state. As a reminder, a value expresses the expected return based on a policy, which is the sum of all the rewards that can be expected by starting in a given state and following a policy thereafter. In other words, it estimates all following states and actions as they produce the rewards. The follow-up state impacts the estimation of a whole chain of future events, and, based on the current policy, this chain can differ big times dependent of the state that is used as a starting point. In consequence, it can be stated that an error of the synthetic reward is less tragic than an error of the synthetic follow-up state. However, both components can be interpolated as long as the reward function and the state transition function are interpolable. In environments that include a movement of the agent in a k-dimensional space it can be expected that the same action performed in two states that have a spacial proximity to each other results in two follow-up states that share a similar spacial proximity to each other. As rewards are typically awarded for reaching some points in the k-dimensional space, state-action-pairs that share similar state-transition properties usually also share similar rewards. Synthetic generated rewards and follow-up states (e.g. generated by interpolation) are denoted as R_{syn} and S_{syn}. The last component to analyze is the terminal tag that indicates if an episode is over. This information is required for the update rule of Q-Learning, as no estimation of the action-value of the follow-up state is required in this case. This needs to be classified as critical as the follow-up state, as an update that includes the estimated expected return of a whole trajectory when actually none should be considered is on the same level as estimating a wrong trajectory. It is able to be interpolated however, as long as the same spatial conditions hold that are required for the reward and the state transition. A synthetic done tag is denoted as d_{syn}. Following the analysis from above, a synthetic experience is defined as follows:

$$e_{\text{syn}} = (S_q, A_q, R_{\text{syn}}, S_{\text{syn}}, d_{\text{syn}}), \tag{4.1}$$

whereas the last three components are interpolable but the interpolation is coupled to different risks that is reasoned by the potential impact of the interpolation error.

4.1.2 Discrete and Non-deterministic Environments

Upon analyzing the potential for interpolating different parts of an experience, it was discovered that two components, namely the follow-up state and the terminal tag, pose a higher risk due to the potential harm caused by errors. Based on this insight, an initial proof of concept was developed, exclusively interpolating the rewards and employing exact matches of observed follow-up states and terminal tags. This approach is obviously limited to very specific classes of problems that require two concrete properties. The first one is that follow-up states and terminal tags need to be accessible such that for any synthetic experience that is to be generated a pair of them exists. As continuous state-spaces do not fulfill this requirement, they are ruled out. That is because a state-space of this form inherits an infinite number of possible states and it is not very likely that a state-action pair that equals the queried one has been observed before. However, discrete state-spaces meet the requirement as there exist a limited number of possible states, and, consequently, with a buffer large enough, it is possible that a state-action pair can be found that equals the query point. The second property that is required is that the interpolation of the reward is beneficial in some way, while having several experiences at hand that equal the query point. This is the case for non-deterministic environments as the execution of a concrete action in a given state does not always result in the same reward, follow-up state and terminal tag. The initial proof of concept is therefore developed for *discrete and non-deterministic environments*.

FrozenLake is one example of a non-deterministic world in which an action $A_t \in \mathcal{A}$ realized in a state $S_t \in \mathcal{S}$ may not lead consistently to the same follow-up state $S_{t+1} \in \mathcal{S}$. FrozenLake is basically a grid world consisting of an initial state I, a final state G and frozen, as well as unfrozen tiles. The unfrozen tiles equal holes H in the lake and if the agent falls into one of such, he has to start from the initial state again. If the agent reaches G he receives a reward of 1. The set of possible actions $\mathcal{A}$ consists of the four cardinal directions $\mathcal{A} = \{N, E, S, W\}$. Executing a concrete action (e.g. N) only results with a probability of $\frac{1}{3}$ in the corresponding field, but it is also possible that the agent instead performs one of the orthogonal actions (in the example: W or E) with the same probability of $\frac{1}{3}$ for each case. This behavior makes the environment non-deterministic. Because there is a discrete number of states the agent can reach, the problem can be denoted as discrete. The environment used for evaluation is the "FrozenLake8x8-v1" environment from OpenAI Gym [Bro+16a] depicted in Figure 4.1.

In addition to the described version, the reward function is changed to return a reward of -1 in case of falling into a hole and 5 for reaching the goal. As the first adaption is crucial for the presented approach (see below), the second change helps the learner to solve the environment. Both changes intensify the received rewards and therefore the experienced transitions. Assigning a negative reward to the end of an episode (hole) makes it possible to calculate an average reward containing additional value (see below). By testing different final rewards (goal) it could be observed that the agent performs best with a reward of 5.

The decision to focus on the presented environment was taken because: (1) it is a relatively well-known problem in the RL community (OpenAI Gym) and (2) the presented approach is designed for discrete and non-deterministic environments.

The non-deterministic character of the problem is coupled with some difficulties that are described in the following paragraph: If an action is chosen that leads the agent in the direction of the goal, but because of the slippery factor it is falling into a hole, it additionally receives a negative reward and creates the following experience: $e_t = (S_t, A_t, -1, S_{t+1})$. If this experience is used for a Q update, it misleadingly shifts the action-value away from a positive value. The slippery factor for executing a neighboring action be denoted as c_{slip}, the resulting rewards for executing the two neighboring actions as R^{right}_{t+1} and R^{left}_{t+1} and the reward for executing the intended action as R^{int}_{t+1}. This formalization enables the definition of the true expected reward

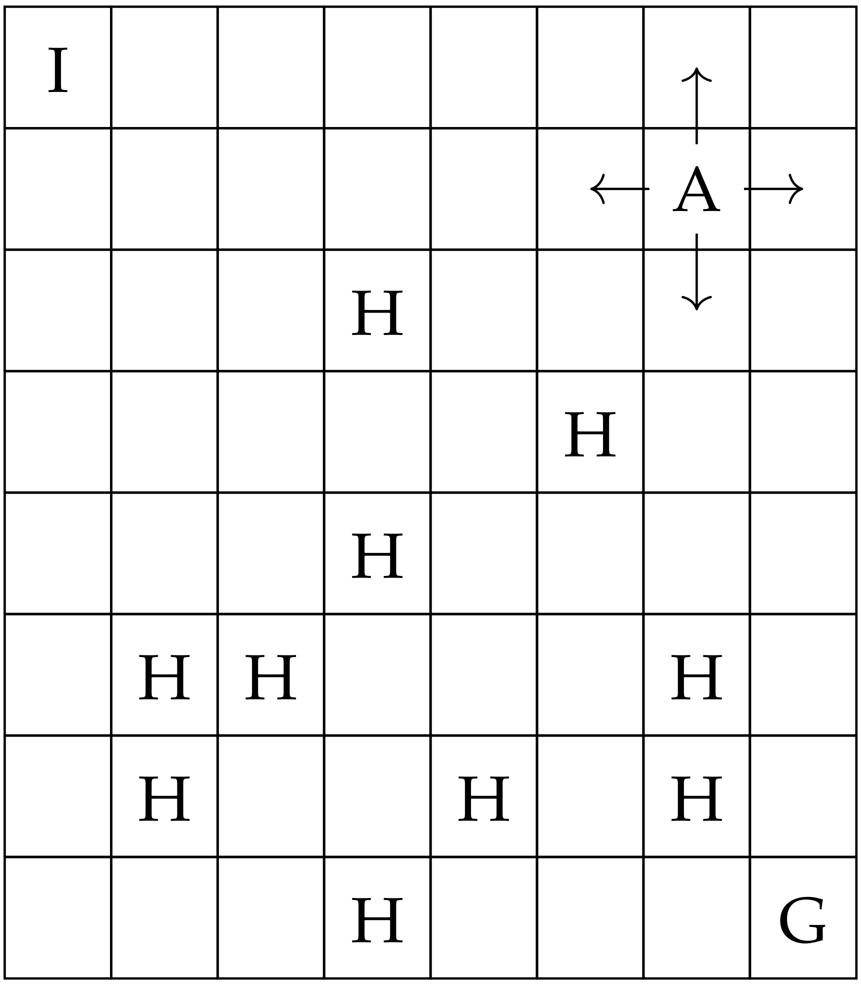


FIGURE 4.1: The FROZENLAKE8x8 environment from OpenAI Gym [Bro+16a].

for executing A_t in S_t as follows:

$$R_{t+1}^{\text{exp}} = \frac{c_{\text{slip}}}{2} \times R_{t+1}^{\text{right}} + \frac{c_{\text{slip}}}{2} \times R_{t+1}^{\text{left}} + (1 - c_{\text{slip}}) \times R_{t+1}^{\text{int}}. \tag{4.2}$$

Following Equation 4.2 the experience that takes the state-transition function into account and that does not confuse the learner is defined as the expected experience e_t^{exp}:

$$e_t^{\text{exp}} = (S_t, A_t, R_{t+1}^{\text{exp}}, S_{t+1}, d_t). \tag{4.3}$$

The learner will converge its estimate of the action value $Q(S_t, A_t)$ after seeing enough experiences to:

$$Q(S_t, A_t) = q_*(S_t, A_t) = R_{t+1}^{\text{exp}} + \gamma \max_{a'} q_*(S_{t+1}, a'). \tag{4.4}$$

Incorporating received (misleading) experiences into a Q update introduces the challenge of oscillation, caused by the non-deterministic nature of the environment. This non-determinism can result in rewards and follow-up states that lead the agent in completely opposite directions, such as moving closer to the goal or terminating the episode in an unfavorable state, like a hole. In the original environment (reward of 0 for falling into a hole), this effect would still occur due to the identical information present in the different follow-up states. However, if the learner exclusively receives experiences in the form of e_t^{exp}, the time required to converge to q_* could be reduced.

4.1.3 Linear Interpolation of Rewards

The intention of the presented solution is to reduce the amount of training by the generation of synthetic experiences that are as similar as possible to e_t^{exp}. As a current limitation, the focus is set on estimating R_{t+1}^{exp} while actual observed follow-up states S_{t+1} and terminal tags d_t are used. As all considerations for S_{t+1} also hold for the terminal tag, d_t is neglected in the following. The presented approach is possible because the environment is discrete (this represents a mandatory precondition of the algorithm). Discrete environments provide a limited number of states, and, more importantly, a limited number of corresponding follow-up states. Thus, it is possible to observe and remember both the states and their follow-ups. Continuous environments, would also require to predict the follow-up state in addition to the reward and for this first investigation of the concept of interpolated experiences a simple approach was chosen. To compute an accurate estimation of e_t^{exp}, an estimation of R_{t+1}^{exp} is required first. As introduced in Algorithm 1, the general task is to generate a synthetic experience for a query point $x_q = (S_q, A_q)$ that has been drawn by a query function. Thus, instead of computing e_t^{exp} and R_{t+1}^{exp}, an experience and a reward for x_q have to be found, which are denoted as e_q^{exp} and R_q^{exp}.

The set of all rewards that belong to the experiences that start in the same state S_q and execute the same action A_q of a drawn query point x_q be defined as:

$$\mathcal{R}_q := \{r_n \in \{r|(s,a,r,s') \in D_t \wedge a = A_q \wedge s = S_q\}\}. \tag{4.5}$$

The set of experiences that equals x_q in its state-action pairs is basically the set of nearest neighbors NN_q, but with a NN distance of zero. In other words a set of exact matches in terms of x_q.

The rewards in $\mathcal{R}_q$ are used to perform a linear interpolation of R_{syn} (which correlates with the computation of a simple average in this case), and is denoted as R_q^{avg}. This value holds as a good estimation of R_q^{exp}.

$$R_q^{\text{avg}} = \frac{\sum_{r \in \mathcal{R}_q} r}{|\mathcal{R}_q|} \tag{4.6}$$

Based on this definition, it is possible to define e_q^{avg} as the estimation of e_q^{exp} in the following form:

$$e_q^{\text{avg}} = (S_q, A_q, R_q^{\text{avg}}, S_{q+1}), \tag{4.7}$$

with

$$e_q^{\text{avg}} \approx e_q^{\text{exp}}. \tag{4.8}$$

The accuracy of this interpolation correlates with the number of transitions stored in the replay buffer that match the query point x_q. This comes from the fact that the effect of outliers can be mitigated with enough normal distributed samples. To achieve this, an algorithm was defined that triggers an interpolation after every step the agent takes. A query point x_q is drawn via a query function F from the state space, and all matching experiences:

$$D_{\text{match}} := \{e_t \in D_t | S_t = S_q\}, \tag{4.9}$$

for that hold that their starting point S_t is equal to the query state S_q, are collected from the replay buffer. Then for every action $a \in \mathcal{A}$ all experiences that satisfy $A_t = a$ are selected from D_{match}:

$$D^a_{\text{match}} := \{e_t | e_t \in D_{\text{match}} \land A_t = a\}. \tag{4.10}$$

The different actions a are the query actions A_q, and in this concrete example all actions from the action-space are used. however, what actions are used is decided by the query function and can differ from this example. The resulting transitions are used to compute an average reward value R_q^{avg}. Utilizing this estimation, a synthetic experience e_q^{avg} for every distinct next state:

$$\mathcal{S}_{t+1} \in \{s' | (s, a, r, s') \in D^a_{\text{match}}\}, \tag{4.11}$$

is created. This results in a minimum of 0 and a maximum of 3 synthetic experiences per action and sums up to a maximum of 12 synthetic transitions per interpolation depending on the amount of stored transitions in the replay buffer. As with the amount of stored real transitions, that can be seen as the combined knowledge of the model, the quality of the interpolated experiences may get better, a parameter $c_{\text{start\_inter}}$ is introduced, that determines the minimum amount of stored experiences before the first interpolation is performed. The associated pseudocode, which provides a concrete implementation of Algorithm 1, is depicted in Algorithm 2. Synthetic experiences that have been generated that way are stored in a separate buffer and can be used for training from that moment on. The concrete architecture of this buffer is explained in the next section.

4.2 Interpolated Experience Replay

Q 3 raises the question of how synthetic experiences, once generated, can be used to assist a DQN with learning. In the former section, a way of how to generate (meaningful) synthetic experiences for discrete and non-deterministic environments

Algorithm 2 Reward averaging in IER

```
1: Initialize $D$
2: Initialize $D^{\text{inter}}$
3: while Training do
4:     Store Experience $e$ in $D$
5:     if $|D| \geq c_{\text{start\_inter}}$ then
6:         Find $S_q$ based on Query Function $F$
7:         Select all $e_t$ that match $S_t = S_q$ from $D$
8:         Store Results in $D_{\text{match}}$
9:         for all $a = A_q \in \mathcal{A}$ do
10:            Select all $e_t$ that match $A_t = A_q$ from $D_{\text{match}}$
11:            Store Results in $D^{a}_{\text{match}}$
12:            Compute $R^{\text{avg}}_t$
13:            for all distinct $S_{q+1}$ in $D^{a}_{\text{match}}$ do
14:                Generate $e^{\text{avg}}_q = (S_q, A_q, R^{\text{avg}}_q, S_{q+1})$
15:                Add $e^{\text{avg}}_q$ to $D^{\text{inter}}$
```

was presented. In the following section, the so-called *Interpolated Experience Replay* (IER) is introduced that provides a mechanism for how to use them.

4.2.1 Interpolation Component

Stein et al. introduce their Interpolation Component (IC) in [Ste+17]. This architecture is used as an underlying basic structure for the interpolation tasks and will be presented in the following chapter.

The IC, depicted in Figure 4.2, serves as an abstract pattern and inherits a *Machine Learning Interface* (MLI), an *Interpolant*, an *Adjustment Component*, an *Evaluation Component* and the *Sampling Points* (SP). The MLI acts as interface to attached ML components as well as a controller for the IC. When a sample is received, it is handed to the Adjustment Component, there, following a decision function, it is added to or removed from SP. If an interpolation is required, the Interpolation Component fetches required sampling points from SP and computes, depending on an interpolation technique, an output. The Evaluation Component provides a so-called trust-level as a metric of interpolation accuracy.

For the approaches presented in the current work, the SP was replaced with a replay buffer and basically the whole functionality of ER. It is realized by a FiFo queue with a maximum length. This queue represents the classic replay buffer and is filled only with real experiences that have been generated by interaction with the environment. To store synthetic experiences, another queue, a so-called *ShrinkingMemory*, is introduced. This second storage is characterized by its decreasing size. Starting at a predefined maximum it gets smaller depending on the length of the real experience queue. The *Interpolated Experience Replay* (IER) has a total size, comprising the sum of the lengths of both queues as shown in Figure 4.3. If this size is reached, the length of the ShrinkingMemory is decreased and the oldest items are removed. This process

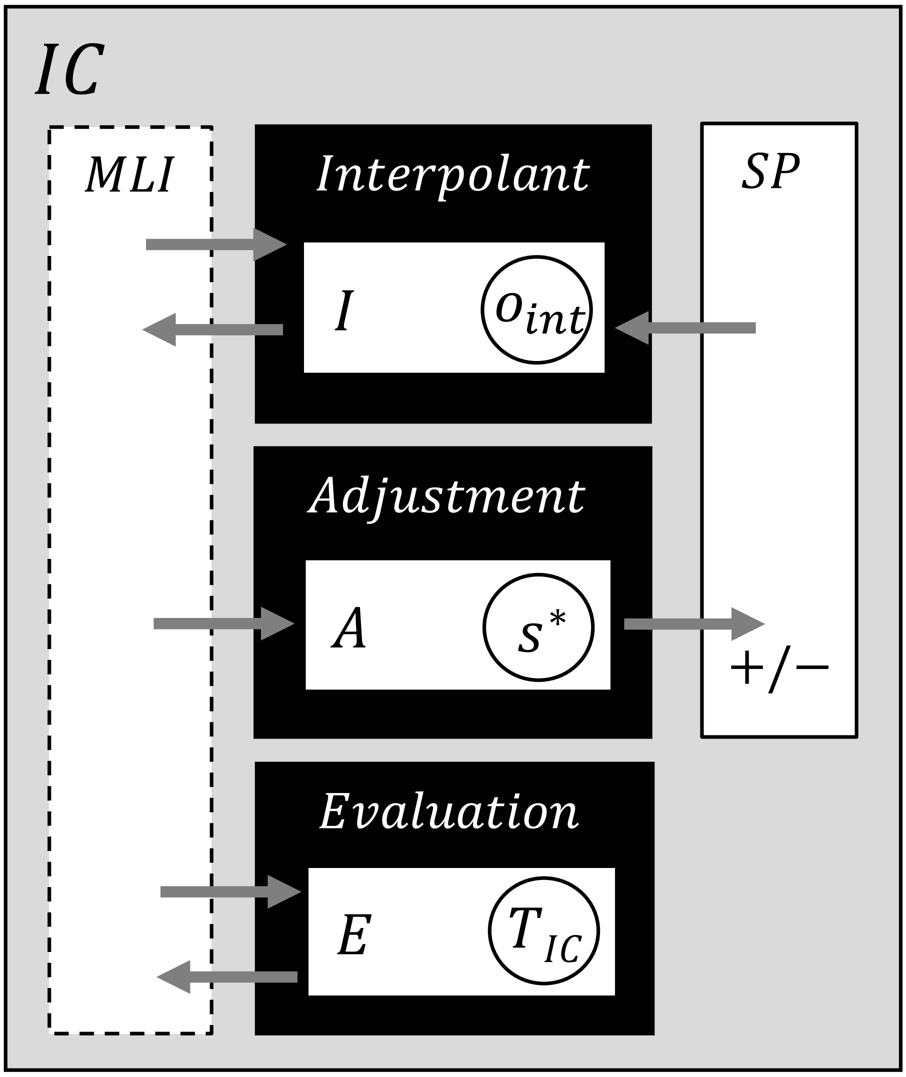


FIGURE 4.2: Schematic of the Interpolation Component from Stein et al. [Ste+17].

continues until either the real-valued queue reaches its maximum length and there is space available for interpolated experiences, or the IER becomes filled with real experiences. As interpolation involves a significant amount of extra work, it might seem counterproductive to discard interpolated experiences. However, this decision was made based on two reasons:

1. When the learner comes near convergence, experiences whose distribution is determined by a query function might harm the real distribution that is derived by following the actual policy. Based on this thought, it can be assumed that the learner benefits more from real experiences as time goes by.

2. The quality of the interpolated experiences is uncertain and a poorly performed interpolation could potentially harm the learner more than a misleading real experience. By discarding interpolated experiences and regularly replacing them with new ones, attempts are made to mitigate this potential effect.

Additionally, a minimum length for the interpolated storage is introduced, that is never fallen below. This results in a differing behavior to the above explained procedure. If the ShrinkingMemory is instructed to reduce its length, it does this only until it reaches this threshold. The maximum length of the IER therefore consists of the real experience buffers maximum length $s_{\text{er\_max}}$ and the minimum length of the synthetic part $s_{\text{syn\_min}}$.

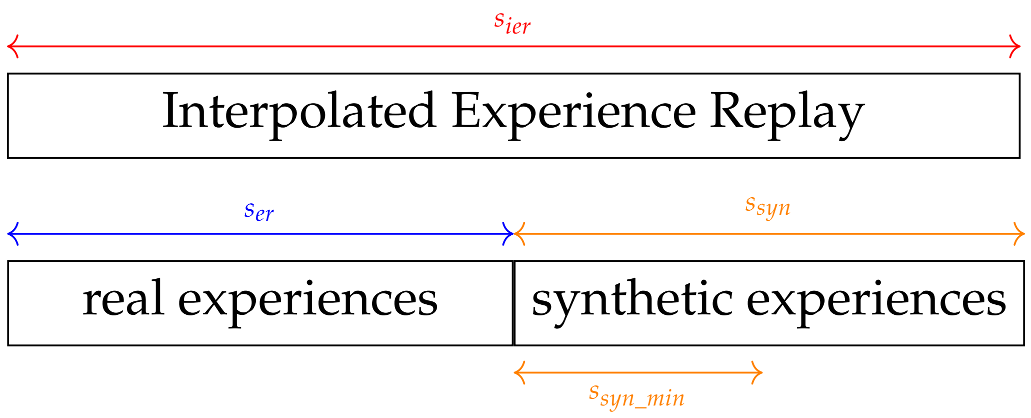


FIGURE 4.3: Intuition of Interpolated Experience Replay memory.

The IER algorithm as described in subsection 4.1.3 is located in the Interpolant, and, as stated above, executed in every step. An exhaustive search would require a computation time of $O(n)$ and therefore is not practical for large sized IERs, because this operation is executed in every single step. A possible solution for this problem is to employ a k-d tree, which represents a multidimensional data structure (cf. section 2.6). By using such a tree, the computation time could be decreased to $O(\log n)$. As the examined problem is very small, and consists out of $|\mathcal{S}| = 64$ discrete states, another approach is used to reduce the computation time further on to $O(1)$. To achieve this a dictionary $dict : K \to V$ of size $|\mathcal{S}| * 3 = 192$ is used that has the following keys:

$$K := \{(S_t, A_t) | S_t \in \mathcal{S}, A_t \in \mathcal{A}\}, \tag{4.12}$$

and corresponding values:

$$V := \left\{R_t^{\text{avg}}, \mathcal{S}_{t+1}\right\}, \tag{4.13}$$

with:

$$\mathcal{S}_{t+1} = \left\{S_{t+1} \in \{S' | (S_t, A_t, R_t, S') \in D^a_{\text{match}}\} | A_t = a\right\}. \tag{4.14}$$

This equals an entry for every state-action pair with associated average rewards and distinct next states of all observed transitions. The dictionary is updated after every step of the agent.

To evaluate the quality of computed interpolations an appropriate metric could be used in the Evaluation part. This is not implemented yet and left for future work.

4.2.2 Modifications and Query Functions

The general architecture of the IER presented above provides a basic functionality but is limited in terms of the synthetic buffer, as a minimum size, if set, can only be a constant value. As previously discussed, the decision to implement the shrinking design was motivated by two specific reasons, one of which relates to the potential harm caused by interpolated experiences that are affected by noise when the learner approaches convergence. A fixed minimum size will never reach zero, whereas a minimum size of zero might get reached too early and small amounts of synthetic experiences could help the learner even after the real valued buffer filled up. To address the described problematic, the parameter ζ is introduced, allowing for fine-tuning. This parameter enables finer adjustment of the minimum size in later stages and is defined as follows:

$$\zeta_i = \max\left[\zeta_{\text{init}} - \left[\frac{\zeta_{\text{init}}}{M} \times i\right], 0\right], \tag{4.15}$$

for every episode i. ζ_{init} is the initial value that is assigned to ζ and a value of 1 serves as a good initialization. The overarching concept of the presented parameter is to linearly decrease the minimum size of the synthetic buffer. Thus, ζ_i decreases until the episode M is reached and has a range of $0 \leq \zeta_i \leq \zeta_{\text{init}}$. To bring ζ into play, the minimum size of the synthetic buffer per episode is defined as:

$$s_{\text{syn\_min},i} = \lfloor s_{\text{syn\_min}} \times \zeta_i \rfloor, \tag{4.16}$$

with $s_{\text{syn\_min}}$ being the minimum size that is set as a parameter in the beginning, and $s_{\text{syn\_min},i}$ being the allowed minimum size at episode i. As can be observed in Equation 4.16, the choice of ζ_{init} manipulates the parameter $s_{\text{syn\_min}}$ and a value of 1 keeps the original value. The minimum size while using ζ can be understood as a linear decreasing value starting at $s_{syn\_min} \times \zeta_{init}$ that reaches zero at episode M.

Even if Algorithm 2 provides a concrete way of how to generate synthetic experiences based on knowledge stored in the replay buffer, it still owes an answer to the question of how x_q is queried and consequently how F is defined. An naive approach would be to sample S_q and A_q uniformly from the state- and action-space, provided that they are known. However, an in-depth evaluation of different query functions in [PSH21] revealed that the naive approach is not very efficient. This might come from the fact that synthetic experiences are added to the buffer distributed in a completely different way (uniformly random) than the real experiences that are generated by following the policy π. Even if the amount of exploration increases big times as even states get queried that the agent did not visit before or at least rarely, it turned out that it is way more beneficial to stick to the policy distribution is some form. Additionally, if the state- and/or action-space grows big, a naive random sampling would generate query points with no accessible sampling points in the majority of times.

Based on the insights from [PSH21] three concrete query function have been defined and the query function F can be concertized to $F(D_t)$ which takes the replay buffer as input. As the replay buffer stores the real experiences that have been generated by following the policy, it inherits this distribution which can be exploited.

The query functions that are used in this work are the following:

1. **Policy Distribution (PD):** To query a state-action pair x_q, a previously observed experience is drawn uniformly at random form the replay buffer D_t. As the distribution present in the replay buffer was generated by following an emerging policy π, this can be seen as drawing query points *off-policy* from the behavior policy π:
$$F_{\text{PD}}(D_t) = (s, a, \_, \_, \_) \sim U(D_t). \tag{4.17}$$

2. **Last State (LS):** The returned query point x_q is st state-action-pair from the last observed experience e_t that has been stored in the replay buffer D_t. As this mechanism returns the state-action-pairs in the same order as the agent faces them, this can be seen as drawing query points *on-policy* from the behavior policy π:
$$F_{\text{LS}}(D_t) = (s, a, \_, \_, \_) \leftarrow D_t[-1]. \tag{4.18}$$

3. ***-Area (*-area):** This query function is an advancement to an arbitrary other query function and shifts a drawn query point to a location that is somewhere in the local neighborhood. The * indicates that it can be used on top of e.g. PD or LS (or any other query function) by performing a transformation operation $T(x_q)$:
$$F_{\text{*-area}}\big(F_*(D_t)\big) = T\big(F_*(D_t)\big), \tag{4.19}$$
whereas $T(x_q)$ needs to be defined for each specific problem individually.

When the linear reward interpolation approach was introduced above, a synthetic experience was generated for each action in $\mathcal{A}$ and each unique follow-up state that has been observed so far. As discussed above, this results in a huge amount of synthetic experiences for the FROZENLAKE problem. To enable a finer tuning of this inflationary generation, different action-selections are introduced, which are presented in the following:

1. **On-Policy (OnPol):** Instead of the generation of a synthetic experience for every action in $\mathcal{A}$ the query state S_q is presented to the actual policy and a synthetic experience is generated for the action the is chosen by it and for every corresponding unique follow-up state.

2. **One Next state (ONS):** Independently of the choice of A_q, for every A_q exactly one synthetic experience is generated. The corresponding follow-up state is

drawn uniformly at random from the set of all follow-up states that have been observed after x_q was executed. Mention that the follow-up state is drawn from the set of all observed follow-up states (not the set of unique follow-up states), which maps the observed distribution of the environments dynamics.

3. **On-Policy & One Next State (OnPolONS):** This action-selection function combines OnPol and ONS and generates exactly one synthetic experience per interpolation step.

4.3 Evaluation

The former sections introduced the functionality and the general concept of the IER and presented a concrete implementation for linear interpolation of rewards for discrete and non-deterministic environments. Even if the research questions **Q** 1 - **Q** 3 have been answered part wise and in a conceptual way, the final proof that the presented ideas correlate with reality still remains to be provided. The following section performs an evaluation with the goal of showing that IER does actually work and is able to assist the learner such that the sample efficiency and the overall performance can be increased (**Q** 4).

4.3.1 FrozenLake Environment

As mentioned above, the FROZENLAKE problem serves as a good environment to evaluate the linear reward interpolation IER on. To gain deeper insights and make the evaluation results more impactful, three different state-encodings have been evaluated. These are introduced in the following:

1. **State Vector Encoding (VE):** The state vector encoding is realized with an array of the length of the state-space ($|\mathcal{S}| = 64$). The whole vector is filled with zeros and the entry that corresponds to the actual state is set to 1. This results in an input layer of size 64.

2. **Coordinates Encoding (CE):** The coordinates state encoding is realized via a vector with two entries that hold the value for the normalized x- and y-coordinate. An input layer of size two is used here.

3. **Local Knowledge Encoding (LKE):** In the local knowledge state encoding the agent receives a vector with 8 entries that corresponds to the surrounding fields of the actual state. The different state types are shown in Table 4.1.

   An input layer of size 8 was utilized. In this encoding the problem of perceptual aliasing [WB91] is faced, as some states have the exact same encoding but in fact are different. The complexity of the problem increases because of these states.

TABLE 4.1: Overview of the values for the different types of tiles in the LKE state-encoding.

tile type	Value
initial state	0
final state	1
frozen	2
hole	3
out of state space	4

A graphical illustration of the different encodings can be observed in Figure 4.4.

With these different state-encodings at hand, the **-area* query function needs to be defined for each one individually:

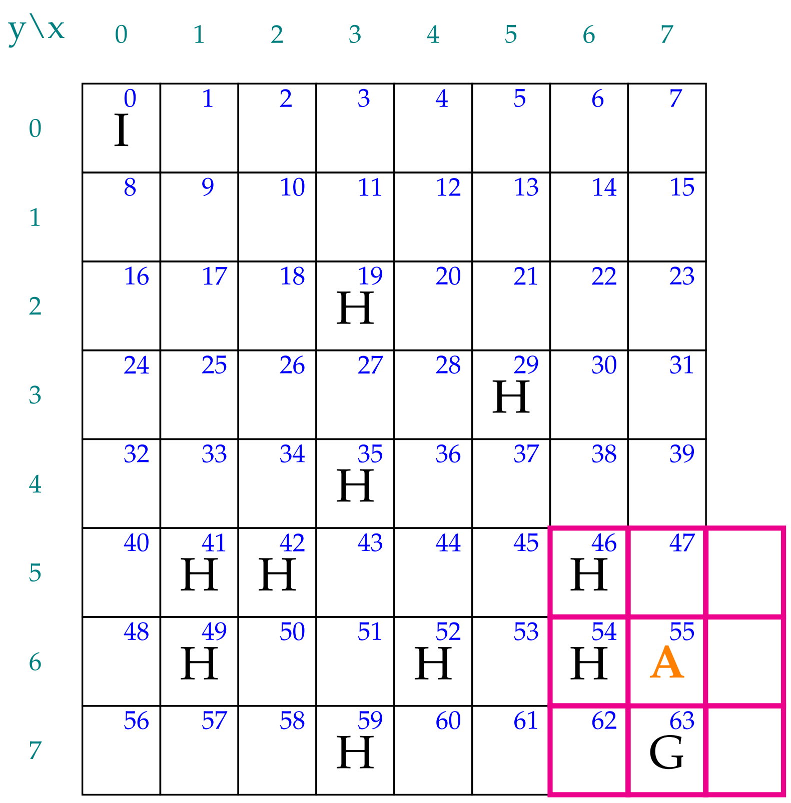
1. **VE & CE:** As for VE and CE a unique position in the grid is given, it is possible to calculate a set of potential points that consists of the original point S_q and all legal surrounding points. To do so, independently of the actual encoding, the coordinates of x_q are identified at first. Afterwards, a random number is drawn for every dimension from the set $\{-1, 0, 1\}$ and added to the coordinate of original S_q. Values below zero and above 7 are clipped to 0 zero and 7.. By doing so, the *-area query function draws a state from the set of x_q and its neighbors and consequently explores the state-space in a corridor around the behavior policy.

2. **LKE:** Unfortunately, the same approach is not applicable to LKE. That is, because no information about the actual position in the grid is provided and, as mentioned above, similar encoded states exists that are actually different from each other (perceptual aliasing). To enable some sort of additional exploration nevertheless, the following approach was found: At first, one of the 8 positions in the state-encoding is chosen. Afterwards, the corresponding entry is replaced with a random integer between 2 and 4 (frozen, hole, out of space). To prevent the generation of states that are not present in the environment, a check in the state-dictionary (cf. subsection 4.2.1) is performed and if the new generated state has not been observed before, then it is discarded.

4.3.2 Experiments

To perform an in-depth evaluation of the linear reward interpolation approach the following configurations were tested against a DQN baseline with standard ER:

1. **Def:** This configuration represents the original idea of an interpolated buffer that has the same maximum size as the real valued buffer which decreases to zero as the amount of true experiences grows. Synthetic experiences are generated in every step for the whole action-space and every observed follow-up state.



state vector encoding = [0,. . . ,0,1,0,. . . ,0]

coordinates encoding= $[\frac{7}{7}, \frac{6}{7}] = [1, 0.86]$

local knowledge encoding = [3,3,2,4,4,4,1,2]

FIGURE 4.4: A graphical illustration of the three different state encodings. Every tile is represented by an index for the VE encoding. A coordinate system is used for the CE encoding and the surrounding 8 tiles represent the LKE encoding. An example for the state A is given for every encoding in the bottom.

2. **Def-OnPol:** Same as Def configuration, but synthetic experiences are generated for the action that the current policy predicts for the query state and all corresponding observed follow-up states.

3. **Def-ONS:** Same as Def configuration, but exactly one synthetic experience is generated for every action in the action-space. For those experiences follow-up states are drawn at random from the collection of all corresponding observed follow-up states.

4. **Del-area:** Same as Def configuration, but the query points are drawn according to the *-area query function (cf. subsection 4.2.2).

5. **Def-area-OnPol:** Same as Def-OnPol, but the query points are drawn according to the *-area query function.

6. **Del-area-ONS:** Same as Def-ONS, but the query points are drawn according to the *-area query function.

7. **Full:** This configuration has set the minimum size of the synthetic buffer to the same value as the real valued buffer. This corresponds to an equal amount of synthetic and real experiences as soon as both buffers filled up. Synthetic experiences are generated in every step for the whole action-space and every observed follow-up state.

8. **Full-area:** Same as Full configuration, but the query points are drawn according to the *-area query function.

9. **Zeta-OnPolONS:** This configuration has set the minimum size of the synthetic buffer to half of the size of the real valued buffer. Additionally, ζ is used to reduce the minimum size of the synthetic buffer to zero over a long period. Exactly one synthetic experience is generated per step for the action that is predicted by the current policy and one follow-up state drawn at random from the collection of all corresponding observed follow-up states.

10. **Zeta-area-OnPolONS:** Same as Zeta-OnPolONS configuration, but the query points are drawn according to the *-area query function.

All configurations have been tested for the three different state-encodings (VE, CE and LKE) and different exploration phases ($t_{\text{expl}} = \{250, 500, 750, 1000\}$). Both, the configurations as well as the baseline, were repeated 40 times with different random seeds. An ϵ-greedy exploration was used to explore the state-space. A complete overview of the used hyperparamets is given in the next section.

4.3.3 Hyperparameters

The hyperparameters that have been used for both, the baseline as well as the different IER configurations, have been tuned by a hyperparameter search. Most hyperparameters are shared between all configurations and the baseline to ensure comparability. Table 4.2 provides a complete overview of the hyperparameters that are shared by all experiments:

As only the IER configurations make use of a synthetic buffer, $s_{\text{syn\_min}}$ is of no relevance for the baseline or rather set to zero. The same thing holds for the point in time when interpolation starts. Both hyperparameters are shared for all IER configurations however. The chosen values can be observed in Table 4.3.

Different architectures of the DNNs are used for each state-encoding and shared for all IER configurations and the baseline. Table 4.4 provides an overview of the architectures.

The minimum size of the synthetic buffer $s_{\text{syn\_min}}$ differs for the following three configurations: Def, Full, Zeta. All configurations that are combinations of them (*-OnPol, *-ONS, etc.) share these values. The Zeta configurations use ζ while for

TABLE 4.2: An overview of the hyperparameters that are shared by all configurations and the baseline.

Parameter	Values
Learning rate α	0.0005
Gamma γ	0.99
Epsilon start	1
Epsilon min	0
Tau τ	0.25
Soft target update	True
Double DQN	True
Dueling Architecture	True
Real valued buffer size $s_{\text{er\_max}}$	50,000
Start learning at size of buffer	300
Minibatch size	32

TABLE 4.3: An overview of the hyperparameters that are shared by all configurations.

Parameters	Values
Synthetic buffer size $s_{\text{syn\_max}}$	50,000
Start interpolation at size of buffer	100

TABLE 4.4: An overview of the neural network architectures that are shared by all configurations and the baseline of a state-encoding.

State-encoding	Input	Hidden Layer	Output
VE	64	[32, 32]	8
CE	2	[128, 256]	8
LKE	8	[64, 64]	8

all other configurations no ζ was used. Table 4.5 provides an overview of the used parameters.

TABLE 4.5: An overview of the different hyperparameters for the different IER configurations.

Configuration	$s_{\text{syn\_min}}$	ζ	M
Def	0	-	-
Full	50,000	-	-
Zeta	25,000	1	1500

4.3.4 Statistical Analysis

For all experiments, a *minimum average reward over 100 consecutive episodes* (MAR) was identified that, when achieved by the agent, indicates that the problem is solved. The number of episodes until that criterion is fulfilled for the first time is called the *time to solution* (TTS). Furthermore, after a maximum of 4000 episodes, if the solution criterion was not achieved, runs are aborted and the task subsequently considered as *not solved in time*.

The main interest of the evaluation is on the required iterations until a problem is solved, the sample efficiency and if IER outperforms original ER (in the following referred to as *ER*) in this aspects (**Q** 4). To be able to provide a scientific answer to that research question, the following evaluation questions (*E-Qs*) have been defined for each learning task $\mathcal{T}$ based on the data collected in the conducted experiments:

E-Q1 If task $\mathcal{T}$ was not solved in time by *some* of the runs for at least one of IER or ER, the *solution rates* are the first metric to be regarded: How do the two approaches compare with respect to the rate of runs that *did* solve the task in time? This entails: Should it be believed that IER's solution rate on task $\mathcal{T}$ is greater than the one of ER or vice versa? How confident can one be about this given the data that has been collected?

E-Q2 Based on only the subset of runs that *did* solve the task in time the question that is tried to be answered is: How does IER's TTS compare to the one of ER? This entails: Should one believe that IER solves task $\mathcal{T}$ faster than ER? If so, then by how much? How confident can one be given the data that has been collected?

E-Q3 What are likely values of the TTS for FU-DQN and DQN for a task $\mathcal{T}$?

In order to be able to make statistical statements like these, the data from the experiments is analyzed using two Bayesian models[1] which provide posterior distributions that can be queried flexibly. At that, both models are provided by the *cmpbayes* Python library [Pät22].

Statistical model for the solution rates

In order to compare two *solution rates* p_i, $i \in \{\text{IER}, \text{ER}\}$ on a task $\mathcal{T}$ (E-Q1), a simple beta-binomial model is employed. This model consists of a binomial model for $n_{\text{solved},i}$, the number of times that $\mathcal{T}$ was solved by method i out of the number of runs performed n (the latter being the distribution's *number of trials* parameter which is fixed to a value of 40), and a beta distribution prior on the binomial distribution's *trial success probability* parameter which corresponds to the solution rate p_i. The prior belief of all values of p_i is encoded as being equally likely by choosing the beta prior parameters such that a uniform distribution is received.

[1]Null-hypothesis significance tests are deliberately not used due to their many flaws, an overview of which are given in [Ben+17].

The overall model for a pair of solution rates p_i (i. e. $i \in \{\text{IER}, \text{ER}\}$) looks like this:

$$n_{\text{solved},i} \sim \text{Binomial}(n, p_i) \tag{4.20}$$

$$p_i \sim \text{Beta}(a, b) \tag{4.21}$$

$$a = 1, \quad b = 1 \tag{4.22}$$

The beta-binomial model provides the posterior distribution of the parameters p_i which can then be queried. Since the most interesting fact is whether the solution rate of IER is greater than the one of ER, the probability $p\left(p_{\text{FU-DQN}} > p_{\text{DQN}}\right)$ is computed which can be done straightforwardly by drawing large-enough samples from the two distributions of the p_i, pairing them at random, and then counting how many of these pairs fulfill the predicate.

Statistical model for the TTS units

The model used for comparing *two sets of TTS units* (one set for IER and one for ER; used for answering E-Q2 as well as E-Q3) for a task $\mathcal{T}$ is inspired by the model described by Kruschke [Kru13] but differs in that it is based on two gamma distributions[2] (one for each set of TTS units denoted as random variables TTS_i, $i \in \{\text{IER}, \text{ER}\}$, in the following), which are reparametrized for a mean and shape parameter. The shape parameter corresponds to the usual gamma distribution rate parameter β_i whereas the mean parameter is defined as $\mu_i = \alpha_i^\gamma / \beta_i$ (where α_i^γ is the usual shape parameter). As a prior for the mean parameter, a broad exponential distribution is used—parametrized such that 90 % of means lie in $[0, 8000]$ where the upper bound stems from multiplying the maximum number of episodes by two (it can be stated quite confidentially that the real mean TTS is smaller than 8000 for the tasks analyzed using this model). Since a mean TTS of 0 can be ruled out a priori, this exponential distribution is further shifted by one hundredth of the minimum of the measured units. The prior on the shape parameter β_i is a uniform distribution chosen such that the variance of the distribution is at least l times the variance of the data and at most u times the variance of the data (note that β_i does not directly correspond to the variance of the distribution but its bounds can be derived given the mean μ_i). Just like Kruschke in [Kru13], these are chosen rather noncommittally as one-thousandth and a thousand.

[2]Note that since TTS values can never be negative, other common distributions like normal or student's t distributions can not be expected to model the data well.

The full model for a pair of TTS units TTS_i (i. e. $i \in \{\text{IER}, \text{ER}\}$) looks like this[3]:

$$TTS_i \sim \text{Gammma}(\mu_i, \beta_i) \tag{4.23}$$

$$\mu_i - \frac{\min(TTS_i)}{100} \sim \text{Exp}(\lambda_i) \tag{4.24}$$

$$\beta_i \mid \mu_i \sim \mathcal{U}\left(\frac{\mu_i}{u\text{Var}(TTS_i)}, \frac{\mu_i}{l\text{Var}(TTS_i)}\right) \tag{4.25}$$

$$\lambda_i = -\frac{\ln(i - 0.9)}{u_\mu} \tag{4.26}$$

$$l = 1000^{-1}, \quad u = 1000, \quad u_\mu = 8000 \tag{4.27}$$

The parameters of interest in this model are the μ_i (i. e. the means of the distribution of the TTSs) and even more so their *difference* as the goal is to estimate not only their ordering but also the magnitude of any effects. Since the model provides a posterior distribution for the μ_i, they can be sampled, paired at random and then subtracted to obtain a sample of the distribution of $\mu_{\text{ER}} - \mu_{\text{IER}}$. Provided that the samples are large enough, it is possible to reason about the underlying distribution itself and, for example, compute probabilities of this difference being positive or negative (E-Q2). Also, the individual distributions of means μ_i, contain the information required to answer (E-Q3).

Note that the *cmpbayes* library [Pät22] uses Markov Chain Monte Carlo (MCMC) sampling to obtain the samples for this model (four independent chains of 10000 samples each are sampled after a warm-up phase of 1000) and that the usual checks were performed to detect whether the sampler behaved well and converged (trace plots, posterior predictive checks, $\hat{R}$ values, effective sample size).

Practical equivalence

Since differences of the TTS that do not have practical significance are not of interest, a *region of practical equivalence* (rope) is defined around a TTS difference of 0. Since practical equivalence in terms of the TTS can be expected to differ between tasks (some are harder and thus the agent naturally requires more training episodes), the rope was chosen task-dependently as 1% of ER's median performance m_T on the task T:

$$\text{ROPE}_T = [-0.01 m_T, 0.01 m_T] \tag{4.28}$$

4.3.5 Results

For the sake of space, only the best performing configurations for the three state-encodings and the four different exploration lengths will be presented in this section.

[3]This is slightly simplified for the sake of space. The actual model used also factors in the runs that were aborted after episode 4000 as *censored data*. See the *cmpbayes* library [Pät22] for details.

A complete overview of the results of the conducted experiments is provided in Appendix A.

The MAR for the FROZENLAKE problem (all state-encodings) was set to a value that indicates that the agent was able to solve the problem 85 times out of 100 consecutive episodes. Such a value was chosen because of the non-deterministic nature of the problem. Even following a good policy could lead the agent into a whole and therefore, a success rate of 100 % is very hard to achieve.

Analysis of the results for the VE state-encoding

The VE state-encoding is considered to be the easiest problem (for DQN in general) because there exists one input node for every possible state and only one of them provides a signal. On the other hand, less generalization might be possible as inputs do not align to each other in a numerical way.

However, as can be seen in Table 4.6, for all exploration lengths without 750 there was found a configuration that is expected to solve the problem in less episodes than ER with a probability greater than 90 %. Only for an exploration length of 750, the best found configuration (Def-OnPol) is expected to perform better (at least as good) as ER with a probability of 41.08 % (47,2 %) which is both below a probability of 50 %. This might be attributed to the fact that an exploration length of 750 is a highly effective parameterization for ER. Consequently, IER may not be particularly useful in this case. While Zeta-OnPolONS appears two times in the best performing configurations, also other configurations such as Def-area-ONS performed well. The only configuration that performed badly (but still good for an exploration length of 1000) was Full-area. For the choice of the query function, no definitive statement can be made regarding whether LS or PD is clearly better, as both performed similarly well. Overall there is strong evidence towards IER outperforming ER for $t_{\text{expl}} = [250, 500, 100]$ in terms of TTS.

TABLE 4.6: Probabilities (in %, rounded to two decimal places) for the best performing configurations in the **VE** state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes.

Config	t_{expl}	Query Function	ER better	practical equivalent	IER better
Def	250	LS	8.51	0.15	91.35
Zeta-OnPolONS	500	PD	0.00	0.00	100.00
Def-OnPol	750	LS	52.81	6.12	41.08
Zeta-OnPolONS	1000	PD	0.57	0.52	98.92

A look at Table 4.7 shows the results of the comparison of the solution rates using the beta-binomial model. The results align with the findings of the TTS comparison, indicating that configurations with high probabilities of outperforming ER in terms of TTS also tend to have higher probabilities of achieving a greater solution rate. Overall there can be observed a definite tendency towards the solution rate of IER exceeding the one of ER for $t_{\text{expl}} = [250, 500, 100]$ for the VE state-encoding.

TABLE 4.7: Probability (in %) for the best performing IER configurations in the **VE** state-encoding that the solution rate of IER is greater than the solution rate of ER.

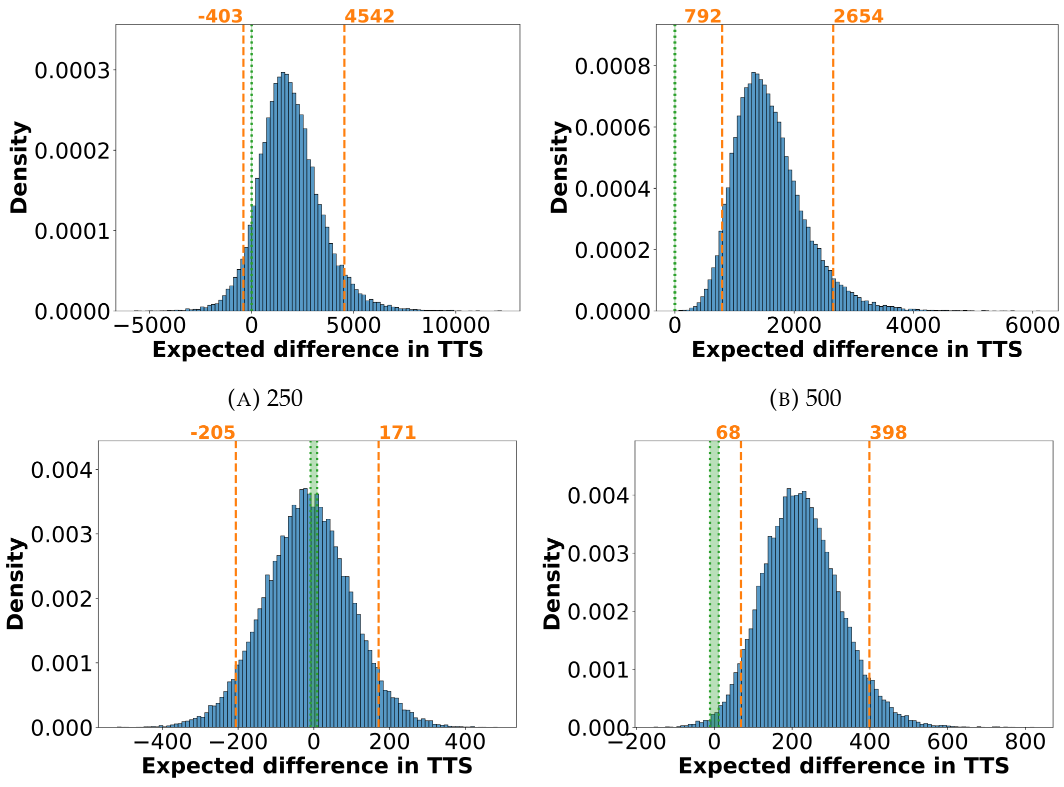
Config	t_{expl}	**Query Function**	$p\,(p_{\text{IER}} > p_{\text{ER}})$
Def	250	LS	91
Zeta-OnPolONS	500	PD	100
Def-OnPol	750	LS	50
Zeta-OnPolONS	1000	PD	94

Figure 4.5 displays the distributions of the means of the TTS distribution for the four best performing configurations for the VE state-encoding. The graphs are visualizations of the results from Table 4.6. The plot for $t_{\text{expl}} = 750$ shows that the probability mass is distributed around the rope which indicates practical equivalent behavior. Moreover, the majority is located slightly to the left which aligns with a higher probability of ER outperforming IER in that case. For the other three distributions, it is visible that the majority of the probability mass is located to the right of the rope. Figure 4.5 also shows the central 95 % HDPI which indicates that for $t_{\text{expl}} = 500$ there is very high confidence to state the true value of the mean difference lies between 792 and 2654 episodes.

Table 4.8 gives summary statistics of the distributions over the TTS means (the distributions of μ_i for $i \in \{\text{ER}, \text{IER}\}$). The most likely values are reported (mode of the distribution over mean values μ_i) as well as a measure of uncertainty (once more, 95 % HPDIs). It can be observed, that ER requires less time when t_{expl} decreases while 750 being the best configuration. IER on the other hand has smaller mean TTSs for most of the exploration lengths and converges at around 1100 mean TTS beginning with a t_{expl} of 500. The differences of the mean TTSs align with the peaks of the probability distributions of the corresponding plots in Figure 4.5.

TABLE 4.8: Most likely values and uncertainties for the mean TTS of IER and ER for the best performing configurations of the **VE** state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}.

Config	t_{expl}	**IER mean TTS**		**ER mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Def	250	3046	[2122, 4737]	4229	[3404, 7483]
Zeta-OnPolONS	500	1071	[937, 1323]	2442	[1930, 3795]
Def-OnPol	750	1057	[943, 1213]	1055	[933, 1192]
Zeta-OnPolONS	1000	1169	[1127, 1214]	1394	[1244, 1563]



(A) 250 (B) 500 (C) 750 (D) 1000

FIGURE 4.5: Density plots of the posterior distribution of $\mu_{\text{DQN}} - \mu_{\text{FU-DQN}}$ for best performing IER configurations of the **VE** state-encoding. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent.

Analysis of the results for the CE state-encoding

The CE state-encoding reduces the amount of input nodes and increases the signals at the same time. Furthermore, signals describe coordinates and neighboring coordinates have spatial dependencies that are present in the inputs.

In general, IER demonstrated excellent performance on the CE encoding, with both the Def-area and Def-area-onPol configurations consistently showing probabilities of 100 % in favor of IER in terms of TTS for almost all configurations and exploration lengths. Table 4.9 shows the results for Def-area and the LS query function, but PD performed equally well. Overall, LS performed better on this state-encodings for the configurations that did not reach a probability of 100 %. The worst performing configurations have been Zeta-* and Zeta-area-*. All other configurations performed well when using the LS query function.

Table 4.10 shows that the solution rate is favored for IER as well.

Figure 4.6 confirms the results from above with the probability masses being to the right of the rope for all exploration lengths. It can be stated with very high confidence that IER outperforms ER in the CE state-encoding in terms of TTS as well

TABLE 4.9: Probabilities (in %, rounded to two decimal places) for the best performing configurations in the **CE** state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-area	250	LS	0.00	0.00	100.00
Def-area	500	LS	0.00	0.00	100.00
Def-area	750	LS	0.00	0.00	100.00
Def-area	1000	LS	0.00	0.00	100.00

TABLE 4.10: Probability (in %) for the best performing IER configurations in the **CE** state-encoding that the solution rate of IER is greater than the solution rate of ER.

Config	t_{expl}	**Query Function**	$p\left(p_{\text{IER}} > p_{\text{ER}}\right)$
Def-area	250	LS	75
Def-area	500	LS	100
Def-area	750	LS	100
Def-area	1000	LS	100

as the solution rate.

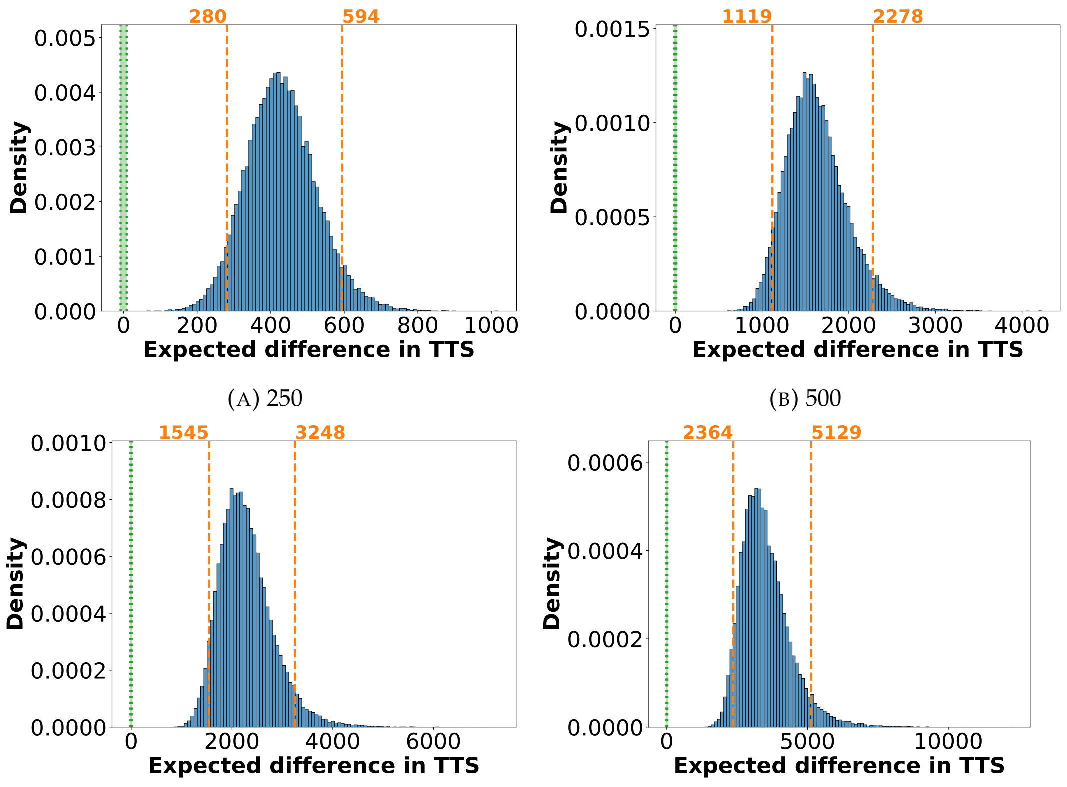
While Table 4.11 confirms the previously mentioned results, it also reveals a distinct behavior of the state encoding compared to the VE state encoding used earlier. Contrary to the expected decrease, the mean TTS shows an increase with higher values of t_{expl}. This effect is more pronounced in ER but is also noticeable in IER.

TABLE 4.11: Most likely values and uncertainties for the mean TTS of IER and ER for the best performing configurations of the **CE** state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}.

Config	t_{expl}	**IER mean TTS**		**ER mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Def-area	250	649	[601, 699]	1038	[941, 1234]
Def-area	500	980	[921, 1053]	2518	[2112, 3250]
Def-area	750	1463	[1377, 1568]	3549	[3022, 4715]
Def-area	1000	1738	[1605, 1867]	4885	[4098, 6889]

Analysis of the results for the LKE state-encoding

The LKE state-encoding is the most complex one as the global knowledge that was present in the former two encodings is replaced with a local knowledge and no spatial dependency is present between neighbouring states.



(A) 250 (B) 500 (C) 750 (D) 1000

FIGURE 4.6: Density plots of the posterior distribution of $\mu_{\text{DQN}} - \mu_{\text{FU-DQN}}$ for best performing IER configurations of the **CE** state-encoding. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent.

While for all exploration lengths a configuration could be found that outperforms ER in terms of TTS with a probability of at least 64 %, the majority of configurations performed poorly and had a 100 % probability of being worse than ER. For the Full-area configuration and the Full configuration for $t_{\text{expl}} = 1000$ LS not a single run was able to solve the problem in time and consequently no statistical model was computable. However, the configurations that have shown acceptable to good performance are: Def-ONS, Def-area-ONS, Zeta-OnPolONS and Zeta-area-OnPolONS. Table 4.12 shows the best configurations and overall it can be observed that ONS and Zeta-* are preferred configurations for the LKE encoding. Furthermore, LS appears to outperform the PD query function, although the difference is not substantial.

In terms of solution rates, Table 4.13 reveals that the found configurations can be expected to outperform ER. This mirrors the observed results from the former state-encodings.

Figure 4.7 displays the probability distribution of the means of the differences in TTS and, once more, confirms the results from Table 4.12. Even if the majority of the probability mass is to the right of the rope, it is not as obvious as with the CE encoding. Furthermore, the span of the HDPI is much smaller when compared with

TABLE 4.12: Probabilities (in %, rounded to two decimal places) for the best performing configurations in the **LKE** state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Zeta-area-OnPolONS	250	LS	29.88	5.83	64.29
Zeta-OnPolONS	500	LS	12.91	3.68	83.41
Def-ONS	750	PD	16.60	4.50	78.90
Zeta-OnPolONS	1000	LS	5.32	2.40	92.28

TABLE 4.13: Probability (in %) for the best performing IER configurations in the **LKE** state-encoding that the solution rate of IER is greater than the solution rate of ER.

Config	t_{expl}	**Query Function**	$p\,(p_{\text{IER}} > p_{\text{ER}})$
Zeta-area-OnPolONS	250	LS	60
Zeta-OnPolONS	500	LS	94
Def-ONS	750	PD	86
Zeta-OnPolONS	1000	LS	78

the CE plots.

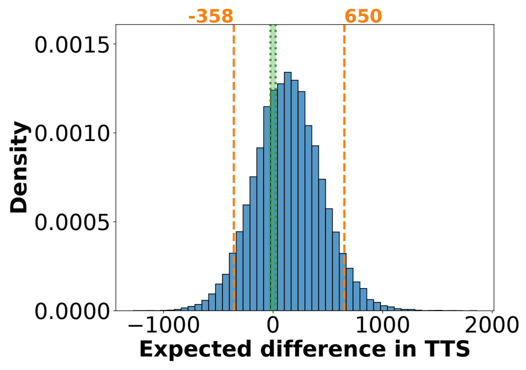
The raw mean values of the TTSs reveal, that an increase of t_{expl} does not have any remarkable influence on the TTS. The time that is needed to solve the LKE encoding on average is at around 3000 episodes for ER and a few hundred episodes below for the best performing IER configurations.

TABLE 4.14: Most likely values and uncertainties for the mean TTS of IER and ER for the best performing configurations of the **LKE** state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}.

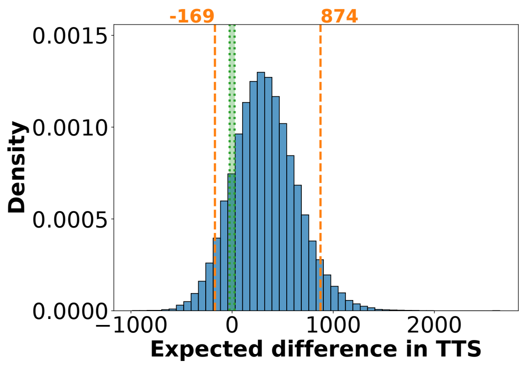
Config	t_{expl}	**IER mean TTS**		**ER mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Zeta-area-OnPolONS	250	2860	[2556, 3219]	3043	[2654, 3406]
Zeta-OnPolONS	500	2910	[2645, 3198]	3183	[2832, 3721]
Def-ONS	750	2739	[2528, 3051]	2983	[2676, 3443]
Zeta-OnPolONS	1000	2854	[2619, 3258]	3240	[3033, 3695]

4.4 Discussion

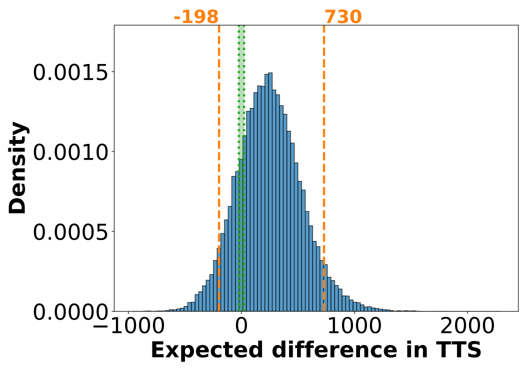
The evaluation results of the Linear Reward Interpolation for discrete and non-deterministic environments on three different state-encodings of the FROZENLAKE8x8 problem showed that the general concept of IER does work. It was possible to find an IER configuration for all encodings and almost all exploration lengths that has a



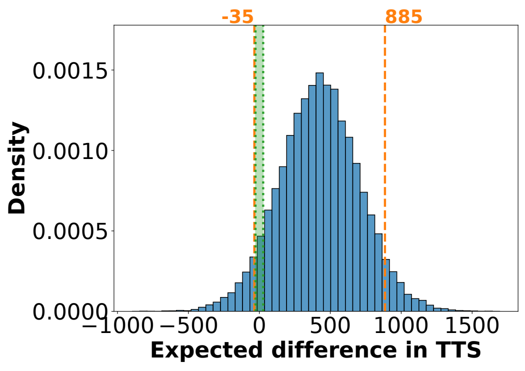
(A) 250



(B) 500



(C) 750



(D) 1000

FIGURE 4.7: Density plots of the posterior distribution of $\mu_{\text{DQN}} - \mu_{\text{FU-DQN}}$ for best performing IER configurations of the **LKE** state-encoding. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent.

positive probability (> 50 %) towards solving the problem in less episodes than the ER baseline. Most of the configurations even have a very high probability (> 80 %) towards outperforming ER in terms of TTS. It was however, not possible to identify a superior configuration and query function that performed best in all investigated problems. In fact, IER with linear reward interpolation is an algorithm that comes with a high number of hyperparameters that need to be tuned individually for every problem. The three investigated state-encodings represent different challenges and difficulties and the proof-of-concept approach was able to solve all of them in a satisfying way. It was possible to show that the right IER configuration is able to increase the solution rate of a problem when compared with a baseline ER. This can be interpreted such that IER with linear reward interpolation is able to help the learner to solve a problem more consistently. This might be reasoned by an increased exploration that is able to guide the policy out of local maxima.

In general it can be said that IER with linear reward interpolation provides a solution for exploiting gathered knowledge to generate meaningful synthetic experiences (**Q** 1), an initial answer to the question of what parts of an experience can be interpolated (**Q** 2) and how synthetic experiences can be used to assist a DQN with learning (**Q** 3).

As the presented IER approach was able to solve the problems in less TTS than ER, it can be stated quite confidently that *the usage of synthetic experiences in combination with a DQN results in faster convergence* (**Q** 4) or at least in a solution of the defined problem in less episodes. If less episodes are required to solve a problem, then observed samples (i.e. experiences) have more impact and consequently the sample efficiency is increased (**Q** 4). Furthermore, as the usage of synthetic experiences is beneficial and true experiences are used to generate them, the sample efficiency is increased even further.

However, the IER approach with linear reward interpolation is a proof-of-concept that is designed for discrete and non-deterministic environments and will only work in such. This is of course a restriction that prevents the IER approach from being used in most of the problems out there. The main restriction is that follow-up states are not interpolated in this initial version of IER, and the reason behind this decision was discussed in section 4.1. However, if this restriction could be lifted, then IER would be ready for continuous environments as well.

Chapter 5

Interpolated Experience Replay

The former chapter introduced an initial proof-of-concept IER that performed well and showed that the concept of using synthetic experiences in a replay buffer can be beneficial. However, only the reward of the synthetic experience was interpolated and the follow-up state, as well as the done tag have been copied from observed experiences. This is a major restriction and makes the algorithm only usable in discrete and non-deterministic environments. To enable IER for continuous environments as well, this restriction has to be lifted. The following chapter will take a close look at the problem of the follow-up state interpolation and try to find a method to generate synthetic follow-up states so that synthetic experiences can be generated for any position in a continuous state-space. With such a solution at hand, IER is developed to a state that is usable in a wide range of problems and beyond a proof-of-concept. Continuous environments require sophisticated nearest neighbor searches and consequently methods that can do that. Furthermore, as the buffer is changing constantly, solutions for fixed datasets can not be used or need to be adapted. An in-depth evaluation proofs that IER works in continuous environments.

Parts of the following text have already been published in a similar form in [PSH22b].

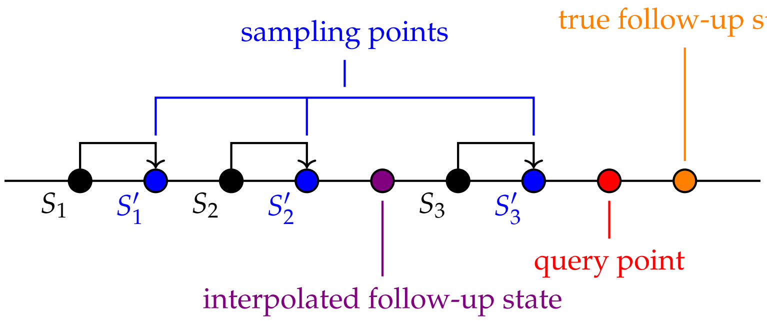
5.1 Interpolation of the Follow-up State

The analysis of the individual components of an experience, regarding their interpolability (cf. subsection 4.1.1), revealed that certain elements, such as the follow-up state and the terminal tag, pose higher risks when interpolated compared to others, such as the reward. If one is willing to take the risk of interpolating the follow-up state (and the terminal tag), then the interpolation error should be as small as possible.

5.1.1 Problem Description

A naive approach to interpolate a follow-up state (and a terminal tag) would be the usage of the raw values of the observed follow-up states of NN_q as sampling points. In a very simple environment, a state is a position on a line and the action-space is defined as $\mathcal{A} = \{\text{move-left}, \text{move-right}\}$. Figure 5.1 depicts a situation that might occur and that the naive interpolation of raw follow-up states is not able to solve in a satisfying way. The situation is such that a query point x_q is drawn with a query action $A_q ==$ move-right. All available experiences however, that are stored in the replay buffer and located within a given distance (NN_q) start to the left (S_*) and also all corresponding follow-up states (S'_*) are located to the left of x_q as well. When the raw values of S'_* hold as sampling points, then the corresponding interpolated follow-up state would be somewhere in between them, as can be seen in the example as the violet point. However, as the action is move-right, the true follow-up state that corresponds to x_q should be somewhere to the right (orange point). The displayed example is in fact a case of extrapolation instead of interpolation as the searched point is located outside of the sampling points. Unfortunately, there is no way to detect such cases in a reliable way without using domain knowledge that is not available to the agent. Furthermore, extrapolation is a much more complex problem than interpolation and comes with a lot of other difficulties that rather complicate the main task of generating a synthetic follow-up state instead of reducing risk and interpolation errors. Of course, such a case will not appear too frequently and one might argue, that such cases could be ignored if most of the follow-up state interpolations are exact enough, but as discussed in subsection 4.1.1 the follow-up state is a crucial part of the learning update and thus the experience itself. As the goal is to reduce risk and not to increase it (even in eventually small amounts of cases) the naive approach of the interpolation of raw follow-up states is not suitable for IER.

FIGURE 5.1: Illustration of the follow-up state interpolation dilemma.

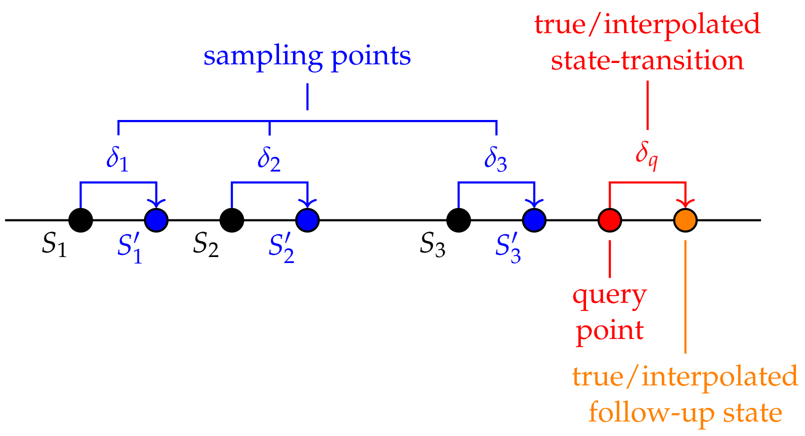


If the terminal tag is interpolated by using observed tags from NN_q, it might appear a problematic situation as well. In the simple environment from above, the goal is defined at some position, and reaching some point to the right of that point terminates a run. Dependent on NN_q, it might appear the case when an experience would not reach that point, but the interpolated terminal tag is set to true. Such an experience could irritate the learner as (depending on the reward function) no (positive) reward is received and the expected positive action-value of the next action that reaches the goal is missing. The other case would be that an experience would reach the goal but because of the sampling points, the terminal tag is set to false and thus, the TD-error would inherit an action-value of a state that has never been visited as it is impossible to do so. Of course such a synthetic experience would irritate the learner as well. In contrast to the extrapolation dilemma from above, the problematic cases for the terminal tag can only appear near the goal state. This reduces the likelihood of occurrence, making it small enough to be acceptable. Furthermore, as the agent is guided towards the goal state, it visits states close to the goal state more frequently; this reduces the likelihood of occurrence even further. Even if some interpolations of the terminal tag might be misleading, it can be stated quite confidentially that the amount of them will be small enough to be neglected.

5.1.2 Transition-Delta Interpolation

As presented above, using raw observed follow-up states as sampling points can produce unrecognized extrapolation cases that can not be handled sufficiently. To nevertheless be able to generate interpolations of the follow-up state, another solution is required. Figure 5.2 shows the same situation as before, but the state-transition-deltas are used as sampling points instead. As a state-transition always directly correlates with the corresponding start state the state-transition-delta holds as a good value to interpolate. An action chosen in a given state leads to a concrete follow-up state, and thus, some form of dependency can be assumed here. Furthermore, extrapolation cases can not appear, no matter of how NN_q is distributed. An interpolated state-transition-delta can be added to S_q and as it was interpolated by sampling points that executed A_q, the produced synthetic follow-up state S_{syn} can be expected to be within a neglectable distance of the true value. This also holds for more complex environments (higher dimensional) as long as S_{t+1} is directly dependent from S_t and A_t. State-transition-deltas interpolated for more complex state-spaces might not match the true value exactly, but, as mentioned above, the error is expected to be small.

FIGURE 5.2: Illustration of the interpolation of the state-transition-delta to generate stable follow-up state interpolations.



The state transition delta is computed as follows:

$$\delta_t = S_{t+1} - S_t, \tag{5.1}$$

and with an interpolation of the state-transition-delta δ_q at hand, the synthetic follow-up state S_{syn} can be defined as:

$$S_{\text{syn}} = S_q + \delta_q. \tag{5.2}$$

Research question **Q** 2 asks what parts of an experience can be interpolated and what needs to be considered by doing so. As discussed above, it is possible to interpolate the reward and the terminal tag straight ahead. The follow-up state itself can be generated by interpolating the state-transition-delta, and thus, all parts of an experience can be generated by means of interpolation. If these interpolations are good enough to assist the learner in a positive way however, is a question that still remains to be answered by an evaluation.

5.2 Interpolated Experience Replay for Continuous Environments

Having a solution available for interpolating the follow-up states and terminal tags, IER can be effectively prepared for continuous state-spaces. Nevertheless, further adjustments are necessary and will be presented in the subsequent sections.

5.2.1 Nearest Neighbor Search

While in discrete state-spaces a simple exact match query was enough to generate NN_q, this is not sufficient enough for continuous state-spaces any more. The state-space is infinite and thus much bigger than a limited discrete one. An exact match query would result in zero matches most of the time, as even a very small difference in one dimension of the state-space could exclude a potential neighbor. Sophisticated NN algorithms as introduced in section 2.6 are required.

As already mentioned in section 2.6, the presented algorithms are designed for fixed datasets and are therefore inflexible and not ready for constantly changing data. The solution that was used in this work was to rebuild the tree in a constant interval L. To ensure a behavior that is as close as possible to a naive approach that is a simple list, the whole IER buffer was split into the following parts:

1. A *list of real experiences* that holds only true experiences that have been generated by interaction with the environment. This list is used for minibatch queries.
2. A *tree-structure of real experiences* that is used for NN searches to interpolate synthetic experiences.
3. A *shrinking memory of synthetic experiences* that is a list.

To have access to all true samples at all times, and store true experiences that are not part of the tree structure yet, a list of the true experiences is maintained at all times.

The size of that list is exactly L bigger than the size of the tree. When the tree is rebuild, then the list holds all experiences that are in the tree plus L experiences that are not part of the tree yet. The new tree will be build out of all experiences from the list without the first L experiences. When a minibatch needs to be retrieved, then samples are drawn at random from the combined list of real and synthetic experiences.

An in-depth analysis of the rebuild functionality will discuss the benefits and disadvantages: First of all, rebuilding is costly and should be kept to a minimum. On the other hand, accurate interpolations require a buffer that is as actual as possible to have access to relevant sampling points. Thus, a compromise is required and the hyperparameter L needs to be tuned depending on the problem and the size of the buffer. Observed experiences in between two build operations are not part of the tree and can also not appear in a NN search during that time. However, experiences are stored in the tree-structure as soon as the rebuild happens and are (dependent of the time of occurring in relation to the last rebuild) longer/shorter part of the buffer as they would be in the naive case. In the best case, an experience is added to the buffer at the time it was observed because at that moment a rebuild was triggered and remains there L time steps longer as it would be part of the buffer in the naive case. Thus, in the best case, samples remain L times longer in the buffer. The worst case is when an experience is added to the buffer L times after it was observed, because the rebuild happened just before the observation and is excluded exactly at the time the naive approach would replace it. Thus, in the worst case, samples remain L times shorter in the buffer. On average experiences are included into the tree $\frac{L}{2}$ time steps later than the occurrence, and remain there $\frac{L}{2}$ time steps longer as in the naive case, which turns out to be the same duration on average delayed by $\approx \frac{L}{2}$ time steps.

It can be seen, that the time a sample can be used for NN searches is the same as with the naive approach (on average). However, this time is delayed and the choice of a huge L would result in interpolations with out-dated data. Such a behavior could possibly harm the learning process and needs to be avoided. The choice of L should therefore be small enough. On the other hand, a choice of L that is too small would cost a disproportional amount of resources and should therefore be avoided as well. As mentioned above, L is a hyperparameter that needs to be tuned individually.

The set of NN_q is searched in the tree-structure and all experiences that start (S_t) within a predefined radius nn_{thresh} are selected.

5.2.2 Query Functions

An evaluation of different query functions in [PSH21] revealed that the IER benefits from a sampling method that follows the distribution that is created by the policy. An off policy equivalent of this distribution is present in the real experience buffer as it was created by the policy. In discrete environments, it was enough to draw an experience (PD) or take the last observed one (LS) out of the real valued buffer and use the received state-action pair as query point. This is no longer reasonable for continuous environments, because the number of states is much bigger, and, the state-space should be explored instead of just interpolating points for which the exact corresponding reward and follow-up state is already known. The method needs to be adjusted to work again, and to achieve this, the query point is drawn within a radius r_{sq} around the state received from the drawn real example. To simplify this hyperparameter for multidimensional state spaces, it is defined as follows: $0 \leq r_{sq} \leq 1$. When the maximum value of each dimension is known, it is possible to determine a corresponding radius for each dimension:

$$r_{sq}^{d} = r_{sq} * S_{max}^{d} \, \forall \, d \in S_{dim}, \tag{5.3}$$

with S_{dim} being the dimensionality of the state space. If the state space is defined as infinite or not known for one or more dimensions the maximum discovered value for the appropriate dimension(s) can be tracked and used instead. To follow an on policy variation of the distribution (LS) the state-action pair that was recently created by the policy can be used instead of sampling from the buffer.

5.2.3 IER algorithm

To be able to ensure at least some accuracy of the conducted interpolations, a parameter min_{spoints} is introduced that specifies the minimum required amount of sampling points before an interpolation is performed. Algorithm 2 from chapter 4 can be adapted to work in continuous environments and is transformed to Algorithm 3. The presented algorithm represents the final version of IER and is ready to be used in continuous environments. IER for continuous environments provides an answer to **Q** 1 - **Q** 3.

5.3 Evaluation

The presented algorithm extends the proof-of-concept from chapter 4 to a version that is theoretically ready to work in continuous environments. However, it still remains the question if IER can be beneficial in terms of episodes that are required to solve a problem with continuous state-space. The following section will perform an evaluation that investigates if IER can reduce the required time to solution and consequently increase sample efficiency (**Q** 4).

5.3.1 MountainCar Environment

The MOUNTAINCAR environment from OpenAi Gym [Bro+16a] is a prominent benchmark problem that was chosen for evaluational purposes. In this problem an agent,

Algorithm 3 IER for continuous environments

```
1: Initialize $D, D^{\text{inter}}$
2: Initialize $nn_{\text{thresh}}, min_{\text{spoints}}$
3: Initialize $s_{\text{syn\_min}}, s_{\text{syn\_max}}, s_{\text{er\_max}}$
4: while Training do
5:     Store Experience $e$ in $D$
6:     if $|D| \geq c_{\text{start\_inter}}$ then
7:         Find $x_q = (S_q, A_q)$ based on Query Function $F$
8:         Get $NN_q = \{e_t | d(S_q, S_t) \leq nn_{thresh} \wedge A_t = A_q\}$ from $D$
9:         if Not $|NN_q| \geq min_{\text{spoints}}$ then
10:            Continue
11:        Interpolate $R_{\text{syn}}$ based on $NN_q$
12:        Interpolate $\delta_q$ based on $NN_q$
13:        Compute $S_{\text{syn}} = S_q + \delta_q$
14:        Interpolate $d_{\text{syn}}$ based on $NN_q$
15:        Generate $e_{\text{syn}} = (S_q, A_q, R_{\text{syn}}, S_{\text{syn}}, d_{\text{syn}})$
16:        Store $e_{\text{syn}}$ in $D^{\text{inter}}$
17:        $sp = \max[s_{\text{syn\_min}}, s_{\text{er\_max}} - |D|]$
18:        while $|D^{\text{inter}}| > \min[sp, s_{\text{syn\_max}}]$ do
19:            Remove $e$ from $D^{\text{inter}}$
20:            $sp = \max[s_{\text{syn\_min}}, s_{\text{er\_max}} - |D|]$
```

the mountain car, starts in between two hills and has to reach the top of the right hill. The action-space consists of three actions: $\mathcal{A} = \{\text{move-left}, \text{move-right}, \text{do nothing}\}$. As the car on its own is not powerful enough to drive up the hill straight away, it has to build up height on one side and use speed that is acquired by driving downwards to build up more height on the other side. By exploiting this effect the agent is able to finally reach the goal on the right hill. Figure 5.3 illustrates the environment.

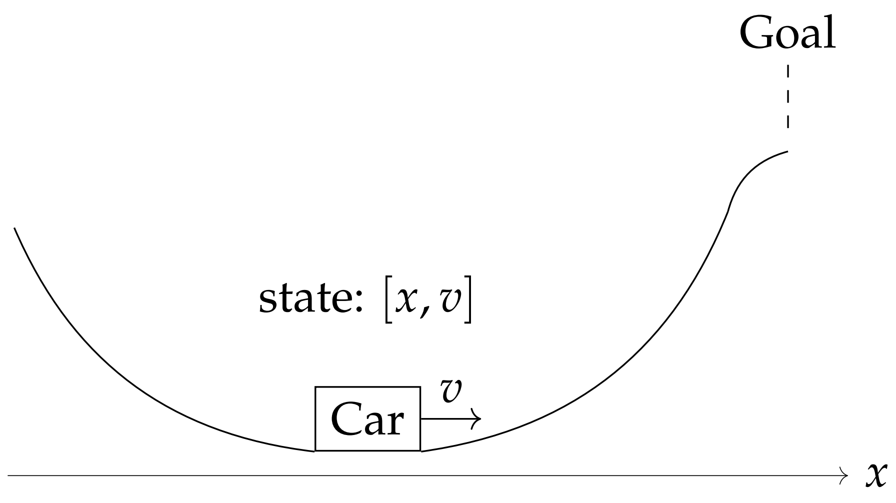


FIGURE 5.3: MOUNTAINCAR environment.

A state is defined as the position and the velocity at a time step. An episode is over if the agent reaches the goal or a maximum time limit of 200 steps is exceeded. An episode is considered as terminated (terminal tag) only if the goal was reached.

Every step is rewarded with a reward of -1, and the positive incentive for reaching the goal is the terminal tag which results in no estimation of the action-value of a follow-up state. As only negative rewards are received until the goal was reached once, the exploration in the beginning is crucial and bad or too less exploration might result into a high time to solution (or even never being able to solve the problem).

5.3.2 Experiments

Several different IER configurations have been evaluated that, to some extend, have also been part of the evaluation in section 4.3. Note that ONS is not possible for continuous states-spaces as the follow-up state is interpolated instead of being drawn from observed ones. The different configurations are presented in the following:

1. **Def:** This configuration is the same as in chapter 4. The synthetic buffer is filled as long as the real valued buffer is not full yet.
2. **Def-OnPol:** This configuration is the same as in Def with the addition that interpolations are performed for exactly one action that is retrieved by the actual policy.
3. **Def-zeta:** This configuration is the same as Def but ζ is used to prolong the time that interpolations are generated and added to the buffer.
4. **Del-zeta-OnPol:** This configuration is the combination of Def-zeta and Def-OnPol.
5. **Full:** This configuration is the same as in chapter 4. The synthetic buffer is filled to the same amount as the maximum size of the real valued buffer and this continues for the whole training.
6. **Full-OnPol:** This configuration is the same as Full with the addition that interpolations are performed for exactly one action that is retrieved by the actual policy.

5.3.3 Hyperparameters

The hyperparameters that are shared by IER and ER have been tuned by a hyperparameter search. Most hyperparameters are shared by IER and ER to ensure comparability and are presented in Table 5.1

As only the IER configurations make use of a synthetic buffer, $s_{\text{syn\_min}}$ is of no relevance for the baseline or rather set to zero. The same thing holds for the point in time when interpolation starts, the search radius for the query point, the interpolation threshold and the minimum required amount of sampling points. All of these hyperparameters are shared by all IER configurations. The chosen values can be observed in Table 4.3:

TABLE 5.1: An overview of shared hyperparameters of IER and ER for the MOUNTAINCAR task.

Parameter	**Values**
Learning rate α	0.00015
Gamma γ	0.99
Epsilon start	1
Epsilon min	0
Tau τ	0.4
Soft target update	True
Double DQN	True
Dueling Architecture	True
Real valued buffer size $s_{\text{er\_max}}$	50,000
Start learning at size of buffer	300
Minibatch size	32
Hidden Layer	[50, 200, 400]

TABLE 5.2: An overview of the hyperparameters that are shared by all IER configurations.

Parameters	**Values**
Synthetic buffer size $s_{\text{syn\_max}}$	50,000
Start interpolation at size of buffer	100
Query radius r_{sq}	0.05
Interpolation threshold nn_{thresh}	0.005
Minimum amount of sampling points min_{spoints}	2

All IER configurations use an IDW interpolation (cf. subsection 2.5.2) with a power parameter of $p_{\text{idw}} = 2$. Further more, when IER was used, a ball-tree (cf. subsection 2.6.2) managed true experiences to speed up NN searches. The tree was rebuilt every 2000 steps. The minimum size of the synthetic buffer $s_{\text{syn\_min}}$ is different for each configuration. All configurations that are combinations of them (*-OnPol) and that are not listed separately below share these values. The Zeta configurations use ζ while for all other configurations no ζ was used. Table 5.3 provides an overview of the used parameters:

TABLE 5.3: An overview of the different hyperparameters for the different IER configurations.

Configuration	$s_{\text{syn\_min}}$	ζ	**M**
Def	0	-	-
Def-zeta	20,000	1	1500
Full	50,000	-	-

5.3.4 Statistical Analysis

The statistical analysis of the results follows the structure that was presented in subsection 4.3.4. Runs have been repeated 40 times with different random seeds and were terminated after 4000 episodes.

5.3.5 Results

For the sake of space, only the best performing configurations for the four different exploration lengths are presented in this section. A complete overview of the results of the conducted experiments is provided in Appendix B.

The MAR for the MOUNTAINCAR problem is defined in OpenAI Gym as an average value of -110 over 100 consecutive episodes.

Table 5.4 shows the best performing IER configurations for the following explorations lengths $t_{\text{expl}} = \{250, 500, 750, 1000\}$. A very short exploration time of 250 episodes was very good for ER and IER was not able to outperform it here, as the probability that IER solves the problem faster than ER is only at around 23 % whereas the probability that ER solves the problem faster is at around 75 %. Even if there is at least some evidence towards IER outperforming ER in terms of TTS, it can be expected that IER will perform worse in most of the cases. Furthermore, Def-OnPol LS was the best performing configuration and all other IER configurations performed even worse. However, for increased exploration phases IER shows very strong evidences ($\geq$ 90 %) in favor of IER. This holds for all evaluated configurations. The LS query function is thereby performing better than PD in most of the cases and seems to be a better choice in general.

TABLE 5.4: Probabilities (in %, rounded to two decimal places) for the best performing configurations on the MOUNTAINCAR problem of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-OnPol	250	LS	75.54	1.04	23.42
Def-OnPol	500	LS	1.82	0.30	97.88
Def	750	LS	3.60	0.53	95.86
Def-zeta-OnPol	1000	PD	3.49	0.41	96.10

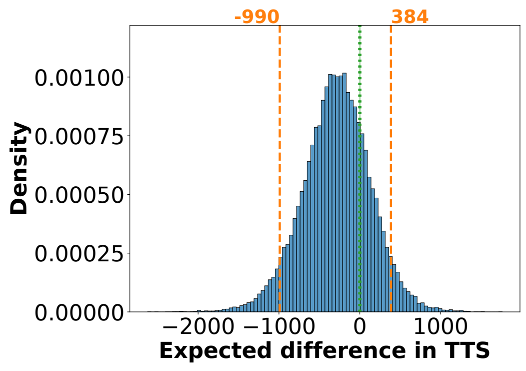
Table 5.5 shows the probability that the solution rate of IER is bigger than the solution rate of ER. Similar to the results from section 4.3, the configurations that outperform ER in terms of TTS have a high probability to have a higher solution rate in general.

The histograms in Figure 5.4 show the probability distribution of the difference of the means of the TTS of IER and ER and confirm the results from above. An exploration phase that is longer than 250 episodes favors IER and the majority of

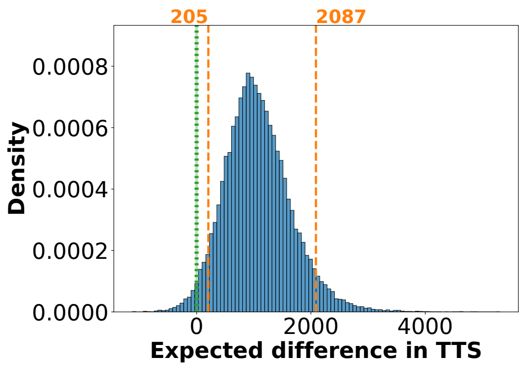
TABLE 5.5: Probability (in %) for the best performing IER configurations on the MOUNTAINCAR problem that the solution rate of IER is greater than the solution rate of ER.

Config	t_{expl}	**Query Function**	$p\,(p_{\text{IER}} > p_{\text{ER}})$
Def-OnPol	250	LS	28
Def-OnPol	500	LS	93
Def	750	LS	93
Def-zeta-OnPol	1000	PD	97

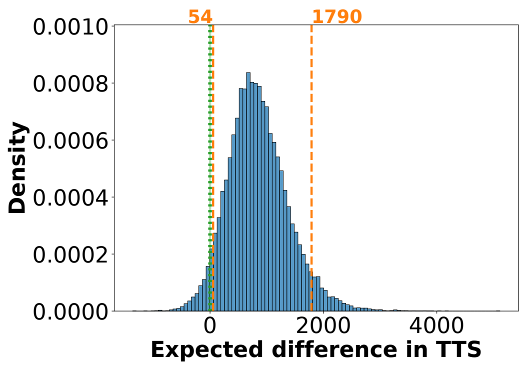
the probability mass is to the right of the green area that indicates the rope. The Def-OnPol LS configuration at $t_{\text{expl}} = 500$ for example can be expected to solve the MOUNTAINCAR problem between 205 and 2087 episodes faster in 95 % of the times. A short exploration phase of $t_{\text{expl}} = 250$ favors ER as the majority of the probability mass is to the left of the rope.



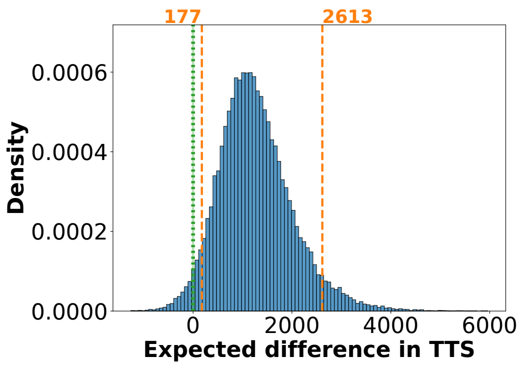
(A) 250



(B) 500



(C) 750



(D) 1000

FIGURE 5.4: Density plots of the posterior distribution of $\mu_{\text{DQN}} - \mu_{\text{FU-DQN}}$ for best performing IER configurations on the MOUNTAINCAR problem. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent.

An overview of the most likely TTSs of the best performing IER configurations is given in Table 5.6 and shows that ER performs best in the $t_{\text{expl}} = 250$ case while

IER is able to outperform in the other cases. Even if ER is able to solve the MOUNTAINCAR problem very well with an exploration phase of 250, it can be seen that IER performs good when given enough time to generate synthetic experiences.

TABLE 5.6: Most likely values and uncertainties for the mean TTS of IER and ER for the best performing configurations on the MOUNTAINCAR problem given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}.

Config	t_{expl}	**IER mean TTS**		**ER mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Def-OnPol	250	1909	[1551, 2596]	1688	[1350, 2196]
Def-OnPol	500	2240	[1852, 2783]	3142	[2642, 4248]
Def	750	2213	[1961, 2792]	2941	[2532, 4041]
Def-zeta-OnPol	1000	2908	[2507, 3664]	3799	[3331, 5465]

5.4 Discussion

IER for continuous environments provides an answer to the question of how gathered knowledge in form of stored experiences can be exploited to generate meaningful synthetic experiences (**Q** 1). Furthermore, it was shown that all parts of an experience can be interpolated, even if the generation of the synthetic follow-up state requires the interpolation of the state-transition-delta to reduce expected interpolation errors and overcome the problem of potential occurring extrapolation cases (**Q** 2). It was also shown how synthetic experiences can be used to assist a DQN with learning (**Q** 3).

Q 4 raises the question if synthetic experiences can be used in combination with a DQN to speed up convergence and increase sample efficiency. The evaluation showed that a baseline with original ER is able to solve the MOUNTAINCAR problem better than IER in general, but when exploration is increased IER started to outperform ER and was able to solve the problem faster and more consistently. Thus, it can be expected that IER requires a sufficient amount of time to generate experiences that are good enough that they can help with exploration. Furthermore, at least in the investigated problem, it seems that the LS query function performs better than PD which might be reasoned by the fact that LS stays closer to the distribution that is generated by the policy. The best performing configuration was Def-*, which is the one that generates lesser synthetic experiences as the Full configuration and it seems to be better when the amount of them is only high in early phases when exploration is high.

Overall, it can be said, that IER is able to assist a DQN and reduce TTS when exploration is high and the agent has a hard time to explore the problem-space. In such cases, IER can increase sample efficiency as not only the problem is solved in less episodes, but also true experiences are used to generate synthetic ones by means of interpolation. However, interpolation is costly and thus, IER seems like a good option in scenarios where a lot of exploration is required but on the other hand is expensive (like in real-world scenarios).

Chapter 6

Full-Update Deep Q-Network

The former sections introduced how synthetic experiences can be generated by means of interpolation based on stored real experiences as sampling points. Those interpolated experiences have then been included into the replay buffer while the training functionality of the DQN has not been changed. However, adding synthetic samples to the buffer is not the only way that they can be used to assist a DQN. In this section, the *Full-Update DQN* (FU-DQN) is presented that generates synthetic experiences to do training updates that cover the whole action space. An evaluation on several different environments with a sparse reward setting shows that FU-DQN is able to solve these problems more consistently and even faster than the original approach.

Parts of the following text have already been published in a similar form in [Pil+23].

6.1 Full-Update DQN

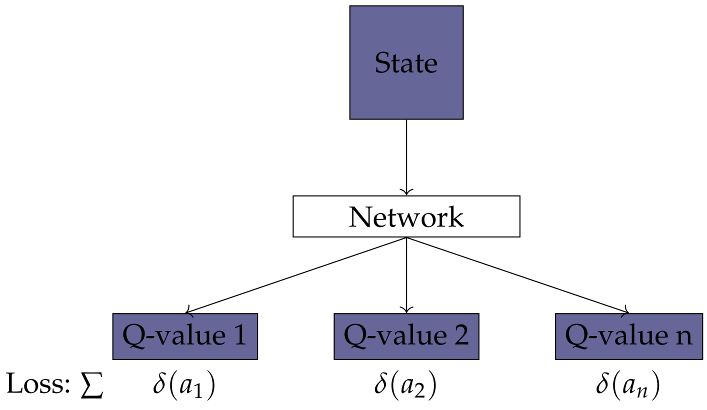
As mentioned above, IER is not the only possibility to make use of synthetic experiences. FU-DQN uses them to perform updates that do take the whole action-space into account.

6.1.1 The Unused Potential of a DQN Update

A DQN model is built such that it accepts a state as input and predicts Q-values for every action in the action-space. This design was chosen, because the agent needs access to all Q-values of a current state in order to pick the action that maximizes the expected return which is estimated by the Q-value (cf. section 2.4). An alternative design would accept a state-action pair as input and predict the single corresponding Q-value. To find the best action, a feed-forward operation would then be necessary for every possible action, which increases the required computational effort. It is possible to reduce this amount to one when the first mentioned design is used. In each training step, experiences are drawn from the ER buffer and the TD error is computed for each of them to be used as a loss for updating the network. Since DQNs have an output node for every action in the action-space $\mathcal{A}$, the loss for every action could be calculated given the TD error of that action. However, as can be seen in Equation 2.20, to calculate the TD error for an action, the immediate reward R_t as well as the follow-up state S_{t+1} are required. These values are only known for the action that has been actively chosen for the corresponding experience and, therefore, the losses for the unknown actions are treated by setting them to zero. By doing so, the back-propagated loss can be attributed clearly to the chosen action.

In Figure 6.1 it can be observed, that the loss for one sample out of a minibatch is computed by the sum of the TD errors from all output nodes.

FIGURE 6.1: Graphical illustration of a DQN loss.



This can be concretised like shown in Figure 6.2. For each subloss, the corresponding TD error needs to be calculated. The problem is that only one experience is at hand and this experience inherits the reward and the follow-up state for exactly one action that was executed at the time the experience was generated. Let this action be a_2 and thus, as already explained above, the TD errors for a_1 and a_2 are set to zero.

FIGURE 6.2: Intuition of a DQN update.

$$\begin{tikzcd}
 & Q(s,a_1) \arrow[dl, bend right] & -r_{a1} + \gamma * \max_a Q(s'_{a1}, a; \theta) \\
L(s) & Q(s,a_2) \arrow[l] & -r_{a2} + \gamma * \max_a Q(s'_{a2}, a; \theta) \\
 & Q(s,a_3) \arrow[ul, bend left] & -r_{a3} + \gamma * \max_a Q(s'_{a3}, a; \theta)
\end{tikzcd}$$

This mechanism makes perfectly sense as the required values for all actions that are not part of the given experience are unknown and not achievable at the time of the update. But in general, the structure of a DQN is such that an update for all actions would be possible if these values would be known. Even estimates of these values would allow such an update, of course the potential errors need to be addressed somehow. With such estimations at hand, it would be possible to perform an update that takes into account *all the possible actions*, and therefore, would give the learner a broader understanding of the environment dynamics in a single update operation. Depending on how the data needed additionally is obtained, the sample efficiency as well as the amount of required training operations could be improved.

6.1.2 Updates for the Whole Action-Space

An algorithm that is able to perform DQN updates that take into account the whole action-space is FU-DQN. This extension of DQN is able to utilize already stored data in the replay buffer to generate the required data for the missing action-losses by employing a *Generation Function*. First, the loss for a minibatch (a multiset) of experiences $\{(s, a, r, s')\} = \{e\} = B$ at iteration i is defined as

$$L_B(\theta_i) = \frac{1}{|B|} \sum_{e \in B} \delta_{e,i}^2 \tag{6.1}$$

with

$$\delta_{e,i} = \left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right), \tag{6.2}$$

where, like in the original work on DQNs, θ_i are the parameters of the Q-function approximation at iteration i and θ_i^- are the parameters of the target network (cf. section 2.4). Note that if B_i is chosen as the batch drawn uniformly from the ER buffer, $e \sim \mathcal{U}(D) \forall e \in B_i$, in iteration i, the original loss function is retrieved [Mni+15]:

$$L_{B_i}(\theta_i) = \frac{1}{|B_i|} \sum_{e \in B_i} \delta_{e,i}^2 \approx \mathbb{E}_{e \sim \mathcal{U}(D)} \left[\delta_{e,i}^2 \right] \tag{6.3}$$

Next, synthetic batches are introduced: A *synthetic batch* $\widetilde{B}$ *for a batch* B *based on the generation function* I is a multiset

$$\widetilde{B}_I = \{(s, a', r_*, s'_*) \mid (s, a, r, s') \in B, a' \neq a, (s, a', r_*, s'_*) \leftarrow I(D; s, a')\} \tag{6.4}$$

where the generation function has been used to generate synthetic experiences for the state-action pairs (s, a') based on the current ER buffer D. Such a synthetic batch inherits $|\mathcal{A}| - 1$ synthetic experiences for every real experience in B_i. These synthetic experiences start in the same state s as the corresponding real experience does but consider all actions $a' \in \mathcal{A} \setminus \{a\}$ that differ from the action that is part of the corresponding real experience. As mentioned before, the synthetic rewards r_* and follow-up states s'_* are generated by I. $\mathcal{B}_{iI}$ is further defined as the union of the real batch B_i and its synthetic batch based on the generation function I, $\mathcal{B}_{iI} = B_i \cup \widetilde{B}_{iI}$. This leads to the following loss for batch B_i at iteration i:

$$\mathcal{L}_{B_i}(\theta_i) = L_{\mathcal{B}_{iI}}(\theta_i) \tag{6.5}$$

6.1.3 Generation Function I

Note that I may use interpolation but that it is in general not limited to do so. It may be any method that is able to generate a prediction of the reward and the follow-up state based on stored (i. e. observed) experiences and given a state-action-pair. Thereby, it is not mandatory that only data is used that is actually present in the ER buffer but model-based approaches may be used as well. Other conceivable methods are tied to the IER approaches. As introduced above, they use the stored samples in the ER buffer as sampling points for interpolations, which is also the approach utilized for the evaluations. To be more specific, the generation function $I(D, s, a)$ is defined for the purpose of this work as follows: Given a state s and an action a, all neighbours that are located within a predefined distance around s and executed action a are searched in the replay buffer D. Those nearest neighbours are considered as sampling points for the interpolation of the synthetic reward r_* and the follow-up state s'_*. Similar to IER, IDW interpolation is used that weights sampling points that are closer to the query point stronger than points that are further away. The output of $I(D, s, a)$ is therefore a synthetic experience $e_* = (s, a, r_*, s'_*)$, which is the same as $e_{\text{syn}} = (S_q, A_q, R_{\text{syn}}, S_{\text{syn}})$. In the edge case where the number of nearest neighbours is 0, no meaningful interpolation is possible and the specific subloss for that state-action-pair is set to zero such that no update is performed for the corresponding action. The same is done if the number of nearest neighbours is 1 and that sampling point does not perfectly match the query point. Figure 6.3 shows how the generation function I is used to generate missing values of the TD errors of the actions that have not been part of the given experience.

FIGURE 6.3: A graphical intuition of how the generation function I is used to generate missing values for TD errors fo actions that have not been part of the given experience.

$$\begin{tikzcd}
 & \delta_{a1} \arrow[dl, bend right] & I(D; s, a_1) \arrow[l] \\
L(s) & \delta_{a2} \arrow[l, "a_2 == A_t"'] & \\
 & \delta_{a3} \arrow[ul, bend left] & I(D; s, a_3) \arrow[l]
\end{tikzcd}$$

6.1.4 Handle Interpolation Noise

The generated data has to be expected to be noisy and, possibly, biased without being able to determine the magnitude of either error. This is due to the fact that the quality of an interpolation depends on the sampling points and the ability of the reward-/state-transition-function to be interpolable. However, as mentioned above, I is not forced to be an interpolation with other conceivable in-place options being models of the environment dynamics, which, depending on the type of model and its goodness-of-fit (e.g. untrained vs. well trained vs. given pre training) might produce untrustworthy estimates as well. To counteract this effect, weights are introduced into (6.1) which yields the following updated definition:

$$L_B(\theta_i) = \frac{1}{|B|} \sum_{e \in B} w_e \delta_{e,i}^2 \tag{6.1a}$$

For any real experience e, the corresponding weight should be $w_e = 1$ which corresponds to not changing their contribution to the loss function. For synthetic experiences, on the other hand, w_e should probably be chosen such that $w_e \leq 1$ since the goal is to compensate for errors in data generation. Finding adequate values for the w_e represents an intricate problem not solvable at runtime. In the experiments the w_e are treated as hyperparameters and set

$$w_e = \begin{cases} 1, & e \text{ is a real experience} \\ w, & e \text{ is a synthetic experience,} \end{cases} \tag{6.6}$$

that is, all the synthetic experiences are discounted by the same weight.

6.1.5 Expected Benefits

The proposed update mechanism considers the whole action space for every state that has been part of a sampled experience. If the generated experiences would be free from noise, this could be compared to performing the amount of $|\mathcal{A}|$ DQN updates, one for every action in $\mathcal{A}$. As discussed above, noise has to be expected in the generated experiences and, therefore, this comparison should not hold true. However, as FU-DQN tries to compensate the noise by using weights and the generations are trustworthy at least to some degree, the true increased efficiency can be expected to be higher than the efficiency of only updating a single action but also below the efficiency of updating all actions at once.

Another benefit is the increased exploration that comes with an update that considers the entire action space. While DQN would require the same states with different actions to be sampled again, FU-DQN updates them in one flush. Furthermore, since synthetic experiences are weighted less, this exploration's impact on the update is not overwhelming (depending on w), and the main focus of the training remains on the trajectories that are generated by following the policy. Both mentioned points are expected to improve the performance of the learner as less experiences are required to have the same training effect that DQN achieves with more samples. Thus, less iterations are expected to be required to solve a problem, which also indicates an increased sample efficiency.

In terms of research questions provides FU-DQN a different way than IER of how synthetic experiences can be used to assist a DQN with learning (**Q** 3). Instead of adding such experiences to the buffer and draw them at random, FU-DQN generates exactly these synthetic experiences that are required at the moment of the update. A drawback is of course the fact that once generated synthetic experiences are not stored and thrown away after the update. As interpolation can be quite costly, this is not ideal. If FU-DQN can help speed up convergence and reduce the required time to solution and consequently increase sample efficiency (**Q** 4) can be answered only after the evaluation.

6.2 Evaluation

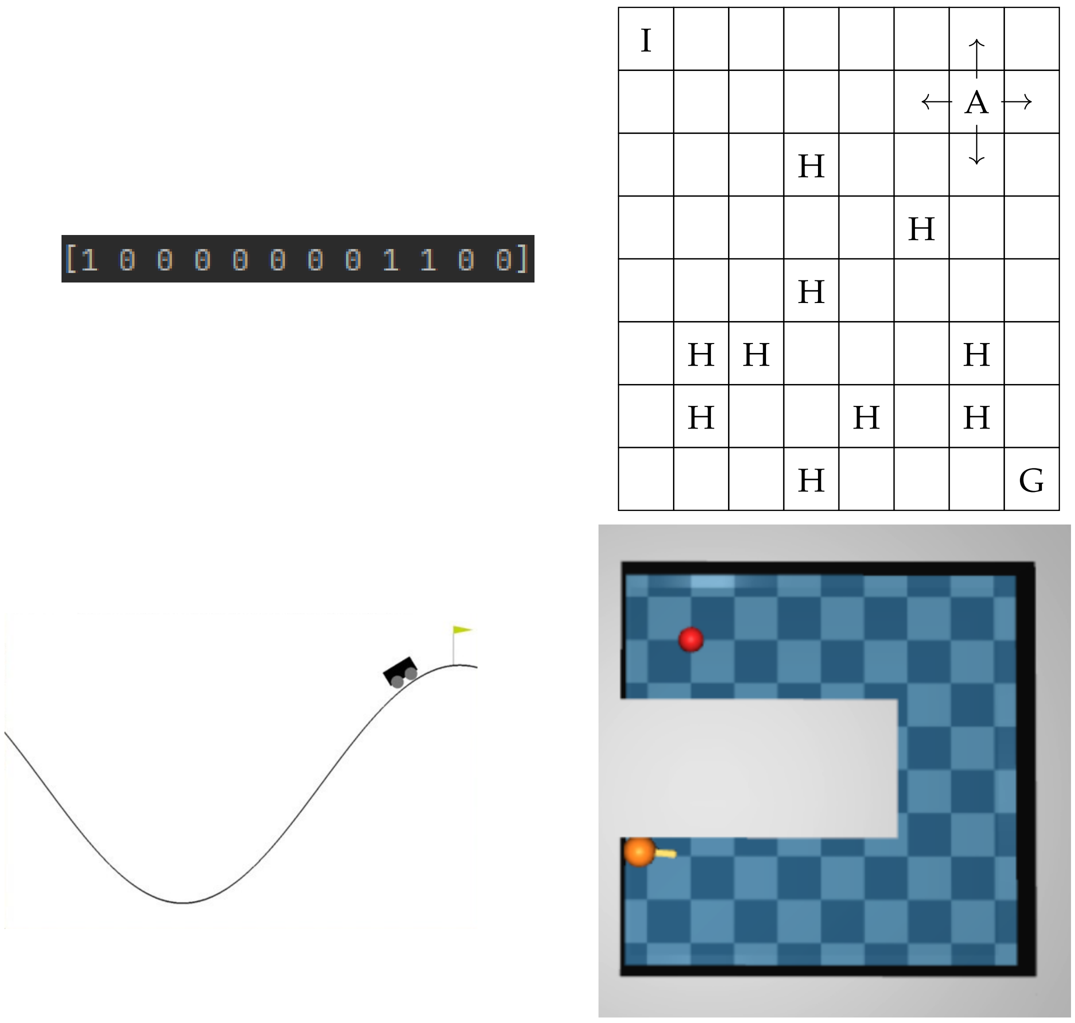
In the former sections, the idea behind FU-DQN and its functionality have been introduced. Even if the approach of updating the whole action space in one update operation is expected to help with exploration and therefore should reduce the time to solution, this can only be stated with high confidence after an in-depth evaluation. To do so, several environments that share a sparse reward function and differ in characteristics and difficulty are evaluated against an original DQN baseline (in the following referred to as *DQN*).

6.2.1 Experiments

For evaluation purposes four environments that have different characteristics and address different problem specifications have been identified. However, they all have in common a sparse reward signal that is focused on a (large) positive reward for the terminal state. The four environments are:

1. CHAIN-N: discrete, deterministic
2. FROZENLAKE: discrete, non-deterministic, three different rewards
3. MOUNTAINCAR: continuous, deterministic, three actions
4. UMAZE: continuous, deterministic, 8 actions

The first problem is mentioned by Andrychowicz et al. [And+17] as a motivating example for their Hindsight Experience Replay technique. The general environment is a simple bit-flipping problem and is described by an array of the size N where every entry can be either one or zero. Therefore, the state-space is defined as $\mathcal{S} = \{0, 1\}^N$ and the action-space as $\mathcal{A} = \{0, 1, \ldots, N-1\}$ where the i-th action flips the i-th bit. While Andrychowicz et al. investigated the area of multi-objective RL and therefore draw a new goal for every episode, a target state is only drawn once, in the beginning and kept for the rest of the run. A reward of -1 is received until the state equals the goal and a reward of 1 is retuned. This problem is used as



```
[1 0 0 0 0 0 0 0 1 1 0 0]
```

FIGURE 6.4: A graphical intuition of: (1) CHAIN-N, (2) FROZENLAKE 8x8, (3) MOUNTAINCAR and (4) UMAZE.

a very basic discrete and deterministic problem that can easily be increased in difficulty by increasing the size N. As mentioned by Andrychowicz et al. a DQN is only reasonably able to solve this kind of problem up to a size of $N = 13$.

The FROZENLAKE 8x8 environment from the OpenAI Gym [Bro+16b] is a non-deterministic grid world and has already be introduced in subsection 4.3.1. The final state returns a reward of 5, falling into a hole a reward of -1 and any other state a reward of 0. The difficulty of this problem comes from its non-deterministic nature.

The first two environments focus on discrete state-spaces. A continuous one is found in the MOUNTAINCAR environment from the OpenAI Gym [Bro+16b]. This problem has already been introduced in subsection 5.3.1 In comparison to the former problems MOUNTAINCAR has a continuous state-space but also never returns a positive reward.

The last problem is the *PointUMaze-v1* environment from the Mujoco-Maze repository [Kan21]. In this problem, a ball has to manoeuvre from a start position to a goal in a U-shaped world. Originally, the action-space is continuous, but because common DQNs are not able to handle such spaces, it was discretized to 8 actions. The speed the ball is moving with has been fixed to a constant value and different angles that it can change its direction with are used as actions. A state is originally encoded in six different parts. The first two are coordinates that describe the position of the ball, the next one describes the actual direction that the ball faces and is encoded in pi. The last three describe the angular velocity. The direction that the ball faces poses a problem for interpolation and NN searches, as a value of π is the same as the value of $-\pi$ in this case and a value of π is close to a value of $-\pi - \epsilon$ whereas ϵ is a small number. These values are very far away form each other when treated as spatial distances, while they are in fact near to each other. To enable NN searches and interpolations for that part of the state, it is converted to its absolute value and a sevenths value is added to the state which is a boolean that indicates if π is positive or negative. A reward of -0.0001 is received in each step in which the goal has not been reached yet; entering the goal yields a reward of 1. In comparison with the MOUNTAINCAR problem, which is continuous as well, both the state- and the actions-space of this problem are larger.

A graphical intuition of the four learning tasks is given in Figure 6.4. All experiments used an ϵ-greedy exploration scheme and a DQN. For each particular environment, the same set of hyperparameters is used for both FU-DQN and DQN which have been tuned individually. For each learning task, 20 FU-DQN and 20 DQN agents are initialized (each seeded with a different random seed), each of them placed in an independent copy of the environment and statistics are collected of their respective performance (rewards received).

6.2.2 Hyperparameters

The hyperparameters that are shared by FU-DQN and DQN have been tuned by a hyperparameter search for each problem and are presented in Table 6.1 and Table 6.2.

FU-DQN specific hyperparameters are the distance threshold that was used for the NN search as well as the weights that have been used to counteract interpolation noise. Those hyperparameters are presented in Table 6.3. The reason why CHAIN-N and MOUNTAINCAR have a NN distance threshold of zero is because an exact match query was used instead of a true interpolation. This method is more like a local model of the environment as experiences are searched in the buffer that match the exact situation and action. Because the underlying environments are discrete, this approach is possible. Furthermore, by using this technique, interpolation errors are not possible and the generated values can be trusted to 100 %, which is why the value for w_e is set to 1.

TABLE 6.1: Shared hyperparameters of the CHAIN-N and FROZENLAKE task.

Parameter	CHAIN-N	FROZENLAKE
Learning rate α	0.0002	0.0005
Gamma γ	0.95	0.95
Epsilon start	1	1
Epsilon min	0	0
Tau τ	0.03	0.25
Soft target update	True	True
Double DQN	True	True
Dueling Architecture	True	True
ER buffer size	5,000	100,000
Start learning at size of buffer	100	200
Minibatch size	32	32
Hidden Layer	[256]	[128, 256]
Exploration length t_{expl}	250	250

TABLE 6.2: Shared hyperparameters of the MOUNTAINCAR and UMAZE task.

Parameter	MOUNTAINCAR	UMAZE
Learning rate α	0.00015	0.0001
Gamma γ	0.99	0.95
Epsilon start	1	1
Epsilon min	0	0
Tau τ	5	0.15
Soft target update	False	True
Double DQN	True	True
Dueling Architecture	True	True
ER buffer size	50,000	200,000
Start learning at size of buffer	2,000	2,000
Minibatch size	32	32
Hidden Layer	[50, 200, 400]	[50, 200, 400]
Exploration length t_{expl}	500	200

TABLE 6.3: Specific hyperparameters of FU-DQN for each specific task.

Task	nn_{thresh}	w_e
CHAIN-N	0	1
FROZENLAKE	0	1
MOUNTAINCAR	0.005	0.05
UMAZE	0.1	0.1

6.2.3 Statistical Analysis

The statistical analysis of the results follows the structure that was presented in subsection 4.3.4. There are some differences in the parameters nevertheless. Runs were repeated for 20 times instead of 40, this is reasoned by time limitations. Also experiments were only run for 3000 episodes before they were terminated, which results in a value of $u_\mu = 6000$.

6.2.4 Results

The four different problems are evaluated seperately in the following subchapters.

Analysis of the CHAIN-N tasks

For the CHAIN-N problems three different complexities have been investigated, that are, $N \in \{12, 20, 30\}$. While the CHAIN-12 problem should be solvable even for the DQN approach, the state space of the latter two is rather large. As the FU-DQN makes use of stored experiences, the hypothesis arose that the performance could be improved with a single perfect episode in the buffer. Therefore, all experiments are also run with the first episode having been guided directly towards the goal state, in the following denoted as CHAIN-N *w/ hints*. If an episode is able to find the target state within N steps a final reward of 1 is reported and 0 otherwise. This task's MAR is set to 1.0 for all tested values of N (i. e. the problem is considered as solved as soon as the agent is able to find the goal state in 100 consecutive episodes). As already mentioned above, for the generation function I a perfect match search in the ER buffer was used and since perfect matches can be trusted, a weight of $w = 1$ was used.

First, the solution rates are compared using the beta-binomial model as described in Table 6.4 and the results are reported in Table 6.5. The values can be interpreted as follows: For CHAIN-12, the collected data tells that there is a 82 % probability of the solution rate of FU-DQN being larger than the one of DQN. Overall, Table 6.4 shows that there is a definite tendency towards the solution rate of FU-DQN exceeding the one of DQN on the CHAIN-N tasks, unless the problem is easy enough that both approaches are able to solve it within 3000 episodes (which is the case for CHAIN-12 w/ hints) or so difficult that neither of them can (CHAIN-30).

Let's now turn the attention to the difference in the TTS. To do so, the gamma-distribution-based model from Section 4.3.4 is used. The probabilities obtained from that model when factoring in the rope defined in Section 4.3.4 are given in Table 6.5. It can be seen that, for CHAIN-12, the probability that FU-DQN solves the tasks faster than (at least as fast as) DQN is 94.12 % (94.94 %). CHAIN-20 and CHAIN-30 were not solved in time by any of the runs which is why this analysis cannot be performed for these tasks. For CHAIN-12 and CHAIN-20 w/ hints, very strong evidence (over 99 %) can be seen in favor of FU-DQN. CHAIN-30 w/ hints was only solved in time by very few runs (2 for DQN and 5 for FU-DQN, corresponding to 90 % and 75 % of data being censored). Even if, for both approaches, the problem seems to be too difficult to be solved within time consistently, there is some slight evidence (63.86 %) towards FU-DQN performing at least as well as DQN.

TABLE 6.4: Probability (in %) for each task that the solution rate of FU-DQN is greater than the solution rate of DQN. A * indicates that for at least one of the methods, not a single run was able to solve the problem in time.

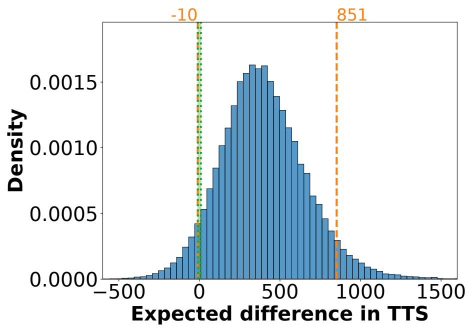
Experiment	$p\,(p_{\text{FU-DQN}} > p_{\text{DQN}})$
CHAIN-12	82
CHAIN-20*	76
CHAIN-30*	50
CHAIN-12 w/ hints	50
CHAIN-20 w/ hints	88
CHAIN-30 w/ hints	87
FROZENLAKE	76
MOUNTAINCAR	77
UMAZE	50

TABLE 6.5: Probability (in %, rounded to two decimal places) of FU-DQN performing better or worse in terms of TTS than DQN—or practically equivalent (with respect to a rope). Only takes into account runs that finished within 3000 episodes. A * indicates that the test was not possible, because, for at least one of the two methods, not a single one of the runs was able to solve the problem in time.

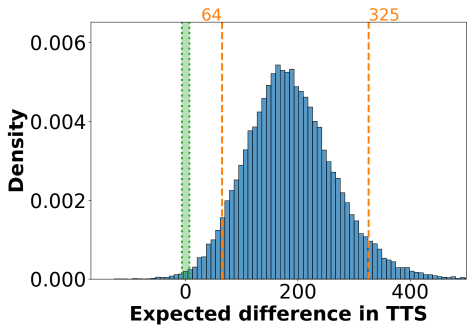
Experiment	DQN better	pract. equiv.	FU-DQN better
CHAIN-12	5.06	0.82	94.12
CHAIN-20	$*$	$*$	$*$
CHAIN-30	$*$	$*$	$*$
CHAIN-12 w/ hints	0.4	0.26	99.34
CHAIN-20 w/ hints	0.01	0.02	99.97
CHAIN-30 w/ hints	36.15	2.05	61.81
FROZENLAKE	0.0	0.01	99.99
MOUNTAINCAR	3.24	0.50	96.26
UMAZE	16.1	4.46	79.44

Figure 6.5 displays the distributions of the difference of the means of the TTS distributions for the three main CHAIN-N tasks that were comparable using the TTS model. The distribution for CHAIN-30 w/ hints is not plotted, because the amount of censored data is above 75 %. This visualizes the results from Table 6.5: Only for the CHAIN-12 problem there is considerable probability mass to the left of 0 indicating that DQN may, with a rather low probability, be better than FU-DQN. Figure 6.5 also shows the central 95 % HPDI which indicates, that for CHAIN-12 one can be very confident that the true value of the mean difference lies between -10 and 851 episodes.

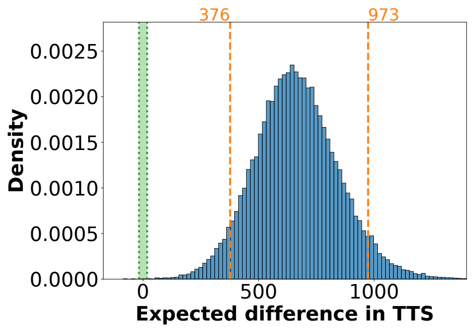
Table 6.6 gives summary statistics of the distributions over the TTS means (the



(A) CHAIN-12



(B) CHAIN-12 w/ hints



(C) CHAIN-20 w/ hints

FIGURE 6.5: Density plots of the posterior distribution of $\mu_{\text{DQN}} - \mu_{\text{FU-DQN}}$ for the three CHAIN-N problems that the gamma-distribution-based model is applicable to. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope.

distributions of μ_i for $i \in \{\text{DQN}, \text{FU-DQN}\}$). The most likely value (mode of the distribution over mean values μ_i) is reported as well as a measure of uncertainty (once more, 95 % HPDIs). It can be observed that the TTSs of FU-DQN are smaller than the TTSs of DQN. Furthermore, the differences of the mean TTSs align with the distributions in Figure 6.5.

Analysis of the results for the FROZENLAKE task

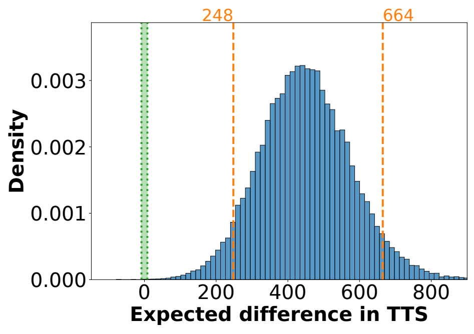
For the FROZENLAKE task, similar to the CHAIN-N task, successful runs are reported with a final reward of 1. The MAR is set to 0.8 which corresponds to a success rate of 80 %. Because of the non-deterministic nature of the problem it is possible to fall into a hole despite of using a good policy. For the generation function I, the linear reward interpolation (cf. chapter 4) is used, with the additional restriction that exactly one interpolation is required and thus, one follow-up state from the distribution of stored possible follow-up states is drawn (cf. ONS). I is trusted (discrete environment) and a weight of $w = 1$ is used.

Table 6.4 shows that, with respect to solution rate, FU-DQN is favoured with a probability of 76 % of performing better. Further on, Table 6.5 indicates that FU-DQN

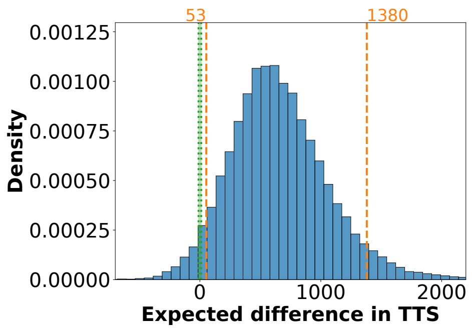
TABLE 6.6: Most likely values and uncertainty for the mean TTS of DQN and FU-DQN given as the modes and 95 % HPDIs of the estimated distributions of μ_{DQN} and $\mu_{\text{FU-DQN}}$. A * indicates that the statistical model was not computed, because, for at least one of the two methods, not a single one of the runs was able to solve the problem in time.

Experiment	DQN's mean TTS		FU-DQN's mean TTS	
	Mode	95 % HPDI	Mode	95 % HPDI
CHAIN-12	1316	[1087, 1816]	964	[815, 1252]
CHAIN-20	*	*	*	*
CHAIN-30	*	*	*	*
CHAIN-12 w/ h/	651	[551, 793]	477	[432, 525]
CHAIN-20 w/ h/	1902	[1706, 2213]	1270	[1129, 1444]
CHAIN-30 w/ h/	3871	[3462, 6384]	3616	[3190, 5642]
FROZENLAKE	989	[854, 1214]	577	[475, 686]
MOUNTAINCAR	1893	[1495, 2661]	1243	[1073, 1661]
UMAZE	1892	[1672, 2161]	1683	[1447, 1994]

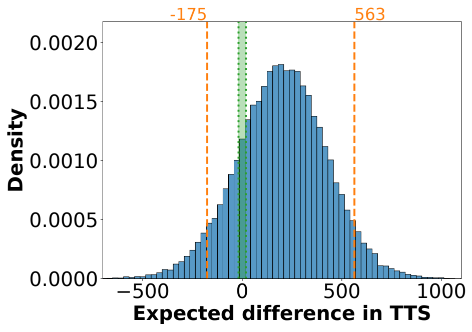
can be expected to solve the FROZENLAKE task faster than DQN with a confidence of almost 100 %. These results are backed by Figure 6.6a.



(A) FROZENLAKE



(B) MOUNTAINCAR



(C) UMAZE

FIGURE 6.6: Density plots like Figure 6.5 but for the FROZENLAKE, MOUNTAINCAR and UMAZE experiments.

Table 6.6 shows that the most likely mean TTS of FU-DQN is smaller than the TTS of DQN, while the difference in TTS aligns with the distributions in Figure 6.6.

This can be observed for all of the tasks that are remaining to be discussed.

Analysis of the results for the MOUNTAINCAR task

For the MOUNTAINCAR experiment, the MAR was used as defined by the OpenAI Gym [Bro+16b] which is -110. As generation function I an IDW interpolation was used. The sampling points are gathered by a nearest-neighbour search in a radius around the query state of $r_q = 0.005$. A weight of $w = 0.05$ was used.

First, the solution rate is compared. Here again, FU-DQN is favoured with 77 % as can be observed in Table 6.4. Regarding the TTS, Table 6.5 shows that there is a high probability of around 96 % of FU-DQN solving the task faster than DQN. When looking at Figure 6.6b, it can be seen that while the central 95 % HDPI is made up of all positive values, it is also considerably wider than for the other tasks indicating a higher uncertainty.

Analysis of the results for the UMAZE task

For the UMAZE task, a similar reward metric as for CHAIN-N and FROZENLAKE was used. A fixed value of 0.1 for the speed parameter increases the state-space and difficulty. Because of that, the value for the MAR was set to 0.9 which equals a solution rate of 90 %. The weight value was set to $w = 0.1$. For the generation function I an IDW interpolation with $r_q = 0.1$ was used.

Table 6.4 shows that the solution rate model reports the solution rates of the approaches to be equal. In contrast to some other experiments above, this is not due to all or none of the runs solving the task, but from an equal count of successful runs. According to the TTS model, the probability that FU-DQN solves the problem faster than DQN is 79.44 % while the probability of solving it at least as fast is 83.9 %. Figure 6.6c confirms this statement visually with the majority of the probability mass lying to the right of the rope.

6.3 Discussion

In summary, it can be reported that *there is some evidence that FU-DQN's solution rate exceeds the one of DQN* (E-Q1): All the values in Table 6.4 are at least 50 % (which corresponds to the two solution rates being the same) and if only considering the tasks that *both* methods were able to solve in time at least once, the values are actually at least 76 % with the exception of the UMAZE task (50 %).

Furthermore (E-Q2), the analysis shows that there are considerably high amounts of evidence (>94 %) for the statement *that the TTS of FU-DQN is shorter than that of DQN on most of the tasks considered* with the only exceptions being the CHAIN-30 w/ hints and the UMAZE tasks. The statistical analysis concludes that there is only some evidence (62 %) towards FU-DQN being faster for the former whereas there are considerable but not high amounts of evidence (slightly less than 80 %) for FU-DQN outperforming DQN on UMAZE. It is assumed that this is due to those problems being the ones with the largest state-spaces—and thus the highest difficulty. When looking at the mean TTSs of all tasks (E-Q3), then it can be observed that FU-DQN was able to solve the problems in less iterations than DQN. This encourages the statements from above

As FU-DQN uses experiences to generate synthetic ones in addition to ER, and it solves the investigated problems in less iterations than DQN, it can be concluded that it is able to increase sample efficiency in a beneficial way. As mentioned above, an update of FU-DQN updates the action-value function for all actions whereas the same effect would require a single update of DQN for every action. However, the factor of less required iterations for FU-DQN over DQN is smaller than this. There are several reasons for this, the most obvious one being that non-experience action-updates are weighted less to counteract noise (cf. subsection 6.1.5). Overall, the analysis shows that FU-DQN is able to outperform DQN at least on the tasks that DQN is able to solve consistently and possibly on the other tasks as well.

When considering the research questions, then FU-DQN presents a way of using synthetic experiences in combination with a DQN, such that faster convergence can be achieved and consequently the sample efficiency can be increased (**Q** 4). Furthermore, the evaluation results show once more that the generation of synthetic experiences by means of interpolation based on stored experiences is a beneficial way of how to exploit knowledge in form of stored experiences (**Q** 1). Also it was shown that all parts of an experience can be interpolated, with the addition that a synthetic follow-up state should be generated by an interpolated state-transition (**Q** 2). And, as mentioned before, FU-DQN represents an alternative approach, next to IER, of how synthetic experiences can be used to assist a DQN with learning (**Q** 3).

Chapter 7

Semi-Model-Based Reinforcement Learning

It still remains the question to be answered, if the combination of synthetic experiences with model-free RL algorithms is to be classified as model-based or model-free RL (**Q** 5). In the following section, an analysis of the presented approaches of this work in the context of the definitions of model-based/free RL is conducted.

Parts of the following text have already been published in a similar form in [PSH22a].

7.1 Semi-Model-Based Reinforcement Learning

First, a categorization of interpolation based methods that make use of ER into the three main categories of model-based RL from subsection 2.3.3 is conducted:

1. **Type of model**: As IER/FU-DQN uses true experiences stored in the real-valued replay buffer as sampling points to interpolate rewards and follow-up states, it can be classified as *forward model*.

2. **Region in which the model is valid**: Interpolated experiences are created from surrounding sampling points and the interpolation model is discarded afterwards. This falls clearly into the *local* region category.

3. **Estimation method**: As no model is learned that is parametrized, the parametric class can be excluded. As stated in subsection 2.3.3 replay buffer methods could be considered as model-based approaches, but for the scope of this work they are not classified as such. This assumption is taken as the range of predictions that these "models" can make is very limited to, not only the exact experienced transitions of the learners past, but additionally only to those that are currently stored in the buffer. This means that no form of generalization is possible as well. The second argument also holds for parametric and exact methods like Dyna, but in contrast, these methods are capable of covering a bigger area of the state space (they do not forget) and in addition can model stochastic state-transitions way better than a simple fraction of past transitions

does. In fact, experiences in a replay buffer can be seen as true experiences, even though off-policy ones. This assumption is retrieved because they are not directly generated from an interaction with the environment but drawn from a memory. If they are considered true experiences, they can not be simulated ones and considering the definition that model-based methods use synthetic samples to learn, replay buffer methods do not satisfy this requirement. This consideration could be interpreted to be true for tabular models as well, but the main difference here is that replayed experiences are drawn from the distribution created from a developing/converging policy and can therefore be seen as off-policy experiences while reused experiences from a tabular model do not follow such a distribution. Indeed, there are researchers who describe the usage of replay methods as a way to avoid a conventional model [VS15]. So, the underlying replay buffer functionality does not fit in here, even if the line is thin. Interpolation, which is used on top, can not be considered as an exact approach apart from the values equal to the sampling points and therefore does not fit the non-parametric and exact class as well. The only remaining candidate among the traditional classes is non-parametric and approximate. Here, the line is very thin again. The general concept of interpolation based on stored true experiences could be classified as a non-parametric model. Then again, interpolation is a different thing than approximation (cf. subsection 2.5.1) and it remains at least discussable if such methods fit in here. Nevertheless, the simulated experiences are used in a replay-based manner which is regarded as model-free learning in general (see above). As interpolated samples are added into the mix (IER) this is of course not true anymore for all updates and IER combines planning and learning in a stochastic manner which at least differs from classic model-based RL approaches in the way that they usually have a clear separation of these phases. For FU-DQN, no synthetic experiences are added to the buffer, but the update combines synthetic and real experiences as well. In conclusion, IER/FU-DQN is not considered as a whole to meet the requirements of a non-parametric model to the fullest.

The question may arise why IER/FU-DQN uses interpolation instead of approximation in the first place. Looking at the two different situations from subsection 2.5.1, IER/FU-DQN clearly fits the second one. The replay buffer is of limited size and throws away experiences when its maximum size is reached. First of all, the amount of sampling points for an interpolation is restricted to the replays maximum size. And furthermore, predictions are needed for very specific local areas and as the true experiences are expected to be distributed over the state space this number can be expected to be rather small. Of course all of this depends on the chosen maximum size of the replay buffer and the problems state space, but it can be expected to rather have few than many sampling points and this scenario favours interpolation. In classic RL, sampling points usually come without noise (non-deterministic environments are an exceptional case) which requires that the points are matched exactly, also favouring interpolation. Of course in real world scenarios (that are OC systems often applied to) this is not true, but the first point still holds.

Focusing on value-based and model-free RL techniques like DQN, IER/FU-DQN also expresses some beneficial features of model-based approaches. The generation and usage of synthetic experiences is of course one of them. In early exploration phases, the learner is assisted with interpolated transitions that try to cover unexplored areas. This effect could be classified as model-based RL, but over time the generation of synthetic experiences is reduced (IER) and the method focuses on model-free Q-updates. Under the assumption that interpolation would fit the model-based RL definition, IER still would be a method that shifts from model-based to model-free and might even shift back if concept drifts occur. FU-DQN makes use of interpolated experiences for the whole training process and it was even stated, that the generation function I could also be a model. If a model is used, then FU-DQN has clearly to be classified as model-based, but when interpolation is used the classification remains unclear.

According to [Jac04] classification and categorization can be defined and distinguished as follows: Classification involves the assignment of each entity to one and only one class, whereas categorization is described as the process of dividing the world into groups of entities that share some similarities. The main difference is that classes can not overlap whereas categories can.

In conclusion, IER/FU-DQN (-with interpolation) could be categorized as a model-free, as well as a model-based RL method depending on the point of view and the definition of single aspects, as in fact the line is very thin. But on the other hand, if one wants to classify it he would need to commit for one class, and, following the former argumentation, this is neither easy nor explicit. Therefore, a class right in-between model-based and model-free is introduced and called *semi-model-based*. Following this new option, IER/FU-DQN (-with interpolation) is to be classified as a semi-model-based RL approach.

7.2 Interpolation-Based RL in Organic Computing

An architectural approach for OC systems is the generic *Observer/Controller*-architecture, more specifically the *Multi-Layer Observer/Controller*-architecture (MLOC) [MT17]. Even if there do exist other approaches beside it, the focus is on this one because it is the architecture that is tied to OC the closest and has even been set into close relation to RL [Ste+18]. The MLOC architecture is composed of 4 layers. At the bottom sits *layer 0* which is the productive system and, in the OC domain, is often called *System under Observation and Control* (SuOC). Here manageable resources are encapsulated via well-defined interfaces that enable monitoring (observation) as well as configuration (control). The SuOC is considered to be deployed in a *Non-Stationary Environment* (NSE) that continuously challenge the MLOC with unforeseen situations and external disturbances. *Layer 1* is called the runtime adaptation layer and deploys a feedback control loop by periodically observing the internal system state and adapting the SuOC accordingly. A form of online RL approach is often situated in this layer. *Layer 2* is known as the offline learning layer and its main functionality is the monitoring of layer 1. If a critical situation (unknown state or disturbance) is detected, then an internal model of the SuOC is used to find a solution. The model can be an upfront known simulation or even a parametrized model that is trained during runtime. Critical exploration is outsourced to this layer so that situations in which the SuOC might be harmed can be reduced to a minimum. The last layer is called collaboration layer and is responsible for communicating with entities from the outside such as the user or neighbouring MLOC instances.

An IER implemented into an OC system based on a MLOC architecture would look the following way: First of all, the online learning DQN instance is located in layer 1 and continuously interacts with the SuOC to generate new insights in the form of true experiences. The replay buffer used on this layer shall be D_{l1} and this buffer is composed of true and simulated experiences and has a maximum length of l_1. When the maximum length is reached, new samples replace old ones in a FIFO manner. Minibatches drawn at random from D_{l1} are used continuously to perform Q-updates and train the online DQN. The true experiences generated by interacting with the SuOC follow the trajectory $\tau = ((S_1, A_1), \ldots, (S_n, A_n))$ with $t = 1 \ldots n$. Even if neural networks are capable of generalisation, it requires a lot of samples (in form of experiences) to understand the dynamics of the environment. This results in a lot of exploration which can be costly in real world scenarios. Reasons for that are among others energy, abrasion and possible damage to either the OC system or entities in the environment. To reduce exploration, synthetic experiences can be generated in layer 2. Therefore, the real experiences stored in D_{l1} are copied to a sampling-point-buffer called D_{l2}. This storage is of length l_2 and can be bigger than l_1 resulting in $l_2 \geq l_1$. The benefit of remembering true experiences longer than layer 1 does is an increased accuracy for interpolations, whereas sticking too long to old samples in layer 1 can result in unstable learning. Asynchronously, in an offline manner, layer 2 triggers interpolations in areas surrounding τ. To realize that, a state S_t is drawn uniformly from τ and the querypoint S_q is drawn uniformly from the corresponding ball of radius r: $B_r(s_i) = \{y \in \mathbb{R}^n : |x - y| \leq r]\}$. Using the interpolation technique described in chapter 5 simulated experiences can be generated and added to D_{l1} which helps with the exploration around τ and therefore can result in reduced exploration needed in the real world. This effect was shown in detailed evaluations in [PSH21; PSH22b] and in chapter 4 and chapter 5.

As mentioned above, the system is considered to operate in a NSE and therefore concept drifts (changes in the world dynamics) can be expected to occur which require the agent to relearn the environments dynamics at least for local areas. Consequentially, exploration is required to adapt. A mechanic that is able to detect concept drifts would trigger the reset of D_{l2}, and probably also D_{l1}, and the new dynamics could be learned with a reduced amount of exploration. An approach that would reduce exploration even more would require a technique that is able to not only detect when a concept drift occurs, but also where. In this case only stored experiences from that area would be deleted and all the still valid sampling points could be kept.

The MLOC architecture described above is typically realized with rule-based learning approaches and one representant of such is the Organic Traffic Control (OTC) [Pro+11]. The authors use Learning Classifier Systems (LCSs) to generate rules that configure traffic lights for a crossroad. Here, on layer 1, an online version of an LCS reacts in real time to changing traffic volumes. The rules feature evaluation metrics such as fitness, expected payoff, expected error and experience. The expected payoff resembles a state-action-value and is mainly used, among the other metrics, to optimize the rule set. According to the definition of model-free RL methods from subsection 2.3.2 the learning approach on layer 1 can be classified as such. If unforeseen or unknown situations occur, the offline layer is triggered. Here a Genetic Algorithm (GA) generates new rules of minimum quality in interaction with a simulation of the underlying environment. A simulation that is given upfront holds as model, but a GA performs optimization and does not learn. So, layer 2 on its own can not be classified as model-based RL. In combination with layer 1 on the other hand it fulfills the requirements of storing a global solution and the usage of a model. Another requirement says that simulated experiences have to be used, and this is not the case, instead the model is used to produce new rules. Furthermore, in contrast to the typical model-based RL approach, the triggering of the model is restricted to special cases instead of a general assistance for the global solution.

The classification of this system in one of the classes model-free or model-based is not easy and obvious and the OTC tends to be in-between both. Therefore, it could be classified as a semi-model-based RL approach. Furthermore, the OC system that implements IER follows the argumentation of section 7.1 and turns out to be a semi-model-based RL approach as well. In conclusion, it can be seen that the MLOC architecture pairs well with approaches of this type, as the separation between the online component and the offline and model-based component encourages them.

Chapter 8

Conclusion

In the end, a conclusion examines if hypothesis 1 can be confirmed based on the two presented approaches and the evaluation results. Afterwards, a summary will give an overview of the contents that have been presented in the dissertation and the work is concluded with a short outlook of potential future work.

8.1 Conclusion

As the five research questions have been identified with the scope of answering hypothesis 1, they will be investigated each in detail first.

Q 1: *How can gathered knowledge in form of experiences be exploited to generate meaningful synthetic experiences?*

Both, IER and FU-DQN use stored experiences in the replay buffer as sampling points to generate synthetic experiences by means of interpolation. It is important to have a method that finds all relevant sampling points for a given query point x_q and that is as efficient as possible in terms of complexity as this procedure is repeated many times. A dictionary solution is used for discrete environments, while a form of k-d tree does the trick for continuous environments. With a set of NNs at hand, an appropriate interpolation method can generate synthetic experiences. The evaluations of IER and FU-DQN prove that the generated experiences are meaningful enough to assist the learner in a way that can reduce the required time to solution.

Q 2: *Which parts of an experience $e_t = (S_t, A_t, R_{t+1}, S_{t+1}, d_{t+1})$ can be interpolated, and what needs to be considered by doing so?*

In general, all parts of an experience can be interpolated, but there are some parts that are more critical than others. In a proof-of-concept implementation for continuous non-deterministic environments only the reward is interpolated while the follow-up state and the terminal tag are copied from matching observed experiences. The reward is also the part of an experience that is the least critical when it is interpolated. As discussed in subsection 4.1.1, this is reasoned by the fact that its impact on the TD-error is tied to its exact value. However, for IER for continuous environments and FU-DQN the follow-up state and the terminal tag are successfully interpolated as well. As mentioned above, the influence that these parts have on the TD error is greater than that of the reward which comes from the recursive nature of the bellman equation that is part of the TD-error. An error in the estimate of a follow-up state produces a false chain of future states and rewards and a similar thing holds for the terminal tag. A small example visualizes the problem of unwanted and unrecognized extrapolation cases that might occur when follow-up states are interpolated as raw values (cf. subsection 5.1.1). A solution to this problem is the interpolation of the state-transition-delta instead. By doing so all cases are interpolation cases and no extrapolation cases can occur. With such a synthetic state-transition-delta at hand, a synthetic follow-up state can be computed.

Q 3: *How can synthetic experiences be used to assist a Deep Q-Network with learning?*

Two different ways of how synthetic experiences can be used to assist a DQN have been proposed during the former sections. The first one is the *Interpolated Experience Replay* that stores synthetic experiences into the buffer next to real experiences that have been generated by interaction with the environment. The update functionality of a DQN remains unchained and minibatches of both, synthetic as well as real experiences are drawn at random from the buffer. Different configurations of the IER buffer have been presented, but the main idea is that the buffer that holds only synthetic experiences shrinks over time when the amount of real experiences grows. This is reasoned by the expectation that interpolations can suffer from errors while true experiences can be trusted at any time. In the beginning, when the agent knows little about the problem space, synthetic experiences can assist with exploration but later on, when the learner has build up some wisdom (good enough policy cf. chapter 1) the focus should be on the true experiences. It turned out that IER comes with a lot of hyperparameters that need to be tuned for problems individually.

The second presented approach is the *Full-Update DQN* that makes use of the fact that the original DQN structure is build such that a training update over the whole action space would be possible in general. This is not done in the original DQN as important values for the computation of the TD error for all actions but one are missing at the time of the update. If those values, or at least estimates of them, are available, then a training update could cover the whole action space. This is what FU-DQN does. Synthetic experiences are interpolated the same way as for IER but not stored in the buffer and only used once for a *full update*.

Q 4: *Does the usage of synthetic experiences in combination with a Deep Q-Network result in faster convergence and consequently in an increased sample efficiency?*

The evaluation of linear reward interpolation IER for discrete and non-deterministic environments (cf. section 4.3) shows that synthetic experiences can be of great assistance in such cases. Three different state-encodings of the MOUNTAINCAR problem represent different aspects like local and global state-spaces and encodings that encourage generalization resp. do not. For each state-encoding and four different exploration lengths, it was possible to find an IER configuration that reduces the required time to solution. Furthermore, the configurations that did so, also enabled a more consistent solution of the problem as well. However, one exception was the VE state-encoding with an exploration length of 750 episodes. Here, the baseline performed better than IER and an exploration length of 750 on that state-encoding seems to be an optimal configuration that leaves no potential for improvements. Overall, linear reward interpolation IER is able to reduce the amount of required episodes until a discrete and non-deterministic problem is considered as solved and thus, can help increase convergence.

IER for continuous environments was evaluated on the MOUNTAINCAR problem (cf. section 5.3), again with four different exploration lengths. It turned out that the baseline is able to solve the task very well on a very low exploration length and, again, here it seems that there is no room for improvements left. When the exploration time is increased on the other hand, IER starts to outperform ER. The assumption here is that IER requires problems that need a sufficiently high exploration time for IER to bring noticeable positive effects. Nevertheless, IER is able to reduce the required time to solution in continuous environments that are complex enough. In such cases, the convergence can be speed up as well.

At last, FU-DQN was evaluated on four different problems (cf. section 6.2) that all share a sparse reward setting. For all these problems, FU-DQN was able to reduce the required amount of episodes until a task was considered as solved. Also, convergence could be speed up.

As discussed before, the effect, that the usage of synthetic experiences can reduce the required amount of episodes (samples) to solve a task, increases the sample efficiency, as less samples are required to gain a similar result that a comparable baseline can achieve without any synthetic experiences. Furthermore, true experiences (samples) are used many times to generate synthetic samples which assist the learner and so, sample efficiency of true experiences (samples) is increased even further.

Q 5: *Is the combination of model-free Reinforcement Learning methods that make use of Experience Replay with synthetic experiences that have been generated by means of interpolation classifiable as model-free or model-based Reinforcement Learning?*

As discussed in chapter 7, methods that combine the usage of synthetic experiences with a replay buffer and model-free RL algorithms are not easily classified into one of the two classes model-based and model-free. An in-depth investigation of different aspects of such methods and how they align with the definition of model-based RL made clear, that no definite answer can be given to this question. To escape this dilemma, the new class semi-model-based was introduced, that is located somewhere in between the former two categories. Model-free RL algorithms that use synthetic experiences in combination with a replay buffer are to be classified as semi-model-based.

Let's now come back to hypothesis 1. With a (positive) answer to all of the identified research questions, the hypothesis can be answered as follows:

Yes! Stored samples in the ER buffer can be used as sampling points for the interpolation of synthetic experiences. Such synthetic experiences are meaningful and can help a learner, but the methods that have been proposed to do so are heavily reliant on correct hyperparameters.

Yes! These synthetic experiences can be used to assist the learner to speed up learning and increase sample efficiency. Two different methods have been proposed that offer a way to include synthetic experiences into the learning process. Both are able to speed up learning such that the required amount of episodes to solve a task can be reduced. Consequently, these methods increase sample efficiency.

Even if the amount of required episodes can be reduces, interpolation remains an operation that can be very costly in terms of computation. Therefore, semi-model-based RL approaches that make use of interpolation are best to be used when the amount of actual exploration actions should be kept as small as possible. This fits well to real-world scenarios which aligns well to OC in general.

8.2 Summary

The present work dealt with the interpolation of synthetic experiences that are used in combination with the model-free RL algorithm DQN. In the context of MDPs, experiences can be treated as knowledge of an underlying environment and its dynamics. This knowledge can be exploited to generate synthetic experiences by means of interpolation that cover yet unexplored areas in the state-space and can therefore help with exploration.

At first, the theoretical background of RL was provided in form of an extensive overview. Starting with the general concepts of finite MDPs, the background section covered types of RL and introduced some value-based RL algorithms like Q-Learning and DQN. Also, the concept of interpolation was covered as well as some methods for efficient nearest neighbour searches. After a short summary of related work that is of relevance for the usage of synthetic experiences in combination with DQN, two concrete ways of how to use synthetic experiences have been presented.

Those are, IER and FU-DQN. While IER adds synthetic experiences to the replay buffer and leaves the update mechanism of a DQN unchanged, FU-DQN uses synthetic experiences to enable updates that span over the whole action space. For IER, a proof-of-concept version for discrete and non-deterministic environments was presented first. An in-depth evaluation revealed that this approach works well and has potential to be extended for continuous environments. It turned out, that the follow-up state is a crucial part of an experience that can suffer from unrecognized extrapolation cases when interpolated straight away. An interpolation of state-transition-deltas instead solved this problem and made IER ready for continuous environments. An evaluation of the MOUNTAINCAR problem revealed that IER can be beneficial, but it requires sufficiently long enough explorations phases. FU-DQN exploits that the architecture of a DQN is build such that an update is possible for the whole action space in theory. As at the time of an update data is only available for one specific action (determined by the present experience) this is not done in the original approach. FU-DQN generated the missing values based on stored experiences in the replay buffer and can perform updates that cover the whole action space. An evaluation on four different problems that share a sparse reward signal showed that FU-DQN can be of great benefit.

In the end, methods that combine synthetic experiences, a replay buffer and model-free RL algorithms have been tried to fit into either the model-based or the model-free category. It turned out that this classification is not possible unambiguously. Thus, the new class semi-model-based was introduced to fit such cases.

8.3 Outlook

The evaluation of IER for continuous environments showed that ER outperforms IER on a small exploration phase, while IER performs better on larger exploration phases. The assumption is that IER requires high exploration phases and consequently more complex problems to be beneficial. Because of time restrictions, this assumption could not be verified with further evaluations. This should be done in the future to confirm this assumption. Furthermore, the presented approaches are limited to state-spaces that are interpolable and, for example, is not usable for images. It might be possible however, to use the output of the classification layer of a Convolutional Neural Network as a representation of the state-input and use those values as sampling points for interpolations. This would of course require a lot of further research but seems like a good point to start with future work. As mentioned above, interpolation can be very costly and so far, no investigation of the actual required computation time was done. A comparison of the effect of the reduced required time with the increased required computation time would be a good thing to have. In the context of real world scenarios (OC), inputs have to be expected to be noisy. This is reasoned by the sensors that produce these values and that can suffer from measurement uncertainties and shifts. Also, sensors need to be re-calibrated in regular intervals. In such environments, interpolation could be of great benefit, even for states, that have already been observed as values suffer from noise. Semi-model-based approaches that make use of interpolation could be a good fit here. Such scenarios could easily be simulated by adding noise to states. An evaluation of IER and/or FU-DQN in such scenarios would be of great interest.

Appendix A

Evaluation Results from Linear Reward Interpolation

A.1 Solution Rates

TABLE A.1: Probability (in %) for each IER configuration in the VE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	Query Function	Vanilla	OnPol	ONS	OnPolONS
Def	250	LS	91	59	87	*
Def	250	PD	50	81	94	*
Def	500	LS	68	50	94	*
Def	500	PD	93	84	89	*
Def	750	LS	13	50	35	*
Def	750	PD	13	8	50	*
Def	1000	LS	24	67	50	*
Def	1000	PD	83	50	49	*
Def-area	250	LS	50	75	58	*
Def-area	250	PD	50	32	87	*
Def-area	500	LS	69	98	94	*
Def-area	500	PD	19	51	94	*
Def-area	750	LS	8	22	1	*
Def-area	750	PD	1	5	7	*
Def-area	1000	LS	50	36	67	*
Def-area	1000	PD	24	82	65	*
Full	250	LS	51	*	*	*
Full	250	PD	74	*	*	*
Full	500	LS	97	*	*	*
Full	500	PD	94	*	*	*
Full	750	LS	34	*	*	*

(To be continued)

TABLE A.1: Probability (in %) for each IER configuration in the VE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**	**ONS**	**OnPolONS**
Full	750	PD	34	*	*	*
Full	1000	LS	35	*	*	*
Full	1000	PD	82	*	*	*
Full-area	250	LS	19	*	*	*
Full-area	250	PD	50	*	*	*
Full-area	500	LS	4	*	*	*
Full-area	500	PD	50	*	*	*
Full-area	750	LS	4	*	*	*
Full-area	750	PD	33	*	*	*
Full-area	1000	LS	82	*	*	*
Full-area	1000	PD	66	*	*	*
Zeta	250	LS	*	*	*	50
Zeta	250	PD	*	*	*	81
Zeta	500	LS	*	*	*	77
Zeta	500	PD	*	*	*	100
Zeta	750	LS	*	*	*	13
Zeta	750	PD	*	*	*	13
Zeta	1000	LS	*	*	*	24
Zeta	1000	PD	*	*	*	94
Zeta-area	250	LS	*	*	*	67
Zeta-area	250	PD	*	*	*	13
Zeta-area	500	LS	*	*	*	93
Zeta-area	500	PD	*	*	*	96
Zeta-area	750	LS	*	*	*	22
Zeta-area	750	PD	*	*	*	49
Zeta-area	1000	LS	*	*	*	50
Zeta-area	1000	PD	*	*	*	94

TABLE A.2: Probability (in %) for each IER configuration in the CE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**	**ONS**	**OnPolONS**
Def	250	LS	19	75	10	*
Def	250	PD	17	50	18	*
Def	500	LS	99	95	97	*
Def	500	PD	85	6	24	*
Def	750	LS	98	87	82	*
Def	750	PD	9	2	14	*
Def	1000	LS	100	33	82	*
Def	1000	PD	18	17	3	*
Def-area	250	LS	75	50	10	*
Def-area	250	PD	75	50	31	*
Def-area	500	LS	100	100	91	*
Def-area	500	PD	100	100	98	*
Def-area	750	LS	100	100	100	*
Def-area	750	PD	100	100	99	*
Def-area	1000	LS	100	100	100	*
Def-area	1000	PD	100	100	100	*
Full	250	LS	50	*	*	*
Full	250	PD	50	*	*	*
Full	500	LS	100	*	*	*
Full	500	PD	95	*	*	*
Full	750	LS	100	*	*	*
Full	750	PD	75	*	*	*
Full	1000	LS	96	*	*	*
Full	1000	PD	41	*	*	*
Full-area	250	LS	75	*	*	*
Full-area	250	PD	75	*	*	*
Full-area	500	LS	100	*	*	*
Full-area	500	PD	100	*	*	*
Full-area	750	LS	100	*	*	*
Full-area	750	PD	100	*	*	*
Full-area	1000	LS	100	*	*	*
Full-area	1000	PD	100	*	*	*
Zeta	250	LS	*	*	*	10
Zeta	250	PD	*	*	*	30

(To be continued)

TABLE A.2: Probability (in %) for each IER configuration in the CE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**	**ONS**	**OnPolONS**
Zeta	500	LS	*	*	*	70
Zeta	500	PD	*	*	*	24
Zeta	750	LS	*	*	*	6
Zeta	750	PD	*	*	*	0
Zeta	1000	LS	*	*	*	24
Zeta	1000	PD	*	*	*	5
Zeta-area	250	LS	*	*	*	11
Zeta-area	250	PD	*	*	*	3
Zeta-area	500	LS	*	*	*	50
Zeta-area	500	PD	*	*	*	23
Zeta-area	750	LS	*	*	*	42
Zeta-area	750	PD	*	*	*	34
Zeta-area	1000	LS	*	*	*	87
Zeta-area	1000	PD	*	*	*	8

TABLE A.3: Probability (in %) for each IER configuration in the LKE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**	**ONS**	**OnPolONS**
Def	250	LS	1	0	31	*
Def	250	PD	0	8	31	*
Def	500	LS	2	4	77	*
Def	500	PD	0	1	68	*
Def	750	LS	0	0	60	*
Def	750	PD	0	0	86	*
Def	1000	LS	0	8	69	*
Def	1000	PD	0	8	69	*
Def-area	250	LS	0	3	50	*
Def-area	250	PD	0	2	40	*
Def-area	500	LS	0	13	77	*
Def-area	500	PD	0	4	60	*
Def-area	750	LS	0	1	31	*
Def-area	750	PD	0	8	23	*

(To be continued)

TABLE A.3: Probability (in %) for each IER configuration in the LKE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**	**ONS**	**OnPolONS**
Def-area	1000	LS	0	1	12	*
Def-area	1000	PD	0	1	17	*
Full	250	LS	0	*	*	*
Full	250	PD	0	*	*	*
Full	500	LS	0	*	*	*
Full	500	PD	0	*	*	*
Full	750	LS	0	*	*	*
Full	750	PD	0	*	*	*
Full	1000	LS	0	*	*	*
Full	1000	PD	0	*	*	*
Full-area	250	LS	0	*	*	*
Full-area	250	PD	0	*	*	*
Full-area	500	LS	0	*	*	*
Full-area	500	PD	0	*	*	*
Full-area	750	LS	0	*	*	*
Full-area	750	PD	0	*	*	*
Full-area	1000	LS	0	*	*	*
Full-area	1000	PD	0	*	*	*
Zeta	250	LS	*	*	*	50
Zeta	250	PD	*	*	*	78
Zeta	500	LS	*	*	*	94
Zeta	500	PD	*	*	*	77
Zeta	750	LS	*	*	*	31
Zeta	750	PD	*	*	*	50
Zeta	1000	LS	*	*	*	78
Zeta	1000	PD	*	*	*	24
Zeta-area	250	LS	*	*	*	60
Zeta-area	250	PD	*	*	*	70
Zeta-area	500	LS	*	*	*	50
Zeta-area	500	PD	*	*	*	68
Zeta-area	750	LS	*	*	*	23
Zeta-area	750	PD	*	*	*	60
Zeta-area	1000	LS	*	*	*	24
Zeta-area	1000	PD	*	*	*	24

A.2 Probabilities

TABLE A.4: Probabilities (in %, rounded to two decimal places) for each configuration in the VE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def	**250**	**LS**	8.51	0.15	**91.35**
Def	**250**	**PD**	48.26	0.37	**51.37**
Def	**500**	**LS**	20.69	0.56	**78.75**
Def	**500**	**PD**	0.97	0.07	**98.96**
Def	750	LS	98.32	0.44	1.24
Def	750	PD	99.08	0.27	0.65
Def	1000	LS	79.45	3.61	16.94
Def	**1000**	**PD**	2.56	1.61	**95.83**
Def-OnPol	**250**	**LS**	40.03	0.37	**59.60**
Def-OnPol	**250**	**PD**	18.35	0.29	**81.36**
Def-OnPol	**500**	**LS**	47.82	0.74	**51.44**
Def-OnPol	**500**	**PD**	8.11	0.36	**91.53**
Def-OnPol	750	LS	52.81	6.12	41.08
Def-OnPol	750	PD	99.54	0.14	0.32
Def-OnPol	**1000**	**LS**	23.28	5.92	**70.80**
Def-OnPol	1000	PD	48.52	6.33	45.14
Def-ONS	**250**	**LS**	10.65	0.23	**89.12**
Def-ONS	**250**	**PD**	2.98	0.07	**96.96**
Def-ONS	**500**	**LS**	1.10	0.08	**98.82**
Def-ONS	**500**	**PD**	2.80	0.16	**97.04**
Def-ONS	750	LS	70.97	4.56	24.47
Def-ONS	750	PD	73.33	4.70	21.96
Def-ONS	1000	LS	52.76	6.44	40.80
Def-ONS	1000	PD	51.53	6.48	41.99
Def-area	250	LS	50.83	0.32	48.85
Def-area	**250**	**PD**	20.01	0.32	**79.67**
Def-area	**500**	**LS**	24.32	0.58	**75.10**
Def-area	500	PD	84.01	0.36	15.62
Def-area	750	LS	99.78	0.06	0.16

(To be continued)

TABLE A.4: Probabilities (in %, rounded to two decimal places) for each configuration in the VE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-area	750	PD	99.99	0.00	0.01
Def-area	1000	LS	56.31	6.55	37.14
Def-area	1000	PD	80.07	3.70	16.23
Def-area-OnPol	**250**	**LS**	27.25	0.32	**72.43**
Def-area-OnPol	250	PD	68.81	0.33	30.85
Def-area-OnPol	**500**	**LS**	0.14	0.02	**99.84**
Def-area-OnPol	**500**	**PD**	44.71	0.71	**54.58**
Def-area-OnPol	750	LS	96.68	0.82	2.50
Def-area-OnPol	750	PD	99.90	0.01	0.08
Def-area-OnPol	1000	LS	64.25	5.43	30.33
Def-area-OnPol	**1000**	**PD**	3.05	1.81	**95.15**
Def-area-ONS	**250**	**LS**	38.99	0.35	**60.66**
Def-area-ONS	**250**	**PD**	12.04	0.21	**87.76**
Def-area-ONS	**500**	**LS**	0.88	0.04	**99.08**
Def-area-ONS	**500**	**PD**	1.01	0.06	**98.93**
Def-area-ONS	750	LS	99.99	0.00	0.01
Def-area-ONS	750	PD	99.62	0.10	0.28
Def-area-ONS	**1000**	**LS**	23.28	5.90	**70.82**
Def-area-ONS	**1000**	**PD**	23.74	6.07	**70.20**
Full	250	LS	55.55	0.36	44.09
Full	**250**	**PD**	26.23	0.36	**73.41**
Full	**500**	**LS**	0.24	0.01	**99.76**
Full	**500**	**PD**	1.95	0.13	**97.92**
Full	750	LS	81.83	3.54	14.62
Full	750	PD	78.41	3.75	17.84
Full	1000	LS	70.47	4.91	24.62
Full	**1000**	**PD**	16.43	5.34	**78.23**
Full-area	250	LS	69.99	0.26	29.74
Full-area	250	PD	56.81	0.32	42.87
Full-area	500	LS	97.99	0.06	1.95

(To be continued)

TABLE A.4: Probabilities (in %, rounded to two decimal places) for each configuration in the VE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Full-area	**500**	**PD**	46.80	0.69	**52.51**
Full-area	750	LS	99.74	0.07	0.19
Full-area	750	PD	86.97	2.56	10.47
Full-area	**1000**	**LS**	1.76	1.21	**97.03**
Full-area	**1000**	**PD**	10.92	3.72	**85.36**
Zeta-OnPolONS	**250**	**LS**	29.65	0.37	**69.97**
Zeta-OnPolONS	**250**	**PD**	16.31	0.26	**83.43**
Zeta-OnPolONS	**500**	**LS**	13.95	0.43	**85.62**
Zeta-OnPolONS	**500**	**PD**	0.00	0.00	**100.00**
Zeta-OnPolONS	750	LS	98.25	0.36	1.39
Zeta-OnPolONS	750	PD	98.37	0.50	1.14
Zeta-OnPolONS	1000	LS	77.79	3.91	18.31
Zeta-OnPolONS	**1000**	**PD**	0.57	0.52	**98.92**
Zeta-area-OnPolONS	**250**	**LS**	33.20	0.28	**66.52**
Zeta-area-OnPolONS	250	PD	79.62	0.22	20.16
Zeta-area-OnPolONS	**500**	**LS**	1.57	0.14	**98.29**
Zeta-area-OnPolONS	**500**	**PD**	0.42	0.04	**99.54**
Zeta-area-OnPolONS	750	LS	93.31	1.50	5.19
Zeta-area-OnPolONS	750	PD	85.95	3.12	10.94
Zeta-area-OnPolONS	**1000**	**LS**	36.17	6.37	**57.46**
Zeta-area-OnPolONS	**1000**	**PD**	1.81	1.50	**96.69**

TABLE A.5: Probabilities (in %, rounded to two decimal places) for each configuration in the CE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def	250	LS	76.45	3.49	20.06
Def	250	PD	96.58	0.74	2.67
Def	**500**	**LS**	1.20	0.24	**98.56**
Def	**500**	**PD**	24.79	1.79	**73.42**
Def	**750**	**LS**	0.77	0.20	**99.04**
Def	750	PD	90.41	0.71	8.88
Def	**1000**	**LS**	0.02	0.01	**99.98**
Def	1000	PD	60.57	1.56	37.87
Def-OnPol	**250**	**LS**	15.51	3.84	**80.65**
Def-OnPol	250	PD	96.58	0.77	2.65
Def-OnPol	**500**	**LS**	8.48	1.03	**90.49**
Def-OnPol	500	PD	98.45	0.12	1.44
Def-OnPol	**750**	**LS**	16.43	1.57	**82.00**
Def-OnPol	750	PD	97.05	0.24	2.71
Def-OnPol	1000	LS	52.87	1.72	45.41
Def-OnPol	1000	PD	64.99	1.41	33.61
Def-ONS	250	LS	99.80	0.04	0.16
Def-ONS	250	PD	99.93	0.02	0.05
Def-ONS	**500**	**LS**	3.57	0.53	**95.89**
Def-ONS	500	PD	89.86	0.76	9.38
Def-ONS	**750**	**LS**	19.20	1.82	**78.98**
Def-ONS	750	PD	85.07	0.94	13.99
Def-ONS	**1000**	**LS**	14.46	1.11	**84.43**
Def-ONS	1000	PD	89.59	0.55	9.87
Def-area	**250**	**LS**	0.00	0.00	**100.00**
Def-area	**250**	**PD**	0.00	0.01	**99.99**
Def-area	**500**	**LS**	0.00	0.00	**100.00**
Def-area	**500**	**PD**	0.00	0.00	**100.00**
Def-area	**750**	**LS**	0.00	0.00	**100.00**
Def-area	**750**	**PD**	0.00	0.00	**100.00**

(To be continued)

TABLE A.5: Probabilities (in %, rounded to two decimal places) for each configuration in the CE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-area	**1000**	**LS**	0.00	0.00	**100.00**
Def-area	**1000**	**PD**	0.00	0.00	**100.00**
Def-area-OnPol	**250**	**LS**	10.05	2.61	**87.34**
Def-area-OnPol	**250**	**PD**	1.29	0.58	**98.12**
Def-area-OnPol	**500**	**LS**	0.00	0.00	**100.00**
Def-area-OnPol	**500**	**PD**	0.00	0.00	**100.00**
Def-area-OnPol	**750**	**LS**	0.00	0.00	**100.00**
Def-area-OnPol	**750**	**PD**	0.00	0.00	**100.00**
Def-area-OnPol	**1000**	**LS**	0.00	0.00	**100.00**
Def-area-OnPol	**1000**	**PD**	0.00	0.00	**100.00**
Def-area-ONS	250	LS	98.69	0.31	1.00
Def-area-ONS	250	PD	48.28	5.34	46.38
Def-area-ONS	**500**	**LS**	4.40	0.59	**95.01**
Def-area-ONS	**500**	**PD**	2.32	0.37	**97.31**
Def-area-ONS	**750**	**LS**	0.00	0.00	**100.00**
Def-area-ONS	**750**	**PD**	0.07	0.01	**99.91**
Def-area-ONS	**1000**	**LS**	0.00	0.00	**100.00**
Def-area-ONS	**1000**	**PD**	0.00	0.00	**100.00**
Full	250	LS	51.83	5.63	42.54
Full	250	PD	93.60	1.43	4.98
Full	**500**	**LS**	0.04	0.01	**99.95**
Full	**500**	**PD**	12.64	1.38	**85.99**
Full	**750**	**LS**	0.62	0.15	**99.23**
Full	**750**	**PD**	29.81	2.12	**68.07**
Full	**1000**	**LS**	1.11	0.17	**98.72**
Full	**1000**	**PD**	36.86	1.76	**61.38**
Full-area	**250**	LS	13.69	4.17	**82.14**
Full-area	**250**	**PD**	0.92	0.54	**98.55**
Full-area	**500**	**LS**	0.00	0.00	**100.00**
Full-area	**500**	**PD**	0.00	0.00	**100.00**

(To be continued)

TABLE A.5: Probabilities (in %, rounded to two decimal places) for each configuration in the CE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Full-area	**750**	**LS**	0.00	0.00	**100.00**
Full-area	**750**	**PD**	0.00	0.00	**100.00**
Full-area	**1000**	**LS**	0.00	0.00	**100.00**
Full-area	**1000**	**PD**	0.00	0.00	**100.00**
Zeta-OnPolONS	250	LS	96.88	0.63	2.49
Zeta-OnPolONS	250	PD	92.16	1.62	6.22
Zeta-OnPolONS	**500**	**LS**	40.60	2.25	**57.15**
Zeta-OnPolONS	500	PD	91.91	0.57	7.51
Zeta-OnPolONS	750	LS	93.94	0.40	5.66
Zeta-OnPolONS	750	PD	98.95	0.10	0.95
Zeta-OnPolONS	1000	LS	64.99	1.44	33.57
Zeta-OnPolONS	1000	PD	88.40	0.62	10.98
Zeta-area-OnPolONS	250	LS	99.49	0.11	0.40
Zeta-area-OnPolONS	250	PD	99.99	0.00	0.01
Zeta-area-OnPolONS	500	LS	71.33	1.68	26.99
Zeta-area-OnPolONS	500	PD	87.46	0.82	11.72
Zeta-area-OnPolONS	750	LS	59.40	1.98	38.62
Zeta-area-OnPolONS	750	PD	70.96	1.50	27.54
Zeta-area-OnPolONS	**1000**	**LS**	9.10	0.90	**90.01**
Zeta-area-OnPolONS	1000	PD	77.30	1.11	21.60

TABLE A.6: Probabilities (in %, rounded to two decimal places) for each configuration in the LKE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def	250	LS	99.82	0.04	0.13
Def	250	PD	99.95	0.01	0.04
Def	500	LS	98.75	0.27	0.98
Def	500	PD	99.94	0.02	0.04
Def	750	LS	99.98	0.01	0.01
Def	750	PD	99.99	0.00	0.01
Def	1000	LS	100.00	0.00	0.01
Def	1000	PD	100.00	0.00	0.00
Def-OnPol	250	LS	99.85	0.04	0.10
Def-OnPol	250	PD	96.16	0.95	2.89
Def-OnPol	500	LS	98.40	0.37	1.22
Def-OnPol	500	PD	99.19	0.20	0.60
Def-OnPol	750	LS	99.96	0.02	0.02
Def-OnPol	750	PD	99.90	0.02	0.08
Def-OnPol	1000	LS	91.48	2.42	6.10
Def-OnPol	1000	PD	91.21	2.33	6.46
Def-ONS	250	LS	71.88	4.40	23.72
Def-ONS	250	PD	81.16	3.50	15.34
Def-ONS	**500**	**LS**	14.96	3.38	**81.65**
Def-ONS	500	PD	47.60	5.87	46.53
Def-ONS	**750**	**LS**	40.00	6.46	**53.54**
Def-ONS	**750**	**PD**	16.60	4.50	**78.90**
Def-ONS	**1000**	**LS**	16.81	4.93	**78.25**
Def-ONS	**1000**	**PD**	21.45	6.91	**71.64**
Def-area	250	LS	100.00	0.00	0.00
Def-area	250	PD	100.00	0.00	0.00
Def-area	500	LS	99.99	0.00	0.01
Def-area	500	PD	100.00	0.00	0.00
Def-area	750	LS	100.00	0.00	0.00
Def-area	750	PD	100.00	0.00	0.00

(To be continued)

TABLE A.6: Probabilities (in %, rounded to two decimal places) for each configuration in the LKE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-area	1000	LS	99.93	0.03	0.05
Def-area	1000	PD	100.00	0.00	0.00
Def-area-OnPol	250	LS	99.28	0.21	0.51
Def-area-OnPol	250	PD	98.89	0.32	0.79
Def-area-OnPol	500	LS	89.93	1.91	8.15
Def-area-OnPol	500	PD	97.60	0.61	1.80
Def-area-OnPol	750	LS	99.29	0.17	0.53
Def-area-OnPol	750	PD	94.40	1.24	4.36
Def-area-OnPol	1000	LS	99.21	0.24	0.54
Def-area-OnPol	1000	PD	99.18	0.29	0.54
Def-area-ONS	250	LS	49.00	6.01	44.99
Def-area-ONS	250	PD	77.18	4.32	18.50
Def-area-ONS	**500**	**LS**	36.58	5.93	**57.48**
Def-area-ONS	500	PD	45.59	5.50	48.91
Def-area-ONS	750	LS	71.88	4.51	23.61
Def-area-ONS	750	PD	88.76	2.45	8.79
Def-area-ONS	1000	LS	92.04	2.34	5.62
Def-area-ONS	1000	PD	91.42	2.58	6.00
Full	250	LS	100.00	0.00	0.00
Full	250	PD	100.00	0.00	0.00
Full	500	LS	100.00	0.00	0.00
Full	500	PD	100.00	0.00	0.00
Full	750	LS	100.00	0.00	0.00
Full	750	PD	100.00	0.00	0.00
Full	1000	LS	*	*	*
Full	1000	PD	100.00	0.00	0.00
Full-area	250	LS	*	*	*
Full-area	250	PD	*	*	*
Full-area	500	LS	*	*	*
Full-area	500	PD	*	*	*

(To be continued)

TABLE A.6: Probabilities (in %, rounded to two decimal places) for each configuration in the LKE state-encoding of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. A * indicates that that the test was not possible, because, no a single run of the IER configuration was able to solve the problem in time. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Full-area	750	LS	*	*	*
Full-area	750	PD	*	*	*
Full-area	1000	LS	*	*	*
Full-area	1000	PD	*	*	*
Zeta-OnPolONS	**250**	**LS**	36.81	5.61	**57.58**
Zeta-OnPolONS	**250**	PD	35.03	6.15	**58.81**
Zeta-OnPolONS	**500**	**LS**	12.91	3.68	**83.41**
Zeta-OnPolONS	**500**	**PD**	20.23	4.40	**75.37**
Zeta-OnPolONS	750	LS	76.93	3.94	19.13
Zeta-OnPolONS	750	PD	60.96	5.95	33.09
Zeta-OnPolONS	**1000**	LS	5.32	2.40	**92.28**
Zeta-OnPolONS	1000	PD	74.95	5.23	19.82
Zeta-area-OnPolONS	**250**	**LS**	29.88	5.83	**64.29**
Zeta-area-OnPolONS	**250**	**PD**	36.98	6.52	**56.51**
Zeta-area-OnPolONS	500	LS	60.81	5.21	33.98
Zeta-area-OnPolONS	**500**	**PD**	33.13	5.67	**61.20**
Zeta-area-OnPolONS	750	LS	81.33	3.48	15.20
Zeta-area-OnPolONS	750	PD	61.69	5.64	32.67
Zeta-area-OnPolONS	1000	LS	60.68	6.80	32.51
Zeta-area-OnPolONS	1000	PD	74.31	5.83	19.86

A.3 Mean TTS

TABLE A.7: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the VE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
ER	250	4229		[3404, 7483]	
Def	250	3046	[2122, 4737]	4181	[3290, 7564]
Def-ONS	250	3070	[2293, 4862]	2460	[1962, 3910]
Def-OnPol	250	4217	[3081, 6946]	3291	[2480, 5613]
Def-area	250	4421	[3311, 7804]	4001	[2941, 4806]
Def-area-OnPol	250	3578	[2744, 6226]	5416	[3891, 9522]
Def-area-ONS	250	4293	[3010, 6699]	2730	[2259, 5060]
Full	250	4208	[3372, 8404]	3453	[2774, 6000]
Full-area	250	5395	[4144, 7300]	4127	[3395, 8492]
Zeta-OnPolONS	250	4268	[3076, 5340]	3130	[2424, 5428]
Zeta-area-OnPolONS	250	3910	[2883, 6632]	5931	[4477, 8853]
ER	500	2442		[1930, 3795]	
Def	500	2089	[1600, 2912]	1383	[1201, 1948]
Def-ONS	500	1392	[1186, 1951]	1578	[1273, 2149]
Def-OnPol	500	2388	[1936, 3679]	1681	[1417, 2442]
Def-area	500	2069	[1674, 3006]	3576	[2538, 5034]
Def-area-OnPol	500	1270	[1059, 1679]	2530	[1879, 3627]
Def-area-ONS	500	1417	[1149, 1909]	1458	[1234, 1936]
Full	500	1304	[1065, 1710]	1512	[1255, 2056]
Full-area	500	3915	[3466, 7713]	2662	[1914, 3679]
Zeta-OnPolONS	500	1843	[1509, 2690]	1071	[937, 1323]
Zeta-area-OnPolONS	500	1514	[1246, 2017]	1332	[1128, 1786]
ER	750	1055		[933, 1192]	
Def	750	1348	[1173, 1657]	1405	[1205, 1710]
Def-ONS	750	1115	[978, 1314]	1136	[1003, 1282]
Def-OnPol	750	1057	[943, 1213]	1476	[1247, 1825]
Def-area	750	1587	[1295, 1924]	1810	[1510, 2373]
Def-area-OnPol	750	1314	[1132, 1564]	1589	[1348, 2012]
Def-area-ONS	750	1782	[1489, 2363]	1468	[1265, 1833]
Full	750	1142	[1020, 1364]	1179	[1005, 1352]
Full-area	750	1524	[1302, 1970]	1206	[1043, 1407]

(To be continued)

TABLE A.7: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the VE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Zeta-OnPolONS	750	1386	[1169, 1659]	1341	[1174, 1671]
Zeta-area-OnPolONS	750	1246	[1087, 1503]	1186	[1056, 1326]
ER	1000	1394		[1244, 1563]	
Def	1000	1498	[1334, 1792]	1195	[1106, 1294]
Def-ONS	1000	1396	[1262, 1590]	1390	[1254, 1586]
Def-OnPol	1000	1302	[1195, 1448]	1377	[1245, 1570]
Def-area	1000	1418	[1273, 1601]	1519	[1336, 1794]
Def-area-OnPol	1000	1424	[1281, 1669]	1187	[1114, 1299]
Def-area-ONS	1000	1299	[1197, 1449]	1321	[1190, 1453]
Full	1000	1404	[1303, 1704]	1285	[1189, 1402]
Full-area	1000	1178	[1090, 1271]	1231	[1126, 1392]
Zeta-OnPolONS	1000	1511	[1323, 1787]	1169	[1127, 1214]
Zeta-area-OnPolONS	1000	1333	[1205, 1533]	1198	[1158, 1256]

TABLE A.8: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the CE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
ER	250	1038		[941, 1234]	
Def	250	1179	[1011, 1407]	1360	[1168, 1612]
Def-OnPol	250	967	[861, 1091]	1344	[1171, 1567]
Def-ONS	250	1548	[1340, 1865]	1581	[1398, 1894]
Def-area	250	649	[601, 699]	732	[678, 790]
Def-area-OnPol	250	932	[803, 1074]	838	[742, 957]
Def-area-ONS	250	1502	[1236, 1866]	1073	[937, 1245]
Full	250	1088	[974, 1219]	1262	[1141, 1435]
Full-area	250	993	[914, 1047]	856	[787, 936]
Zeta-OnPolONS	250	1377	[1170, 1698]	1303	[1111, 1525]

(To be continued)

TABLE A.8: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the CE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Zeta-area-OnPolONS	250	1514	[1283, 1783]	1848	[1553, 2214]
ER	500	2518		[2112, 3250]	
Def	500	1763	[1601, 2136]	2283	[1976, 2762]
Def-OnPol	500	2091	[1795, 2448]	3768	[3170, 5152]
Def-ONS	500	1907	[1706, 2279]	3115	[2712, 4008]
Def-area	500	980	[921, 1053]	1178	[1059, 1340]
Def-area-OnPol	500	1177	[1095, 1278]	1289	[1188, 1427]
Def-area-ONS	500	1883	[1644, 2351]	1871	[1630, 2232]
Full	500	1674	[1542, 1855]	2151	[1934, 2503]
Full-area	500	1225	[1167, 1296]	1239	[1166, 1310]
Zeta-OnPolONS	500	2440	[2109, 3000]	3331	[2778, 4110]
Zeta-area-OnPolONS	500	2745	[2406, 3461]	2955	[2633, 4003]
ER	750	3549		[3022, 4715]	
Def	750	2632	[2335, 2986]	4760	[3920, 6024]
Def-OnPol	750	2994	[2705, 3783]	5093	[4385, 7091]
Def-ONS	750	3116	[2802, 3803]	4204	[3700, 5815]
Def-area	750	1463	[1377, 1568]	1500	[1352, 1617]
Def-area-OnPol	750	1618	[1491, 1777]	1947	[1750, 2246]
Def-area-ONS	750	1981	[1782, 2281]	2215	[1956, 2655]
Full	750	2583	[2352, 2953]	3363	[2935, 4045]
Full-area	750	1493	[1431, 1567]	1571	[1446, 1720]
Zeta-OnPolONS	750	4243	[4002, 7489]	5490	[4752, 8536]
Zeta-area-OnPolONS	750	3873	[3286, 4709]	3960	[3353, 5280]
ER	1000	4885		[4098, 6889]	
Def	1000	2937	[2646, 3397]	5364	[4476, 7255]
Def-OnPol	1000	4986	[4284, 6826]	5199	[4505, 7678]
Def-ONS	1000	3877	[3405, 5364]	6853	[5302, 10313]
Def-area	1000	1738	[1605, 1867]	2167	[1955, 2428]
Def-area-OnPol	1000	1942	[1810, 2146]	2362	[2112, 2645]
Def-area-ONS	1000	2310	[2047, 2596]	2643	[2418, 3029]
Full	1000	3569	[3102, 4150]	4744	[4086, 6037]

(To be continued)

TABLE A.8: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the CE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Full-area	1000	1940	[1813, 2057]	2290	[2087, 2523]
Zeta-OnPolONS	1000	4929	[4419, 7780]	6044	[5200, 10341]
Zeta-area-OnPolONS	1000	3867	[3291, 5034]	6036	[4806, 8585]

TABLE A.9: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the LKE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
ER	250	3043		[2654, 3406]	
Def	250	4101	[3722, 5411]	4369	[3893, 5393]
Def-OnPol	250	4504	[3786, 5925]	3777	[3217, 4476]
Def-ONS	250	3069	[2829, 3693]	3232	[2942, 3789]
Def-area	250	5868	[5146, 9623]	5910	[4709, 8591]
Def-area-OnPol	250	3829	[3484, 4628]	3899	[3417, 4849]
Def-area-ONS	250	2928	[2672, 3428]	3213	[2948, 3597]
Full	250	5950	[5243, 9720]	6418	[5341, 10451]
Full-area	250	*	*	*	*
Zeta-OnPolONS	250	2978	[2546, 3362]	2862	[2607, 3253]
Zeta-area-OnPolONS	250	2860	[2556, 3219]	2833	[2629, 3267]
ER	500	3183		[2832, 3721]	
Def	500	4173	[3718, 5386]	5252	[4317, 6830]
Def-OnPol	500	4219	[3666, 4979]	4272	[3779, 5681]
Def-ONS	500	2812	[2557, 3303]	3170	[2882, 3651]
Def-area	500	5619	[4771, 7979]	5910	[4848, 8239]
Def-area-OnPol	500	3626	[3307, 4423]	3917	[3622, 4652]
Def-area-ONS	500	3129	[2835, 3484]	3163	[2852, 3643]
Full	500	5703	[4914, 8602]	6700	[5685, 11595]
Full-area	500	*	*	*	*

(To be continued)

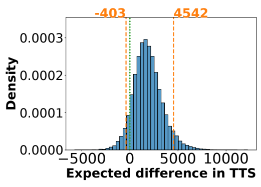
TABLE A.9: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the LKE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
Zeta-OnPolONS	500	2910	[2645, 3198]	2984	[2671, 3336]
Zeta-area-OnPolONS	500	3240	[2978, 3802]	2992	[2790, 3479]
ER	750	2983		[2676, 3443]	
Def	750	4801	[4101, 6479]	4682	[4254, 6103]
Def-OnPol	750	4603	[3968, 5712]	4085	[3788, 5365]
Def-ONS	750	2971	[2714, 3260]	2739	[2528, 3051]
Def-area	750	4932	[4442, 6680]	4854	[4449, 6285]
Def-area-OnPol	750	3991	[3535, 4944]	3589	[3181, 4299]
Def-area-ONS	750	3097	[2877, 3678]	3397	[3119, 3778]
Full	750	5400	[4918, 8539]	5317	[4787, 7716]
Full-area	750	*	*	*	*
Zeta-OnPolONS	750	3238	[2938, 3685]	3086	[2847, 3442]
Zeta-area-OnPolONS	750	3197	[2979, 3784]	3106	[2824, 3503]
ER	1000	3240		[3033, 3695]	
Def	1000	5209	[4490, 7100]	6674	[5075, 9679]
Def-OnPol	1000	3691	[3422, 4264]	3679	[3412, 4320]
Def-ONS	1000	2993	[2770, 3464]	3119	[2924, 3437]
Def-area	1000	4507	[4092, 5425]	5923	[5079, 9014]
Def-area-OnPol	1000	4127	[3768, 5013]	4077	[3758, 4803]
Def-area-ONS	1000	3731	[3440, 4277]	3652	[3464, 4045]
Full	1000	*	*	6579	[5916, 12689]
Full-area	1000	*	*	*	*
Zeta-OnPolONS	1000	2854	[2619, 3258]	3555	[3215, 4018]
Zeta-area-OnPolONS	1000	3416	[3109, 3867]	3446	[3251, 3898]

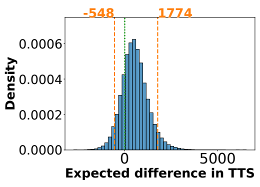
A.4 Probability Distributions

Density plots of the posterior distribution of $\mu_{\mathrm{ER}} - \mu_{\mathrm{IER}}$ for all IER configurations that the gamma-distribution-based model is applicable to. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent. Only successful runs were taken into account.

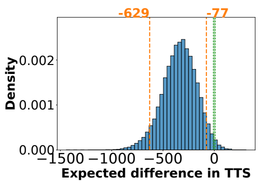
A.4.1 VE state-encoding



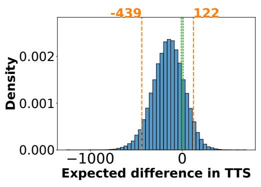
(A) Def 250 LS



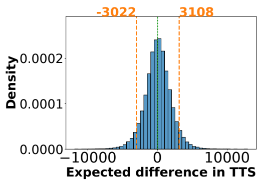
(B) Def 500 LS



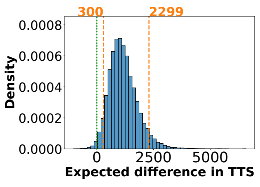
(C) Def 750 LS



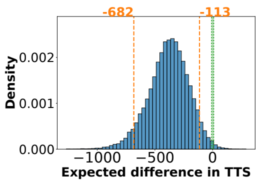
(D) Def 1000 LS



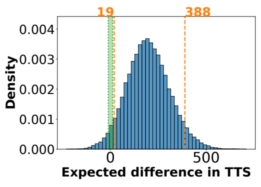
(E) Def 250 PD



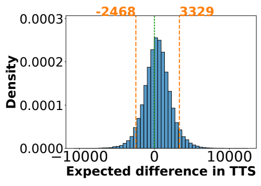
(F) Def 500 PD



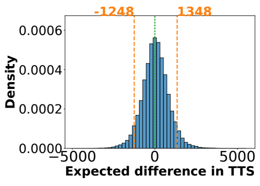
(G) Def 750 PD



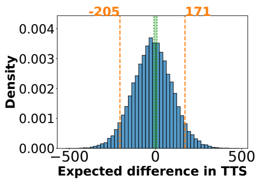
(H) Def 1000 PD



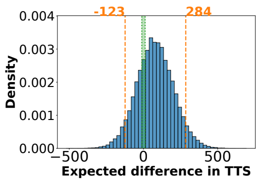
(I) Def-OnPol 250 LS



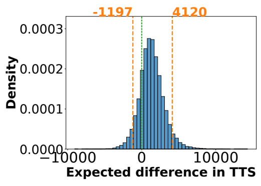
(J) Def-OnPol 500 LS



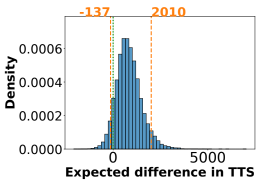
(K) Def-OnPol 750 LS



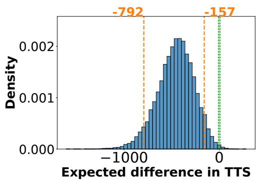
(L) Def-OnPol 1000 LS



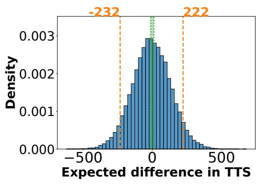
(M) Def-OnPol 250 PD



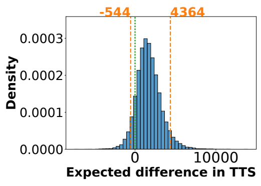
(N) Def-OnPol 500 PD



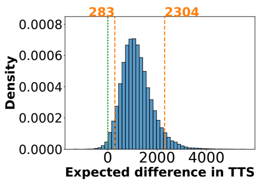
(O) Def-OnPol 750 PD



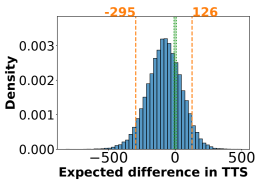
(P) Def-OnPol 1000 PD



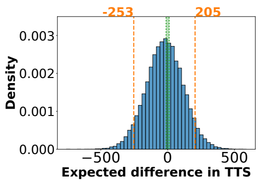
(Q) Def-ONS 250 LS



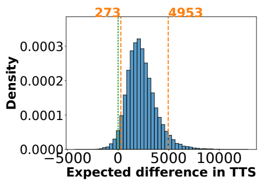
(R) Def-ONS 500 LS



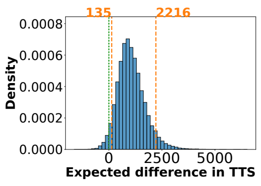
(S) Def-ONS 750 LS



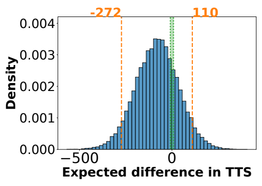
(T) Def-ONS 1000 LS



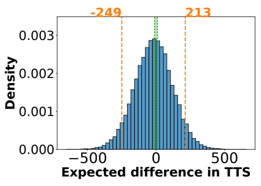
(U) Def-ONS 250 PD



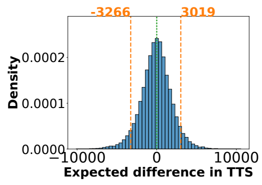
(V) Def-ONS 500 PD



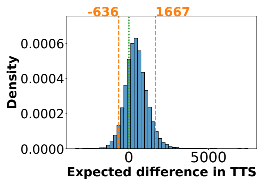
(W) Def-ONS 750 PD



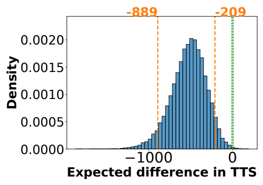
(X) Def-ONS 1000 PD



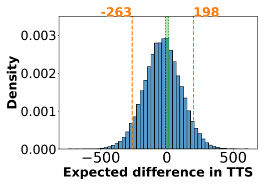
(Y) Def-area 250 LS



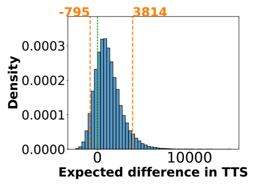
(Z) Def-area 500 LS



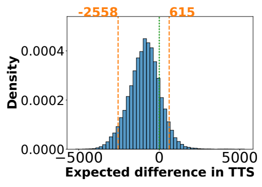
(AA) Def-area 750 LS



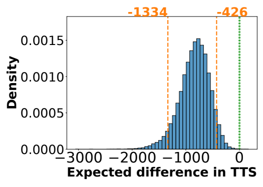
(AB) Def-area 1000 LS



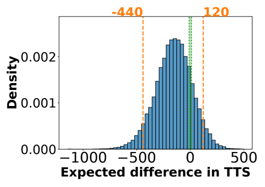
(AC) Def-area 250 PD



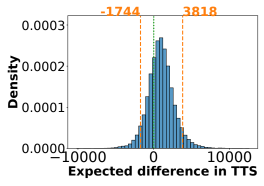
(AD) Def-area 500 PD



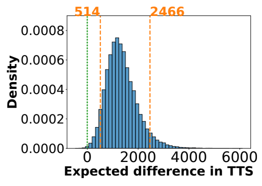
(AE) Def-area 750 PD



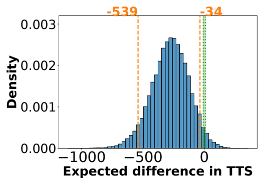
(AF) Def-area 1000 PD



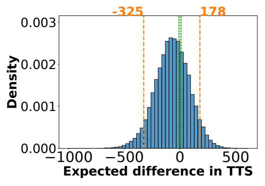
(AG) Def-area-OnPol 250 LS



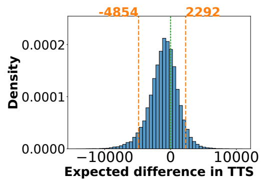
(AH) Def-area-OnPol 500 LS



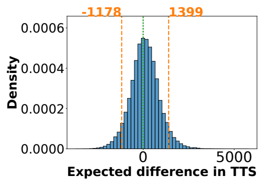
(AI) Def-area-OnPol 750 LS



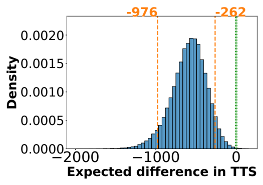
(AJ) Def-area-OnPol 1000 LS



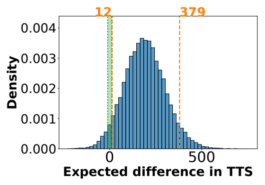
(AK) Def-area-OnPol 250 PD



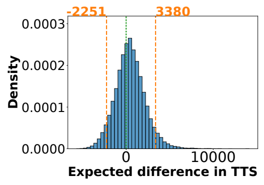
(AL) Def-area-OnPol 500 PD



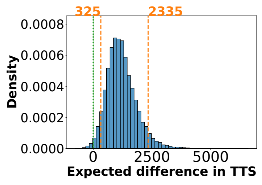
(AM) Def-area-OnPol 750 PD



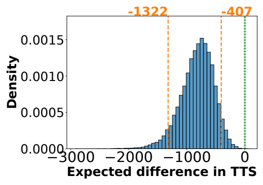
(AN) Def-area-OnPol 1000 PD



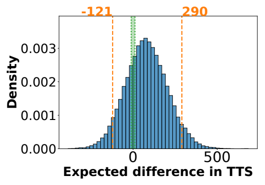
(AO) Def-area-ONS 250 LS



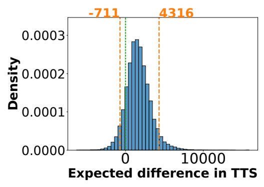
(AP) Def-area-ONS 500 LS



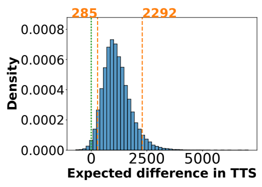
(AQ) Def-area-ONS 750 LS



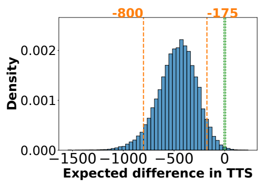
(AR) Def-area-ONS 1000 LS



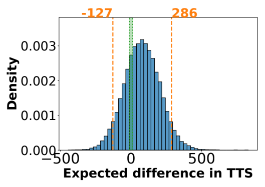
(AS) Def-area-ONS 250 PD



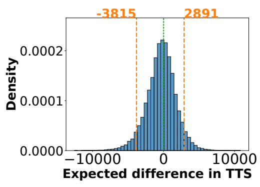
(AT) Def-area-ONS 500 PD



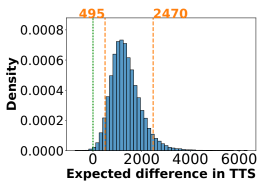
(AU) Def-area-ONS 750 PD



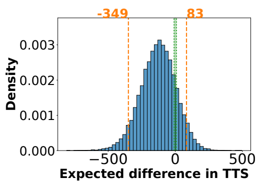
(AV) Def-area-ONS 1000 PD



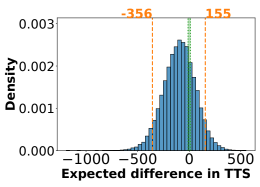
(AW) Full 250 LS



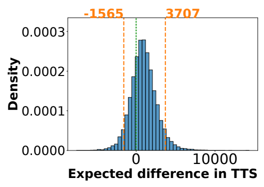
(AX) Full 500 LS



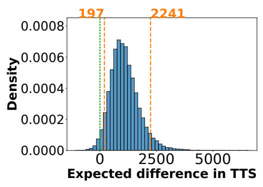
(AY) Full 750 LS



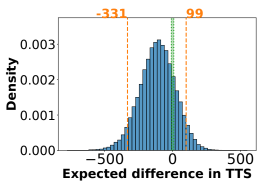
(AZ) Full 1000 LS



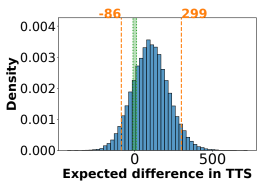
(BA) Full 250 PD



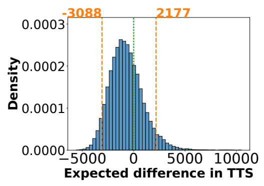
(BB) Full 500 PD



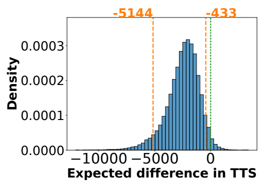
(BC) Full 750 PD



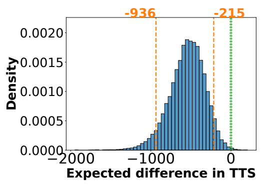
(BD) Full 1000 PD



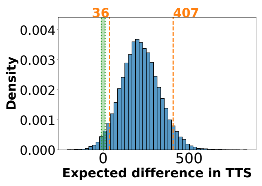
(BE) Full-area 250 LS



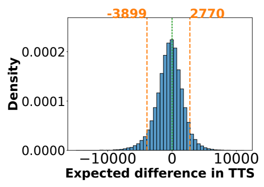
(BF) Full-area 500 LS



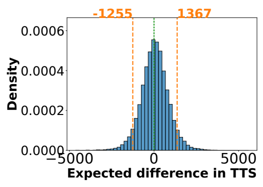
(BG) Full-area 750 LS



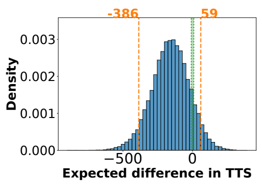
(BH) Full-area 1000 LS



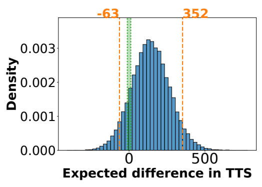
(BI) Full-area 250 PD



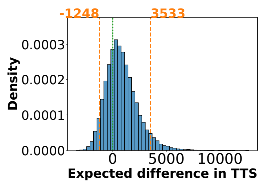
(BJ) Full-area 500 PD



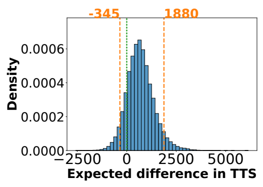
(BK) Full-area 750 PD



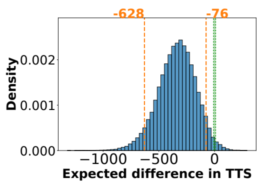
(BL) Full-area 1000 PD



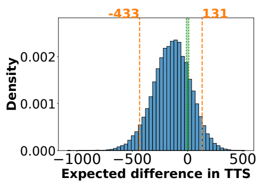
(BM) Zeta 250 LS



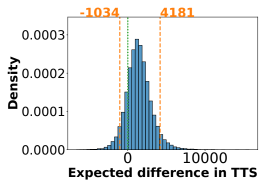
(BN) Zeta 500 LS



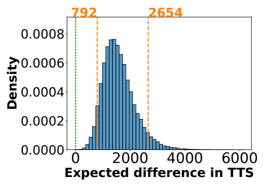
(BO) Zeta 750 LS



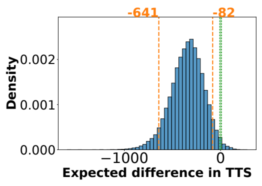
(BP) Zeta 1000 LS



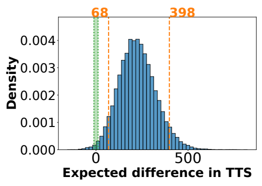
(BQ) Zeta 250 PD



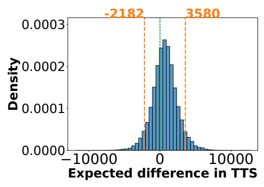
(BR) Zeta 500 PD



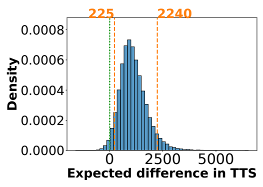
(BS) Zeta 750 PD



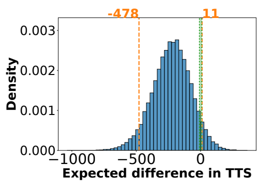
(BT) Zeta 1000 PD



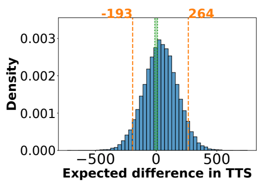
(BU) Zeta-area 250 LS



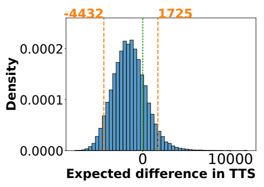
(BV) Zeta-area 500 LS



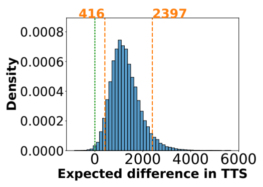
(BW) Zeta-area 750 LS



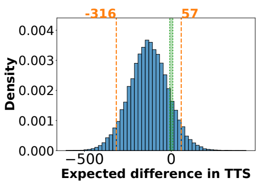
(BX) Zeta-area 1000 LS



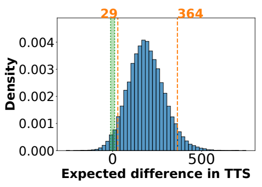
(BY) Zeta-area 250 PD



(BZ) Zeta-area 500 PD

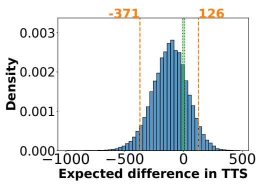


(CA) Zeta-area 750 PD

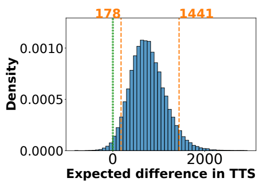


(CB) Zeta-area 1000 PD

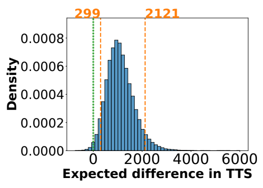
A.4.2 CE state-encoding



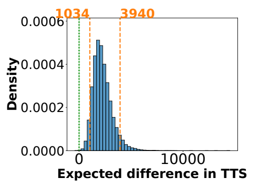
(A) Def 250 LS



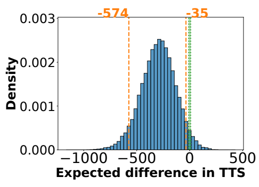
(B) Def 500 LS



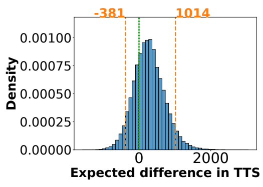
(C) Def 750 LS



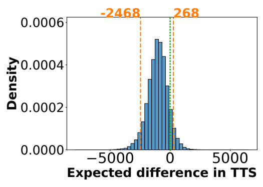
(D) Def 1000 LS



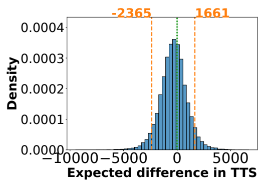
(E) Def 250 PD



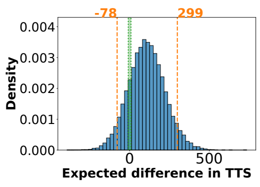
(F) Def 500 PD



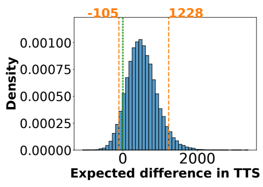
(G) Def 750 PD



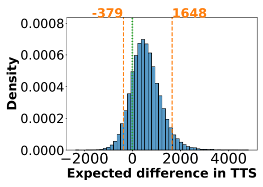
(H) Def 1000 PD



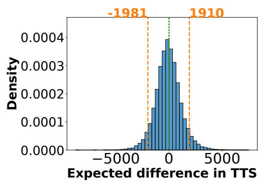
(I) Def-OnPol 250 LS



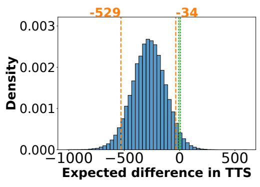
(J) Def-OnPol 500 LS



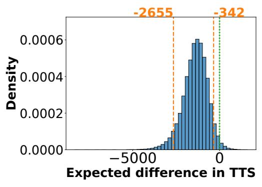
(K) Def-OnPol 750 LS



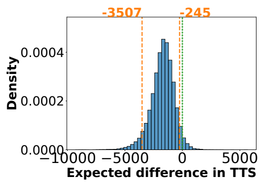
(L) Def-OnPol 1000 LS



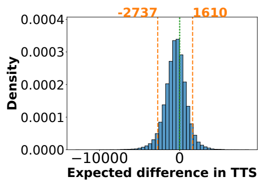
(M) Def-OnPol 250 PD



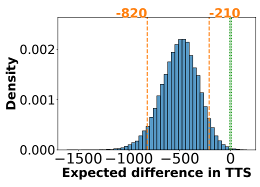
(N) Def-OnPol 500 PD



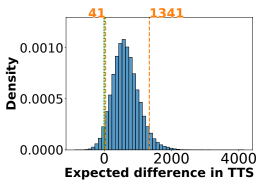
(O) Def-OnPol 750 PD



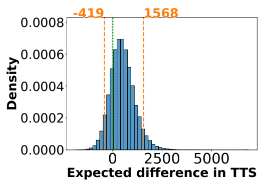
(P) Def-OnPol 1000 PD



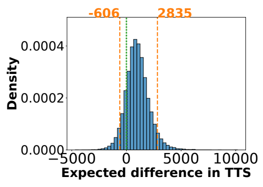
(Q) Def-ONS 250 LS



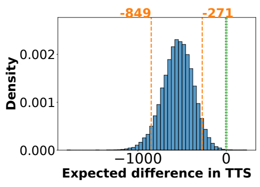
(R) Def-ONS 500 LS



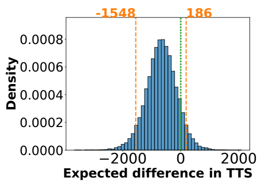
(S) Def-ONS 750 LS



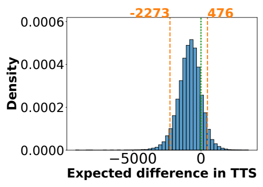
(T) Def-ONS 1000 LS



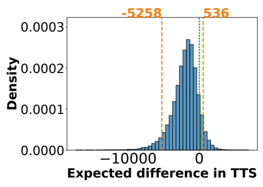
(U) Def-ONS 250 PD



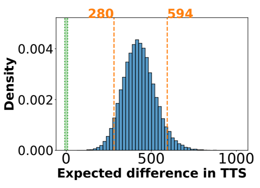
(V) Def-ONS 500 PD



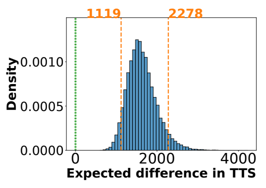
(W) Def-ONS 750 PD



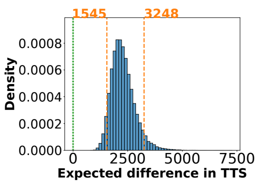
(X) Def-ONS 1000 PD



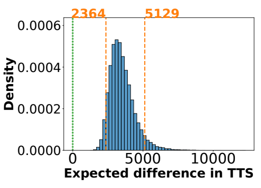
(Y) Def-area 250 LS



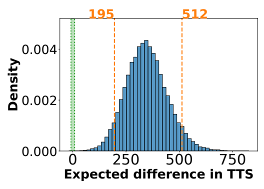
(Z) Def-area 500 LS



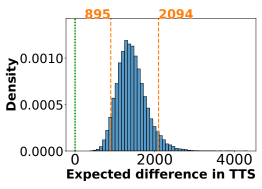
(AA) Def-area 750 LS



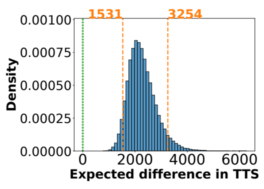
(AB) Def-area 1000 LS



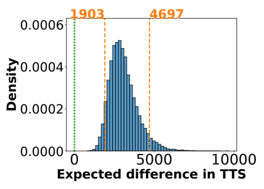
(AC) Def-area 250 PD



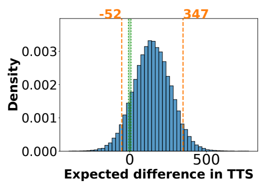
(AD) Def-area 500 PD



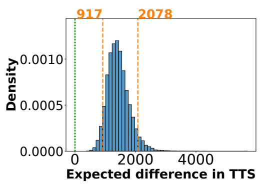
(AE) Def-area 750 PD



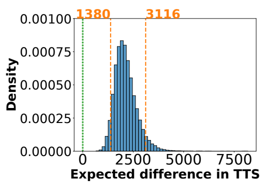
(AF) Def-area 1000 PD



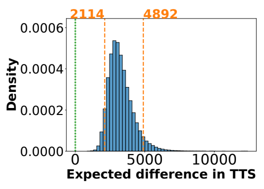
(AG) Def-area-OnPol 250 LS



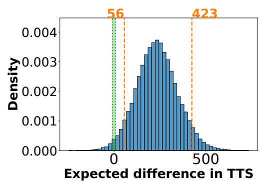
(AH) Def-area-OnPol 500 LS



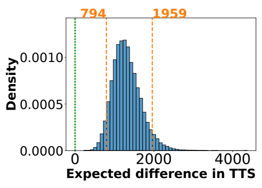
(AI) Def-area-OnPol 750 LS



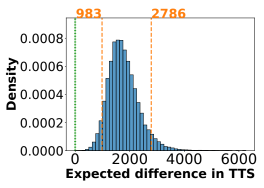
(AJ) Def-area-OnPol 1000 LS



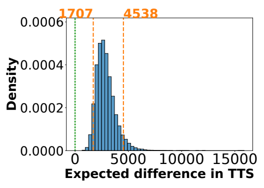
(AK) Def-area-OnPol 250 PD



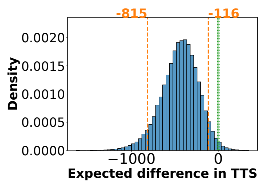
(AL) Def-area-OnPol 500 PD



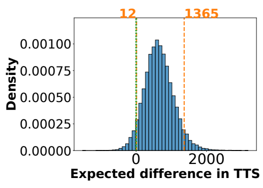
(AM) Def-area-OnPol 750 PD



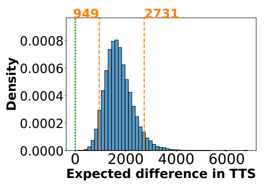
(AN) Def-area-OnPol 1000 PD



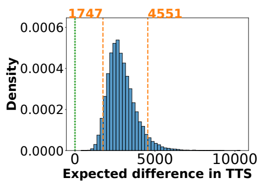
(AO) Def-area-ONS 250 LS



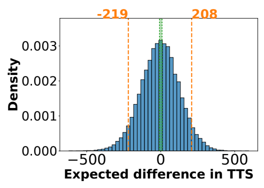
(AP) Def-area-ONS 500 LS



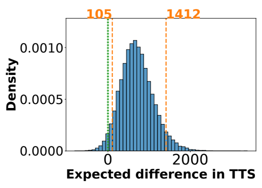
(AQ) Def-area-ONS 750 LS



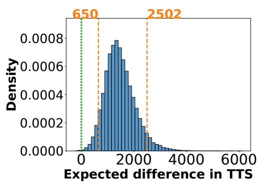
(AR) Def-area-ONS 1000 LS



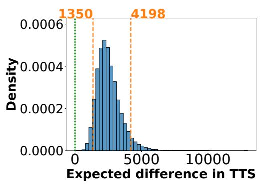
(AS) Def-area-ONS 250 PD



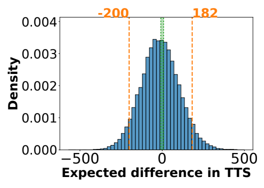
(AT) Def-area-ONS 500 PD



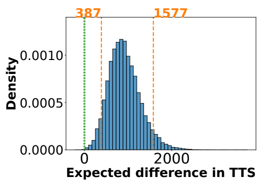
(AU) Def-area-ONS 750 PD



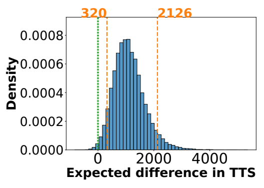
(AV) Def-area-ONS 1000 PD



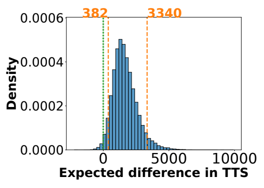
(AW) Full 250 LS



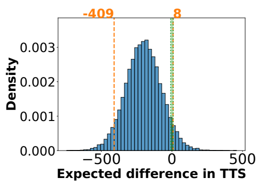
(AX) Full 500 LS



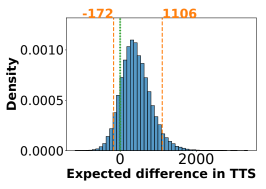
(AY) Full 750 LS



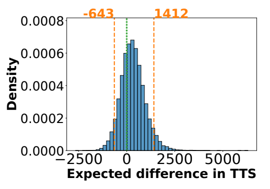
(AZ) Full 1000 LS



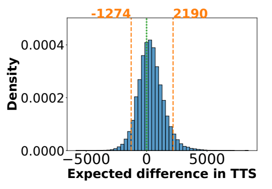
(BA) Full 250 PD



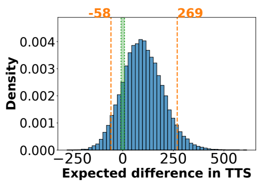
(BB) Full 500 PD



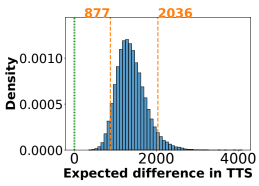
(BC) Full 750 PD



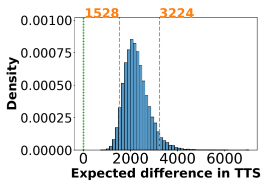
(BD) Full 1000 PD



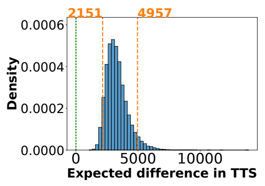
(BE) Full-area 250 LS



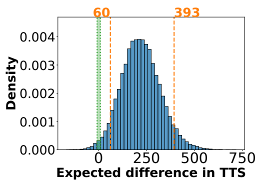
(BF) Full-area 500 LS



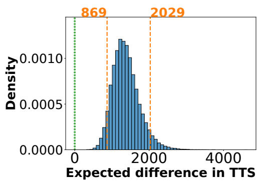
(BG) Full-area 750 LS



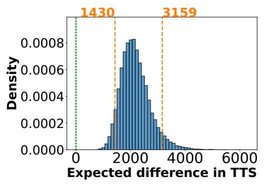
(BH) Full-area 1000 LS



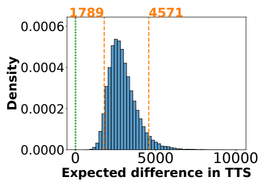
(BI) Full-area 250 PD



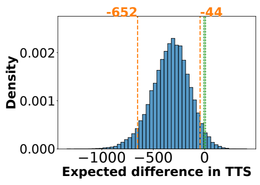
(BJ) Full-area 500 PD



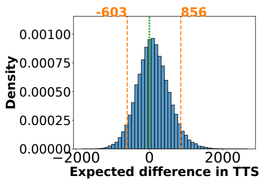
(BK) Full-area 750 PD



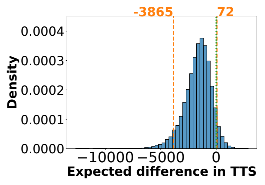
(BL) Full-area 1000 PD



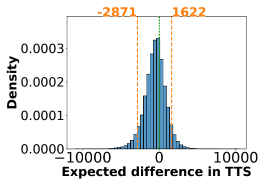
(BM) Zeta 250 LS



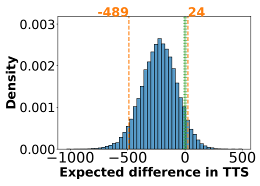
(BN) Zeta 500 LS



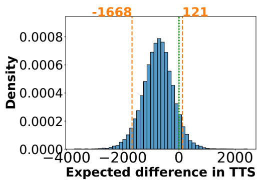
(BO) Zeta 750 LS



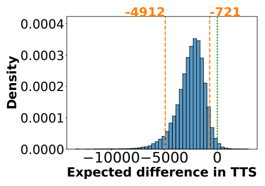
(BP) Zeta 1000 LS



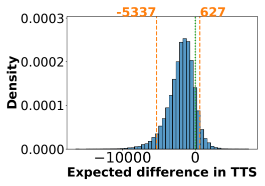
(BQ) Zeta 250 PD



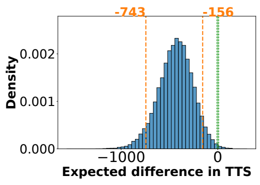
(BR) Zeta 500 PD



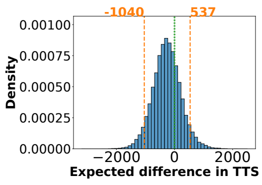
(BS) Zeta 750 PD



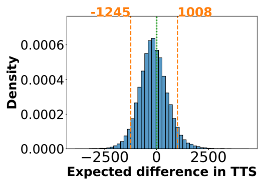
(BT) Zeta 1000 PD



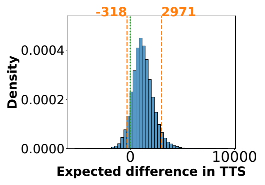
(BU) Zeta-area 250 LS



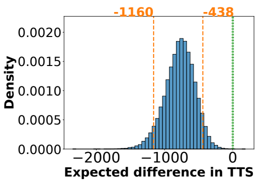
(BV) Zeta-area 500 LS



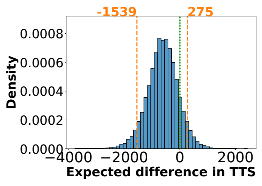
(BW) Zeta-area 750 LS



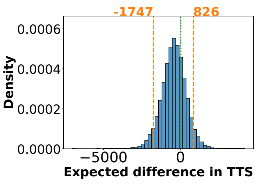
(BX) Zeta-area 1000 LS



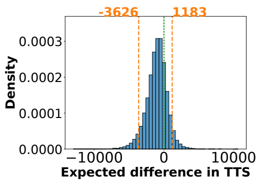
(BY) Zeta-area 250 PD



(BZ) Zeta-area 500 PD

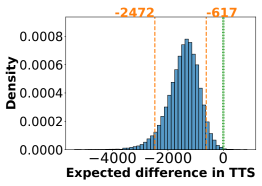


(CA) Zeta-area 750 PD

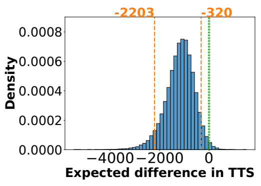


(CB) Zeta-area 1000 PD

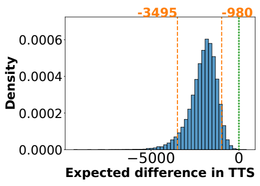
A.4.3 LKE state-encoding



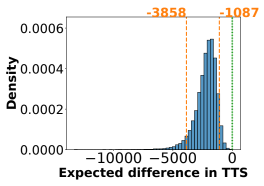
(A) Def 250 LS



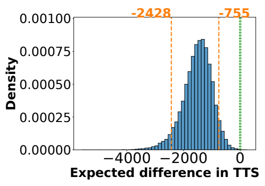
(B) Def 500 LS



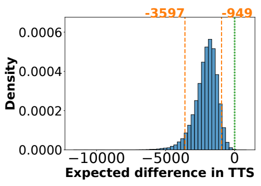
(C) Def 750 LS



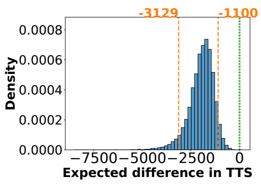
(D) Def 1000 LS



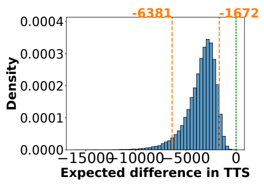
(E) Def 250 PD



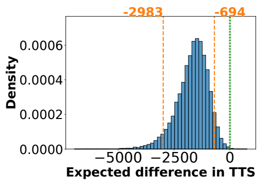
(F) Def 500 PD



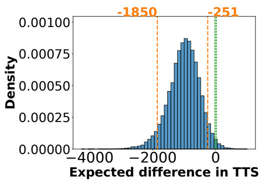
(G) Def 750 PD



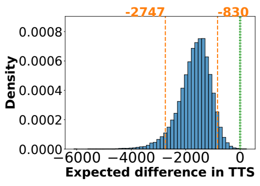
(H) Def 1000 PD



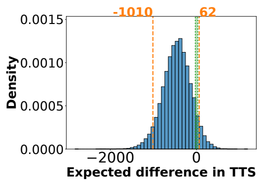
(I) Def-OnPol 250 LS



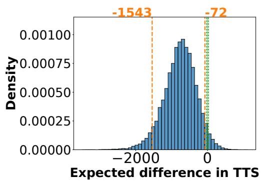
(J) Def-OnPol 500 LS



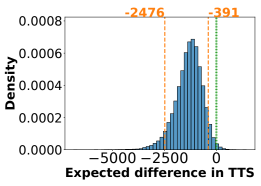
(K) Def-OnPol 750 LS



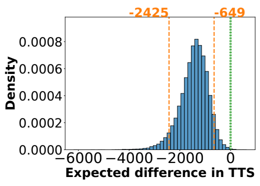
(L) Def-OnPol 1000 LS



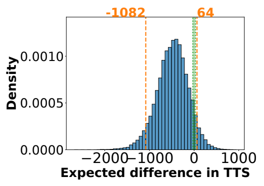
(M) Def-OnPol 250 PD



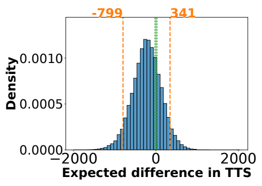
(N) Def-OnPol 500 PD



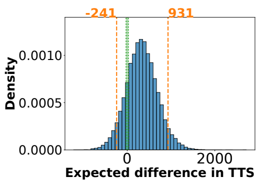
(O) Def-OnPol 750 PD



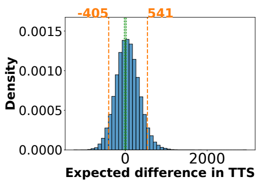
(P) Def-OnPol 1000 PD



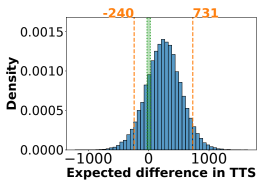
(Q) Def-ONS 250 LS



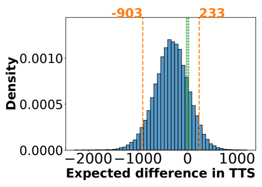
(R) Def-ONS 500 LS



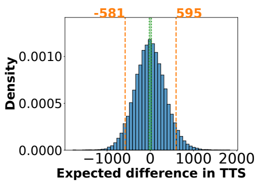
(S) Def-ONS 750 LS



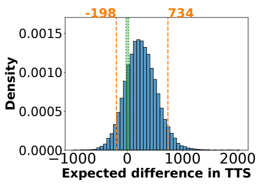
(T) Def-ONS 1000 LS



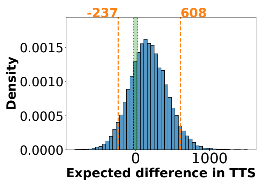
(U) Def-ONS 250 PD



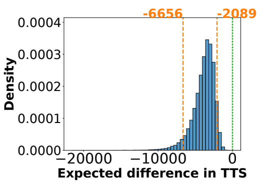
(V) Def-ONS 500 PD



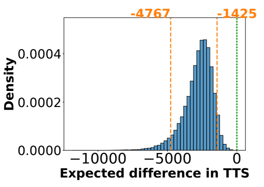
(W) Def-ONS 750 PD



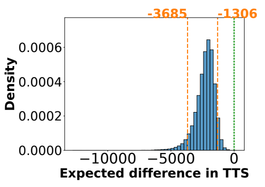
(X) Def-ONS 1000 PD



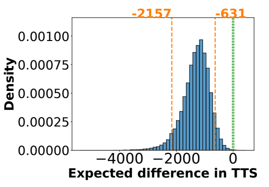
(Y) Def-area 250 LS



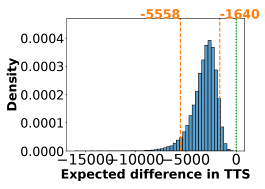
(Z) Def-area 500 LS



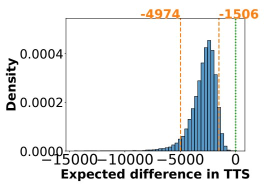
(AA) Def-area 750 LS



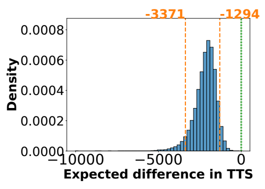
(AB) Def-area 1000 LS



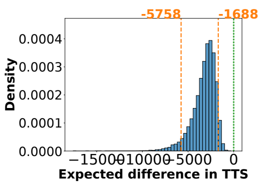
(AC) Def-area 250 PD



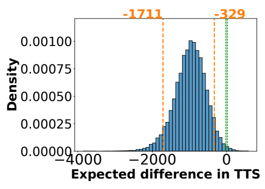
(AD) Def-area 500 PD



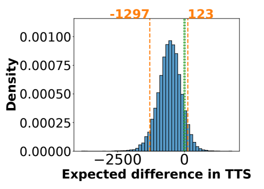
(AE) Def-area 750 PD



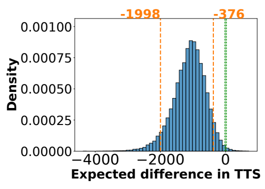
(AF) Def-area 1000 PD



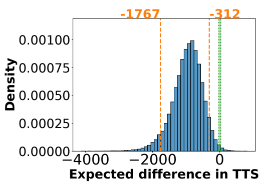
(AG) Def-area-OnPol 250 LS



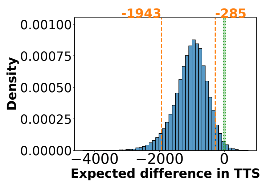
(AH) Def-area-OnPol 500 LS



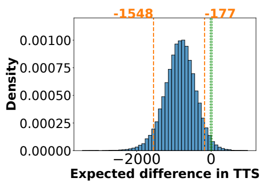
(AI) Def-area-OnPol 750 LS



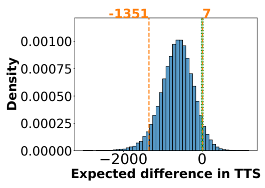
(AJ) Def-area-OnPol 1000 LS



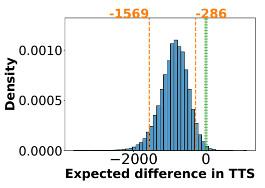
(AK) Def-area-OnPol 250 PD



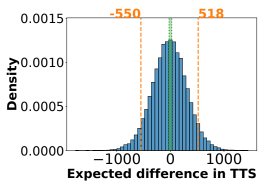
(AL) Def-area-OnPol 500 PD



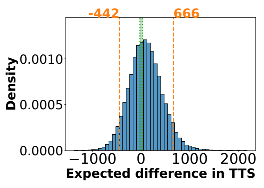
(AM) Def-area-OnPol 750 PD



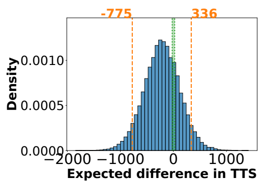
(AN) Def-area-OnPol 1000 PD



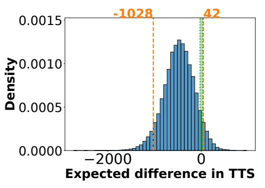
(AO) Def-area-ONS 250 LS



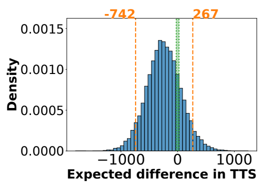
(AP) Def-area-ONS 500 LS



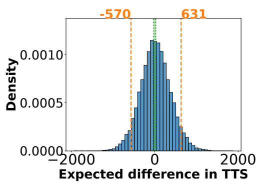
(AQ) Def-area-ONS 750 LS



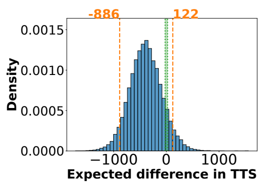
(AR) Def-area-ONS 1000 LS



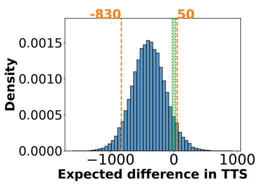
(AS) Def-area-ONS 250 PD



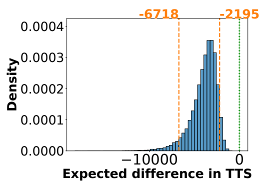
(AT) Def-area-ONS 500 PD



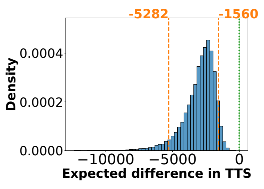
(AU) Def-area-ONS 750 PD



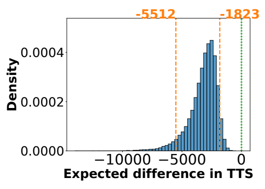
(AV) Def-area-ONS 1000 PD



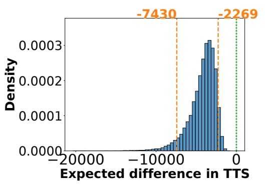
(AW) Full 250 LS



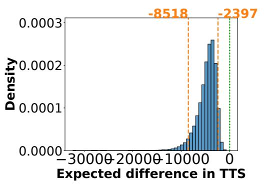
(AX) Full 500 LS



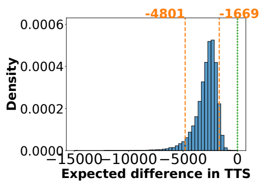
(AY) Full 750 LS



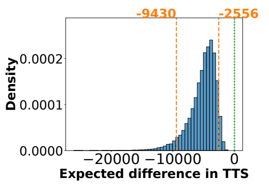
(AZ) Full 250 PD



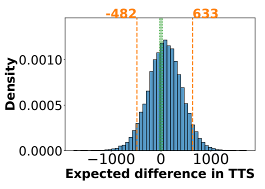
(BA) Full 500 PD



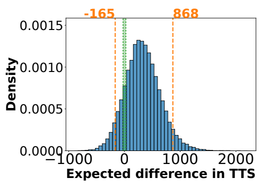
(BB) Full 750 PD



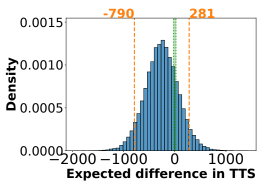
(BC) Full 1000 PD



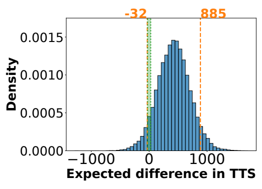
(BD) Zeta 250 LS



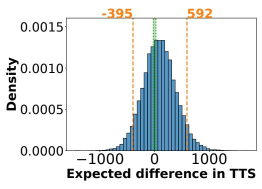
(BE) Zeta 500 LS



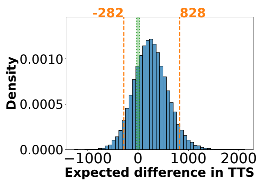
(BF) Zeta 750 LS



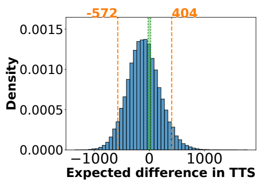
(BG) Zeta 1000 LS



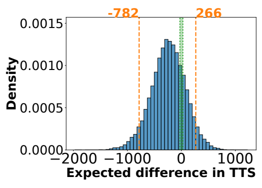
(BH) Zeta 250 PD



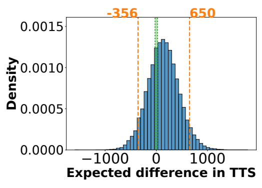
(BI) Zeta 500 PD



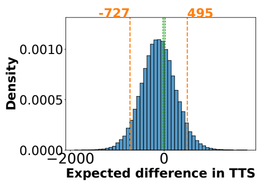
(BJ) Zeta 750 PD



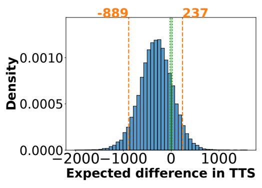
(BK) Zeta 1000 PD



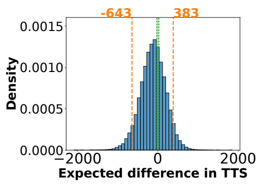
(BL) Zeta-area 250 LS



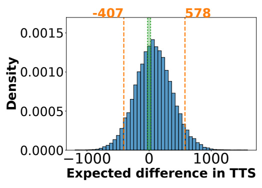
(BM) Zeta-area 500 LS



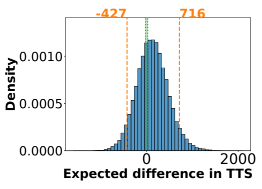
(BN) Zeta-area 750 LS



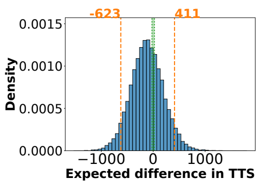
(BO) Zeta-area 1000 LS



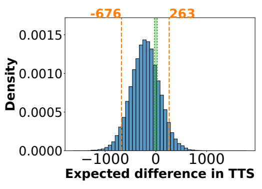
(BP) Zeta-area 250 PD



(BQ) Zeta-area 500 PD



(BR) Zeta-area 750 PD



(BS) Zeta-area 1000 PD

Appendix B

Evaluation Results from IER for Continuous Environments

B.1 Solution Rates, Probabilities and TTS

TABLE B.1: Probability (in %) for each IER configuration in the VE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**
Def	250	LS	9	28
Def	250	PD	3	1
Def	500	LS	84	93
Def	500	PD	33	76
Def	750	LS	93	41
Def	750	PD	19	41
Def	1000	LS	94	94
Def	1000	PD	91	91
Def-zeta	250	LS	0	9
Def-zeta	250	PD	6	4
Def-zeta	500	LS	41	93
Def-zeta	500	PD	25	93
Def-zeta	750	LS	32	32
Def-zeta	750	PD	41	13
Def-zeta	1000	LS	87	86
Def-zeta	1000	PD	97	96
Full	250	LS	3	2
Full	250	PD	0	1
Full	500	LS	68	42
Full	500	PD	4	67

(To be continued)

TABLE B.1: Probability (in %) for each IER configuration in the VE state-encoding that the solution rate of IER is greater than the solution rate of ER. A * indicates that the query function was not evaluated for the corresponding configuration.

Config	t_{expl}	**Query Function**	**Vanilla**	**OnPol**
Full	750	LS	9	33
Full	750	PD	59	50
Full	1000	LS	58	87
Full	1000	PD	75	59

TABLE B.2: Probabilities (in %, rounded to two decimal places) for each configuration in the MountainCar problem of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. Bold entries indicate a probability greater than 50% for IER performing better than ER.

Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def	250	LS	94.45	0.29	5.27
Def	250	PD	98.79	0.07	1.13
Def	**500**	**LS**	8.91	0.97	**90.12**
Def	500	PD	63.01	1.59	35.39
Def	**750**	**LS**	3.60	0.53	**95.86**
Def	750	PD	81.61	0.95	17.44
Def	**1000**	**LS**	3.92	0.49	**95.58**
Def	**1000**	**PD**	8.44	0.82	**90.74**
Def-OnPol	250	LS	75.54	1.04	23.42
Def-OnPol	250	PD	99.68	0.01	0.31
Def-OnPol	**500**	**LS**	1.82	0.30	**97.88**
Def-OnPol	**500**	**PD**	12.41	1.21	**86.38**
Def-OnPol	750	LS	57.70	1.65	40.64
Def-OnPol	750	PD	51.81	1.57	46.62
Def-OnPol	**1000**	**LS**	4.67	0.56	**94.77**
Def-OnPol	**1000**	**PD**	9.32	0.89	**89.79**
Def-zeta	250	LS	99.89	0.00	0.11
Def-zeta	250	PD	98.25	0.12	1.64
Def-zeta	**500**	**LS**	42.44	1.84	**55.72**
Def-zeta	500	PD	78.89	1.24	19.87
Def-zeta	750	LS	71.13	1.31	27.56
Def-zeta	750	PD	58.12	1.63	40.25

(To be continued)

TABLE B.2: Probabilities (in %, rounded to two decimal places) for each configuration in the MountainCar problem of IER performing better or worse in terms of TTS than ER—or practically equivalent (with respect to a rope). Only taken into account runs that finished within 4000 episodes. Bold entries indicate a probability greater than 50% for IER performing better than ER.

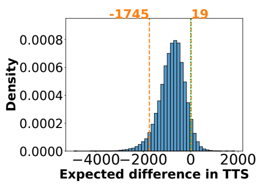
Config	t_{expl}	**Query Function**	**ER better**	**practical equivalent**	**IER better**
Def-zeta	**1000**	**LS**	7.40	0.63	**91.97**
Def-zeta	**1000**	**PD**	2.51	0.30	**97.18**
Def-zeta-OnPol	250	LS	93.83	0.32	5.85
Def-zeta-OnPol	250	PD	99.02	0.07	0.90
Def-zeta-OnPol	**500**	**LS**	3.64	0.57	**95.78**
Def-zeta-OnPol	**500**	**PD**	3.21	0.50	**96.29**
Def-zeta-OnPol	750	LS	65.73	1.47	32.80
Def-zeta-OnPol	750	PD	86.61	0.74	12.65
Def-zeta-OnPol	**1000**	**LS**	11.74	0.90	**87.36**
Def-zeta-OnPol	**1000**	**PD**	3.49	0.41	**96.10**
Full	250	LS	98.47	0.09	1.44
Full	250	PD	99.99	0.00	0.01
Full	**500**	**LS**	18.58	1.56	**79.86**
Full	500	PD	96.00	0.26	3.73
Full	750	LS	91.32	0.54	8.14
Full	750	PD	49.11	1.88	49.02
Full	**1000**	**LS**	41.85	1.73	**56.43**
Full	**1000**	**PD**	26.32	1.51	**72.17**
Full-OnPol	250	LS	99.56	0.02	0.42
Full-OnPol	250	PD	99.73	0.01	0.26
Full-OnPol	**500**	**LS**	44.75	1.75	**53.50**
Full-OnPol	**500**	**PD**	22.07	1.57	**76.36**
Full-OnPol	750	LS	79.42	1.21	19.37
Full-OnPol	750	PD	48.80	1.69	49.50
Full-OnPol	**1000**	**LS**	8.44	0.74	**90.82**
Full-OnPol	**1000**	**PD**	42.89	1.70	**55.41**

TABLE B.3: Most likely values and uncertainties for the mean TTS of IER and ER for all configurations of the VE state-encoding given as the modes and 95 % HDPIs of the estimated distributions of μ_{ER} and μ_{IER}. A * indicates that the statistical model was not computed, because, for the IER configuration not a single one of the runs was able to solve the problem within time.

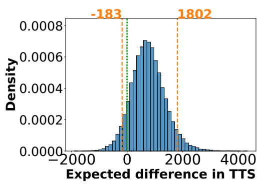
Config	t_{expl}	**LS mean TTS**		**PD mean TTS**	
		Mode	**95 % HPDI**	**Mode**	**95 % HPDI**
ER	250	1688		[1350, 2196]	
Def	250	2340	[1870, 3389]	2792	[2166, 4015]
Def-OnPol	250	1909	[1551, 2596]	2882	[2410, 4746]
Def-zeta	250	3268	[2696, 5540]	2711	[2102, 3740]
Def-zeta-OnPol	250	2145	[1850, 3242]	2769	[2209, 4059]
Full	250	2583	[2129, 3981]	3971	[2987, 5658]
Full-OnPol	250	3124	[2332, 4332]	2989	[2468, 4648]
ER	500	3142		[2642, 4248]	
Def	500	2490	[2068, 3215]	3373	[2759, 4865]
Def-OnPol	500	2240	[1852, 2783]	2484	[2072, 3414]
Def-zeta	500	3072	[2461, 4333]	3722	[3117, 5147]
Def-zeta-OnPol	500	2464	[2018, 2923]	2237	[1953, 2908]
Full	500	2743	[2255, 3534]	4688	[3763, 6932]
Full-OnPol	500	3079	[2451, 4474]	2714	[2311, 3621]
ER	750	2941		[2532, 4041]	
Def	750	2213	[1961, 2792]	3962	[2975, 5250]
Def-OnPol	750	3150	[2680, 4147]	2942	[2498, 4254]
Def-zeta	750	3414	[2839, 4525]	3253	[2685, 4151]
Def-zeta-OnPol	750	3154	[2688, 4611]	3777	[3149, 5362]
Full	750	3889	[3292, 5747]	3244	[2608, 3892]
Full-OnPol	750	3537	[3032, 4668]	3014	[2532, 4033]
ER	1000	3799		[3331, 5465]	
Def	1000	2858	[2591, 3684]	3082	[2662, 4010]
Def-OnPol	1000	2907	[2502, 3794]	3095	[2733, 4039]
Def-zeta	1000	2856	[2558, 3994]	2804	[2482, 3567]
Def-zeta-OnPol	1000	3211	[2750, 4145]	2908	[2507, 3664]
Full	1000	3915	[3430, 4857]	3504	[3042, 4663]
Full-OnPol	1000	3081	[2654, 4030]	3753	[3416, 4916]

B.2 Probability Distribution

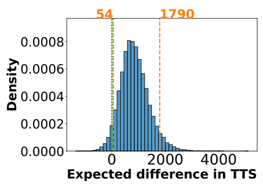
Density plots of the posterior distribution of $\mu_{\text{ER}} - \mu_{\text{IER}}$ for all IER configurations that the gamma-distribution-based model is applicable to. Orange colors indicate the central 95 % HPDI (also known as the 95 % *credible interval*; 95 % of central probability mass lies within these bounds). The green area indicates the rope in which a difference is treated as practical equivalent. Only successful runs were taken into account.



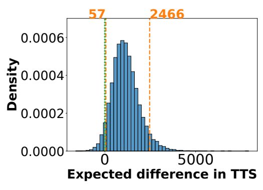
(A) Def 250 LS



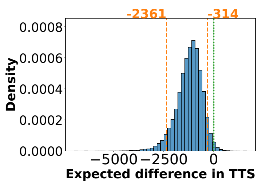
(B) Def 500 LS



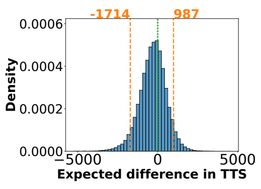
(C) Def 750 LS



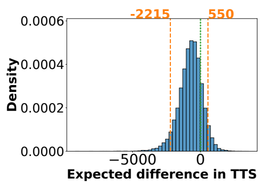
(D) Def 1000 LS



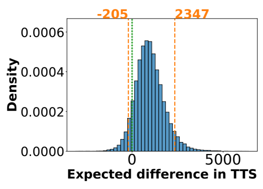
(E) Def 250 PD



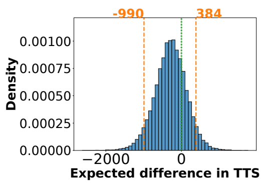
(F) Def 500 PD



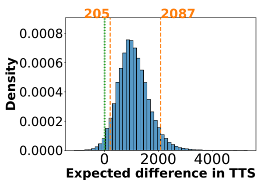
(G) Def 750 PD



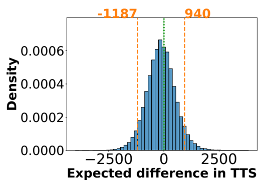
(H) Def 1000 PD



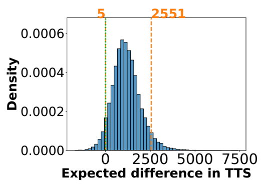
(I) Def-OnPol 250 LS



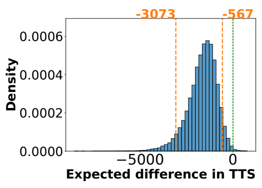
(J) Def-OnPol 500 LS



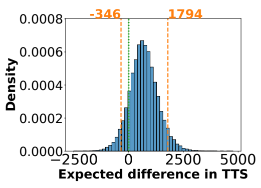
(K) Def-OnPol 750 LS



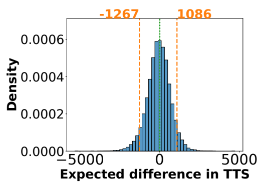
(L) Def-OnPol 1000 LS



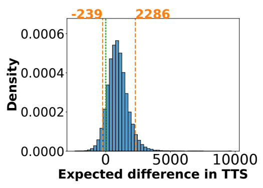
(M) Def-OnPol 250 PD



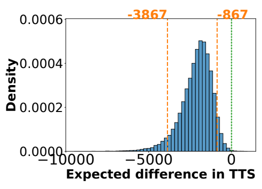
(N) Def-OnPol 500 PD



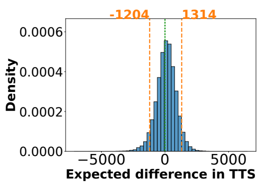
(O) Def-OnPol 750 PD



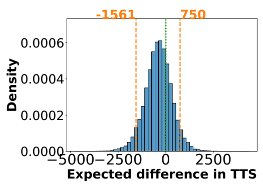
(P) Def-OnPol 1000 PD



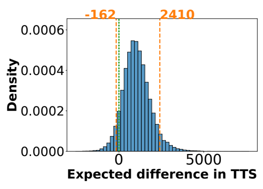
(Q) Def-zeta 250 LS



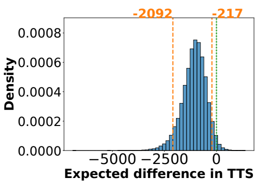
(R) Def-zeta 500 LS



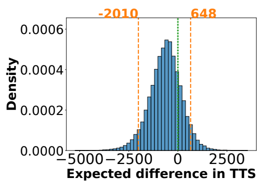
(S) Def-zeta 750 LS



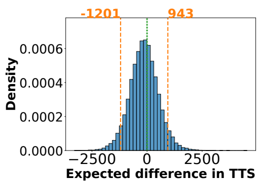
(T) Def-zeta 1000 LS



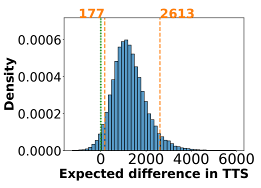
(U) Def-zeta 250 PD



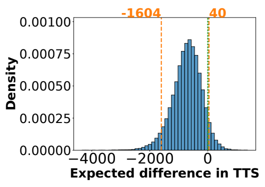
(V) Def-zeta 500 PD



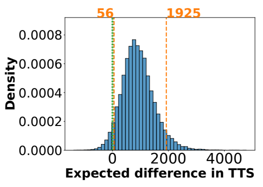
(W) Def-zeta 750 PD



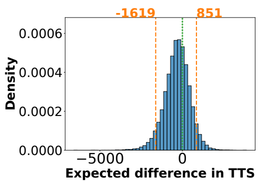
(X) Def-zeta 1000 PD



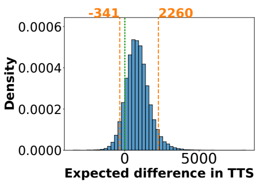
(Y) Def-zeta-OnPol 250 LS



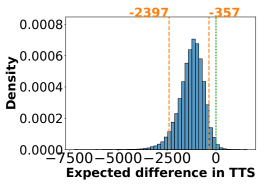
(Z) Def-zeta-OnPol 500 LS



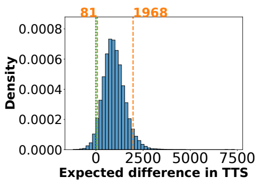
(AA) Def-zeta-OnPol 750 LS



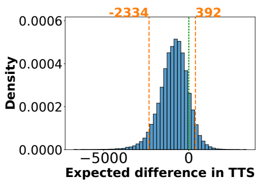
(AB) Def-zeta-OnPol 1000 LS



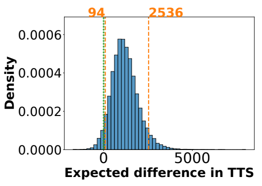
(AC) Def-zeta-OnPol 250 PD



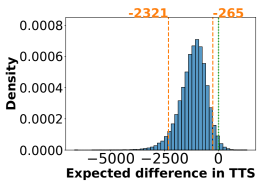
(AD) Def-zeta-OnPol 500 PD



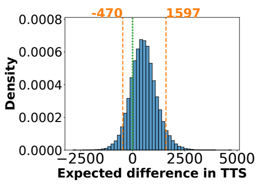
(AE) Def-zeta-OnPol 750 PD



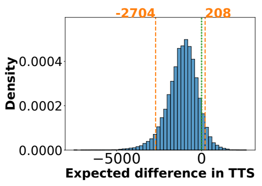
(AF) Def-zeta-OnPol 1000 PD



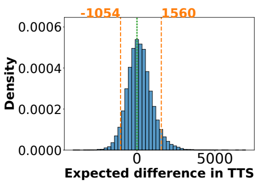
(AG) Full 250 LS



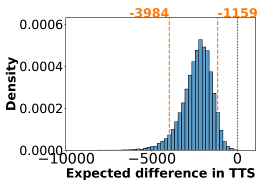
(AH) Full 500 LS



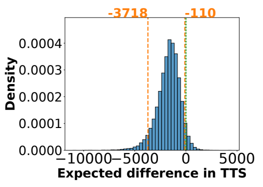
(AI) Full 750 LS



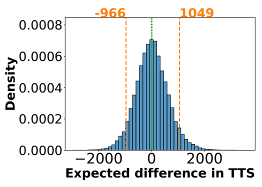
(AJ) Full 1000 LS



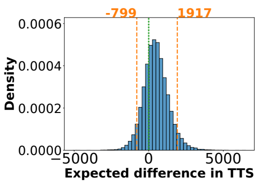
(AK) Full 250 PD



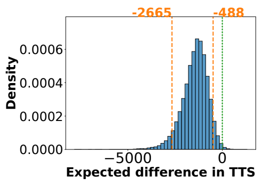
(AL) Full 500 PD



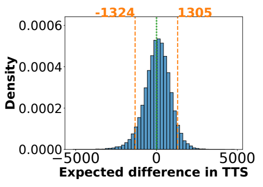
(AM) Full 750 PD



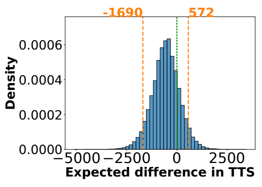
(AN) Full 1000 PD



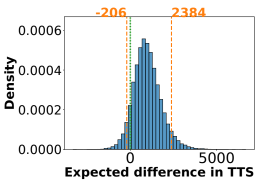
(AO) Full-OnPol 250 LS



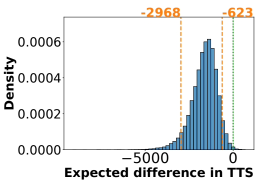
(AP) Full-OnPol 500 LS



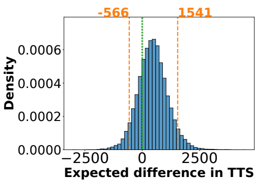
(AQ) Full-OnPol 750 LS



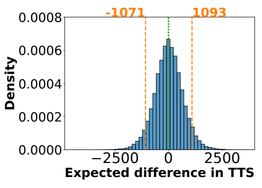
(AR) Full-OnPol 1000 LS



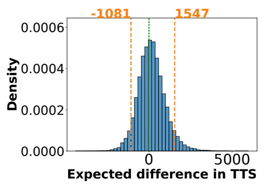
(AS) Full-OnPol 250 PD



(AT) Full-OnPol 500 PD



(AU) Full-OnPol 750 PD



(AV) Full-OnPol 1000 PD

Bibliography

[And+17] Marcin Andrychowicz, Filip Wolski, Alex Ray, et al. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, et al. Vol. 30. Curran Associates, Inc., 2017.

[Ben+17] Alessio Benavoli, Giorgio Corani, Janez Demar, and Marco Zaffalon. “Time for a Change: A Tutorial for Comparing Multiple Classifiers through Bayesian Analysis”. In: *Journal of Machine Learning Research* 18.77 (2017), pp. 1–36.

[Ber+19] Christopher Berner, Greg Brockman, Brooke Chan, et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680.

[BO10] Bernard W Balleine and John P O’Doherty. “Human and Rodent Homologies in Action Control: Corticostriatal Determinants of Goal-Directed and Habitual Action”. In: *Neuropsychopharmacology* 35.1 (Jan. 2010), pp. 48–69. ISSN: 1740-634X. DOI: 10.1038/npp.2009.131.

[Bro+16a] Greg Brockman et al. *OpenAI Gym*. 2016.

[Bro+16b] Greg Brockman, Vicki Cheung, Ludwig Pettersson, et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[BW01] Martin V. Butz and Stewart W. Wilson. “An Algorithmic Description of XCS”. In: *Advances in Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 253–272. ISBN: 978-3-540-44640-8.

[CCD08] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. “Supervised Learning”. In: *Machine learning techniques for multimedia: case studies on organization and retrieval* (2008), pp. 21–49.

[De +15] Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuka. “The Importance of Experience Replay Database Composition in Deep Reinforcement Learning”. In: *Deep Reinforcement Learning Workshop, NIPS*. 2015.

[DR11] Marc Deisenroth and Carl E Rasmussen. “PILCO: A Model-Based and Data-Efficient Approach to Policy Search”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Citeseer, 2011, pp. 465–472.

[FBF77] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. "An Algorithm for Finding Best Matches in Logarithmic Expected Time". In: *ACM Transactions on Mathematical Software (TOMS)* 3.3 (1977), pp. 209–226.

[For+17] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, et al. "Noisy Networks for Exploration". In: *CoRR* abs/1706.10295 (2017). arXiv: `1706.10295`.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[Gu+17] Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, et al. "Interpolated Policy Gradient: Merging on-Policy and off-Policy Gradient Estimation for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3846–3855.

[Hei+23] Michael Heider, David Pätzel, Helena Stegherr, and Jörg Hähner. "A Metaheuristic Perspective on Learning Classifier Systems". In: *Metaheuristics for Machine Learning: New Advances and Tools*. Ed. by Mansour Eddaly, Bassem Jarboui, and Patrick Siarry. Singapore: Springer Nature Singapore, 2023, pp. 73–98. ISBN: 978-981-19388-8-7. DOI: `10.1007/978-981-19-3888-7\_3`.

[Hes+17] Matteo Hessel, Joseph Modayil, Hado van Hasselt, et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. DOI: `10.48550/ARXIV.1710.02298`.

[Jac04] Elin K Jacob. "Classification and Categorization: A Difference That Makes a Difference". In: (2004).

[Kan21] Yuji Kanagawa. *Mujoco-Maze*. 2021.

[Kir+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. "Overcoming Catastrophic Forgetting in Neural Networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[Kru13] John K. Kruschke. "Bayesian Estimation Supersedes the t Test." In: *Journal of Experimental Psychology: General* 142.2 (2013), pp. 573–603. DOI: `10.1037/a0029146`.

[LA14] Sergey Levine and Pieter Abbeel. "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Advances in neural information processing systems* 27 (2014).

[LaV+98] Steven M LaValle et al. "Rapidly-Exploring Random Trees: A New Tool for Path Planning". In: (1998).

[Li19] Yuxi Li. "Reinforcement Learning Applications". In: *CoRR* abs/1908.06973 (2019). arXiv: `1908.06973`.

[Lil+16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, et al. "Continuous Control with Deep Reinforcement Learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[Lin+20] Junfan Lin, Zhongzhan Huang, Keze Wang, et al. *Continuous Transition: Improving Sample Efficiency for Continuous Control Problems via MixUp*. 2020. DOI: 10.48550/ARXIV.2011.14487.

[Lin92] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.3 (May 1992), pp. 293–321. ISSN: 1573-0565. DOI: 10.1007/BF00992699.

[Lin93] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1993.

[LSC19] Su Young Lee, Choi Sungik, and Sae-Young Chung. "Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, et al. Vol. 32. Curran Associates, Inc., 2019.

[MA93] Andrew W Moore and Christopher G Atkeson. "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time". In: *Machine learning* 13.1 (1993), pp. 103–130.

[Mit97] Tom M Mitchell. "Machine Learning. 1997". In: *Burr Ridge, IL: McGraw Hill* 45.37 (1997), pp. 870–877.

[MMO95] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex: Insights from the Successes and Failures of Connectionist Models of Learning and Memory." In: *Psychological review* 102.3 (1995), p. 419.

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013).

[Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[Mni+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, 2016, pp. 1928–1937.

[Moe+20] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. *Model-Based Reinforcement Learning: A Survey*. 2020. DOI: 10.48550/ARXIV.2006.16712.

[MRG+87] James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*. Vol. 2. MIT press, 1987.

[MSU11] Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer. *Organic Computinga Paradigm Shift for Complex Systems*. Springer Science & Business Media, 2011.

[MT17] Christian Müller-Schloer and Sven Tomforde. *Organic Computing - Technical Systems for Survival in the Real World*. Birkhäuser, 2017. ISBN: 978-3-319-68476-5. DOI: 10.1007/978-3-319-68477-2.

[Omo89] Stephen M Omohundro. *Five Balltree Construction Algorithms*. International Computer Science Institute Berkeley, 1989.

[ONe+10] Joseph O'Neill, Barty Pleydell-Bouverie, David Dupret, and Jozsef Csicsvari. "Play It Again: Reactivation of Waking Experience and Memory". In: *Trends in Neurosciences* 33.5 (2010), pp. 220–229. ISSN: 0166-2236. DOI: 10.1016/j.tins.2010.01.006.

[Ope23] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[Pät22] David Pätzel. *Cmpbayes Python Library*. 2022.

[Pen+18] Baolin Peng, Xiujun Li, Jianfeng Gao, et al. "Deep Dyna-q: Integrating Planning for Task-Completion Dialogue Policy Learning". In: *arXiv preprint arXiv:1801.06176* (2018). arXiv: 1801.06176.

[Pil+23] Wenzel Pilar von Pilchau, David Pätzel, Anthony Stein, and Jörg Hähner. "Deep Q-Network Updates for the Full Action-Space Utilizing Synthetic Experiences". In: *2023 International Joint Conference on Neural Networks (IJCNN), to Appear*. 2023.

[Pro+11] Holger Prothmann, Sven Tomforde, Jürgen Branke, et al. "Organic Traffic Control". In: *Organic ComputingA Paradigm Shift for Complex Systems*. Springer, 2011, pp. 431–446.

[PSH20] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Bootstrapping a DQN Replay Memory with Synthetic Experiences". In: *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020), November 2-4, 2020*. Ed. by Juan Julian Merelo, Jonathan Garibaldi, Christian Wagner, et al. 2020. ISBN: 978-989-758-475-6. DOI: 10.5220/0010107904040411.

[PSH21] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Synthetic Experiences for Accelerating DQN Performance in Discrete Non-Deterministic Environments". In: *Algorithms* 14.8 (2021). ISSN: 1999-4893. DOI: 10.3390/a14080226.

[PSH22a] Wenzel Pilar von Pilchau, Anthony Stein, and Jörg Hähner. "Semi-Model-Based Reinforcement Learning in Organic Computing Systems". In: *Architecture of Computing Systems*. Ed. by Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck. Cham: Springer International Publishing, 2022, pp. 241–255. ISBN: 978-3-031-21867-5. DOI: 10.1007/978-3-031-21867-5_16.

[PSH22b] Wenzel Pilar von Pilchau, Anthony Stein., and Jörg Hähner. "Interpolated Experience Replay for Continuous Environments". In: *Proceedings of the 14th International Joint Conference on Computational Intelligence (IJCCI 2022) - NCTA*. SciTePress / INSTICC, 2022, pp. 237–248. ISBN: 978-989-758-611-8. DOI: 10.5220/0011326900003332.

[Row07] Jennifer Rowley. "The Wisdom Hierarchy: Representations of the DIKW Hierarchy". In: *Journal of Information Science* 33.2 (Apr. 2007), pp. 163–180. ISSN: 0165-5515. DOI: 10.1177/0165551506070706.

[San21] Ryan M Sander. "Interpolated Experience Replay for Improved Sample Efficiency of Model-Free Deep Reinforcement Learning Algorithms". PhD thesis. Massachusetts Institute of Technology, 2021.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[Sch+15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized Experience Replay". In: *arXiv e-prints* (Nov. 2015), arXiv:1511.05952.

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347.

[She68] Donald Shepard. "A Two-Dimensional Interpolation Function for Irregularly-Spaced Data". In: *Proceedings of the 1968 23rd ACM National Conference*. ACM '68. New York, NY, USA: Association for Computing Machinery, 1968, pp. 517–524. ISBN: 978-1-4503-7486-6. DOI: 10.1145/800186.810616.

[Sil+17] David Silver, Julian Schrittwieser, Karen Simonyan, et al. "Mastering the Game of Go without Human Knowledge". In: *Nature* 550.7676 (2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270.

[Sil+18] David Silver, Thomas Hubert, Julian Schrittwieser, et al. "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[SK11] HansRudolf Schwarz and Norbert Köckler. "Interpolation und Approximation". In: *Numerische Mathematik*. Vieweg+Teubner Verlag, 2011, pp. 91–182. ISBN: 978-3-8348-1551-4. DOI: 10.1007/978-3-8348-8166-3_4.

[SMG21] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. *S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning*. 2021. DOI: 10.48550/ARXIV.2103.06326.

[Ste+17] Anthony Stein, Dominik Rauh, Sven Tomforde, and Jörg Hähner. "Interpolation in the eXtended Classifier System: An Architectural Perspective". In: *Journal of Systems Architecture* 75 (2017), pp. 79–94.

[Ste+18] Anthony Stein, Sven Tomforde, Ada Diaconescu, Jörg Hähner, and Christian Müller-Schloer. "A Concept for Proactive Knowledge Construction in Self-Learning Autonomous Systems". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 204–213. DOI: 10.1109/FAS-W.2018.00048.

[Ste+20] Anthony Stein, Roland Maier, Lukas Rosenbauer, and Jörg Hähner. "XCS Classifier System with Experience Replay". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 404–413. ISBN: 978-1-4503-7128-5. DOI: 10.1145/3377930.3390249.

[Ste19] Anthony Stein. "Interpolation-Assisted Evolutionary Rule-Based Machine Learning - Strategies to Counter Knowledge Gaps in XCS-Based Self-Learning Adaptive Systems". PhD thesis. Universität Augsburg, 2019.

[Sut+99] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in neural information processing systems* 12 (1999).

[Sut90] Richard S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *Machine Learning Proceedings 1990*. Ed. by Bruce Porter and Raymond Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224. ISBN: 978-1-55860-141-3. DOI: 10.1016/B978-1-55860-141-3.50030-4.

[Sut91] Richard S Sutton. "Dyna, an Integrated Architecture for Learning, Planning, and Reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[TR97] J. N. Tsitsiklis and B. Van Roy. "An Analysis of Temporal-Difference Learning with Function Approximation". In: *IEEE Transactions on Automatic Control* 42.5 (May 1997), pp. 674–690. ISSN: 2334-3303. DOI: 10.1109/9.580874.

[TSM17] Sven Tomforde, Bernhard Sick, and Christian Müller-Schloer. "Organic Computing in the Spotlight". In: *CoRR* abs/1701.08125 (2017).

[vGS16] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (2016). DOI: 10.1609/aaai.v30i1.10295.

[Vin+19] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. 2019.

[VS15] Harm Vanseijen and Rich Sutton. “A Deeper Look at Planning as Learning from Replay”. In: *International Conference on Machine Learning*. PMLR, 2015, pp. 2314–2322.

[Wan+16] Ziyu Wang, Tom Schaul, Matteo Hessel, et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. PMLR, 2016, pp. 1995–2003.

[WB91] Steven D. Whitehead and Dana H. Ballard. “Learning to Perceive and Act by Trial and Error”. In: *Machine Learning* 7.1 (1991), pp. 45–83. ISSN: 1573-0565. DOI: 10.1007/BF00058926.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. “Q-Learning”. In: *Machine Learning* 8.3 (1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698.