Improving Sample-Efficiency for Model-Free Off-Policy Deep Reinforcement Learning in Contact-Rich Manipulation

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde einer Doktorin der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

vorgelegt von

Sabrina Hoppe

aus Mainz

Hauptberichter: Prof. Dr. Marc Toussaint
Mitberichter: Prof. Dr. Marco Huber

Tag der mündlichen Prüfung: 07.09.2021

Institut für Parallele und Verteilte Systeme (IPVS)
2021

Abstract

For centuries, humans have been dreaming of intelligent machines which can move and behave like humans. With the industrial revolution, autonomously moving machinery came into existence. The intelligence to make such a machine behave or even learn autonomously is still a huge challenge however. Since first steps in this direction have been taken we have seen a rising demand for intelligent robots in many fields including production automation where robots may help to cope with increasingly flexible and rapidly changing manufacturing processes.

Researchers in the field of Reinforcement Learning investigate how to design algorithms such that agents can learn autonomously. For instance robots can be enabled to figure out the optimal way to perform production tasks by themselves. In this thesis, I will focus on insertion tasks which frequently occur in manufacturing. Today's algorithms for such tasks typically trade off sample efficiency against generalization capabilities. This means that one extreme type of algorithms makes an agent learn quickly by pre-defining a lot of structure or making specific assumptions about the task at hand, but this typically implies that the algorithm becomes very specific to this task. The other extreme type of algorithms, model-free learning methods, are very broadly applicable but then typically require vast amounts of data to learn reasonable behavior.

In this thesis, I will start from flexible, general-purpose model-free Reinforcement Learning algorithms and examine ways how to add small amounts of common sense or human prior knowledge to considerably speed up learning. Such an improvement can be characterized by the type of information that is used as well as by the approach that is chosen to make use of the information. The types of information I will use in this thesis include the robot itself, i.e., for example dynamics information; prior

knowledge about the task, for instance a coarse solution strategy that intuitively seems sensible for humans; and mathematical insights into the type and structure of data that the agent has collected.

I will also suggest a number of methods to integrate such information in a learning process: as a way to make informed choices about new actions for the agent to try (the so-called exploration), as criteria for an engineer on how to formally describe the task (i.e., how to design a suitable Markov Decision Process), and by choosing and adapting the type of function approximation that is used inside the learning process.

All methods that I will present in this thesis have been evaluated on real-world robotic manipulation tasks that have been derived or taken from industrial production plants. The results show that the proposed ways to make use of additional information significantly increase the efficiency of learning processes and can improve their stability even in adverse settings.

Zusammenfassung

Die Menschheit träumt seit Jahrhunderten von intelligenten Maschinen, die sich bewegen und verhalten wie Menschen. Seit der ersten industriellen Revolution gibt es nun ausgefeilte mechanische Systeme, die autonome Bewegungen ermöglichen. Die Intelligenz jedoch, die diese Bewegungen steuern und adaptiv auf die Umgebung anpassen könnte, lässt noch immer auf sich warten. Durch erste Schritte in diese Richtung gibt es aber bereits heute einen steigenden Bedarf für autonome und adaptive Robotik Anwendungen, etwa in der Fertigungsautomatisierung um der steigenden Nachfrage nach flexiblerer Planung und individualisierten Produkten aus automatischer Herstellung gerecht zu werden.

Forscher im Bereich Reinforcement Learning beschäftigen sich allgemeiner mit der Frage, wie man Algorithmen schreiben kann um autonome Agenten zu befähigen, selbst zu lernen. Beispielsweise könnte ein Fertigungsroboter dann selbst herausfinden, welches Verhalten oder welche Strategie eine gegebene Aufgabe am schnellsten und zuverlässigsten löst. In der vorliegenden Doktorarbeit habe ich mich mit Fügeprozessen beschäftigt, wie sie häufig in der heutigen industriellen Fertigung vorkommen. Für Algorithmen, die typischerweise im Reinforcement Learning angewandt werden, können typischerweise Generalisierungsfähigkeit gegen die Effizienz abgewägt werden, mit der verfügbare Informationen verarbeitet werden. Das heißt, es gibt ein ganzes Spektrum an Methoden die zwischen zwei Extremen liegen: Auf der einen Seite sehr spezifische Methoden, die viel Struktur und Vorwissen mitbringen und dadurch höchst effizient an eine spezielle Aufgabe angepasst sind; auf der anderen Seite Algorithmen, die für eine Vielzahl von Aufgaben geeignet sind und dann aber mehr Zeit und Daten brauchen, um sich mit einer spezifischen Aufgabe zurechtzufinden. In dieser Doktorarbeit

werde ich daher von flexiblen, allgemein anwendbaren Methoden starten und zeigen, wie durch Zuhilfenahme einzelner Informationen über die Aufgabe, den Roboter oder menschlicher Intuition der Lernprozess signifikant beschleunigt werden kann.

Die zu diesem Zweck vorgeschlagenen Methoden können von zwei Seiten charakterisiert werden: die Art der verwendeten Information und die Art der Verwendung. Die Informationen, die ich in der folgenden Promotionsschrift verwenden werde, beinhalten den Roboter an sich, z.B. Informationen über seine Dynamik; Vorwissen über die Aufgabe, ausgedrückt etwa von Menschen durch Skizzierung grober Lösungsansätze; und schließlich abstrakterer mathematischer Einsichten in die Art der Aufgabe und der daraus entstehenden Daten. All diese Informationen können verwendet werden, und zwar durch zielgerichtete Exploration des lernenden Agenten, also das gezielte Ausprobieren erfolgversprechender neuer Handlungen; durch eine gezielte formale Charakterisierung der Aufgabe, sodass ein besonders effizient zu lösendes Optimierungsproblem entsteht; und indem Wahlmöglichkeiten bei der Zusammenstellung des Reinforcement Learning Algorithmus genutzt werden, etwa die genaue Art der verwendeten Funktionsapproximation.

In diesem Spielraum aus zur Verfügung stehenden Informationen und Verwendungsmöglichkeiten werde ich in der vorliegenden Arbeit mehrere Methoden vorschlagen und durch Experimente mit echten Robotern evaluieren, die sich aus Fügeaufgaben im industriellen Fertigungsumfeld ergeben haben. Die Ergebnisse zeigen, dass die vorgeschlagenen Wege zusätzliche Informationen im flexiblen Modell-freien Reinforcement Learning zu integrieren tatsächlich zu einer Beschleunigung und unter Umständen auch Stabilisierung des Lernprozesses führen; selbst für nachteilige Situationen und Aufgaben.

The only difference between
stumbling stones and stepping stones
is the way you use them.

— AMERICAN PROVERB

acknowledgments

I am deeply grateful for the support and inspiration that my advisor, Marc Toussaint, has provided to me over the last years. His honest curiosity and passion for research questions combined with in-depth technical expertise provided an enjoyable open space for science. I appreciated the open-mindedness in our discussions as well as the friendly and constructive guidance I received through them.

Daniel Hennes, who joined our lab as a proxy to Marc Toussaint, became my main advisor for roughly one year. I am very grateful for his fresh input to my work which made the first journal paper for this thesis possible.

Additionally, I am thankful for all the time that I got to spend at the Machine Learning and Robotics Lab together with my wonderful colleagues: Andreas, Camille, Carola, Danny, Heiko, Hung, Janik, Jim, Jung-Su, Matt, Oz, Peter, Philipp, Ruth, Vien, Yoojin.

I was employed by Bosch for my PhD thesis and also received immense support from this side: Thomas and Zhongyu gave me the opportunity to work on the topic of this thesis and were a great source of support in setting up the formal framework for my PhD. I am thankful to numerous Bosch colleagues, for scientific input through various reading groups but above all from Michael, Markus and Robert; help in engineering and hardware aspects from Jens, Martin and Andreas; input on scientific writing and proof-reading of this thesis by Eduardo, Ruyu and Zhongyu; and finally the exchange among Bosch PhD students.

Finally, I cannot treasure enough everybody who supported me from the private side of life. You have been the backbone to my perseverance throughout all these years.

Contents

Acronyms **xv**

Symbols **xvii**

1 Introduction and Motivation **1**
1.1 Manipulation Tasks in Production 2
1.2 Challenges in Reinforcement Learning for Real-World Manipulation Tasks 3
1.3 Contributions and Structure of this Thesis 4

2 Background **7**
2.1 Types of Reinforcement Learning 9
2.2 Learning with Q-Values 11
2.3 Function Approximation 12
2.3.1 Bayesian Optimization and Gaussian Process 12
2.3.2 Artificial Neural Networks 14
2.4 Model-Free Off-Policy Deep RL 17

3 Peg in Hole Variants and Challenges **21**
3.1 Tasks 21
3.1.1 Clearance Fit Peg in Hole 23
3.1.2 Double Peg in Hole 24
3.1.3 Form Fit Peg in Hole 26
3.2 Hardware: Robots and Sensors 26
3.2.1 KAWADA Nextage Open Setup 27
3.2.2 Franka Emika CoBot Setup 28

4 Prestudies **31**
4.1 Function Approximation with Discontinuities 31
4.1.1 Motivation 31
4.1.2 Method 33
4.1.3 Results 34
4.1.4 Conclusion 36
4.2 Uncertainty Estimates for Value Networks in Low-Dimensional Spaces 41
4.2.1 Motivation 41
4.2.2 Method 41
4.2.3 Results 42

5 Modalities to Efficiently Learn Peg In Hole Tasks **49**
5.1 Introduction & Motivation 49
5.1.1 Disambiguating Sensory Input 49
5.2 Related Work 50
5.3 Experimental Setup 52
5.4 Method 52
5.4.1 Residual Policy Formulation 54
5.4.2 Data Collection 54
5.4.3 Network Architecture and Training Procedure 55
5.4.4 Control Loop Variants 57
5.4.5 Experimental Results 60
5.5 Discussion 61
5.6 Limitations and Open Questions 62

6 Exploration through Approximate Uncertainty-Based Trajectory Optimization **65**
6.1 Introduction & Motivation 66
6.2 Related Work 67
6.3 Method 69
6.3.1 Advantage Networks 69
6.3.2 Greedy Exploration using Bootstrapped Advantage Networks 71
6.3.3 Training Bootstrapped Advantage Networks 71
6.3.4 Global Exploration using Trajectory Optimization 73
6.4 Experimental Results 74
6.4.1 Exploration Speed 76
6.4.2 Robustness to Corrupted Steering Functions 78
6.4.3 Feasibility on a Real Robot 78
6.4.4 Sample Efficiency for Policy Learning 80
6.5 Discussion 82
6.6 Limitations and Open Questions 83

7 Linking Data Graph Structures to Soft Divergence 85
7.1 Introduction & Motivation . . . 85
7.2 Preliminaries . . . 87
7.3 Related Work . . . 88
7.3.1 Instabilities in RL: the Deadly Triad . . . 88
7.3.2 Graph Perspective on Training Data . . . 90
7.4 Method: Data Graph Structure . . . 91
7.4.1 Characterizing Transitions . . . 91
7.5 Experimental Results . . . 92
7.5.1 Experimental Setup . . . 92
7.5.2 Empirically Assessing Divergence . . . 93
7.5.3 Example I: Types of Transitions . . . 93
7.5.4 Example II: Chains of Transitions . . . 97
7.6 Discussion . . . 98
7.7 Limitations and Open Questions . . . 99

8 Stabilizing DDPG through Q-Graph-based Lower Bounds 101
8.1 Related Work: Constrained Q-learning . . . 101
8.2 Method . . . 102
8.2.1 Q-Graph Implementation . . . 103
8.2.2 Zero Actions . . . 106
8.2.3 Q-Graph Values as Lower Bounds . . . 106
8.2.4 Q-Graph-bounded Q-learning . . . 108
8.3 Experimental Results . . . 110
8.3.1 Baird's Star Example . . . 110
8.3.2 Simulated Clearance Fit Peg Insertion Task . . . 110
8.4 Discussion . . . 121
8.5 Limitations and Open Questions . . . 122

9 System Study: Q-Graph-bounded DDPG for Industrial Assembly 125
9.1 Introduction & Motivation . . . 125
9.2 Industrial Shaft Fitting Task . . . 129
9.3 Related Work . . . 130
9.4 Residual Formulation . . . 131
9.4.1 States . . . 131
9.4.2 Actions . . . 131
9.4.3 Reward . . . 132
9.4.4 QG-DDPG . . . 132
9.5 Experimental Setup . . . 133
9.5.1 Network Details . . . 135
9.5.2 Robot Control . . . 135
9.6 Experimental Results . . . 135

9.6.1 Task Difficulty and Baselines . . . 136
9.6.2 Sample Efficiency and Robustness to Hyperparameters . . 137
9.6.3 Relation to Soft Divergence . . . 138
9.6.4 Robustness to Sparse Rewards and Limited Memory . . . 139
9.7 Discussion . . . 140
9.8 Limitations and Open Questions . . . 142

10 Conclusion **145**

Appendices **161**

A Visual Experiment Documentation **163**
A.1 Visual Evaluation: Results from Chapter 5 . . . 164
A.1.1 Controller A: Vision & Passive Compliance . . . 164
A.1.2 Controller B: Closed (Hard-Coded) Control Loop . . . 165
A.1.3 Controller C: Vision & Compliance Feedback . . . 168
A.2 Visual Evaluation: Results from Chapter 6 . . . 169
A.2.1 Illustration of Steering Function . . . 169
A.2.2 Task Illustration in Simulation . . . 169
A.2.3 Learning Progress on the Real Robot . . . 170
A.3 Visual Evaluation: Results from Chapter 9 . . . 174
A.3.1 Human Baseline . . . 176
A.3.2 Learning Progress . . . 177

Acronyms

GP	Gaussian Process
BO	Bayesian Optimization
RL	Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
QG	Q-Graph
QG-DDPG	QG-bounded Deep Deterministic Policy Gradient
DQN	Deep Q-Networks
MDP	Markov Decision Process
ZA	Zero Action
RBF	Radial Basis Function
MLP	Multi Layer Perceptron
TD	Temporal Difference
EEF	End Effector
ANN	Artificial Neural Network
LQR	Linear Quadratic Regulator
UCB	Upper Confidence Bound
ReLU	Rectified Linear Unit
DoF	Degree of Freedom
tanh	Hypberbolic tangent function

Symbols

Throughout this thesis, I use the following notation: matrices are typeset as bold upper case letters like $\mathbf{W}$; vectors are represented by lower case letters in bold such as $\mathbf{x}$, while regular lower case letters like x represent scalars. Probability distributions as well as the constituents of a Markov Decision Process are typeset as calligraphic upper case letters like $\mathcal{T}$. Sets and tuples are referred to by capital letters in script style, e.g. $\mathscr{W}$.

Indices

i	Set indices, enumerating un-ordered elements
t	Indices of a sequence expressing temporal order
k	Indices of repetitions in an iterative algorithm or loop

Probabilities

$\mathcal{N}$	Normal probability distribution, parameterized by mean μ and variance σ
	μ Mean value of a normal distribution. If estimated from data, it is marked as $\widehat{\mu}$
	σ Standard deviation of a normal distribution. If estimated from data, it is marked as $\widehat{\sigma}$
	κ Scalar trading off μ and σ in the Upper Confidence Bound (UCB)
$\mathcal{U}$	Uniform probability distribution
ϵ	Probability of random actions in so-called ϵ-greedy exploration
p	Dropout probability

Artificial Neural Networks (ANN)

$\mathscr{D}$	Data Set. Elements of the set are indexed with i

NN — Neural network. A function that maps an input $\boldsymbol{\chi}$ to an output $\boldsymbol{\varphi}$ using multiple layers
- $\boldsymbol{\chi}$ — Input to a neural network. Is a tensor of different dimensionality depending on the network architecture
- $\boldsymbol{\varphi}$ — Output of a neural network. Is a tensor of different dimensionality depending on the network architecture
- l — Layer of a neural network, a non-linear function
- $\mathfrak{N}$ — Number of layers in a neural network
- $\mathfrak{n}$ — Index to identify the layers of a neural network, from 0 to $\mathfrak{N}$
- f_{in} — Number of in-going connections of a neuron
- $\mathsf{f}_{\mathsf{out}}$ — Number of out-going connections of a neuron
- $\mathbf{z}$ — Output of a neural network layer. Also referred to as latent representation of the input
- h — Nonlinear activation function used to compute a layer's activation in a Multi Layer Perceptron (MLP)
- W — Weight matrix of a layer in an MLP
- $\mathbf{b}$ — Bias vector of a layer in an MLP
- $\boldsymbol{\theta}$ — All parameters of a neural network flattened and concatenated

$\mathcal{L}$ — Loss function

ℓ_2 — The ℓ_2 metric, also noted as $\ell_2(\mathbf{x}) = ||\mathbf{x}||_2 = \sqrt{\sum_i x_i^2}$. Used to define loss functions, regularization and distance measures

λ — Regularization coefficient such that a loss $\mathcal{L}$is combined with a regularization term ρ as $\mathcal{L} + \lambda \cdot \rho$

B — Number of instances used in bootstrapping

b — Index used for instances in bootstrapping, element of $[0, B-1]$

Reinforcement Learning

$\mathscr{M}$ — Markov Decision Process (MDP)
- $\mathcal{A}$ — Action Space in a MDP $\mathscr{M}$
- $\mathcal{S}$ — State Space in a MDP $\mathscr{M}$
- $\mathcal{T}$ — Transition Probabilities in a MDP $\mathscr{M}$
- $\mathcal{R}$ — Reward Function in a MDP $\mathscr{M}$
- r_{min} — Lower bound of the reward function $\mathcal{R}$
- r_{max} — Upper bound of the reward function $\mathcal{R}$
- $\mathcal{P}_0$ — Initial State Probabilities in a MDP $\mathscr{M}$

T — Maximum number of transitions in one episode

$\mathscr{T}$ — Transition in a MDP. Indexed by t to express the temporal order in which an agent transitions; indexed by i to emphasize it is part of a batch of randomly sampled transitions from a replay memory
- $\mathbf{s}$ — State in a transition $\mathscr{T}$. Element of $\mathcal{S}$

$\mathbf{a}$	Action in a transition $\mathscr{T}$ that was executed from state $\mathbf{s}$. Element of $\mathcal{A}$
$\mathbf{s}'$	State in a transition $\mathscr{T}$ that was reached by executing $\mathbf{a}$ from $\mathbf{s}$. Element of $\mathcal{S}$
r	Reward in a transition $\mathscr{T}$, received after executing $\mathbf{a}$ from $\mathbf{s}$ and reaching $\mathbf{s}'$. Element of $\mathcal{R}$
$\mathbf{a}'$	Notes an action that was taken after a transition, i.e. starting from state $\mathbf{s}'$
π	Policy. A function mapping from state to action: $\mathcal{S} \rightarrow \mathcal{A}$
γ	Discount factor.
R	Return. Sum over discounted rewards
$\mathfrak{A}$	Series of actions an agent has taken
δ	Maximum difference in the return that the agent obtains for different $\mathfrak{A}$
V	State Value
V_{target}	Target State Value in Temporal Difference (TD)-learning
Q	State-Action Value
Q_{target}	Target State-Action Value in TD-learning
$\widehat{Q}$	Approximation to Q, e.g. critic in Deep Deterministic Policy Gradient (DDPG)
A	Advantage. Difference between Q and V
A_{target}	Target Advantage in TD-learning
α	Hyperparameter in the TD formulation of Q-learning, could be seen as a step size

Trajectory Optimization

$\mathscr{W}^*$	Set of waypoints representing an optimal trajectory
$\mathbf{w}$	Waypoints on an optimal trajectory, element of $\mathcal{S}$
η	Number of waypoints in an optimal trajectory
υ	Scaling factor for distance between successive waypoints

Bayesian Optimization (BO) and Gaussian Processes (GP)

f	Objective function to maximize
$\widehat{f}$	Approximation to f
$\mathbf{x}$	Input vector for the objective function f
$\widehat{\mathbf{x}}$	Vector selected by the acquisition function for evaluation by f
m	Mean function of a GP
k	Covariance function of a GP
ι	Length scale parameter of the Matern kernel
ν	Smoothness parameter of the Matern kernel

Observations & Measurements

- $\mathbf{I}$ — Three dimensional tesnor representing an image
- $\mathbf{c}$ — Feedback from a custom compliance feedback containing the six dimensional offset between the grippers and the wrist
  - c_{max} — Manually tuned threshold on $\ell_2(\mathbf{c})$
- Δ — Euclidean distance
- $\mathbf{p}$ — Tip position of the peg in world coordinates

Task and Policy Parametrization

- $\mathbf{g}$ — Goal position, element of $\mathcal{S}$
- ε — Maximum euclidean distance to target in reaching task
- $\mathfrak{z}$ — Policy step in z direction of the task space
- $\boldsymbol{\zeta}$ — Feedforward wrench vector, 6 dimensional, consisting of 3D force and 3D torque
  - $\mathfrak{f}$ — Feedforward force in one dimension that is given as subscript, e.g. $\mathfrak{f}_x$ in x direction
  - τ — Feedforward torque in one dimension that is given as subscript, e.g. τ_x in x direction
- $\widehat{\mathfrak{f}}$ — Estimated external force at the End Effector (EEF) in one dimension given as subscript, e.g. $\widehat{\mathfrak{f}}_x$
- $\widehat{\tau}$ — Estimated torque at the EEF in one dimension given as subscript, e.g. $\widehat{\tau}_x$

Q-Graph

- $\mathcal{G}$ — Q-Graph
  - $\mathcal{E}$ — Edges of the Q-Graph, representing transitions between states
  - $\mathcal{V}$ — Vertices of the Q-Graph, representing states
- $\mathscr{M}^{\mathcal{G}}$ — MDP with finite discrete state and action spaces, induced by data from Q-Graph $\mathcal{G}$ which is a specific subset of the data collected for a learning task with MDP $\mathscr{M}$
  - $Q^{\mathcal{G}}$ — Q-value for a transition in the simplified MDP $\mathscr{M}^{\mathcal{G}}$. Can be used as lower bound to the Q-value in $\mathscr{M}$
  - $\mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}$ — Set of actions that the Q-Graph-induced MDP $\mathscr{M}^{\mathcal{G}}$ contains starting from $\mathbf{s}_1$
- $Q^{\mathcal{G}}_{\text{target}}$ — Target Q-value after bounds from the Q-Graph $\mathcal{G}$ were applied
- LB — Lower bound to a Q-value
  - LB^{AP} — A priori lower bound
- UB — Upper bound to a Q-value
  - UB^{AP} — A priori upper bound

$\mathbf{a}_z$	Zero Action. Element of $\mathcal{A}$
$\mathcal{C}$	Capacity of a Q-Graph, i.e. maximum number of transitions it can hold

Bounds to Q-values after He et al. (2017)

L	Lower bound
U	Upper bound
K	Hyperparameter for He et al. (2017)'s bounds: maximum number of future or past transitions to consider. Counter k can take values between 0 to $\mathrm{K}-1$

List of Figures

2.1 Deep Q-Networks network architecture 18
2.2 Deep Deterministic Policy Gradient network architecture 19

3.1 Clearance fit peg in hole illustration 22
3.2 Double peg in hole illustration 24
3.3 Form fit peg in hole illustration 27
3.4 KAWADANextage Open setup 28
3.5 Franka Emika Panda CoBot setup 29

4.1 Illustration of discontinuities in peg in hole problems 31
4.2 Illustration of discontinuities in contact-rich manipulation 32
4.3 Toy example for discontinuities 34
4.4 Results: MLP on toy example for discontinuities 35
4.5 Results: GP on toy example for discontinuities 40
4.6 2D example for uncertainty estimates 44
4.7 Results: uncertainty for center condition 44
4.8 Results: uncertainty for upper left condition 45
4.9 Results: uncertainty for lower right condition 46
4.10 Results: uncertainty for center condition 47

5.1 Disambiguation of sensory input 51
5.2 Experimental setup 53
5.3 Automatic labelling 55
5.4 Network architecture 56

6.1 Bootstrapped Advantage Network 70
6.2 Results: ablation study 77
6.3 Results: robustness to corrupted steering function 79
6.4 Results: sample efficiency in policy learning 81

7.1 From replay memory to data graph 86
7.2 The 15 subsets of the data graph in example I 95
7.3 Standard deviation of predicted Q-values by transition type 96
7.4 Results: distribution of predicted Q-values for the chain example . 98

8.1 Results on 7-state star problem 111
8.2 Results: hyperparameter search and sample efficiency 113
8.3 Results: standard deviation of predicted Q-values 114
8.4 Results: sample efficiency with Zero Actions 115
8.5 Baseline performance on a grid of hyperparameters 116
8.6 Results: empirical and a priori bounds as baselines 118
8.7 Results: limited graph capacity and non-determinism 120

9.1 Close-up view on the industrial insertion task 128
9.2 Initial and target poses 134
9.3 Results: random baseline 136
9.4 Results: sample efficiency on a grid of hyperparameters 138
9.5 Results: best and worst case learning curves 139
9.6 Evolution of mean Q-value predictions during training 140
9.7 Results: robustness to sparse rewards and limited memory 141

List of Algorithms

1.A Test Time Control Loop: Open Loop, Vision Only 58
1.B Test Time Control Loop: Closed Loop, Vision Only 59
2 Exploration Episode using Approximate Uncertainty-Based Trajectory Optimization 75
3 Exemplar Q-Graph implementation 104
4 QG-bounded Deep Deterministic Policy Gradient (QG-DDPG) . . . 109

List of Tables

5.1 Results: success rate for control loop variants 60

10.1 Summary: sources of information and how to exploit them 146

Chapter 1

Introduction and Motivation

The rich corpus of mythology is full of passages showing that people have already dreamt of artificial creatures in ancient times. In the Iliad, Homer describes Hephaestus' home with mechanical "golden handmaids also who worked for him, and were like real young women, with sense and reason, voice also and strength, and all the learning of the immortals" (Butler S., 1898). These ideas have evolved over hundreds of years in literature before the industrial revolution led to significant progress in engineering. Machines soon came into existence but their "sense and reason" as well as "learning" capabilities are still largely lacking behind.

In the early 20th century, the school of behaviorism lead to new paradigms in psychological research. Contrasting other schools of thought like mentalism, they focused on observable, measurable behavior and therefore provided easily accessible inspiration to contemporaneous research in the field of robotics. Among the most influential ideas in behaviorism is operant conditioning, which describes a learning process in which the strength of a specific behavior is modulated by reward or punishment. In the late 1980s, this thread of psychological research has been combined with methods from computer science such as dynamic programming for optimal control problems (Sutton and Barto, 2018). Together, these ideas gave rise to the research field that is now called Reinforcement Learning

(RL) and that this thesis aims to contribute to. Research in this field investigates how an autonomous agent can learn to act optimally based on observed rewards and punishment.

This type of learning has many aspects. Intelligent behavior or Homer's "sense and reason" is a collection of abilities: one would expect a being to be able to learn, i.e., improve over time, to handle failures, to predict the effect of future actions, to transfer skills to new tasks and so on. In this thesis, I will focus on the speed of learning. On the one hand, fast learning—potentially even from suboptimally presented material—is widely regarded as a sign of intelligence (e.g., Jensen (1989)). On the other hand, many of today's RL algorithms need vast amounts of data to derive decent behavior. In this thesis, I will therefore propose multiple strategies to improve learning speed, i.e., in technical terms increase sample efficiency by making smarter use of broadly available information.

1.1 Manipulation Tasks in Production

We as humans tend to have the strong intuition that everyday tasks such as manipulating things are very easy compared to complex abstract tasks like playing checkers. While computers have been programmed to play checkers better than humans decades ago, the seemingly easy tasks still pose large challenges (Moravec, 1988). To perform a manipulation task, such as inserting a peg into a hole, many intricate processes are involved: Assuming that there is some abstract concept of physical objects, an agent would have to perceive the scene, locate the objects, decide where and how to grasp, and plan how to move the arm considering that the movement may have to change when the objects touch, e.g., because friction can increase. Programming such sophisticated behavior to work for a variety of tasks, objects, materials and backgrounds can be close to impossible. Even just for one specific task, immense engineering effort may be required.

One important field of application for robotic manipulation is production automation: According to the International Federation of Robotics, 44% of the operational robots in industry are used for handling operations which includes tasks like picking, placing, and insertions. More than 400,000 robotic units were estimated to have been newly installed in industry in 2018 alone, corresponding to a rise of 15% compared to 2017 (Müller

and Kutzbach, 2019). This steep increase may further accelerate with progress in robotics leading to more and more tasks which can be automated.

Recent changes in manufacturing industry, such as more individualized products, more product variants, and more agile planning have caused a trend towards smaller lot sizes (Spath et al., 2013). To cope with this required flexibility, re-configurable production systems are needed and create an additional demand for intelligent robots. These machines would ideally be easy and fast to setup, use and re-use on different tasks. Thus, a truly intelligent, learning agent who figures out how to solve a task robustly and is then able to transfer this skill to other tasks seems highly desirable for practical challenges in today's production plants.

1.2 Challenges in Reinforcement Learning for Real-World Manipulation Tasks

The famous *no free lunch theorems* by Wolpert and Macready (1997) state, loosely speaking, that on average over all possible optimization problems, every pair of algorithms performs equally well. This means that there is no single best algorithm which outperforms all others in all cases. Instead, each approach has advantages and disadvantages that can be traded off against each other.

In RL, prior knowledge versus generalization across tasks form a very prominent trade-off: If only a single task is considered, knowledge about the particular task at hand can be used to create a very efficient algorithm. However, such a hand-drafted algorithm for one task is unlikely to generalize to a wide range of other tasks. Similarly, algorithms which do not use any knowledge about the task at hand typically need much more data to infer good behavior.

In practice, the sweet spot is often somewhere in between and only requires to add as much knowledge as possible but keep as much flexibility and generalization capabilities as needed. In this thesis, I will start from very general RL algorithms and experiment with different ways of injecting prior knowledge without loosing too much of the algorithm's generaliza-

tion capabilities and without requiring too much human input for setup. As I will show, already small and relatively abstract pieces of information can speed up the learning process substantially.

1.3 Contributions and Structure of this Thesis

After giving common background information on the field of RL in Chapter 2, I will first give full details and background on the manipulation tasks I have chosen to evaluate all methods for the remainder of this thesis in Chapter 3.

In Chapter 4 I will present systematic prestudies on function approximation that motivate my choice of methods for the subsequent chapters of this thesis. First, I have evaluated how different function approximators deal with discontinuities (Section 4.1). This question is important in the context of RL for manipulation tasks, because physical contact is very likely to introduce discontinuities. For example, the dynamics function but also the desired behavior changes instantly when the end effector gets in touch with a rigid object such as a hard table top. The results show that neural networks have slight advantages over other methods in these cases, but one of the most obvious drawbacks is that networks do not intrinsically offer an uncertainty measure for their predictions. I therefore evaluated different types of uncertainty measures for neural networks in Section 4.2.

My investigations towards the main topic of this thesis, increasing sample efficiency, start in Chapter 5. I have analyzed the relation of input modalities to sample efficiency and show on a dataset of fixed size that using more informative data speeds up learning even if it increases the dimensionality of the learning problem. These results have been published in

> Hoppe, S., Lou, Z., Hennes, D., and Toussaint, M. (2017). Deep learning for manipulation with visual and haptic feedback. IROS Workshop on Frontiers in Contact-Rich Robotic Interaction

In Chapter 6 I will present insights from an analysis on a particular aspect of RL algorithms, namely exploration—which refers to new behavior the agent tries in order to learn more about its environment or task.

I show that algorithms which do not make use of detailed knowledge of the robots dynamics, and therefore are more easily applicable in environments with changing types of contacts, benefit if such knowledge is used to plan exploration trajectories. These results have been published in:

> Hoppe, S., Lou, Z., Hennes, D., and Toussaint, M. (2019). Planning approximate exploration trajectories for model-free reinforcement learning in contact-rich manipulation. *IEEE Robotics and Automation Letters*, 4(4):4042–4047

Starting from Chapter 7 I will take a more abstract perspective on the data that an agent is supposed to learn from. In particular, I propose a graph view on past experience and show that one of the major stability issues for Deep Deterministic Policy Gradient (DDPG), namely soft divergence, can be linked to the graph structure. The graph view furthermore allows to derive novel lower bounds for Q-learning based deep RL methods which, as I will show in Chapter 8, can help to stabilize learning. All details and experiments are also publicly available in:

> Hoppe, S. and Toussaint, M. (2020). Q-graph-bounded Q-learning: Stabilizing model-free off-policy deep reinforcement learning. *arXiv preprint arXiv:2007.0758*

In Chapter 9 I will present a system study on an industrial insertion task and a particularly sample-efficient residual policy formulation. Such a formulation induces a particular type of graph structure, because the agent may be more likely to reach its goal but the average trajectory length increases. I have shown that this provokes soft divergence on the one hand, and can be prevented using my graph-based lower bounds on the other hand. Thus, this concluding system study is based on the insights from the previous chapters and corroborates several effects under real-world conditions. These results have been published in:

> Hoppe, S., Giftthaler, M., Krug, R., and Toussaint, M. (2020). Sample-efficient learning for industrial assembly using Q-graph-bounded DDPG. In *International Conference on Intelligent Robots and Systems*, pages 9080–9087

The system study as well as all the preceding research on Q-Graphs and their relation to soft divergence have been combined and presented in an on-going submission:

Hoppe, S., Giftthaler, M., Krug, R., and Toussaint, M. (2021). Stabilizing deep Q-learning with Q-Graph-based bounds. Under review for the International Journal for Robotics Research (IJRR)

Chapter 2

Background

The term *reinforcement* has first been used in psychology by Behaviorists for systematic changes in an organism's environment which influence its behavior; similar to a child learning from praise or reproach. In computer science, Reinforcement Learning (RL) describes a problem setting in which an agent learns from interacting with its environment (Sutton and Barto, 2018). In contrast to supervised learning, the agent initially has no data to learn from but instead collects its own data over time. RL has therefore been described as a 'way of programming agents by reward and punishment without needing to specify how a task is to be achieved' (Kaelbling et al., 1996). The agent receives reinforcement for its behavior and is striving to maximize the amount of long-term reinforcement. Silver et al. (2021) even hypothesize that 'reward is enough' for intelligence, and thus reinforcement learning could—if their conjecture is true—be 'a direct pathway towards understanding and constructing an artificial general intelligence.'

Formally, the setting is typically described as a Markov Decision Process (MDP) that consists of a state space $\mathcal{S}$, a probability distribution over initial states $\mathcal{P}_0$, an action space $\mathcal{A}$, transition probabilities between states $\mathcal{T}$ and the reward function $\mathcal{R}$ (Sutton and Barto, 2018). At each discrete time step t, the agent can then observe its state $\mathbf{s}_t \in \mathcal{S}$ and choose an action according to a function called policy $\pi(\mathbf{s}_t) = \mathbf{a}_t \in \mathcal{A}$, which leads to a new state $\mathbf{s}_{t+1}$ with probability $\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and an immediate reward $r_t = \mathcal{R}(\mathbf{a}_t, \mathbf{s}_{t+1})$. One such transition is summarized as a tuple

$\mathscr{T} = (\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, where the time index is omitted and state $\mathbf{s}_{t+1}$ is also referred to as $\mathbf{s}'$. The action taken from $\mathbf{s}'$ is not part of the transition anymore but for brevity also referred to as $\mathbf{a}'$.

The discounted sum over all rewards until a terminal state was reached is called return:

$$R = \sum_t \gamma^t r_t,$$

where γ is a so-called discount factor that ensures convergence of this sum even over infinite terms. The discount factor γ can also be interpreted as trading off immediate rewards against those which are farther in the future. Often, RL tasks are designed to be *episodic*: after a certain number of steps T the agent is reset to an initial position even if no terminal state was reached. The agent's goal is to find the optimal policy π^* which maximizes the expected future return $\mathbb{E}[R]$.

Striving for improvement, the agent from time to time needs to *explore* actions that are different from its behavior policy π. In the simplest case, actions can be chosen randomly with some probability ϵ. This leads to a trade-off between exploration and what is called *exploitation*: namely executing safe and rewarding behavior that the agent has already discovered. If actions are continuous, so-called *dithering strategies* can be applied, where some noise is added to the output of the policy. Different noise processes have been examined for this, such as simple Gaussian noise or the temporally correlated Ornstein-Uhlenbeck noise (e.g., in Lillicrap et al. (2015)).

The reinforcement or punishment an agent receives typically has been manually designed by a programmer. The process of defining a good reward function for a MDP such that the agent can learn quickly is called *reward shaping* (e.g., Grześ (2017)). For instance, if the goal position is known, a reward can be inversely proportional to the agent's distance from the goal. Such a distance-based reward could lure the agent towards the goal even if the agent has not actually discovered the goal state yet. To make the agent minimize the number of steps it needs to reach the goal, rewards are often negative at every time step. The return for a long trajectory will therefore decrease with every step the agent needs to take. This is one example how prior knowledge about a task can be used to speed up learning. In this thesis, I will explore further ways to infuse prior knowledge into RL methods.

2.1 Types of Reinforcement Learning

The RL problem of finding an optimal policy for a given task can be approached with a variety of different methods, but there are three main categories:

- *Policy Optimization* methods directly optimize a parameterized policy. That is, a policy gradient is computed that determines how to change the parameters of the policy to increase the expected return. In the simplest case, this expectation can be approximated by the sample mean of gradients for all trajectories and associated return that an agent has experienced so far. More complex algorithms may for example also limit the change in policy for each update step (Schulman et al., 2015, 2017). Algorithms in this category can typically be further differentiated by how the policy gradient is found, e.g., whether or which type of approximations are used. The advantage of policy search methods is that they directly optimize what the agent is seeking to find: the policy. On the downside these algorithms work *on-policy*: that is, they reason about the current policy only and thus, may not be able to learn from data collected in states that the current policy would not encounter or from actions the current policy would never choose.

- *Model-based* methods first identify models of relevant system properties such as the agent dynamics (e.g., Yu et al. (2017)). This can be very efficient because typically supervised learning can be applied. Since these models are trained on all available data irrespective of the current policy, this is an example of *off-policy* methods. In a second step, a variety of robotics methods can utilize the model, e.g., for planning or imaginary rollouts (Sutton, 1990), which can speed up learning and help to restrict rollouts on the actual robot to promising interaction that seem safe to execute. There is always an underlying assumption that the function class used to fit the model is capable to capture the true dynamics. In Section 4.1 I will show that this assumption can be problematic when the dynamics are discontinuous.

- *Model-free* methods do not explicitly identify models but instead rely on *value learning*: A *state value* $V^{\pi}(\mathbf{s})$ represents the expected return for an agent that starts at state $\mathbf{s}$ and follows policy π:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}\left[\sum_{t} \gamma^{t} r_{t}\right] \tag{2.1}$$

Analogously, a *state-action value* $Q^{\pi}(\mathbf{s}, \mathbf{a})$ represents the expected return for an agent in state $\mathbf{s}$ that first executes $\mathbf{a}$ and then follows policy π (Watkins, 1989). The difference in reward the agent receives when choosing action $\mathbf{a}$ rather than $\pi(\mathbf{s})$ in the first step and then follow policy π is also referred to as *advantage*

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}) \tag{2.2}$$

The core of model-free RL methods is then often function approximation to find V or Q (Mnih et al., 2015). Given the correct Q, the optimal policy can be easily inferred:

$$\pi^{*}(\mathbf{s}) = \arg\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \tag{2.3}$$

In the absence of models, all data to work with needs to be collected by the agent through experience. However, the lack of predictive models can make this exploration process dangerous. For instance, there is no model that could be used to predict what consequences one action will have and thus every state-action pair may in principle be tried directly in the real world.
On the positive side, these algorithms can be also applied in domains where models are hard to obtain, for example contact-rich manipulation. Such generality often comes at the cost of decreased sample efficiency though. This means that, because the algorithm can cope with a huge variety of environments, it takes particularly much data to narrow all the possible policies down to the optimal one.

2.2 Learning with Q-Values

One way to estimate Q-values $\widehat{Q}$ from data is Q-iteration: In each repetition k, Q-iteration iterates over all states and actions as follows (Watkins, 1989):

$$\forall \mathbf{s}, \mathbf{a}.\ \widehat{Q}_{k+1}(\mathbf{s}, \mathbf{a}) = \mathcal{R}(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{T}(\mathbf{s}' \mid \mathbf{a}, \mathbf{s}) \cdot \max_{\mathbf{a}' \in \mathcal{A}} \widehat{Q}_k(\mathbf{s}', \mathbf{a}') \quad (2.4)$$

If the transition probabilities $\mathcal{T}$ are not known (or explicitly estimated), often the Temporal Difference (TD) formulation of Q-learning is used. It does not iterate over all state-action pairs but instead only uses past experience which has been stored as a set of trajectories. Each trajectory is a sequence of transitions $\mathscr{T} = (\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ with state $\mathbf{s}$, the action $\mathbf{a}$ that the agent took, the state $\mathbf{s}'$ that the agent reached and the reward r it received. The TD formulation then iteratively updates the estimated Q-values $\widehat{Q}_k$ in iteration k as follows:

$$\widehat{Q}_{k+1}(\mathbf{s}, \mathbf{a}) = (1 - \alpha)\widehat{Q}_k(\mathbf{s}, \mathbf{a}) + \alpha \left[\underbrace{r + \gamma \max_{\mathbf{a}'} \widehat{Q}_k(\mathbf{s}', \mathbf{a}')}_{Q_{\text{target}\,k}} \right] \quad (2.5)$$

where the hyperparameter α determines how far $\widehat{Q}_k$ is moved towards the TD target $Q_{\text{target}\,k}$. This formulation of Q-learning can be seen as a stochastic variant of Q-iteration and for cases where the full state-action space can be iterated it converges under some conditions for the learning rate with probability 1 (Watkins and Dayan, 1992; Jaakkola et al., 1994).

A widely used variant of this formulation uses $\alpha = 1$ and explicitly considers terminal states for which the TD target becomes r:

$$Q_{\text{target}\,k}(\mathbf{s}, \mathbf{a}) = r + \begin{cases} 0, & \text{if } \mathbf{s}' \text{ is terminal} \\ \gamma \cdot \widehat{Q}_k(\mathbf{s}', \pi(\mathbf{s}')), & \text{else} \end{cases} \quad (2.6)$$

which makes use of the definition of π as choosing the action that is optimal with respect to the current estimate of Q:

$$\begin{aligned}\widehat{Q}(\mathbf{s}', \pi(\mathbf{s}')) &= \widehat{Q}_k(\mathbf{s}', \arg\max_{\mathbf{a}'} \widehat{Q}_k(\mathbf{s}', \mathbf{a}')) \\ &= \max_{\mathbf{a}'} \widehat{Q}_k(\mathbf{s}', \mathbf{a}')\end{aligned}$$

2.3 Function Approximation

For small and finite MDPs, Q-values can be represented in tables using one entry per state-action pair. In robotics applications however, both states and actions are typically real-valued. To circumvent discretization issues, function approximation can then be applied instead of tables—may it be to fit a model, to obtain value-functions or to find a parameterized policy.

The following sections will therefore cover two general types of function approximation: Gaussian Processes (GP) and Deep Learning. Finally, deep Reinforcement Learning (RL) methods which use deep function approximators for RL methods will be introduced in detail.

2.3.1 Bayesian Optimization and Gaussian Process

Bayesian Optimization (BO) is a sequential approach for maximizing an objective function that is not analytically known and can only be evaluated at few input vectors $\mathbf{x}$ (Shahriari et al., 2015):

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$$

This optimization problem is solved by iteratively refining an estimate $\widehat{f}$, starting from a prior belief. For each step of refinement, new points $\widehat{\mathbf{x}}$ are determined for which the estimate $\widehat{f}(\widehat{\mathbf{x}})$ is compared to the outcome of the true objective function $f(\widehat{\mathbf{x}})$. The function that is used to choose $\widehat{\mathbf{x}}$ is called *acquisition function*. Given a new objective function evaluation $f(\widehat{\mathbf{x}})$, the current estimate $\widehat{f}$ is updated to a better informed posterior distribution over the space of admissable functions.

One example of an acquisition function is the Upper Confidence Bound (UCB) as suggested by Auer (2002) which I will refer to as UCB_f:

$$\begin{aligned}\widehat{\mathbf{x}} &= \arg\max_{\mathbf{x}} \hat{\mu}(\widehat{f}(\mathbf{x})) + \kappa \cdot \hat{\sigma}(\widehat{f}(\mathbf{x})) \\ &= \arg\max_{\mathbf{x}} \text{UCB}_{\widehat{f}}(\mathbf{x})\end{aligned} \tag{2.7}$$

As can be seen from Eq. (2.7), the UCB uses a coefficient κ to combine two criteria: a high mean value $\hat{\mu}(\widehat{f}(\mathbf{x}))$ indicates that $\mathbf{x}$ may be a good candidate for a maximum, while a high standard deviation $\hat{\sigma}(\widehat{f}(\mathbf{x}))$ additionally means that there is uncertainty to resolve. Combined, an input vector $\mathbf{x}$ which leads to both relatively high mean and standard deviation in the current estimate $\widehat{f}$ is particularly likely to lead to a high function evaluation $f(\mathbf{x})$. With f being the (unknown) reward function, this acquisition function can be applied to trade off exploration and exploitation in RL: a high predicted mean reward for exploitation on the one hand and a large variance as a good opportunity for exploration on the other hand.

A function approximator that is often used with BO are GPs. While multiple views on GPs exist, the following function-space centered definition is most accessible in the context of function approximation (see Williams and Rasmussen (2006) for more details): A GP describes a probability distribution over functions $f(\mathbf{x})$ and is completely specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$:

$$\begin{aligned}m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]\end{aligned}$$

Intuitively, the covariance function k encodes the smoothness of functions f, i.e., how similar $f(\mathbf{x})$ and $f(\mathbf{x}')$ are depending on the similarity of $\mathbf{x}$ and $\mathbf{x}'$.

Generally, the covariance function has a large impact on BO performance. If well tuned, it can lead to very useful generalization between samples, but in special cases it can also pose challenges: if the function to approximate changes quickly in some particular areas (e.g., a cliff the robot must avoid), one may need to use small covariances to make the acquisition function sample close enough to the cliff to obtain informative

data. On the other hand, this would automatically also lead to less generalization in all other areas of the state space and thus decrease sample efficiency. I will come back to this issue in Section 4.1.

2.3.2 Artificial Neural Networks

Artificial Neural Network (ANN) have been investigated as function approximators for a long time. First work in the 1950ies was mainly inspired by brain research. For instance the first perceptron—today still known as a simple type of ANN—was introduced by Rosenblatt (1958) in a journal that, by their current description, publishes articles "that make important theoretical contributions to any area of scientific psychology"[1].

A number of algorithmic improvements starting in the 1980ies have led to breakthroughs for ANNs: On the algorithmic side, algorithms such as backpropagation (Rumelhart et al., 1986) lead to efficient training of ANNs even with complex internal structures. Further improvements include principles such as weight sharing (LeCun et al., 1989) that is mainly used to reduce the number of free parameters in convolutional neural networks and specific initializations that consider the network architecture (Glorot and Bengio, 2010).

In 2001, Kussul et al. (2001) have shown that Rosenblatt's perceptron network architecture could perform similar to state of the art on the MNIST dataset, a widely used benchmark in computer vision, by only few modifications. In the 21st century more and more resources could be used to train networks and allowed researchers to create, share and store huge datasets. In computer vision for example, a lot of progress has been attributed to datasets and challenges such as ImageNet (Deng et al., 2009) which played an important role for the impressive improvements in the field since 2010 (LeCun et al., 2015).

Today it is possible to easily download network weights for popular architectures that resulted from extensive training on these huge datasets. This allows other researchers to work with trained networks without the burden of time-consuming training, and therefore gave rise to what is called *fine-tuning*: Neural networks that have been trained on one task can often quickly adopt to new domains, for instance by fixing the first layers and only re-training the latter ones (Oquab et al., 2014).

[1] `https://www.apa.org/pubs/journals/rev/`, last retrieved: 01.12.2019

Many intricate variants of ANNs exist but this thesis will only work with feed-forward fully connected neural networks—the so-called Multi Layer Perceptrons (MLP), which will be introduced in the following paragraph. Networks represent weight-parameterized non-linear functions

$$\boldsymbol{\varphi} = \mathsf{NN}(\boldsymbol{\chi} \in \mathbb{R}^d, \boldsymbol{\theta})$$

The function is called network because of the modular structure that is used to compute NN (LeCun et al., 2015): It consists of $\mathfrak{N}$ layers l^{n} which are functions that map input from previous layers $\mathbf{z}^{\mathrm{n}}$ to output vectors $\mathbf{z}^{\mathrm{n}+1}$ using a weight matrix W^{n}, a bias vector $\mathbf{b}^{\mathrm{n}}$ and a non-linear activation function h:

$$\mathbf{z}^{\mathrm{n}+1} = h^{\mathrm{n}}(\mathsf{W}^{\mathrm{n}}\mathbf{z}^{\mathrm{n}} + \mathbf{b}^{\mathrm{n}})$$

The input of the first layer is at the same time the input to the network, and the output of the last layer is also interpreted as the output of the network. The full network is therefore a highly non-linear function of the input:

$$\begin{aligned}\boldsymbol{\varphi} &= \mathsf{NN}(\boldsymbol{\chi} \in \mathbb{R}^d, \boldsymbol{\theta}) \\ &= h^{\mathfrak{N}-1}\Big(\mathsf{W}^{\mathfrak{N}-1}\Big[\dots h^1\big(\mathsf{W}^1\big[h^0(\mathsf{W}^0\boldsymbol{\chi} + \mathbf{b}^0)\big] + \mathbf{b}^1\big)\dots\Big] + \mathbf{b}^{\mathfrak{N}-1}\Big)\end{aligned}$$

All weight matrices and bias vectors are flattened and concatenated to form the so-called weights of the network, i.e., its parameters:

$$\boldsymbol{\theta} = \{\mathsf{W}^0, \mathbf{b}^0, ...\mathsf{W}^{\mathfrak{N}-1}, \mathbf{b}^{\mathfrak{N}-1}\}$$

Popular choices for activation functions include the Hypberbolic tangent function (tanh) or Rectified Linear Unit (ReLU):

$$\begin{aligned}\mathsf{tanh}(\mathbf{z}) &= \frac{\exp(2\mathbf{z}) - 1}{\exp(2\mathbf{z}) + 1} \\ \mathsf{ReLU}(\mathbf{z}) &= \max(\mathbf{z}, \mathbf{0})\end{aligned}$$

where the `exp` and `max` operation are applied element-wise to the input vector $\mathbf{z}$.

The weights $\boldsymbol{\theta}$ of a neural network are trained in a *supervised* learning scheme: there is a data set $\mathscr{D} = \{(\mathbf{x}, \mathbf{y})_i\}$ that contains a number of input - output pairs. A loss function $\mathcal{L}$ is used to compare the current output of

the network with the ground truth from the dataset $\mathscr{D}$. A popular choice for a loss function is the ℓ_2 loss function:

$$\mathcal{L}\big(\mathsf{NN}|\mathscr{D}\big) = \ell_2\big(\mathsf{NN}(\mathbf{X}) - \mathbf{Y}\big) = \sqrt{\sum_i \Big(\mathsf{NN}(\mathbf{x}_i) - \mathbf{y}_i\Big)^2} \tag{2.8}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are the concatenation of all $\mathbf{x}_i$ and $\mathbf{y}_i$ respectively.

This loss function can be used to compute gradients $\frac{\delta \ell_2}{\delta \boldsymbol{\theta}}$. Together with the chain rule and a scheme called backpropagation, these gradients can be used to update all network parameters such that the output of the loss function is minimized (Rumelhart et al., 1986).

There is a large body of literature covering the choice of initial network weights, activation functions and their interplay. For instance, initial weights that are drawn from a simple uniform distribution often lead to saturation when activation functions such as tanh or ReLUs are used as activation function (Glorot and Bengio, 2010). Saturation means that a neuron outputs mostly values close to the limit of the range of the function, e.g., such as zero as the lower limit of values that a ReLU activation function yields. In such cases, the gradients often become very flat and make it hard to further optimize the neural network although the current predictions may not be good. Saturation can be limited when the type of activation function is analyzed more closely though. For instance for tanh activations, initial weights that are drawn uniformly from the following interval were shown to be advantageous:

$$\left[-\frac{\sqrt{6}}{\mathsf{f}_{\mathsf{in}} + \mathsf{f}_{\mathsf{out}}}, +\frac{\sqrt{6}}{\mathsf{f}_{\mathsf{in}} + \mathsf{f}_{\mathsf{out}}}\right]$$

where f_{in} and $\mathsf{f}_{\mathsf{out}}$ represent the number of in- and out-going connections to a neuron (Glorot and Bengio, 2010). This is also referred to as *Xavier*-initialization after its inventor Xavier Glorot.

A similar initialization scheme has also been derived by He et al. (2015) for ReLU activation functions which is referred to as *he-uniform*-initialization and will be used for most of the ANNs in this thesis: Initial weights for ReLU activation function are better drawn uniformly from the interval:

$$\left[-\frac{\sqrt{6}}{\mathsf{f}_{\mathsf{in}}}, +\frac{\sqrt{6}}{\mathsf{f}_{\mathsf{in}}}\right]$$

Many more improvements have been suggested to make ANNs learn faster and more stable. For example, normalizing the input to each layer of neurons by standardizing their mean and variance has been shown to improve the learning procedure (Ioffe and Szegedy, 2015). However, as for many empirical findings, the theoretical reason for the apparent improvements is still under debate (Santurkar et al., 2018; Yang et al., 2019).

2.4 Model-Free Off-Policy Deep RL

As discussed in Section 2.2, the key idea in Q-learning is to find a Q-function which assigns the expected future return to state-action pairs.

Since neural networks are powerful function approximators, they can also be used to fit Q-functions. In Deep Q-Networks (DQN), a network takes a state representation as an input and predicts one Q-value per action as an output (see Figure 2.1, Mnih et al. (2015)). This network can then be trained using TD learning as follows: the TD target value is computed as in Eq. (2.6) and compared to the currently estimated Q-values using the expected squared error

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathscr{T}\sim\mathcal{U}(\mathscr{D})}\left[\left(Q_{\mathsf{target}}(\mathbf{s}, \mathbf{a} \mid \boldsymbol{\theta}^{-}) - \widehat{Q}(\mathbf{s}, \mathbf{a} \mid \boldsymbol{\theta})\right)^2\right] \tag{2.9}$$

where the transition $\mathscr{T}$ is a tuple $(\mathbf{s},\mathbf{a},r,\mathbf{s}')$. $\mathscr{D}$ refers to all the experience an agent has collected and is represented as a data storage called replay memory (Mnih et al., 2015). The expectation in Eq. (2.9) is then approximated by averaging the error from a number of randomly sampled transitions (*batch*) from the *replay memory*. Randomly sampling transitions increases the statistical independence of samples in one batch, because

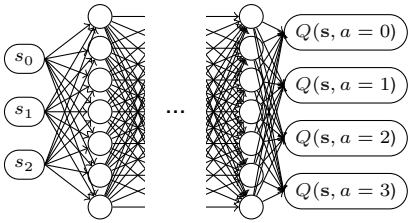
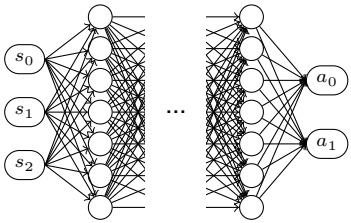


Figure 2.1: DQN network architecture. A state is used as input ($\mathrm{s} = [s_0\ s_1\ s_2]$). The output consists of one Q-value for each possible action a in the finite action space $\mathcal{A}$. Note that the actions are only enumerated and not parameterized. The network structure can vary, e.g., use convolutions in case the state input are images.

it breaks the order in which samples were collected and in most domains data of subsequent time steps is very close. In turn this statistical independence is a requirement for the stochastic optimization methods that are typically used to update network weights.
$\boldsymbol{\theta}^-$ are parameters of a *target network* which is also introduced in Mnih et al. (2015) because learning with TD updates tends to become unstable when the TD targets change too quickly. Therefore, a second network representing a $\widehat{Q}$-function, the so-called target network, is parameterized by $\boldsymbol{\theta}^-$ in Eq. (2.9). The weights of the target network are updated to be equal to those of the $\widehat{Q}$ network, but far less frequently. They therefore do develop as the weights in $\widehat{Q}$ but act as a low pass filter, which helps to stabilize training. However, recent work on divergence in deep RL has found that target networks only tend to delay stability problems but not solve them (Van Hasselt et al., 2018). I will come back to this point in Chapter 7.

Since the network in DQN predicts one Q-value per action, it is only applicable for finitely many actions. In settings with continuous actions—as typically required for robotics applications—an actor-critic architecture called Deep Deterministic Policy Gradient (DDPG) can be used (see Figure 2.2, Lillicrap et al. (2015)): The network that takes both state and action as an input and predicts the Q-value for this pair is referred to as critic. The policy is represented as an actor network that maps states to the optimal action to take. The critic is trained using TD learning and the same loss function as in Eq. (2.9). The current estimate of the critic is then

Actor network:



Critic network:

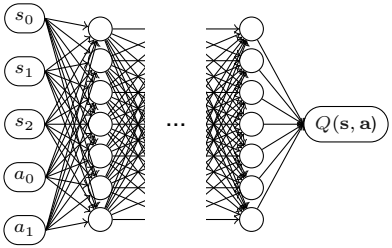


Figure 2.2: DDPG uses two networks: the actor networks represents the policy. It takes a state ($\mathbf{s} = [s_0\ s_1\ s_2]$) as input and predicts the optimal action ($\mathbf{a} = [a_0\ a_1]$). The critic network takes a state-action pair as input ($[\mathbf{s}\ \mathbf{a}] = [s_0\ s_1\ s_2\ a_0\ a_1]$) and predicts the Q-value for this pair.

also used as a training signal for the actor network which in other words means that the actor network π is trained to maximize the Q-value that the critic predicts:

$$\pi(\mathbf{s}) = \max_{\mathbf{a}} \widehat{Q}(\mathbf{s}, \mathbf{a})$$

Once an actor network has been trained, it is computationally much more efficient to evaluate the network at test time than it would be to maximize the highly non-linear Q-function $\widehat{Q}$.

Chapter 3

Peg in Hole Variants and Challenges

All methods I will discuss in the following chapters of this thesis were evaluated on tasks that were taken from or inspired by real industrial use cases for robotic manipulation. In many aspects, evaluations in simulation may have been advantageous but one of the fundamental motivations for this work was the idea that model-free methods are promising in fields where exact models are hard to obtain. Therefore, the idea that realistic simulation environments are available for these tasks seems incompatible. Instead, there is a whole research community (cf. for instance Collins et al. (2019)) who investigates simulation on the one hand, and transfer learning or other techniques to bridge the gap between simulation and the real world which are beyond the scope of this thesis.

In this chapter, I will first describe the tasks in detail and introduce their respective challenges. Afterwards, I will present the two robotic setups that have been used for the experiments throughout this thesis.

3.1 Tasks

All tasks in this doctoral thesis are variants of a paradigm called *peg in hole*. Peg in Hole tasks are well-established in robotics and automation research: one lengthy object has to be inserted into a second object of

inverted shape, see Figure 3.1 for an illustration. This task is one of the universal steps that frequently re-occur in industrial assembly, e.g., for assembly processes as in Sayler (2011).

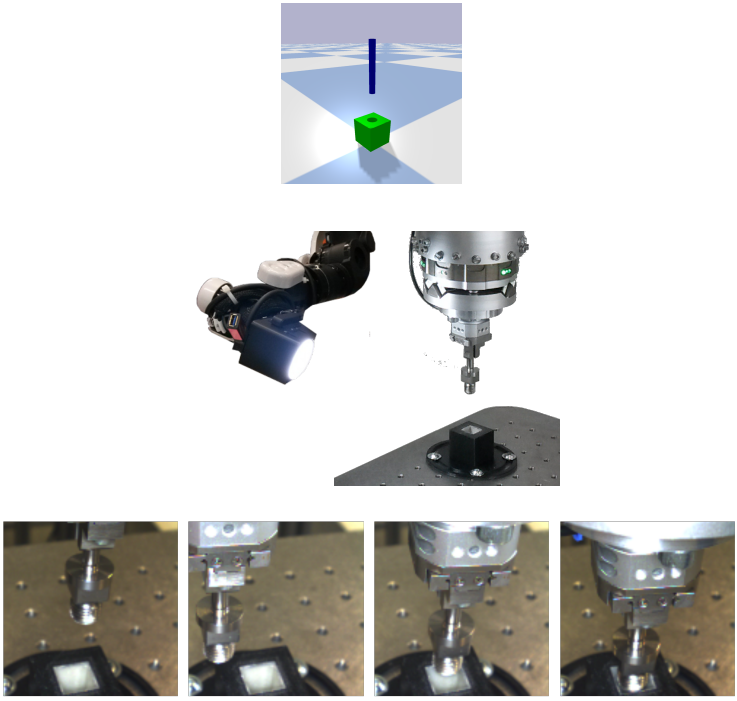


Figure 3.1: Clearance fit peg in hole tasks: one object needs to be inserted into a second object. Top: prototypical simulation screenshot. Middle: real-world setup with a custom passive compliance element as well as the KAWADA's built-in camera inside of an illuminated ring as light source. Bottom: series of images from real-world experiment.

3.1.1 Clearance Fit Peg in Hole

A peg in hole task can, in principle, be solved by avoiding any contact, as long as the hole is slightly wider than the peg. I therefore started my investigations with a simplified setting that is also referred to as *clearance fit* peg in hole. Large portions of the experiments on this task consisted of motion in free air, i.e. contactless. If the robot dynamics are known or a model can be inferred, model-based robot control is straight forward to apply and achieves robust solutions. However, I only regard the task as a step towards more complex tasks. Therefore I already apply model-free methods which promise to still be applicable in contact-rich environments, particularly for tight form fit peg in hole tasks, which are much more likely to occur in real-world industrial tasks.

3.1.1.1 Simulation Details

The simulation environment corresponding to clearance fit tasks was implemented using pybullet[1] and is illustrated in the top row of Figure 3.1. The blue peg is always upright and velocity-controlled: an action represents the three-dimensional offset to the next position. To execute an action, the new reference position is set and the simulation stepped forward until a stable new position is reached. The actions are box-constrained to $[-1, 1]$ in each dimension which corresponds to a movement of 1 cm. The green object has a width of 5 cm and is placed on the bottom of a cubic state space with side length 20 cm. The peg has a diameter of 1 cm while the hole has a diameter of 2 cm. In our default RL setting for this task, the agent receives a distance-based reward $r = \exp(-\frac{\Delta}{0.03}) - 1$, where Δ is the Euclidean distance to the goal position in meters. This reward was shaped such that all possible positive distances lead to a reward in $[-1, 0]$ and thus urge the agent to find solutions with as few steps as possible.

3.1.1.2 Real-World Task Details

In the real-world experiment corresponding to the clearance fit example shown on the bottom of Figure 3.1, a square metal piece of side length 1.3 cm was supposed to be inserted into a 3D-printed object with 1mm clearance. That is the square hole inside the 3D printed object was of

[1] `https://github.com/bulletphysics/bullet3`

side length 1.4 cm. For visual clarity, the object was 3D printed with white on the inside and black on the outside. The robot operated in a 3D rectangular state space with width and length of 3 cm and height of 5 cm. This work space was centered around the 3D-printed object that also had side length 3 cm. To avoid harmful collisions, a compliance element was mounted at the end effector (cf. Subsection 3.2.1) and its feedback used in different control loops which will be discussed in more detail in Subsection 5.4.4.

3.1.2 Double Peg in Hole

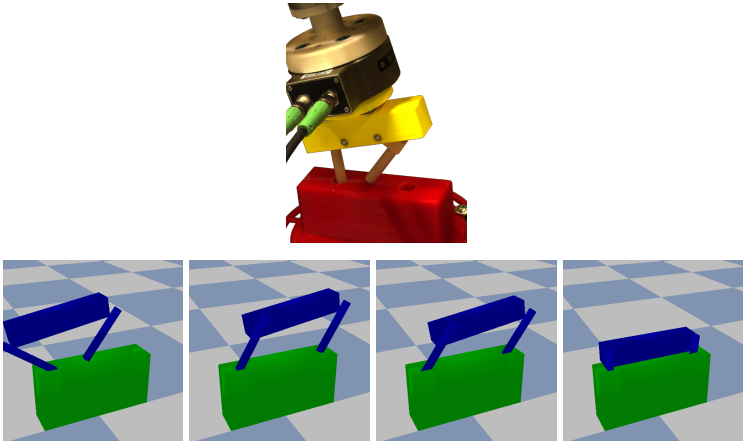


Figure 3.2: Double peg in hole: Extending the classical peg in hole by a second peg enforces contact: one peg must stay in contact until the second peg is inserted as well.

To evaluate my methods in contact-rich tasks, I introduced a so-called *double peg in hole* task. This task principle is similar to the clip insertion investigated in Vecerik et al. (2017). In our setting, instead of a clip, two rigid pegs are mounted on a bar with off-axis joints such that the two pegs are pointing towards the center in free air. The second object consists of two holes that the pegs can only be inserted into when one of the pegs is first inserted a little, then this peg stays in contact while the bar is moved

such that the second peg can also be inserted (see Figure 3.2). The video accompanying the publication presented in Chapter 5 also shows this task as executed on a real robot and can be accessed under `https://www.youtube.com/watch?v=JTfeHhWSb0Y`.

In this extension of the classical peg in hole setting, dynamics models are already harder to obtain: as soon as the pegs touch the second object, the dynamics vary locally depending on the peg tip position (e.g., inside vs. outside of a hole) and further properties such as the surface friction when sliding from one hole to the other. Additionally, the robot may be underactuated for this task: there is no motor to control and maybe also no sensor to measure the individual peg orientations. Formally, this can turn the environment from a MDP into a *Partially Observable* Markov Decision Process (POMDP). However, due to the nature of the task, locally the unobserved Degrees of Freedom (DoF) are often a function of the observed DoFs—in particular, when working with a deterministic policy.

3.1.2.1 Simulation Details

For the simulation task shown at the bottom of Figure 3.2, the blue object can move in a plane, i.e. there are two positional degrees of freedom and one rotational ($\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$). The state is defined by the raw coordinates for these three dimensions. The state space was manually restricted to some box area around the green object and scaled linearly to $[-1, +1]$ in all dimensions before being fed into a network. The initial state of the blue object was uniformly chosen from all states at the upper end of the box and combined with any possible rotation.

Actions are defined as 3D offsets between states. The resulting target poses are tracked by a task space position controller. After a step has been triggered, the simulation is designed to wait until the agent including both pegs has reached a stable position before the next action is applied. The terminal states were manually defined such that only states with both pegs correctly inserted were included. The reward is based on the euclidean distance in state space between the current state s and the goal g:

$$r = \exp\left(-\frac{||\mathbf{s} - \mathbf{g}||_2}{0.3}\right) - 1,$$

where the constant denominator was manually tuned and fixed for all experiments. Analogously to the clearance fit task, this reward function leads to rewards in $[-1, 0]$ and encourages fast solutions with few steps.

3.1.2.2 Real-World Task Details

For real-world evaluation, a 3D printed version of the task was used (see the top row in Figure 3.2). The lower object was 4.5 cm wide. The state space covers 8 cm from right to left, 5 cm from the lowest to the highest point in z direction and a rotation of up to 30°. One action could move the robot by up to 0.5 cm. To avoid collisions, a light-weight 6-axis force-torque sensor (KMS40, Weiss Robotics) was mounted on the robot's wrist. In contrast to the heavier compliance element from the clearance task, this sensor exerted less force on the end effector under extreme rotations. No passive compliance was used in this case though.

3.1.3 Form Fit Peg in Hole

Apart from passively sustained contact as in the previously described double peg in hole task, a robot may also have to actively apply force in order to achieve a peg in hole fitting task. In our final evaluations, I therefore chose a real industrial application: a shaft needs to be inserted into a ball bearing inside an eBike drive unit. To finish the tight form fit a force of about 10N and a determined push are needed to succeed. The task is illustrated in Figure 3.3 and a video from the publication associated with Chapter 9 can be found at `https://www.youtube.com/watch?v=Z_GcNbCWE-E`.

3.2 Hardware: Robots and Sensors

In this section, I will give details about the two robotic setups that I used for the real-world experiments.

Figure 3.3: Form fit peg in hole: the shaft for an eBike drive unit needs to be inserted into a ball bearing. For the last bit of motion, forces of approximately 10N and a determined push are required. The motor housing is turned upside down for better visibility of the ball bearing.

3.2.1 KAWADA Nextage Open Setup

The experiments on clearance fit in Chapter 5 and double peg in hole in Chapter 6 have been conducted with a KAWADA Nextage Open, which for instance in Japan is used in real-world production plants[2] (see Figure 3.4). For all experiments, I used a task space position controller that was provided through the manufacturer's API.

For the evaluation with the clearance fit peg in hole task in Chapter 5, I additionally mounted a custom passive compliance element as described in Rueb and Becker (2016) on one of the robot's wrists. It allows the peg to be passively displaced against the end effector and provides full 6D feedback about the positional and rotational displacement through a serial interface. The full setup, including also the built-in camera and light source on the KAWADA's second wrist, is shown in Figure 3.1.

[2] `http://nextage.kawada.jp/en/gallery/`, last retrieved: 23.08.2020

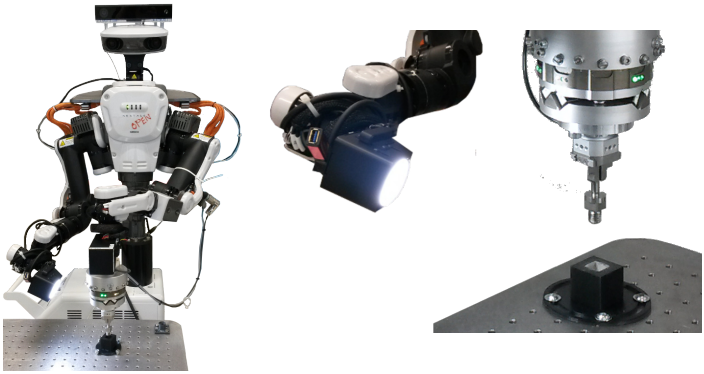


Figure 3.4: KAWADANextage Open with a custom passive compliance element (Rueb and Becker, 2016).

3.2.2 Franka Emika CoBot Setup

Finally, for the experiments in Chapter 9, I used a different setup that achieved compliance through joint torque control: The experiments were performed on a Franka Emika Panda CoBot (see Figure 3.5) with a custom control toolchain running at 1 kHz. For rigid body kinematics, dynamics, and for efficiently computing its derivatives, I employed the open-source library 'Pinocchio' (Carpentier et al., 2019). The required inertial parameters of our Panda robot were identified using an Linear Matrix Inequality (LMI) approach as presented in Sousa and Cortesão (2019). The end-effector contact wrench was estimated at 1 kHz real-time using an extended Kalman filter-based disturbance observer implementation taken from Giftthaler et al. (2018). The state and action definitions for the RL setup will be discussed in detail in Chapter 9.

Figure 3.5: Full setup for the experiments in Chapter 9 using a Franka Emika Panda CoBot and the form fit task with an eBike drive unit.

Chapter 4

Prestudies

In this chapter I will present two smaller prestudies which I conducted before all other experiments in this thesis. They concern very general questions and motivate the broader choice of methods to be examined in the remaining pieces of research.

4.1 Function Approximation with Discontinuities

4.1.1 Motivation

When robotic manipulation tasks are mathematically modelled, physical contact leads to discontinuities in the functions created. This can be illustrated with a schematic peg in hole task that is shown in Figure 4.1.

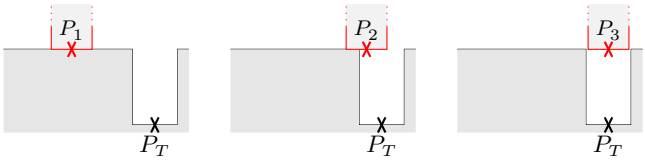


Figure 4.1: Illustration of discontinuities in peg in hole problems.

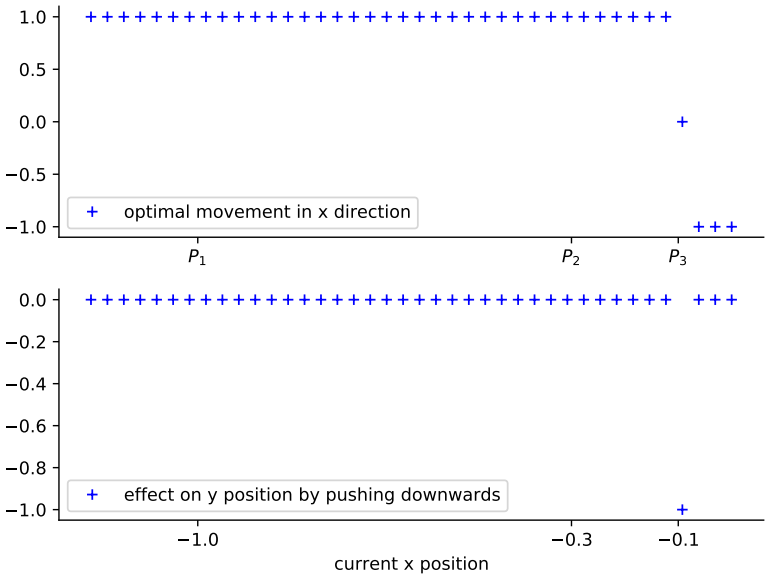


Figure 4.2: Illustration of discontinuities in contact-rich manipulation. This plot uses the task in Figure 4.1 to illustrates the sudden jumps in the x coordinate of the optimal policy (top) as well as in the dynamics function for a given action (pushing down in y direction, bottom).

The task is to insert the peg into a hole, where the agent only knows the tip position $\mathbf{p}_i$ which is supposed to reach the target $\mathbf{p}_T$. To achieve this from point $\mathbf{p}_1$, the shortest path would be to slide to the right through $\mathbf{p}_2$ until $\mathbf{p}_3$ and then to move downwards until $\mathbf{p}_T$ is reached. Thus, the optimal policy in 2D would be to move to the right first ($[1, 0]$) and then downwards ($[0, 1]$) which causes a discontinuity for both the x and y coordinate (see Figure 4.2 for illustration). Similar discontinuities arise when the robots dynamics are considered: The effect of the peg being pushed downwards in $\mathbf{p}_1$ and $\mathbf{p}_2$ is close to zero, but when $\mathbf{p}_3$ is reached the same push downwards will change the y coordinate of the tip. This discontinuity is illustrated in Figure 4.2 as well.

When RL is applied on such a manipulation task, both state and actions are very likely presented in continuous spaces. Thus, function approximation is typically applied and needs to approximate discontinuous functions such as the optimal policy or dynamics described above. I therefore investigated different function approximators and their ability to fit discontinuous functions.

4.1.2 Method

For the experimental evaluation, I considered the following step function:

$$\varsigma(x) = \begin{cases} +1, \text{ if } x \leq 0 \\ -1, \text{ else} \end{cases}$$

To simulate a setting where this function should be approximated, artificial noisy training data was created by sampling 70 values from a Gaussian around the true values:

$$\mathscr{D} = \left\{ \left(\mathbf{x}, \mathcal{N}\big(\varsigma(x), \sigma = 0.05\big) \right)_i \right\}$$

To be closer to an RL setting in which the agent performs strategically orchestrated exploration, 50 out of 70 training points ($\mathbf{x}$) were sampled in close vicinity of the step (i.e., $-0.2 \leq x \leq 0.2$) and only 10 in each of the two remaining intervals ($[-1, -0.2]$ and $[0.2, 1]$). The ground truth as well as sampled training data is illustrated in Figure 4.3.

I evaluated two of the most popular general purpose function approximators: GPs and MLPs. For the GPs, different versions of the Matern kernel were examined using a parameterization by length scale ι as well as an additional hyperparameter ν that controls the smoothness of the resulting function. I experimented with ν in $[0.5, 1.5, 2.5, \inf]$ and ι in $[0.01, 0.1, 1, 2]$. These values include the special cases of a Radial Basis Function (RBF) kernel ($\nu = \inf$) and the absolute exponential kernel ($\nu = 0.5$), both of which are very popular choices for kernel functions in GPs.

For the MLPs, I used a network with five fully connected layers. Each of these layers had 300 nodes and applied ReLU activation functions. All layers were initialized using He uniform sampling and optimized by an

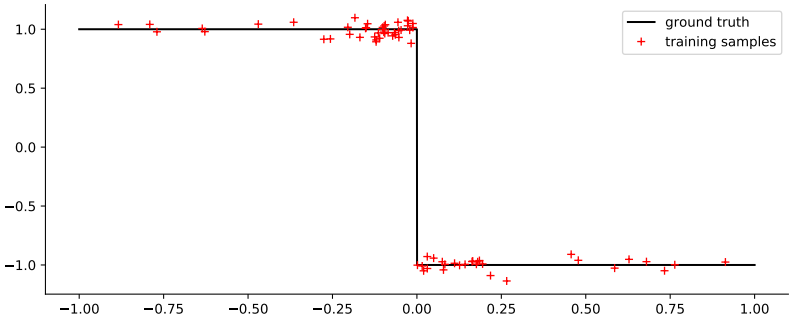


Figure 4.3: Toy example for function approximation on a discontinuous step function.

Adam Optimizer in 2000 training epochs using the full set of training data split into batches of size eight. The learning rate was set to values in $\left[10^{-6}, 10^{-5}, 10^{-4}\right]$.

4.1.3 Results

The results for MLPs are shown in Figure 4.4. The networks are able to learn the step function under all three learning rates. The main difference is that the lowest learning rate leads to very smooth predicted function and thus underfits the data very close to the step. In contrast, the highest learning rate leads to overfitting: the noisy training data is perfectly fit in those areas with less dense training samples.

In Figure 4.5, the results from in total 12 combinations of length scale ι and smoothness ν are illustrated. The mean function is shown as the solid green line, the shade highlights the GP's standard deviation. At first glance, the figure shows that most variants of GPs do not deal with the discontinuity well. Closer inspection shows the following problematic patterns: a smaller ι means that only values which are very close to each other are assumed to have similar values. This is useful for a step function and, as the first column of Figure 4.5 shows, results in a tight fit around the step. However, these function then lack the generalization capabilities in areas with less samples. The same column also shows that the predicted

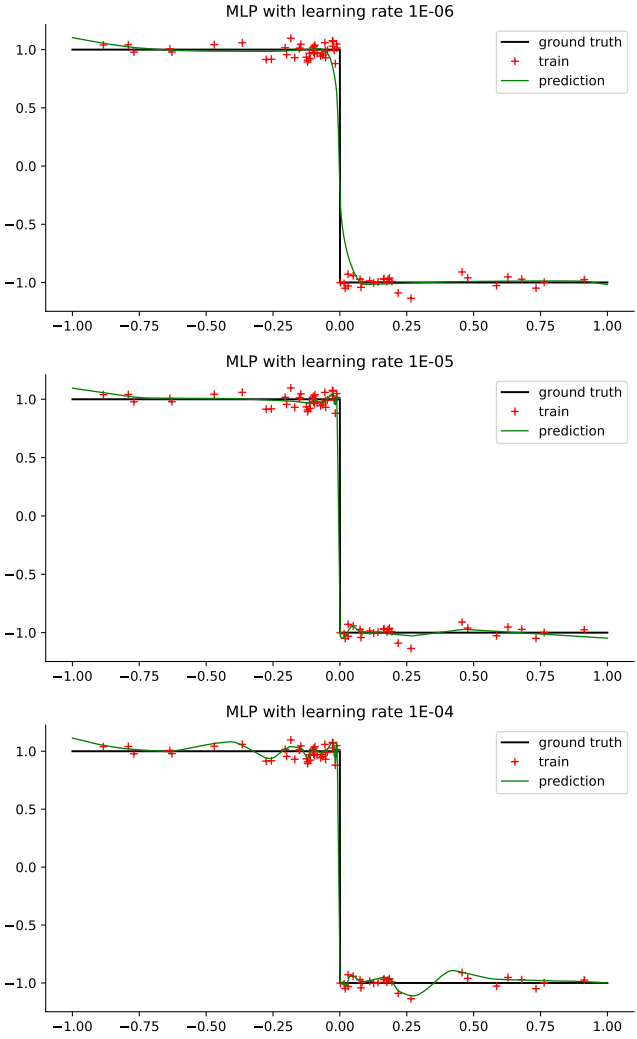
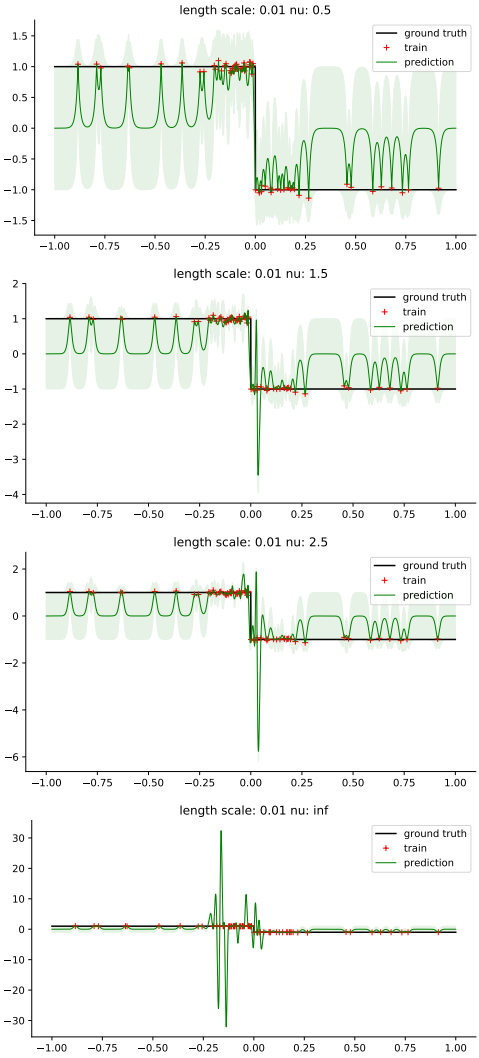


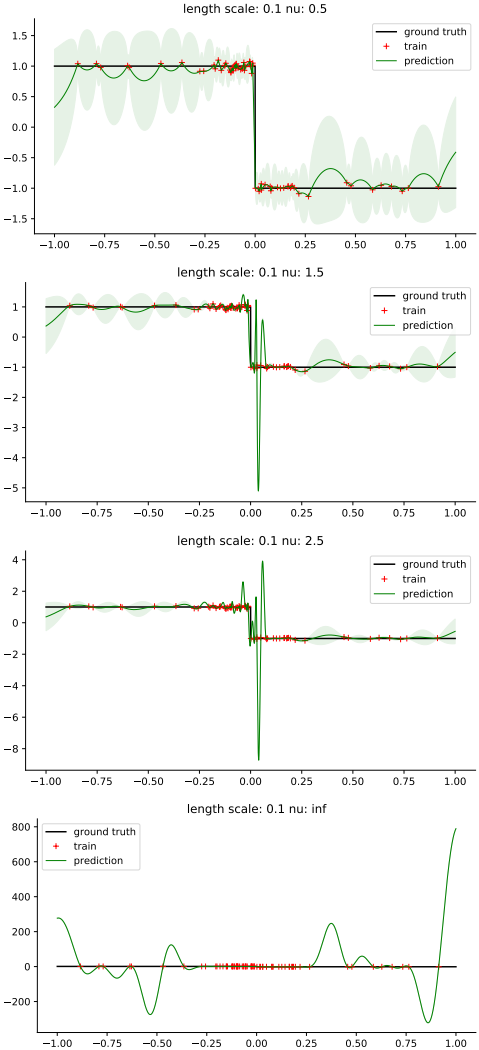
Figure 4.4: MLP results for the prestudy on function approximation with discontinuities: underfitting with the lowest learning rate (top) and overfitting with the highest learning rate (bottom).

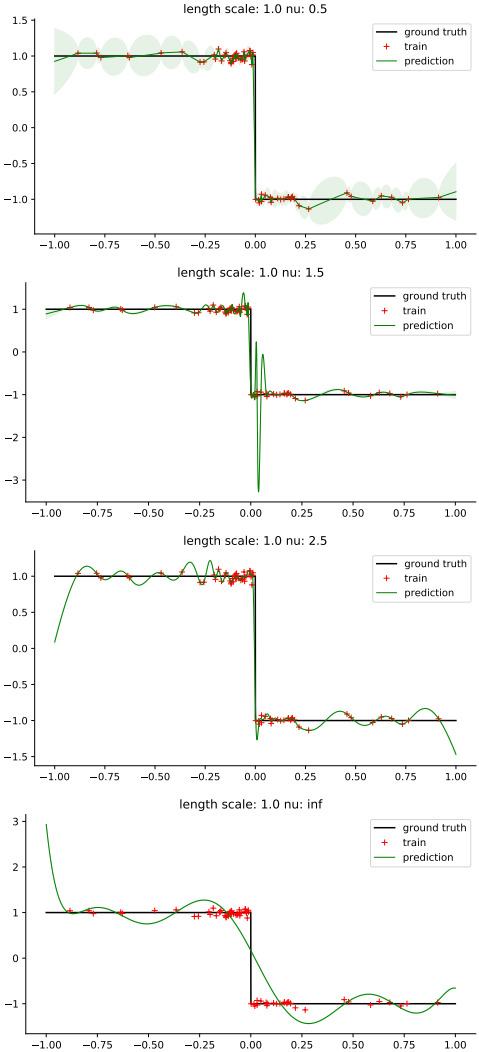
mean value between two points tends toward the global mean training value, even if all neighboring values have different values. For the other extreme, a long length scale of $\iota = 2$, which is shown in the last column of Figure 4.5, the result depends more on the chosen ν. For infinity, which assumes very smooth functions and is equivalent to the RBF kernel, the discontinuous function cannot be represented. For larger values of ν, the fit becomes better but also immediately leads to overfitting of the noisy training samples.

4.1.4 Conclusion

Overall, I note that all MLPs solve the task relatively well with only minor differences, while GPs are much more sensitive to the specific variant used and vary a lot in their performance. There is a large body of research about extensions of GPs, including works on automatic inference of suitable kernels such as Duvenaud et al. (2013) as well as works on non-stationary kernels, e.g., based on ANN as in Wilson et al. (2016). On the other hand there is also a growing body of work that reports of high-dimensional model-free learning methods which cope better with discontinuities by using ANNs; for instance Chebotar et al. (2017) in the area of policy search. I therefore decided to use MLPs for all the upcoming work in this thesis.


length scale: 0.01 nu: 0.5
ground truth
train
prediction
length scale: 0.01 nu: 1.5
ground truth
train
prediction
length scale: 0.01 nu: 2.5
ground truth
train
prediction
length scale: 0.01 nu: inf
ground truth
train
prediction


length scale: 0.1 nu: 0.5
ground truth
train
prediction
length scale: 0.1 nu: 1.5
ground truth
train
prediction
length scale: 0.1 nu: 2.5
ground truth
train
prediction
length scale: 0.1 nu: inf
ground truth
train
prediction


length scale: 1.0 nu: 0.5
ground truth
train
prediction
length scale: 1.0 nu: 1.5
ground truth
train
prediction
length scale: 1.0 nu: 2.5
ground truth
train
prediction
length scale: 1.0 nu: inf
ground truth
train
prediction

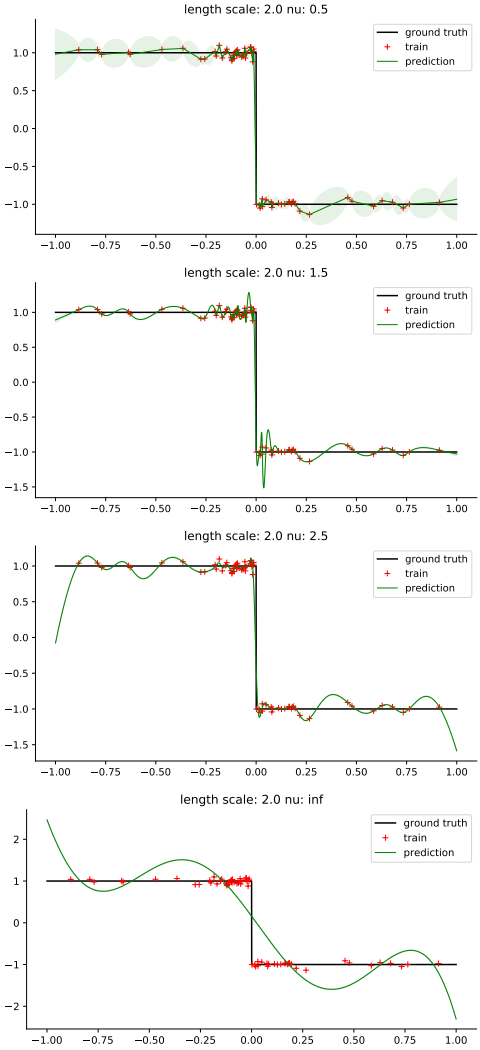


Figure 4.5: GP results for the prestudy on function approximation with discontinuities.

4.2 Uncertainty Estimates for Value Networks in Low-Dimensional Spaces

4.2.1 Motivation

In the previous chapter, I have shown that MLPs can relatively easily be used to approximate discontinuous functions. However, neural networks have one major drawback in comparison to GPs: they do not come with an innate mechanism to determine the prediction uncertainty. In the second prestudy, I therefore reviewed the two most popular methods from the literature that promise to alleviate this issue: dropout as proposed by Gal and Ghahramani (2016a) and bootstrapping by Osband et al. (2016).

When bootstrapping is used to represent uncertainty, a number of independent copies of the same network are set up and trained of different subsets of the training data. For large network structures, it is also possible to use one network with a number of heads to obtain different outputs. If a new test input is then fed to all networks, their outputs can be interpreted as samples from a distribution over outputs.

An alternative to model uncertainty in neural networks is dropout: applying dropout to the input of a layer means that each input node is randomly "dropped", i.e., is set to zero with some probability p. Gal and Ghahramani (2016a) have shown that this procedure can be interpreted as approximate Bayesian inference as thus can provide a theoretically grounded uncertainty estimate.

In the following, I will assess two aspects:

1. how well the methods perform on low-dimensional data, such as pose information of a robot. In prior work, the methods have often been evaluated on high-dimensional input in the form of images.

2. how well the methods perform on discontinuous data, similar to the previous prestudy.

4.2.2 Method

Both aspects were considered in the design of the function to approximate: Thinking of a state value function for a 2D maze in which the agent is supposed to reach a target, I designed the function to approximate such

that it approximately quantifies the length of the shortest path to the goal from every point in the maze—see Figure 4.6 for illustration. This function is defined on a low-dimensional support (2D) and it has significant discontinuities due to the walls of the maze.

Since formal performance criteria for the task of uncertainty estimation are hard to define, I based the experimental evaluation on the following expectations:

1. "good" uncertainty measures would lead to high uncertainty for input ranges with few training data.
2. if the predicted mean values are erroneous for some input, the uncertainty for this input should also be high. This could, for instance, be the case close to discontinuities which may be hard to approximate even with many training samples.
3. "good" uncertainty measures would lead to low uncertainty for all other cases, in particular for cases where many training samples were available and the correct mean function value has been learned.

To assess these criteria, I performed supervised training on four different training sets: for the "full" training data, 1000 samples were sampled uniformly from the full state space; for "center" training data, the 1000 samples were sampled from a Gaussian in the center of the state space; for "upper left" and "lower right", the data was sampled from a narrow Gaussian in one of the state space's corners.

Both methods, i.e., bootstrapping- and dropout-based uncertainty measures, were built on the same network architecture: a MLP with three fully connected layers with 200 nodes each and a ReLU activation function. The networks were trained using an Adam optimizer, a learning rate of 0.001, batch size 16 and 1000 full epochs. For bootstrapping, ten such networks were trained in parallel. Each training sample was uniquely assigned to one of the ten networks. For the dropout experiments, dropout with $p \in \{0.1, 0.25, 0.5\}$ was applied to every layer input at test time. All networks were trained without dropout.

4.2.3 Results

The results for each method on each training set are visualized with two plots: one illustrating the uncertainty, the second one the prediction error.

The results for "center" Figure 4.7, "lower right" Figure 4.9 and "upper left" Figure 4.8 can be used to assess our first performance criterion: in areas with few training samples, the uncertainty (shown on the left plot for each method) should be high. While this is somewhat the case for each method, bootstrapping seems to have the clearest shape. For example, for the "upper left" training data, only bootstrapping also has increased uncertainty in the upper right corner of the state space. This is particularly important for performance criterion (2), that uncertainty should correlate with areas of erroneous predictions. For the same results on "upper left" training data, there seems to be a particularly error-prone area in the upper right corner—but only bootstrapping also shows at least moderately increased uncertainty in this area.

The last training set, "full", has provided the algorithm with uniformly sampled samples of the full function. In this case, most uncertainty should be found around locations that are hard to approximate—e.g., locations of discontinuities. Evaluating the results in Figure 4.7 shows that the uncertainty from bootstrapping clearly focuses around the upper left wall, where the gap in values above and below the wall is particularly large. But also for walls with smaller gaps, some uncertainty is predicted. For all dropout results, the base uncertainty in areas with smooth and continuous function values is higher than for dropout. This is a negative performance indicator as we discussed for our evaluation criterion (3). Additionally, the uncertainty is less significantly increased for areas with discontinuities and the prediction error even for easy to predict areas are higher than for bootstrapping.

Combining these results, we conclude that it is indeed possible to approximate prediction uncertainty even for MLPs. Bootstrapping seems to perform better in a low-dimensional and discontinuous setting, which means that its uncertainty estimates are better correlated with the density of training samples, its uncertainty estimates are better correlated with prediction errors, e.g., for inputs close to jumps in the discontinuous target function, and its uncertainty estimates are lower for uncritical input ranges.

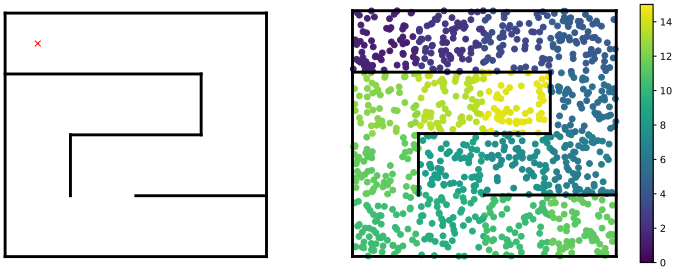


Figure 4.6: 2D example for uncertainty estimates: inspired by a task where an agent has to reach a target position in a 2D maze (left), an optimal state value function is defined. Samples from this state value function were drawn randomly and are illustrated on the right. Note that this function is discontinuous at the walls of the maze.

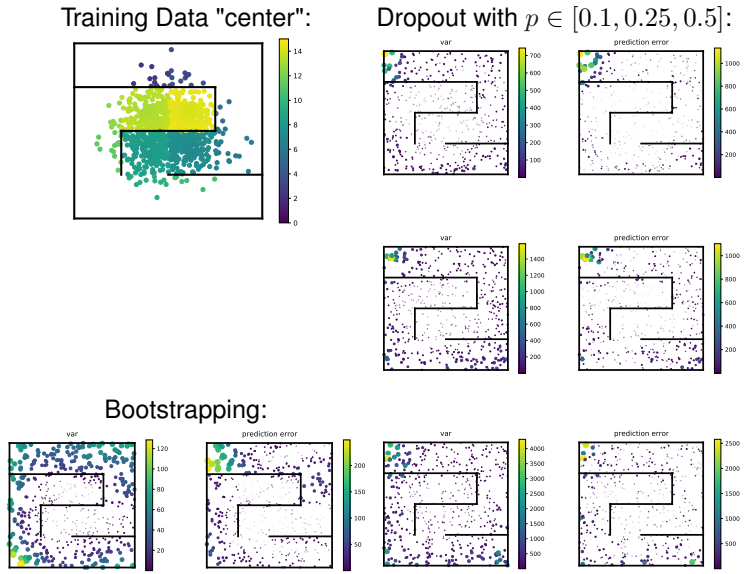


Figure 4.7: Prestudy on uncertainty under discontinuities. Results on "center" training data.

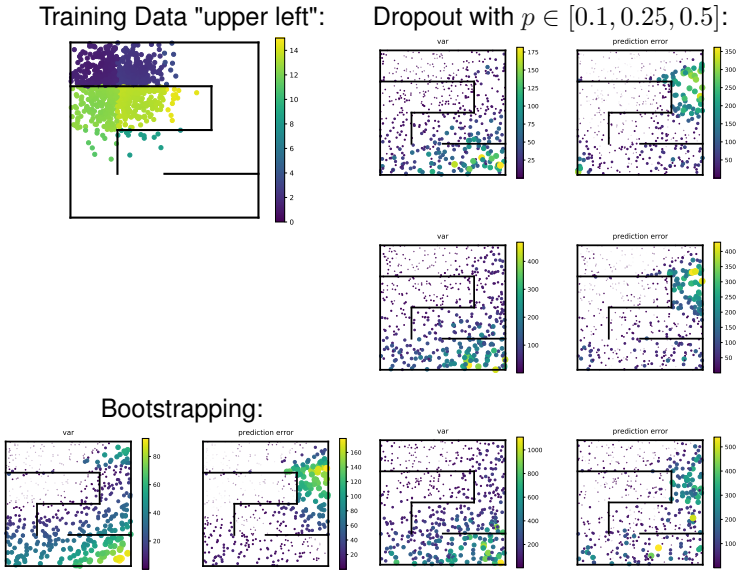


Figure 4.8: Prestudy on uncertainty under discontinuities. Results on "upper left" training data.

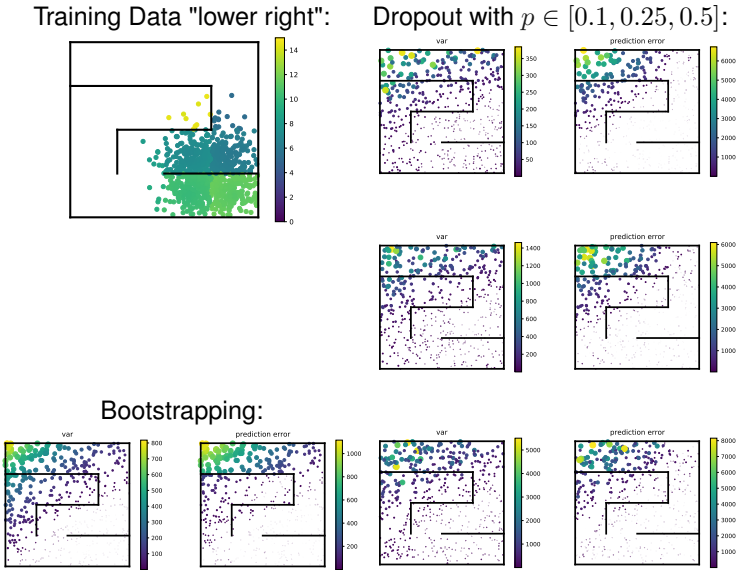


Figure 4.9: Prestudy on uncertainty under discontinuities. Results on "lower right" training data.

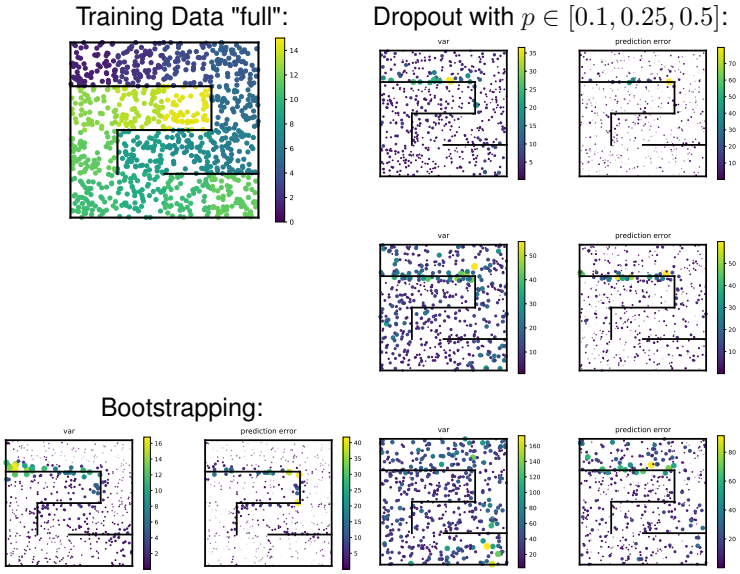


Figure 4.10: Prestudy on uncertainty under discontinuities. Results on "full" training data.

Chapter 5

Modalities to Efficiently Learn Peg In Hole Tasks

5.1 Introduction & Motivation

There is a wide variety of peg in hole tasks with different characteristics. The clearance fit variant is particularly easy to solve because it does not necessarily require contact between the peg and the second object. It is therefore possible to exploit supervised learning which is known to be more efficient than RL. Yet a clearance fit task can be used to illustrate the effects of different modalities, in particular visual and haptic feedback. In this work I therefore investigated a number of control strategies and function approximation strategies to fuse these pieces of information. Building on the results from the prestudy on function approximation under discontinuities (Section 4.1), I decided to work with MLPs as function approximators.

5.1.1 Disambiguating Sensory Input

The purpose of sensory input in a learning setting in general is to observe the robot state and allow the agent to make decisions based on its current state. For peg in hole tasks, this typically means that the pose of the peg relative to the hole should be identified. Which source of information is

most informative also depends on the state of the system as illustrated in Figure 5.1: When the peg is not in contact with the hole, visual information is most likely to describe the system state appropriately. The closer the peg gets to the hole however, the more likely also a vision system reaches its limit: either because the resolution is not enough for fine-grained pose estimation, or a monocular camera angle cannot detect small deviations in the direction that is perpendicular to the image plane. Also depth information may be insufficient in this case.
In contrast, estimated forces at the peg tip are only informative in special cases: without contact between the objects, no information at all is available. Depending on the point of contact, a single force measurement may also not be conclusive enough. For instance, the two peg poses in the middle of Figure 5.1 cannot be disambiguated based on force readings only. If the peg is in contact with a single edge of the hole, the measured forces become informative again: assuming that the peg on the right plot in Figure 5.1 is pushing down in z direction, and assuming that there is no parallel edge at the outer walls of the second object that the peg could be in contact with, then the force information is unique. Overall, one could either manually design or restrict the setting such that force readings are informative enough (as I will do in Chapter 9 to focus on force-driven interaction), or information from multiple sources must be combined as I will do in this chapter. How to best combine sensory information from different sources is an open research problem that is typically summarized as *sensor fusion*. To focus on open questions in this field, I chose to first evaluate different control settings in an otherwise simplistic setting: the task was designed such that supervised learning could be applied, and the clearance was large enough to avoid issues with high contact forces.

5.2 Related Work

Supervised learning is known to be much more sample efficient than RL. This is because the function approximator is trained to fit fixed labels, while in RL these labels are estimated over time and thus are subject to change. In areas such as computer vision and natural language processing, supervised learning combined with ANNs has lead to substantial progress in recent years (Voulodimos et al., 2018; Young et al., 2018). When robots are supposed to learn how to perform a certain task, supervised learn-

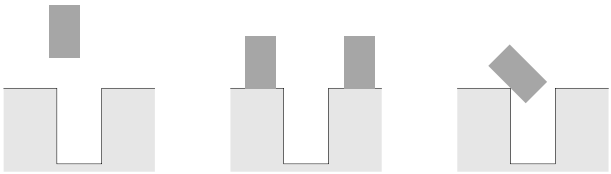


Figure 5.1: Disambiguation of sensory input: On the left, the peg is not in touch with the object surface and thus no haptic feedback can help to locate the peg. In the middle, two peg positions are shown in which the peg is in contact and yet, the haptic feedback is ambiguous. In some cases, as illustrated on the right, haptic feedback can be useful.

ing is much harder to apply: Often, optimal controllers that could create automatic labels are not known—or if they are known, the task is solved and no further learning required. Levine et al. (2016a) have proposed one notable exception: local models are fit using iLQR, an iterative approach to fit Linear Quadratic Regulators (LQR). The LQR problem is a classical setting from control theory, where the optimal controller for a system that is described with linear differential equations is computed under a quadratic cost term. The derived optimal controllers are executed and the resulting experience is recorded, i.e., the resulting transitions consisting of state-action pairs and an associated reward in RL terminology. Combining the experience from all iLQR solutions, a large dataset is formed for supervised learning. In a complex iterative scheme, an ANNs is trained to imitate all iLQR solutions in a single network. Importantly, the network may operate on a different observation or state description than the iLQR problem did. For example, the collected experience may consist of images that the iLQR policies did not use. In the clearance task I used for evaluation in this chapter, the training set for a supervised training setting can be derived more easily, but I will exploit the same idea that additional information can be exploited to create the training data which will not be passed to the network policy.

The aforementioned work has demonstrated the possibility to train neural network policies end-to-end from raw images to control signals directly (Levine et al., 2016a). For complex contact-rich manipulation tasks, haptics play an important role however and it therefore seems promising to jointly work with both (high-dimensional) visual and haptic sen-

sory input. In model-free learning for contact-rich applications, prior work tends to work with more low-dimensional data instead of images, e.g., with a vector including object positions and velocities as well as applied torques as input (Levine et al., 2015). Low-dimensional representations may also be learned from high-dimensional and potentially visual input, e.g., through an autoencoder structure (Van Hoof et al., 2016). In this work however, a model-free controller is designed which learns to combine high-dimensional images with low-dimensional sensory input. Since the time when this work was carried out in 2016, more work has been published on directly integrated multimodal feedback in model-free learning systems. I will discuss those in Section 5.5.

5.3 Experimental Setup

All experiments in this chapter were conducted on a dual arm robot—the KAWADA Nextage (see Figure 5.2 and the introduction in Section 3.2). A custom passive compliance module (Rueb and Becker, 2016) is mounted on one of the wrists. It allows for some displacement between the robot's wrist and End Effector (EEF) and provides full 6D feedback $\mathbf{c}$ about the current positional and rotational displacement through a serial port interface. The second robot arm provided the light source and camera images from a static view point for our experiments.

5.4 Method

As discussed in Section 1.2 in the context of the free lunch theorems, sample efficiency in learning can typically be traded off against generalization. To enable sample efficient learning from multimodal high-dimensional input, I therefore make use of two core assumptions: First, the target position may not be known at test time, but for training I can use it to automatically create a labelled dataset for supervised learning. Second, I exploit the geometric structure of the insertion task by using a residual formulation so the downward part of the motion does not need to be learned or discovered by the agent.

Figure 5.2: System consisting of a camera at the left robot arm and the right arm carrying a compliance element and the metal object to be inserted into the white square hole inside the black target object on the bottom of the image. The hole is slightly larger than the hole; in other words this is a clearance fit (see also Figure 3.1).

5.4.1 Residual Policy Formulation

For many manipulation tasks, it is intuitively possible for humans to sketch a rough policy that represents a good initial guess. For the case of peg insertion, I therefore choose to represent the policy as a residual policy. That is, the policy consists of two policies, one learned and one constant, that are combined.

In this case, the constant part of the task space policy is a hard-coded controller for the z coordinate of the EEF. Different control loops for the z coordinate task will be compared experimentally. For instance, I have experimented with a control loop that also allows the robot to move upwards again in case the contact forces exceed some manually tuned threshold.

The x and y coordinates of the EEF are learned and predicted by a neural network that takes an image of the scene and haptic feedback as input. More precisely, the network outputs are interpreted as relative EEF positions in task space. Overall, this means that only two instead of three positional dimensions need to be learned by the agent, which typically implies it can learn faster due to a smaller search space. A small prestudy had shown that such a residual formulation is particularly useful when the optimal values for x or y are on substantially different scales than the value for z; for instance because the EEF should move slowly in z direction but x and y can be much larger because their offset to the target position is supposed to be reduced quickly.

5.4.2 Data Collection

In assembly tasks for production processes which served as inspiration for the tasks in this doctoral thesis, the target positions is often roughly known: a theoretical value can be given but due to random influences or production tolerances, the true position of the object will deviate and thus be unknown at test time. For training however, it is easy to create a setting in which the target position is known, either because it is hard-coded or because a precise robot arm was used to put the target object in place or simply because the experiment can be repeated with few known objects that therefore do not suffer from random production tolerances.

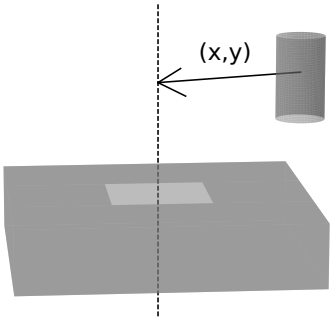


Figure 5.3: Automatic labelling: each randomly sampled position is annotated with the relative position that the EEF (here, only the peg is drawn for clarity) has compared to the center of the hole.

Using the target position, it is possible to automatically create a dataset for supervised training: 10,000 positions were uniformly sampled from a cubic space above the target object as well as 5,000 positions with surface contact. The label for each position is the x and y offset between the EEF and the center of the hole (as illustrated in Figure 5.3).

For the samples without contact (i.e., with the EEF being well above the target object), the robot was maneuvered to the randomly sampled positions. At the target position, an image was taken and stored together with the haptic feedback from the compliance element and the automatically computed label. Exemplar images are shown on the bottom of Figure 3.1.

For the positions which lead to a contact between peg and target object, also the direction of movement matters. Therefore, additionally to the already sampled position to take the image, also a second nearby position was sampled from which the EEF moved towards the first target position. The resulting dataset thus contained various positions and contact angles which lead to different force readings.

5.4.3 Network Architecture and Training Procedure

Based on this dataset, I trained an ANN to map from an image and compliance feedback to the task space movement in x and y direction that centers the EEF above the hole. Preliminary studies had shown that the

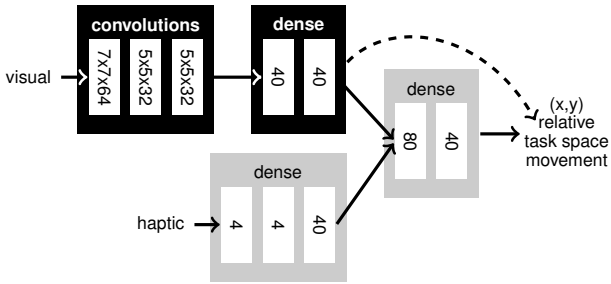


Figure 5.4: Network architecture: the black parts process visual input, i.e., images. It was trained starting from pretrained filters from GoogLeNet trained on ImageNet (Szegedy et al., 2015) to predict the relative task space movement (x, y). In a second step, the output (dashed line) was discarded and instead the gray parts that process haptic feedback from the compliance element were added. To train the grey part, all weights from the black part of the network were frozen. The numbers inside the white blocks specify layer sizes.

training procedure for such a network with different input modalities was very likely to diverge. Therefore, I designed the structure shown in Figure 5.4, which consists of two parts that were trained in separate phases.

First, one stream of the network was set up to process the visual input: pretrained convolutions from GoogLeNet trained on ImageNet (Szegedy et al., 2015) were topped with two fully connected layers. This part of the network is highlighted in black in Figure 5.4 and was trained to predict the relative task space movement the EEF would have to execute to hover above the hole ((x, y), illustrated by the dashed arrow).

In a second stage, the output layer was removed. Instead, a second block (illustrated in gray) was introduced: a fully connected network that processes the haptic input. The output of both the image block and the haptic block has 40 neurons which are then concatenated. Another fully connected block on top was then trained to predict the relative task space movement again, while the already trained parts for visual processing were frozen.

In the experiments section, I will present an ablation study that gives further insights into how important the different parts of the networks are and how they can be combined with different control loops at test time.

Network Implementation Details

I used the Xavier initialization and Batch Normalization for all layers as well as the Adam optimizer and ℓ_2 regularisation with coefficient $\lambda = 0.005$. The fully connected layers were trained using dropout with probability $p = 0.1$ for improved stability. As activation function for the fully connected layers, the exponential linear unit (eLu) function was chosen which is a smooth variant of the ReLU function:

$$\text{eLu}(x) = \begin{cases} \exp(x) - 1, \text{if } x < 0 \\ x, \text{else} \end{cases}$$

5.4.4 Control Loop Variants

Since the network policy only predicts a relative movement in x and y direction, the height of the EEF can be controlled separately in our residual policy formulation. Since in most applications for peg insertions, the hole is in an upright position on a table top, I made the assumption that the EEF starts higher in z direction than the target object's surface. A simple constant policy could then use a constant step (-3) downwards in z direction at each time step. Using the haptic feedback from the compliance element, it is possible to detect when the forces in z direction become too high, that is when the robot gets stuck on the surface of the target object rather than inserting the peg. Based on these assumptions and sources of information, I derived the following controllers which will be compared in the upcoming experiments:

A) a vision-only, open loop baseline.
Only the vision part of the network (black in Figure 5.4) is used, and an open-loop controller decreases z at each step. Due to the constant decrease in z direction, the robot inevitably either reaches the hole or gets stuck on the surface of the object. The haptic feedback the compliance element provides is a six dimensional vector describing the positional and rotational displacement between peg and EEF. If the ℓ_2 norm of the displacement between peg and EEF exceeds a manually tuned threshold c_{max} , the episode is stopped and counted as a failure. The z coordinate of the position of the

EEF is used to detect whether the peg is fully inserted into the hole and thus whether an episode was successful. Pseudocode for one episode under this controller is given in Algorithm 1.A.

Algorithm 1.A Test Time Control Loop: Open Loop, Vision Only

Given:
step size $\mathfrak{Z}$
threshold on compliance feedback c_{max}

1: move EEF to random initial position
2: **while** EEF above target height **do**
3: take image $\mathbf{I}$, get compliance feedback $\mathbf{c}$
4: **if** $||\mathbf{c}||_2 \geq c_{\text{max}}$ **then** ▷ high contact forces
5: **break** ▷ episode failed
6: **end if**
7: network prediction $(x, y) = \text{NN}(\mathbf{I})$
8: move EEF by $(x, y, -\mathfrak{Z})$ ▷ policy step
9: **end while**

B) vision only, closed loop.
Only the vision part of the network is used to predict x and y of the EEF movement again. However, the feedback from the compliance element is used to also increase the z position of the EEF again in case it pushes too hard into the target object, i.e., in case a manually tuned threshold c_{max} for the ℓ_2 norm of the displacement is exceeded. Pseudocode for this variant is given in Algorithm 1.B. Effectively, this is a manually tuned resetting behavior which could theoretically lead to infinite episodes. The episode is therefore stopped after 40 steps, and counted as a failure if the peg is not inserted (indicated by a high z coordinate for the EEF position as for variant A).

C) vision & haptics, closed-loop.
This variant is equal to B but the full network is used, i.e., both visual input in the form of camera images as well as the displacement feedback from the compliance element is used. In Algorithm 1.B, this would modify line 8 to $(x, y) = \text{NN}(\mathbf{c}, \mathbf{I})$.

Algorithm 1.B Test Time Control Loop: Closed Loop, Vision Only

Given:
step size $\mathfrak{Z}$
force correction step size $\mathfrak{Z}_{\text{up}}$
max. steps per episode N
threshold on compliance feedback c_{max}

move EEF to random initial position
for t in $\{0..N\}$ **do** ▷ time steps
  take image $\mathbf{I}$, get compliance feedback $\mathbf{c}$
  if $||\mathbf{c}||_2 \geq c_{\text{max}}$ **then** ▷ high contact forces
    move EEF by $(0, 0, +\mathfrak{Z}_{\text{up}})$ ▷ new in Version B: move up
  end if
  network prediction $(x, y) = \text{NN}(\mathbf{I})$ ▷ Version C: compute $\text{NN}(\mathbf{I}, \mathbf{c})$ instead
  move EEF by $(x, y, -\mathfrak{Z})$ ▷ policy step
  if target height reached by EEF **then**
    break ▷ success
  end if
end for

For all experiments, $\mathfrak{z}$ was fixed at 1.5mm, the upward movement when the robot got stuck, $\mathfrak{z}_{\text{up}}$ was constantly set to 2mm.

5.4.5 Experimental Results

To assess the previously introduced variants of control loops, I executed 100 test episodes (consisting of up to 40 time steps each) for each variant starting from different initial positions above the hole. The numbers in Table 5.1 summarize how many out of these runs were successful: Variant A) with camera input only succeeded in 77/100 runs. Using closed-loop controller B) that moves the EEF up when the contact forces exceeded a manually defined threshold, the success rate increases to 85%. The full network architecture trained on visual and haptic feedback (C) solves the task in all 100 trials.

short		description	success rate
A)	vision +	passive compliance	77/100
B)	A) +	closed loop controller	85/100
C)	B) +	compliance feedback	100/100

Table 5.1: The test time control loop variants A, B and C were run for 100 trials each. One trial was successful if the robot reached the target position in less than 40 steps without getting stuck, i.e., without causing contact forces above a manually defined threshold c_{max}.

For qualitative assessment, all experiments were recorded as videos which allow the following observations:

- First, there is a positive example for variant A) in Subsection A.1.1 showing how the robot successfully centers the peg above the hole.
- Second, there is an example in Subsection A.1.2 where the peg first gets stuck on the left corner of the hole (which would be a failure with controller A). Due to controller B however the EEF is moved up and the insertion trial continues. Finally, in this case, the peg can be successfully inserted.
- The failure case that is also presented in Subsection A.1.2 illustrates that this upward movement is not always enough to solve the task. In the example, the robot gets stuck on the rear left corner and

by successive upward movements gets stuck again multiple times along the rear edge until it reaches the rear right corner where the timeout was reached.

- Such a behavior did not occur with control variant C as illustrated in a similar setting in Subsection A.1.3: the robot also gets stuck in the rear left corner but then is able to interpret the compliance feedback such that it successfully corrects the peg position.

5.5 Discussion

This study emphasizes a number of aspects that contribute to successful and sample-efficient solutions for robotic learning in manipulation tasks:

First, for optimal results, different input modalities need to be combined. On a theoretical level, the examples in Figure 5.1 illustrate why for peg in hole tasks neither vision nor haptics alone can be sufficient to disambiguate the system state. The empirical evaluation has confirmed this hypothesis and demonstrated that successful and robust insertion skills cannot be learned without haptic feedback. Even with an improved test time controller which resets the robot when contact forces exceed some threshold, the resulting policy was significantly inferior to a fully learned policy based on information from both sensory modalities.

Second, learning from multiple modalities may require new network architectures. Training a single network from scratch on input from multiple sources (which often produce raw data across different scales and with different variances) is challenging. Instead I designed an incremental network architecture that could be trained in two phases. The research area of sensor fusion has been quite active in the years after this work was originally carried out. Several papers have proposed novel methods how to use deep learning to extract a joint abstract representation from multiple input sources, e.g., Lee et al. (2019a).

Third, I explored ways to integrate prior knowledge about the task at hand in this experiment: Similar to Levine et al. (2016a), additional knowledge about the task is used during training to create data on which a test time policy with a different input can be trained. More precisely, I used the known target position at train time. Given the target position, a good solution is already known: moving in x and y direction such that the peg

hovers above the hole. Thus, it is possible to employ supervised learning, which is known to be much more efficient than other techniques such as RL. After training, the test time controller takes an image and compliance information as input—but does not need access to the target position.

Fourth, the learning task can be simplified even further through a residual policy formulation: expert knowledge, and in the case of insertion tasks even human common sense, can often provide simple strategies that roughly summarize the desired behavior and only need to be slightly adapted for successful manipulation. In the case of peg insertion that I considered in this work, this simple strategy is a constant movement in z direction. The algorithm therefore does not need to learn this part and can focus on the residuals in x and y direction instead. Since the work presented in this chapter, research on residual policies has progressed also for model-free learning (Silver et al., 2018; Johannink et al., 2019) as I will discuss again in Chapter 9.

5.6 Limitations and Open Questions

The study presented in this chapter is one step towards sample efficient learning in contact-rich robotic manipulation tasks. It has shown how network architectures, prior knowledge and residual policies can contribute to increased sample efficiency. However, manipulation tasks in our every day lives as well as in real-world robotic tasks, e.g., in industrial assembly, are much more sophisticated than the simplified clearance fit peg insertion task that was considered here. The study thus leaves the question open to what degree results from this chapter generalize in more complex settings. The upcoming chapters in this thesis will deal with such tasks of increasing difficulty.

When tasks become more complex, creating labelled data for supervised learning becomes harder. In Chapter 9 for example, I will apply a similar residual policy formulation for a real industrial assembly task. For this particular application no optimal controller is known; even under the assumption of known target positions. Thus, supervised learning is not applicable. The question which of the effects from supervised learning this chapter transfer to an RL setting remains open for now.

Further open questions are beyond the scope of this doctoral thesis: First, the image acquisition I used in this experiment is limited. Certainly, binocular images or depth information could help the agent in performing an insertion task. This shifts the sensor fusion problem from integrating haptics with vision to fusing multiple input channels.
Second, all manually designed components in the experiment could either be improved or replaced by a learning system: all thresholds as well as the test time controller that moves the EEF up in case of contact.
Finally, the disambiguities in system state that I illustrated in Figure 5.1 could also be resolved by keeping a history of observations. Whether such a history of sensory input is still superior when based on multiple input modalities is an interesting open question for future research.

Chapter 6

Exploration through Approximate Uncertainty-Based Trajectory Optimization

For insertion tasks with large clearance, such as in the last chapter, it is often feasible to design good datasets for supervised learning. For more complex tasks, when manually crafted data is not available, RL becomes more appealing: the agent can be enabled to detect solutions by itself. However, the harder the task the less likely a standard RL agent will discover optimal behavior without explicit guidance in a feasible amount of time. In this chapter, I therefore examined whether it is possible to employ model-free deep RL combined with planning to quickly generate informative data for a manipulation task. For planning, I will not use an accurate model but a coarse steering function, which can be seen as one way to inject prior knowledge about the robot at hand into model-free RL.

6.1 Introduction & Motivation

When enough data is available, e.g., by collecting experience from several robots over the span of multiple months, model-free deep RL has been proven to outperform many conventional approaches (Levine et al., 2016b). In particular for contact-rich manipulation tasks, where models are hard to obtain, purely data-driven deep learning approaches have outperformed model-based approaches (Fazeli et al., 2017). However, collecting sufficiently much data real-world settings can be slow, costly and might need human supervision throughout the collection process.

One reason for this sample-inefficiency is exploration, which is notoriously hard for model-free approaches. In general, only dithering strategies can be used. Alternatively, exploration can be circumvented to some degree by exploiting expert demonstrations (Schaal, 1996) or known local optimal controllers (Levine et al., 2016a). In this work, I investigated whether model information can be used to plan optimal exploration behavior. In particular, I use an estimated Bayesian-like Upper Confidence Bound (UCB) on the advantage function to plan optimal exploration trajectories. For an efficient approximation of the UCB, I extend the critic network to consist of separate streams for state-value and advantage, allowing to estimate prediction uncertainty only for the advantage stream using ensembles of networks. The resulting exploration data can then be used analogously to demonstrations in state-of-the-art learning from demonstrations (Vecerik et al., 2017) and, as I will show, outperforms purely model-free RL with dithering exploration in terms of sample efficiency.

The approach is evaluated on the novel double peg in hole paradigm which is discussed in detail in Subsection 3.1.2 and illustrated in Figure 3.2. Task A, with the two pegs stiffly connected to the upper bar, is equivalent to traditional peg in hole: given the right pose, it is possible to insert the peg while mostly avoiding contact. With dangling pegs and off-axis joints (Task B, see Figure 3.2), sustained contact is needed to reach the goal pose. This task principle is similar to the clip insertion investigated in Vecerik et al. (2017) and was also 3D printed for evaluation on a real robot (see `https://youtu.be/JTfeHhWSb0Y` and Section A.2 for key frames).

6.2 Related Work

If prior knowledge about the task is available, model-based RL can use this information for efficient exploration (e.g., Houthooft et al. (2016)). In the presence of contacts however, models are notoriously hard to obtain (Fazeli et al., 2017). Even when models are theoretically available, robotics practitioners might struggle with limited knowledge about the materials at hand or the availability of suitable environment representations.

Without imposing models or structure on the functions to learn, exploration is the main bottleneck. There are different dithering strategies ranging from ϵ-greedy (Sutton and Barto, 2018) and Gaussian noise to more sophisticated processes like the temporally correlated Ornstein-Uhlenbeck (Lillicrap et al., 2015) or adaptive parameter space noise (Plappert et al., 2017). However, all dithering strategies mainly explore locally around states that the current policy tends to visit and it can take a long time until the relevant part of the state space is discovered—even more so, if some area is only reachable after a specific series of actions.

Model-based and model-free methods have also been combined: For instance, guided policy search turns RL into iterative supervised learning from model-based solutions (Levine et al., 2016a). This approach is limited to tasks where model-based solutions are known or easy to obtain though. In this chapter, I will make use of a coarse steering function which can be seen as a simplistic type of model knowledge within a model-free learning framework.

Reward shaping can help to lure the agent into relevant parts of the state space but it typically requires experience and manual effort or otherwise easily leads to unintended local minima (Lowrey et al., 2018). Count-based exploration schemes aim to explore the full state space uniformly by keeping track of which area was visited how frequently (Brafman and Tennenholtz, 2002; Tang et al., 2017). This behavior is often seen as an analogy to human *intrinsic motivation* (Chentanez et al., 2005), but in many cases it is unrealistic to cover the full state space. If an agent *knows what it knows* (Li et al., 2008; Lopes et al., 2012) it can deliberately explore areas it is unsure about.

Model uncertainties from neural networks can formally be obtained from Bayesian Neural Networks that consider distributions over weights. Since those are often intractable, there are approximations such as network ensembles or dropout at test time (Osband et al., 2016; Gal and

Ghahramani, 2016b), which I explored in Section 4.2 already. Using information-theoretic criteria like the UCB, these approximations can be used to speed up learning (Chen et al., 2017). Still, existing strategies for model-free RL only explore greedily for a current state. In this chapter I combine UCB-based exploration with global state space exploration via trajectory optimization to also explore far-away states if they seem promising in terms of their UCB.

To perform global exploration, I utilize a coarse *steering function*: that is, I assume that a known function can return actions that will guide the agent approximately towards a given goal state. In contrast to model-based learning, this steering function is only used for exploration while the learned policy remains model-free. Therefore the method is quite robust to perturbations in this function (as I will show in Subsection 6.4.2). Importantly, this exploration scheme even generates informative data if it does not solve the task, as I will discuss in my evaluations.

Supervised learning from fixed input-output pairs is generally much more sample efficient than iterative RL where changing optimization targets introduce non-stationarity. Instead of model-based solutions also expert demonstrations can help speed up learning (Schaal, 1996) but they are often costly to acquire. When available, already single demonstrations can be enough for robots to learn (Englert and Toussaint, 2018). In this work, I follow the idea that learning from demonstration could be used on top of automatically generated demonstrations. When exploration is separated from policy learning, the modalities and type of information that is used can change between the two stages (similar to the previous chapter or for instance Levine et al. (2016a)). Crucially, exploration can be much faster when the value function in the exploration stage can change quickly without hindering convergence of the actor network which mostly matters in the policy learning stage.

Convergence of the function approximation can be additionally sped up by separately learning the value function and the advantage as opposed to learning their sum, the Q-value (Greensmith et al., 2001). This idea has already been transferred to deep learning by Wang et al. (2016) but their method loses the original semantics of the state value V and the advantage A. I will introduce novel update rules that enable separate training of both streams, preserving their semantics and thus allowing to build policies on the actual advantage.

6.3 Method

The problem of RL can be split into two stages: exploration and learning from demonstration. For the second stage, all samples from the first phase are slowly added to a new replay memory, similar to Vecerik et al. (2017). For the first stage I combine the following components to efficiently guide exploration for DDPG-like Q-learning (Lillicrap et al., 2015), each of which will be discussed in greater detail in one of the following paragraphs:

1. I structurally encode the decomposition of Q-values into state-value and advantage (cf. Eq. (2.2)) in a network architecture with two streams similar to Wang et al. (2016). This reduces the gradient variance which leads to faster and more stable training. To preserve the semantics of both streams, I introduce a novel training update scheme.

2. Exploiting this network structure, an ensemble of networks in the advantage stream can be used to approximate model uncertainty. I will argue that only this portion of uncertainty in the Q-function is relevant for exploration and deriving a policy.

3. Using trajectory optimization under coarsely approximated dynamics, the agent can globally explore areas of high model uncertainty by choosing trajectories such that the sum over the UCB on the advantage for a set of waypoints is maximized.

6.3.1 Advantage Networks

Decomposing the state-action value Q into state-value V and the advantage A is formally equivalent to using V as a control variate which leads to more stable learning if V correlates strongly with Q (but A does not) (Greensmith et al., 2001; Wang et al., 2016). If, for instance in my environment, there is a reward at each time step, then V and Q sum up (discounted) rewards over potentially long trajectories, while the advantage is the difference between V and Q. Effectively this means that V and Q might be in a very large range of absolute values, while A might be on a range of very small numbers. In those cases, V dominates gradients and uncertainty measures derived from Q. If a policy is derived from Q, it is

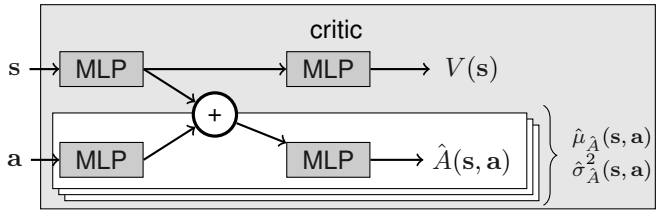


Figure 6.1: Bootstrapped Advantage Network: This network architecture consists of four MLPs, each consisting of four fully connected layers—the first three using leaky ReLU activations (Maas et al., 2013), the last one using linear activations. The advantage stream of the network is bootstrapped, i.e., there are B copies of the lower part of the network.

also dominated by V although actually only A matters:

$$\arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a}) = \arg\max_{\mathbf{a}} \big(A(\mathbf{s},\mathbf{a}) + V(\mathbf{s})\big) = \arg\max_{\mathbf{a}} A(\mathbf{s},\mathbf{a}) \quad (6.1)$$

I structurally encode the split into V and A as two fully separate streams within the network architecture that consists of four MLPs: The inputs, i.e., state and action, are each processed by one MLP. The outputs of both MLPs are added element-wise and fed into a third MLP that predicts the advantage $A(\mathbf{s},\mathbf{a})$. The output of the state-MLP is also fed into the fourth MLP to predict $V(\mathbf{s})$ (see Figure 6.1 for illustration). In contrast to Wang et al. (2016) the network branches are trained separately as described in the following sections.

Since the advantage of an action over the state-values associated with the optimal policy cannot be larger than zero, I use a ReLU activation on the advantage network and change the sign of the output.

6.3.2 Greedy Exploration using Bootstrapped Advantage Networks

Bootstrapping uses ensembles of networks to get an uncertainty estimate (Osband et al., 2016) which can then be used for exploration. In preliminary investigations (cf. Section 4.2), bootstrapping has outperformed dropout-based methods for uncertainty estimates for models with low-dimensional raw input signals.

To determine the next action to take for exploration, acquisition functions inspired by Bayesian Blackbox Optimization (BO) can be used, e.g., the UCB in a Q-learning setting (Chen et al., 2017):

$$\pi(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathcal{A}} \text{UCB}_{\widehat{Q}}(\mathbf{s}, \mathbf{a}) \tag{6.2}$$

$$= \arg\max_{\mathbf{a}\in\mathcal{A}} \left(\hat{\mu}_{\widehat{Q}}(\mathbf{s}, \mathbf{a}) + \kappa \cdot \hat{\sigma}^2_{\widehat{Q}}(\mathbf{s}, \mathbf{a})\right) \tag{6.3}$$

where κ is a hyperparameter that trades off variance and mean[1]. $\hat{\mu}_{\widehat{Q}}$ denotes the estimated mean of the distribution over the predicted Q-values $\widehat{Q}$ and $\hat{\sigma}^2_{\widehat{Q}}$ its variance.

A policy that greedily optimizes Q depends on A but is independent of V:

$$\pi(\mathbf{s}) = \arg\max_{a\in\mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \overset{Eq.\ (6.1)}{=} \arg\max_{a\in\mathcal{A}} A(\mathbf{s}, \mathbf{a}) \tag{6.4}$$

Therefore, I only add bootstrapping to two out of four MLPs, such that a distribution over advantage predictions can be fit, but only a single state-value $\hat{V}(\mathbf{s})$ is predicted. The predicted state-values $\hat{V}$ are needed for optimization, as will be explained in the next section (Eq. (6.6)).

6.3.3 Training Bootstrapped Advantage Networks

Each advantage network is updated independently with data from its separate replay memory (as in previous work that utilizes bootstrapping, e.g., Osband et al. (2016)). The advantage networks and their associated replay memories are both indexed by $b = 0, .., B$. During each episode all

[1] For all evaluations, I fixed κ in the UCB-based exploration to 1.96 which corresponds to the 97.5 percentile under a Gaussian probability distribution.

data is written to one of these replay memories. To update the state-value network, samples are drawn from all replay memories. All networks are trained using stochastic gradient descent to approximate iterative targets A_{target} and V_{target} which are computed using the replay memory with trajectories $\mathscr{T} = (\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ as follows:

$$V_{\text{target}}(\mathbf{s}) = r + \begin{cases} 0, \text{ if } \mathbf{s}' \text{ is terminal} \\ \gamma \cdot \hat{V}(\mathbf{s}'), \text{ else} \end{cases} \tag{6.5}$$

$$A^b_{\text{target}}(\mathbf{s}, \mathbf{a}) = Q^b_{\text{target}}(\mathbf{s}, \mathbf{a}) - \hat{V}(\mathbf{s}) \tag{6.6}$$

where

$$Q^b_{\text{target}}(\mathbf{s}, \mathbf{a}) = r + \begin{cases} 0, \text{ if } \mathbf{s}' \text{ is terminal} \\ \gamma \cdot \widehat{Q}^{\hat{b}}(\mathbf{s}', \pi_{\texttt{greedy}}(\mathbf{s}')), \text{ else} \end{cases} \tag{6.7}$$

$$\widehat{Q}^b(\mathbf{s}, \mathbf{a}) = A^b_{\text{target}}(\mathbf{s}, \mathbf{a}) + \hat{V}(\mathbf{s}) \tag{6.8}$$

$$\hat{b} \sim \mathcal{U}\big(\{1..B\} \backslash b\big) \tag{6.9}$$

$$\pi_{\texttt{greedy}}(\mathbf{s}) = \arg\max_{\mathbf{a} \in \mathcal{A}} \hat{\mu}_{\hat{A}}(\mathbf{s}, \mathbf{a}) \tag{6.10}$$

where $\hat{\mu}_{\hat{A}}$ denotes the empirical mean over the advantage predictions for a given input. $\hat{b} \neq b$ is a uniformly sampled replay memory index. It was introduced in analogy to double Q-learning (van Hasselt, 2010; Van Hasselt et al., 2016) to avoid bias in estimating Q caused by the `argmax` operation to evaluate π in Eq. (6.7). Using $\hat{b}$ has turned out crucial for performance of this method.

Since $\hat{\mu}_{\hat{A}}$ can in general be a highly non-convex function, the `argmax` in Eq. (6.10) to compute $\pi_{\texttt{greedy}}$ was approximated by evaluating 100 sampled actions. Empirically, a higher sample size did not improve performance in my evaluation setting. Moreover, the experiment on the real robot has confirmed that the additional time for this update is negligible compared to the time needed for the robot rollouts.

6.3.4 Global Exploration using Trajectory Optimization

Just using the UCB over the advantage is already enough to derive a greedy exploration policy as in Eq. (6.10). This will explore locally but often it is necessary to go through well-explored areas for several steps before reaching a point of high uncertainty. I therefore use trajectory optimization to compute a series of states throughout the full state space to visit. The resulting exploration is referred to as *global*, because not just locally reachable states but the whole state space is considered. The optimal states to visit are referred to as waypoints $\mathbf{w}$ and selected such that the following sum over UCBs is be maximized:

$$\mathscr{W}^* = \underset{\{\mathbf{w}_i \in \mathcal{S}\}_{i=1}^{\eta}}{\arg\max} \sum_{i}^{\eta} \max_{\mathbf{a}} \mathrm{UCB}_{\hat{A}}(\mathbf{w}_i, \mathbf{a}) \quad \text{subject to} \quad ||\mathbf{w}_{i-1} - \mathbf{w}_i|| \leq \frac{T}{\eta} \cdot v \tag{6.11}$$

In my experiments I fixed the number of waypoints $\eta = 5$, and set the number of steps per rollout $T = 200$. v is a manually tuned maximum step size. The optimization was solved using the Powell method (Powell, 1964).

Given this set of waypoints, the agent is supposed to move towards each of them successively. This could in principle be achieved by exploiting a dynamics model, but contacts and friction are hard to model and some of the physical properties of relevant materials might be unknown. Instead of a fully fleshed model I rely on general domain knowledge for robotic manipulation tasks in form of a *steering function* and a task-space position controller: an action $\mathbf{a}$ in the MDP of the RL formulation represents an offset from the current state $\mathbf{s}$ that is sent to a position controller and thus, the action that is expected to lead the agent to a target state $\mathbf{s}'$ can be computed as

$$\mathbf{a} = \mathbf{s}' - \mathbf{s} \tag{6.12}$$

A state in the upcoming experiments was represented in two positional and one rotational dimension. This is only a coarse approximation of the dynamics and will fail whenever obstacles or contacts come into play. However this steering function is only needed for exploration and never at test time. In Section 6.4, I demonstrate the robustness of exploration under noise and miscalibration of the steering function.

As a result, the agent will subsequently try to reach the waypoints by moving according to the approximate steering function, which leads to piecewise linear motion in configuration space. Since an action in most RL settings is restricted, the agent might effectively take several small steps towards the waypoint. If a waypoint $\mathbf{w}$ is reached, the optimal action w.r.t. the advantage UCB is executed:

$$\mathbf{a}^* = \arg\max_{\mathbf{a} \in \mathcal{A}} \mathrm{UCB}_{\hat{A}}(\mathbf{w}, \mathbf{a}) \tag{6.13}$$

Note that $\mathbf{a}^*$ is the same action that was used to compute the sum of UCBs in Eq. (6.11).

If the agent does not get any closer to a waypoint, e.g., because the steering function fails in the presence of objects, the agent discards this waypoint and moves on to the next waypoint (see Subsection A.2.1 for an illustration including obstacles). When the last waypoint has been reached or discarded, the greedy exploration policy $\pi_{\texttt{greedy}}$ is used as a fallback until the end of the episode.

Empirically, I found it beneficial to start with a small number of episodes using a random policy and introduce some random steps after each waypoint was reached to further explore the area of high uncertainty[2].

The full exploration algorithm is formally given in Algorithm 2.

6.4 Experimental Results

The proposed exploration scheme is designed to generate useful data such that in a second step, these samples can give a head start to train a policy, similar to learning from demonstration.

I therefore evaluated how fast the exploration scheme is able to find successful solutions, how robust the scheme is to a perturbed or miscalibrated steering function, the feasibility of my exploration strategy on a real robot, and if the combination of exploration and learning from demonstration is superior to direct RL from scratch. Most results were obtained from simulation where a larger number of experiments could be conducted.

[2] I used 5 random steps after each waypoint and 10 random episodes in the beginning of each experiment.

Algorithm 2 Exploration Episode using Approximate Uncertainty-Based Trajectory Optimization

Given:
max. number of steps per episode T
target reaching tolerance ε
max. step size υ

$t \leftarrow 0$ ▷ initialize step counter
for all $\mathbf{w}_i \in \mathscr{W}^*$ **do** ▷ $\mathscr{W}^*$ computed with Eq. (6.11)
  $\Delta_{t-1} \leftarrow \inf$ ▷ initialize distance to waypoint
  observe current state $\mathbf{s}$
  $\Delta_t \leftarrow ||\mathbf{s} - \mathbf{w}_i||_2$ ▷ update distance to waypoint
  while $(\Delta_t > \varepsilon)$ and $(\Delta_t \leq \Delta_{t-1})$ **do**
    ▷ while waypoint not reached yet but distance is decreasing
    execute $\mathbf{a}^*$ towards waypoint $\mathbf{w}_i$
      ▷ using the steering function in Eq. (6.12)
    observe current state $\mathbf{s}$
    if $\Delta_t \leq \varepsilon$ **then** ▷ waypoint reached
      execute $\arg\max_{\mathbf{a}} \mathrm{UCB}_A(\mathbf{s}, \mathbf{a})$
    end if
    $t \leftarrow t + 1$
    if $t == T$ **then** ▷ time-out
      return
    end if
    $\Delta_t \leftarrow ||\mathbf{s} - \mathbf{w}_i||_2$ ▷ update distance to waypoint
  end while
end for
for t to T **do** ▷ remaining steps until time-out
  execute $\pi_{\texttt{greedy}}(\mathbf{s})$ ▷ Equation 6.10
end for

All results are based on the double peg in hole tasks described and illustrated in Subsection 3.1.2, where two pegs need to be inserted into an object with two holes. Subsection A.2.2 illustrates a successful insertion strategy in simulation. All tasks use continuous state and action spaces. The states consist of two positional dimensions and one rotational, i.e., effectively the object can be moved and rotated in a 2D plane. The agent uses position control with a reference position that was obtained by summing up the current position of the EEF and the action representing a 3D offset. The reward is based on the Euclidean distance in state space between the current state and goal configuration.

6.4.1 Exploration Speed

I first evaluated the speed of exploration, measured by the number of successful trials generated during exploration, i.e., in off-policy behavior. In an ablation study, I compared the full system (**OURS**) to two baselines:

1. the exact same system but with bootstrapped Q-networks instead of the split into state-value and advantage, changing the number of states in the last MLP such that both types of networks use the same number of parameters (**Q-NET**).

2. the system with advantage networks but with greedy UCB exploration instead of trajectory optimization (**UCB-GREEDY**).

The approach in Chen et al. (2017) is a combination of these two baselines, where a greedy exploration is chosen based on Q-networks.

Each method was tuned with a hyperparameters grid search[3] and ten random seeds per parameter set. Only the set with best performance on average was chosen for evaluation. Figure 6.2 illustrates the mean cumulative number of episodes that ended in a terminal state for different random seeds. Each algorithm was run with 10 different random seeds, the mean performance is shown as a solid line, the shaded area covers one standard deviation of the mean estimator. My full exploration scheme achieves more successful episodes than the same algorithm based on Q networks or a greedy UCB policy.

[3] learning rates in $\{0.0001, 0.001, 0.01\}$, batch sizes $\{32, 64\}$, noise scales $\{0.1, 0.2, 0.3\}$. After each rollout, 50 training steps with stochastic gradient descent were performed.

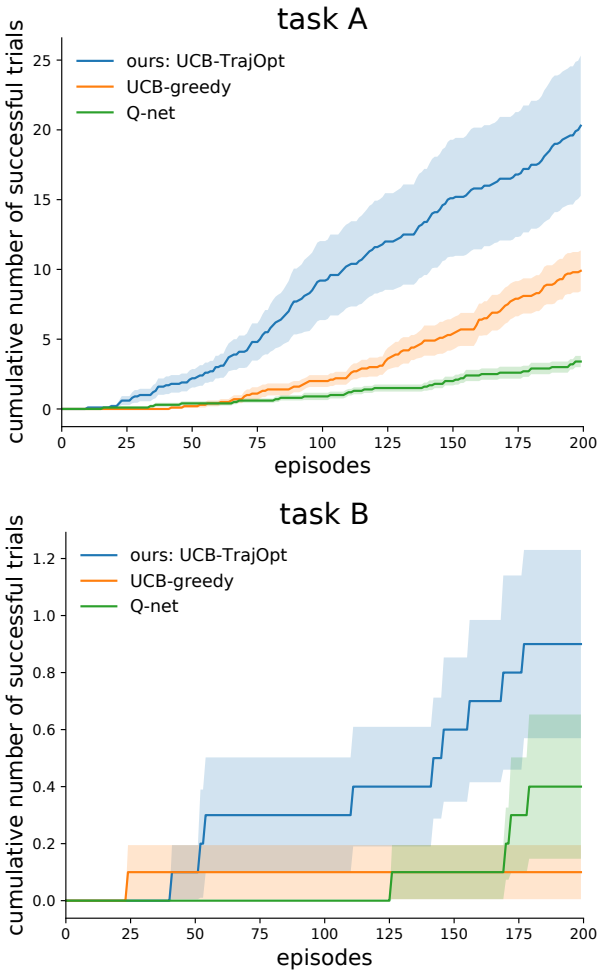


Figure 6.2: Ablation study on off-policy exploration performance: For each algorithm, the same grid search for hyperparameters was applied and only results for the best-performing set of hyperparameters are shown. Task A is the task variant with fixed pegs, task B has dangling pegs as illustrated in Subsection 3.1.2.

6.4.2 Robustness to Corrupted Steering Functions

To perform global state-space exploration, my method exploits a so-called steering function. This is used to compute control inputs, i.e., actions, for an agent to take in order to move towards a given target state. The target states that the agent is supposed to reach in this approach are the waypoints from trajectory optimization during exploration. During test time, the steering function is not used at all. Since the steering function is only used for exploration, I hypothesized that my approach would be robust to perturbations and re-ran the experiment from the previous section on the variant with stiff pegs with different kinds of disturbances: Additive Gaussian noise (**NORMAL-X**) as well as systematic mis-calibrations of 5°, 10° and 30° around one (**X° 1D**) or all three action dimensions (**X° 3D**). All experiments were conducted using the best set of hyperparameters from the previous section under 10 different random seeds. The results in Figure 6.3 show that the system is robust to most disturbances: Gaussian noise does not affect performance on average, presumably because this cancels out over time along a trajectory. Similarly, rotations of 5° and 10° around one axis can be tolerated. The performance decreases for stronger disturbances and is worst for a 30° rotation around all dimensions. Except for this last perturbation, all exploration schemes based on noisy steering functions outperform the best baseline from Figure 6.2 on average.

6.4.3 Feasibility on a Real Robot

I also replicated the experiment from Subsection 6.4.1, i.e., the exploration phase, on a real robot: A KAWADA Nextage Open and a 3D printed version of the harder double peg in hole with dangling pegs as illustrated in the upper row of Figure 3.2.

States and actions were defined exactly as in simulation. More details on the 3D printed objects are given in Figure 3.2. To avoid harmful collisions, a force-torque sensor was installed at the EEF. Whenever the force or torque exceeded a manually set threshold, the current robot movement was stopped and the EEF was moved upwards until the pressure was relieved. Since evaluation on real hardware is too time-consuming to perform statistically sound analyses with multiple hyperparameters and

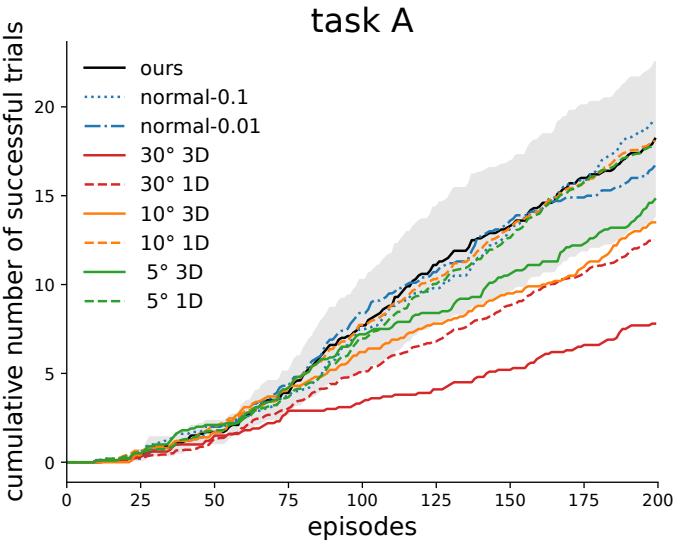


Figure 6.3: robustness of off-policy exploration performance to corrupted steering functions: my method was re-run with different perturbations added to the steering function: Gaussian noise (**NORMAL-STD**) that might cancel out over longer trajectories as well as systematic mis-calibrations of 5°, 10° and 30° around one (**X° 1D**) or all three action dimensions (**X° 3D**). Each line represents the mean over results from ten different random seeds, the gray shade represents the standard deviation around the mean estimator without disturbance. Task A refers to the task variant with stiff pegs.

random seeds, I used those hyperparameters that performed best in simulation. To save time, I only executed 2 random episodes in the beginning though.

The video at `https://youtu.be/JTfeHhWSb0Y` and extracted keyframes in Subsection A.2.3 also allow quantitative visual inspection of the exploration behavior. It demonstrates what the initial episodes with Gaussian noise look like and how this compares to the first exploration using trajectory optimization and piecewise linear movements afterwards. With some more training, the uncertainty estimates start to lure the agent towards exploring promising areas: first a local minimum in the middle between both holes (optimizing the Euclidean distance to the target position), the the agent starts to focus on the area around the holes before it finally finds the target position after less than 1200 steps.

6.4.4 Sample Efficiency for Policy Learning

Exploration data that contains experience from successful behavior intuitively seems useful. Still, here I experimentally verified the hypothesis that the exploration data collected in Subsection 6.4.1 will speed up the training process for a policy.

A DDPG actor-critic setup[4] served as a baseline (**DDPG**) for learning from scratch with Gaussian noise, Ornstein-Uhlenbeck (Lillicrap et al., 2015) and adaptive parameter noise (Plappert et al., 2017). Additional parameters for these noise processes were included in the hyperparameter grid search. For the proposed approach (**OURS**), the experience from my exploration scheme is slowly added into the replay memory (similar to Vecerik et al. (2017)). Except for this experience injection, the same DDPG algorithm as for the baseline is run.

I used the exploration data from the best performing hyperparameter set in my ablation study as exploration samples. Since all configurations were run 10 times, I chose the random seed whose performance was closest to average to keep the data as representative as possible. Empirically, I found it beneficial to discard the initial random episodes and add samples from 10 exploration episodes at each training episode in the policy learning stage.

[4] as implemented in the openAI baselines repository `https://github.com/openai/baselines`

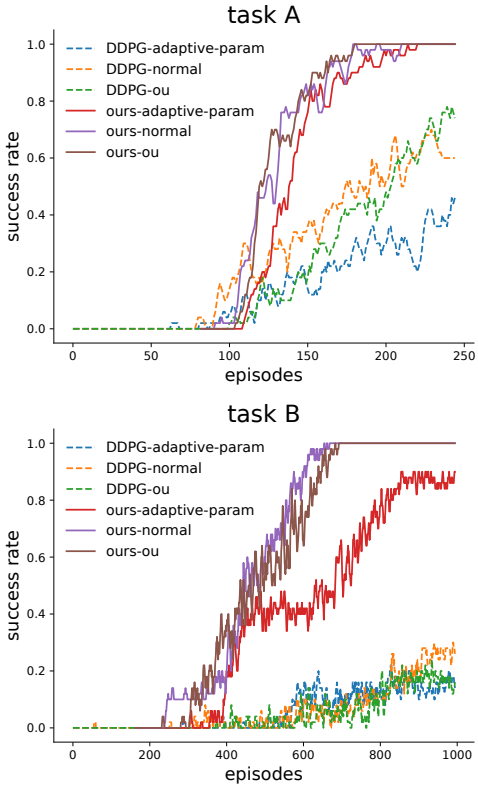


Figure 6.4: Sample efficiency in policy learning: Policies learned from scratch (**DDPG**, dashed lines) are compared to policies that started with data from my exploration scheme (**OURS**, solid lines starting after exploration episodes). Each line represents the ratio of successful policies during on-policy evaluation over ten independent trainings using different random seeds. Using data from my exploration scheme seems to speed up convergence; and the difference becomes more significant for the harder contact-rich manipulation task B (dangling pegs) than for task A (stiff pegs).

Training an actor from the exploration data in a supervised setting without any further environment interaction failed completely—most likely due to compounding errors known in behavioral cloning (e.g., Jiang et al. (2015)). Also training an actor network while exploring with my strategy off-policy was empirically unstable unless a very small learning rate was used (which in turn counteracts the sample efficiency).

I defined a run to be converged if the actor network without noise was able to solve the problem 10 times in a row starting from random initial positions. Figure 6.4 compares the performance of vanilla DDPG trained from scratch to DDPGfD (DDPG from demonstration) using 80 episodes of exploration data for the simpler task and 160 for the harder one.

There were no successful samples in the demonstration data before episode 80 or 160 respectively, but it can be observed that injecting the data from my exploration scheme speeds up learning anyway. Interestingly, the improvement of the full policy trained on my data over training from scratch is larger for the task with dangling pegs, although the steering function should fail more often for dangling than for fixed pegs. Together this indicates that the data created is generally informative, and that the number of successful episodes may not be the best criterion to evaluate the usefulness of data for RL.

6.5 Discussion

To overcome the infamous sample inefficiency of exploration for model-free RL from scratch, I have proposed to use global state space exploration via trajectory optimization to generate informative data in a first step and then use this data in analogy to expert demonstrations in a second step to train a full model-free policy.

The approximate trajectory optimization aims to maximize the UCB on advantages at a set of waypoints that the agent visits sequentially using a coarse steering function. I have also demonstrated the robustness of the approach to noise and systematic mis-calibration of the steering function.

To compute the advantage function and derive an uncertainty measurement over advantages, I designed a network with two branches, which —in contrast to prior work (Wang et al., 2016)— can be trained without loosing the semantic meaning of the branches.

The experimental evaluation has illustrated that the proposed approach rapidly finds successful sample trajectories; and experience collected during this exploration can significantly speed up policy learning even if the exploration does not contain successful samples. Additionally, the largest speed-up is achieved in training full policies for the harder contact-rich manipulation task although significantly less successful samples are found during exploration. Together, these results indicate that data from my exploration scheme is generally more informative than existing strategies.

Beyond its contributions, this work is one step towards integrating prior knowledge into model-free deep RL. In a trade-off between generalization and specificity, this can move model-free deep RL one step closer to efficiency. The loss in generalization seems manageable in many areas of application, e.g., for contact-rich manipulation in robotics, where tasks are almost always performed with robots for which a coarse steering function and a task space controller are available.

6.6 Limitations and Open Questions

The experiments are limited by the type of task performed. It remains open to what degree the results could be transferred to different tasks, in particular because theoretical arguments about which kind of task is easy or hard to solve for deep RL are hard to make. Also it is unclear how much the task formulation and modalities matter, i.e., whether the same kind of effects could be observed for visual input and deeper networks, or for sparse reward formulations. The latter is a particularly important open question, because sparse reward settings make exploration even harder in general.

Using a coarse steering function can be seen as one particular step towards integrating approximate model information into deep RL through exploration. One question left for future work is thus what types of models can be exploited, in particular models that go beyond the coarse steering function I applied here. It would be interesting to see whether approaches from model-learning can be used as well; and how much the quality of exploration data improves when the model accuracy improves. Measuring quality of exploration data however is a full open research topic in itself.

The next chapter will take first steps towards characterizing properties of datasets and exploring how this characterization is linked to success and speed of RL processes.

Chapter 7

Linking Data Graph Structures to Soft Divergence

Exploration in RL aims to collect informative data that an agent can learn from. Improving exploration therefore typically means to collect data which is supposed to improve learning in some form, e.g., improving learning speed, learning stability, the safety of exploration or many more. However, there is barely any formal definition of what makes data useful. In this chapter, I will represent the contents of a replay memory from off-policy deep RL as a graph where nodes correspond to states and edges to transitions between states. This graph perspective allows extracting structural information of the data which can, as I will show, be linked to soft divergence, one of the issues in (deep) Q-learning that are not entirely understood yet and hard to prevent.

7.1 Introduction & Motivation

DQN and DDPG are two state of the art algorithms for deep model-free off-policy RL that have been successfully applied on a wide range of tasks including continuous control tasks both in simulation and the real world (Lilli-

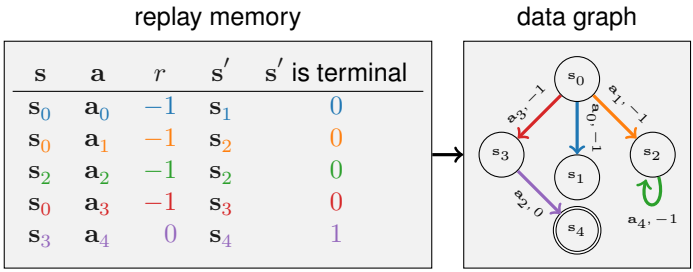


Figure 7.1: The replay memory (left) can be represented as a data graph (right): nodes correspond to states and transitions are represented by edges between states. Additional transition information, such as the reward, can be seen as an annotation to the edge. The structure of the data graph is shown to be linked to soft divergence in Chapter 7 and will be used constructively to improve sample efficiency and robustness of deep model-free off-policy RL in Chapter 8.

crap et al., 2015; Vecerik et al., 2017). Both algorithms however are based on Q-learning (Watkins, 1989), which is still not fully understood. On the contrary, Q-learning even with linear function approximation is known for convergence issues (Baird, 1995). Deep Q-learning combines highly non-linear function approximation with off-policy learning and bootstrapping—a combination that has been termed *deadly triad* by Sutton and Barto (2018) because of the instabilities it is likely to induce. Empirically, deep Q-learning does not seem to fully exhibit these expected divergence issues (Van Hasselt et al., 2018) but its performance can be unreliable and hard to reproduce (Henderson et al., 2018).

With the work I present in this chapter, I aim to extend the community's understanding of when deep Q-learning diverges: I propose to take a graph perspective on the replay memory (see Figure 7.1 for illustration), which allows analyzing its structure and show on small characteristic example problems how specific structures are linked to soft divergence.

7.2 Preliminaries

Building on the definitions for return, state value and state-action value in Chapter 2, the following properties can be derived for state values:

For $\gamma < 1$ and an infinite trajectory with constant reward r, the return forms a geometric series:

$$R = \sum_{t}^{\infty} \gamma^t r = \frac{r}{1-\gamma} \tag{7.1}$$

Thus, under the assumption of a policy π which induces such an infinite trajectory with constant reward, also the state value converges:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}\left[\sum_{t}^{\infty} \gamma^t r\right] = \frac{r}{1-\gamma} \tag{7.2}$$

The state action value converges analogously:

$$\begin{aligned} Q^{\pi}(\mathbf{s},\mathbf{a}) =& \mathbb{E}\left[r_a + \gamma Q^{\pi}(\mathbf{s}', \pi(\mathbf{s}'))\right] \\ =& \mathbb{E}\left[r_a + \gamma V^{\pi}(\mathbf{s}')\right] \\ =& \mathbb{E}\left[r_a + \gamma \sum_{t}^{\infty} \gamma^t r\right] \end{aligned}$$

where r_a refers to the reward that the agent received for executing action $\mathbf{a}$ in the first step which may be different from $\pi(\mathbf{s})$.

This also means, that if the reward function is bounded by r_{min} and r_{max}, the smallest and largest possible Q-value can be computed as

$$\left[\min\left(r_{\text{min}}, \frac{r_{\text{min}}}{1-\gamma}\right), \max\left(r_{\text{max}}, \frac{r_{\text{max}}}{1-\gamma}\right)\right] \tag{7.3}$$

respectively (Lee and Kim, 2015). The min/max operations ensure that also cases when terminal states are reached are covered.

Analogously, state(-action) values for loops with different rewards can be derived. Assume that one loop consists of m ever repeating transitions $(\mathbf{s}_1, \mathbf{a}_1, r_1, \mathbf{s}_2)$ to $(\mathbf{s}_m, \mathbf{a}_m, r_m, \mathbf{s}_1)$ which are induced by a policy π and rewarded equally in each loop traversal. Then the state value can be

computed as follows:

$$
\begin{aligned}
V(\mathrm{s}_1) &= \underbrace{r_1 + \gamma r_2 + ... + \gamma^{m-1} r_m}_{r_L} + \gamma^m r_1 + \gamma^{m+1} r_2 + ... \\
&= r_L + \gamma^m r_L + \gamma^{2m} r_L + ... \\
&= r_L \sum_t^\infty (\gamma^m)^t = \frac{r_L}{1 - \gamma^m}
\end{aligned} \tag{7.4}
$$

7.3 Related Work

RL based on Q-learning has been known to be instable even with linear function approximation for more than 20 years (Baird, 1995). While the problem to date is not fully understood, in particular for more complex nonlinear settings, there are several approaches to characterizing instability and prevent it.

7.3.1 Instabilities in RL: the Deadly Triad

The combination of function approximation, bootstrapping and off-policy RL has been called the *deadly triad* by Sutton and Barto (2018) because of its tendency to diverge. While deep RL methods within the deadly triad tend to be hard to reproduce and evaluate empirically (Henderson et al., 2018), it seems to exhibit soft divergence only (Van Hasselt et al., 2018): that is, following the authors' definitions, deep RL does not actually reach floating point `NaN` values but instead plateaus or oscillates at Q-values that are outside of the theoretically derived bounds as given in Eq. (7.3). My experiments will largely support these findings.

To understand causes and remedies against divergence in the deadly triad, all three properties of the triad have been the focus of investigations.

Function Approximation

Different networks for *function approximation* and update schemes have been linked to convergence: Fu et al. (2019) found large neural networks with compensation for overfitting to be beneficial for learning stability. Mnih

et al. (2015) introduced target networks: These are a second function approximator that is only updated slowly or periodically and therefore serves as a kind of low pass filter on the network parameters. Its values are therefore more stable and lead to more stable target Q-values in TD learning. Besides, a second network can help to counteract maximization bias in Q-learning (Van Hasselt et al., 2016). Also other methods that delay (Fujimoto et al., 2018) or average target values (Anschel et al., 2017) have been shown to stabilize learning. Achiam et al. (2019) theoretically link generalization properties of the Q-function approximator to the stability of learning. I empirically confirm and provide further intuition about this effect in Subsection 7.5.3.

Off-Policy

In policy gradient methods, reducing the impact of *off-policy* data has been beneficial for stability, e.g., by mixing on- and off-policy (Gu et al., 2017) or by constraining the gradient update through a proximity term (Touati et al., 2020). Also in DQN and DDPG, restricting the action space to achieve lower levels of off-policy data have been explored (Fujimoto et al., 2019). Constrained action selection when computing the target Q-values can also stabilize deep RL (Kumar et al., 2019).

Kumar et al. (2020) emphasize that the interaction of off-policy learning and bootstrapping can lead to cases where a state is visited frequently and yet its incorrectly estimated Q-value is not updated because the state that the target value depends on is not visited. They refer to this phenomenon as 'lack of corrective feedback' and derive a re-weighting of transitions from the replay buffer that is supposed to mitigate this issue. The full version of my proposed method, using zero actions, will be able to improve performance with such tail ends of data distributions without downweighting the associated transitions, without an additional error model and without constraining the action selection.

Further questions regarding the impact of off-policy data remain open however: Correcting off-policy samples may have adverse effects as well, as reported by Hernandez-Garcia and Sutton (2019) for SARSA which is an on-policy algorithm that is quite close to Q-learning (Sutton and Barto, 2018). Fedus et al. (2020) found that counter-intuitively, n-step return updates which are not corrected for policy differences are beneficial in off-policy deep RL despite being theoretically ungrounded.

Bootstrapping

Standard Q-learning uses *bootstrapping* as in Eq. (2.6) to estimate a Q-function, i.e., the estimate at one iteration is used to derive the update for the next iteration's estimates. Alternatives to bootstrapping include fixed-horizon TD methods (De Asis et al., 2019) and finite-horizon Monte Carlo updates, in which a Q-value is estimated based on observed Returns from each state. While the resulting estimator for the Q-function has low bias, it comes with high variance. Introducing eligibility traces of different lengths, a spectrum of methods between TD and Monte Carlo methods can be spanned (Sutton and Barto, 2018; Precup et al., 2000), also in a deep learning setting (Munos et al., 2016; Mnih et al., 2016; Amiranashvili et al., 2018). Similarly to Monte Carlo estimates, my proposed method propagates information along full trajectories. However, I do not apply return values as high-variance targets but use them to derive a lower bound for each target Q-value instead.

7.3.2 Graph Perspective on Training Data

Monte Carlo updates can be seen as a special case of graph perspective: data from full episodes is used to derive updates along a trajectory. Episodic backward updates are classical TD updates that are executed along trajectories in reverse order, such that information is quickly propagated through consecutive states (Lee et al., 2019b). To prevent errors from sequential updates of correlated states, a diffusion coefficient is introduced.

Zhu et al. (2019) take a full graph perspective on the agent's experience: using a learned state embedding, episodes with shared states are identified and can benefit from inter-episode information, i.e., the algorithm can combine multiple trajectories from experience. State embeddings have also been combined with k-nearest neighbors as a method to estimate Q-values for unseen states (Blundell et al., 2016). Corneil et al. (2018) use a network model to map states to an abstract tabular model where planning can be easily applied. In my approach, I also use a graph perspective but without a learned embedding inter-episodic information is only exchanged if the exact same state is revisited (up to floating point precision). Note however, that in many cases there are points which are very likely re-visited, e.g., physical corners between objects.

7.4 Method: Data Graph Structure

Despite the continuous state-action space, the networks in DDPG are updated based on a finite set of transitions from the replay memory. It is therefore possible to take a graph perspective on this data: A transition $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ can be seen as an edge between the nodes corresponding to states $\mathbf{s}$ and $\mathbf{s}'$; and can be annotated with action $\mathbf{a}$ and reward r. Any hashing function can be used to encode nodes and detect if the same node is revisited. This is not supposed to introduce any discretization beyond the limits of precision. I refer to the resulting directed graph as *data graph* (see Figure 7.1 for an illustration).

7.4.1 Characterizing Transitions

In the experimental section I will present results that link different types of transitions to different levels of soft divergence. The differentiation between transition types is based on the following definitions:

1. If $\mathbf{s}'$ is terminal, target Q-values for TD learning equal the observed reward r (see Eq. (2.6)) and thus RL reduces to supervised learning. I thus hypothesized that Q-values for such *directly connected* transitions are very unlikely to diverge.

2. Transitions that end in a non-terminal state from which a terminal state is reachable are referred to as *(indirectly) connected*. This class of transitions can further be parameterized by the length of the shortest path that is known to reach a terminal state.

3. If no terminal state is reachable from $\mathbf{s}'$ but there is at least one infinite path from $\mathbf{s}'$, this transition is referred to as *disconnected*. Such an infinite path means in practice that there is at least one loop on the path ahead of $\mathbf{s}'$. If this loop is deterministically induced by a policy and rewarded equally across loop iterations, the resulting state-values can be analytically derived using Eq. (7.4).

4. If no terminal state is reachable from $\mathbf{s}'$ and there is no infinite path from $\mathbf{s}'$, the transition is referred to as a *loose end*. These transitions occur for instance at the end of each episode in episodic learning setups, when the agent does not succeed but is reset to a starting position.

It is insightful to note that Q-values for such transitions are conceptually ill-defined in tabular Q-learning where a state without successors would be defined as terminal. For non-terminal states, a Q-value could be determined under the assumption that further transitions exist (and just have not been discovered yet), but then the Q-value is estimated using bootstrapping from another Q-value that has never been explicitly updated. This phenomenon is one example for what has been referred to as a lack of corrective feedback (Kumar et al., 2020). In other words, the estimate depends only on network initialization and generalization from data for other state-action pairs; cf. also Achiam et al. (2019) who analyze the theoretical link between approximator generalization properties and learning stability.

7.5 Experimental Results

In this chapter, I illustrate how the different types of transitions impact the Q-value estimates in deep off-policy model-free RL using a series of toy examples. The examples differ in the data graph they work on, i.e., they consist of different types of transitions following the definitions in the previous section.

7.5.1 Experimental Setup

The following offline policy evaluation task was set up: An agent can maneuver in a 2D continuous state space with 2D actions such that adding state and action yields the next state $\mathbf{s}' = \mathbf{s} + \mathbf{a}$. For each step, the agent receives a reward of -1 and 0 at the terminal state. The replay memory stays fixed, i.e., is not extended, while a DDPG-like critic network is trained to find an approximation to the Q-function. The policy π is not approximated by an actor network as in DDPG, but instead defined based on the replay memory such that for a given state it selects the action from the full graph that is currently associated with the highest Q-value prediction. Like in DDPG, this means that the actor may choose an action that has not been observed before. As in DQN and DDPG, all network updates are solely derived from the finitely many samples in the replay memory. The

network consists of two layers with four hidden states each, ReLU activations (except on the output) and Xavier-initialization. No target networks or other bells and whistles were used.

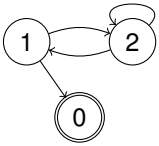
7.5.2 Empirically Assessing Divergence

The training procedure was repeated with ten random seeds that were drawn uniformly from $[0, 1000]$. For further analysis, I propose to compute the standard deviation of Q-values that were predicted from networks based on different random seeds as a measure of divergence: if Q-learning for a transition converges, all Q-values should be identical and thus have a standard deviation close to zero. The more divergence occurs however, the larger the standard deviation becomes. Even if all trials diverge, it is highly unlikely that the resulting Q-values are identical.

All results in the following sections were in line with the finding in Van Hasselt et al. (2018), according to which no *unbounded* divergence occurs (which would cause floating point `NaN`s). Instead, I argue about occurrences of *soft* divergence, i.e., Q-values beyond the realizable range as given by Eq. (7.3).

7.5.3 Example I: Types of Transitions

Let me first illustrate the types of transitions with an exemplar data graph:



It consist of three states, one of which is terminal (double circle). To learn a Q-function from data which can be represented by this graph, I assigned 2D coordinates to each state as follows: $\mathbf{s}_0 = [0, 0]$, $\mathbf{s}_1 = [-1, 1]$, $\mathbf{s}_2 = [1, 1]$ Furthermore, let's assume that the agent could possibly have collected any subset of the four transitions on the graph. This creates $2^4 = 16$ subsets; one of which is empty and therefore ignored. Moreover, each subset contains different types of transitions. These types are illustrated for all subsets by a color coding in Figure 7.2: directly connected transitions, that is those leading to the terminal state 0, are shown in blue.

In my example this is only ever the transition between states 1 and 0. Indirectly connected states are those for which a longer path to the terminal state exists. They are illustrated in orange in Figure 7.2. Here it can be highlighted that the type of a transition depends on the context: The transition from state 2 to state 1 is indirectly connected to the terminal state, only if the transition from state 1 to state 0 exists as well. Otherwise, if the transition from state 1 to state 0 is not present, the transition from state 2 to state 1 becomes a loose end (illustrated in green, e.g., the top left example in Figure 7.2). Loose ends are those transitions that end in a non-terminal state from which no further transitions are known. The remaining transitions, highlighted in red in Figure 7.2, are disconnected transitions: from their end state no path to a terminal state is known, but at least one infinite path (i.e., loop) exists.

To verify that these transition types are linked to different divergence behavior in deep model-free RL, the training that was described in Subsection 7.5.1 has been repeated with multiple random seeds for each of the 15 subsets of the data graph. To empirically assess soft divergence, I evaluated the predicted Q-values for each transition after the training process as follows: First, the standard deviation over Q-value predictions for one specific transition in one example subset from different random seeds was computed. Second, the distribution over predicted Q-values for all transitions of a specific type was illustrated in a boxplot in Figure 7.3. The line inside each box represents the median, the box extends to the quartiles and the whiskers cover 1.5 times the inter quartile range.

Evaluating the distribution of standard deviations reveals a clear link between the structure of the data graph and soft divergence: Q-values for transitions which directly end in a terminal state ('directly connected') are estimated almost perfectly, most likely because Q-learning is reduced to supervised learning in these cases (cf. Eq. (2.6)).

Q-values for transitions ending in states with a longer path to a terminal state, the connected transitions, exhibit only slightly more variance than the directly connected transitions. Presumably the reachable terminal state still acts as an anchor for the Q-value (as long as all transitions on the path are regularly used for updates). Transitions without such an anchor congruously have caused much more variance in their predictions: loose ends and disconnected transitions.

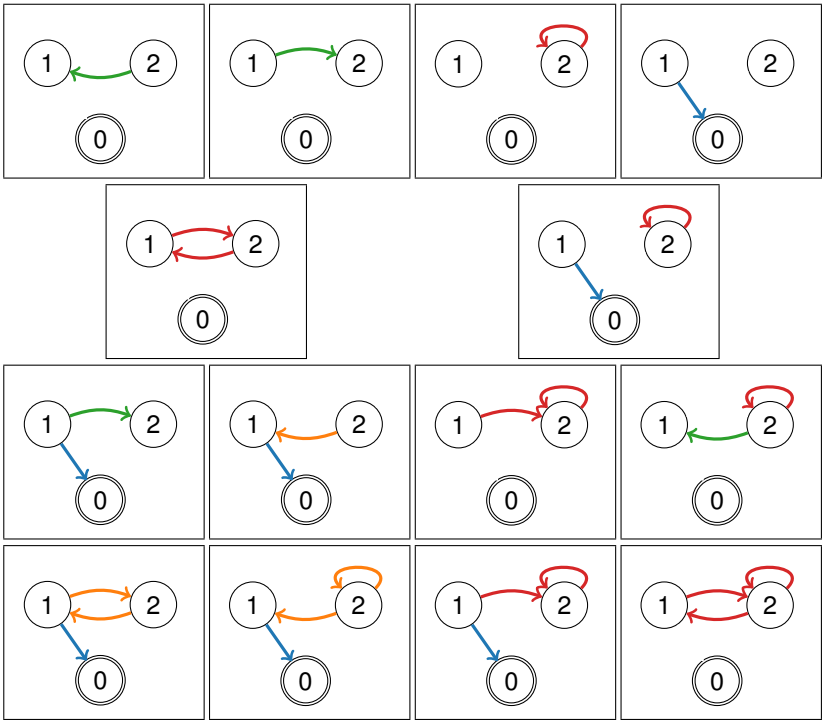


Figure 7.2: The 14 incomplete subsets of the data graph in example I. The 15th subset is empty, the 16th subset is the complete graph. There are three states (with state 0 being terminal) and up to four transitions, which are colored based on the graph structure as directly connected to a terminal state (blue), indirectly connected (orange), disconnected but infinite paths (red) or loose ends (green).

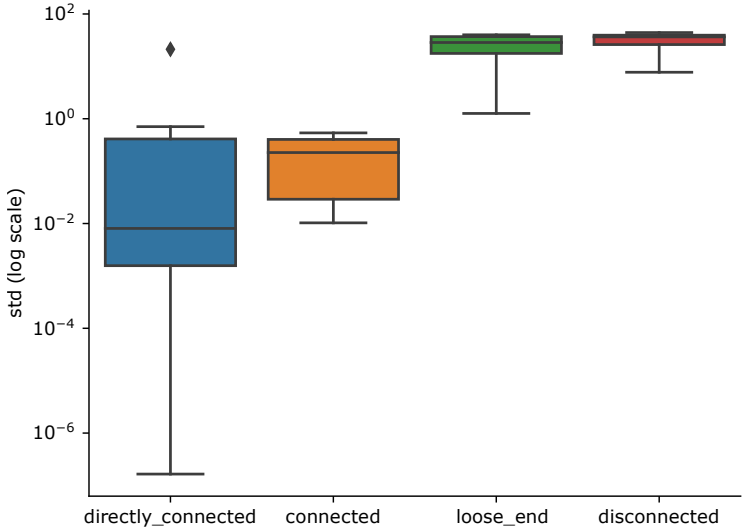


Figure 7.3: Standard deviation over predicted Q-values for each type of transition from all 15 possible subsets of the educational example.

Loose ends, whose Q-values are conceptually ill-defined for tabular Q-learning as discussed above, caused high variance in predictions—a finding which is in line with other works (Kumar et al., 2020; Achiam et al., 2019).

Disconnected transitions occur frequently in practical applications of RL, e.g., when the robot gets stuck in a non-terminal state. Such disconnected transitions caused the highest variance in my experiment. In contrast to loose ends however, the Q-value for these transitions is well-defined under the assumption that all possible transitions are known and can even be computed analytically (cf. Eq. (7.4)).

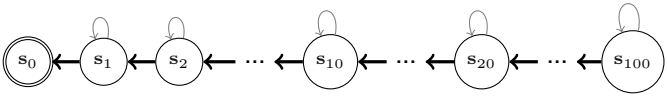
7.5.4 Example II: Chains of Transitions

In the last example, only few states and relatively short paths have been examined (except for the infinite paths on loops). Transitions with a short path to a terminal state barely caused soft divergence. In real world applications however, data graphs tend to me much larger and more complex. From the previous example it remains unclear how this interacts with divergence.

Depending on the learning formulation or domain of application, the structure of the graph can even vary. Residual formulations for example tend to produce paths which often still end in a terminal state, but it may take much more than the one step in the previous example to reach a terminal state.

Therefore, I designed a second example to investigate the impact of long chains of transitions. As mentioned in Subsection 7.4.1, this can be seen as examining an additional parameterization of indirectly connected transitions by the length of the chain ahead.

In this example I thus created a data graph which could have been induced by a replay memory with a single episode of 100 steps. The self-loops are additional options that the policy has when choosing an action.



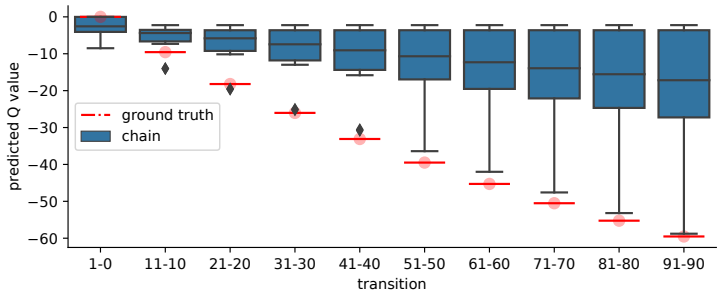


Figure 7.4: Distribution of predicted Q-values for transitions of the chain example.

Using the same setup as in the previous example, I trained a number of critic networks on data from this fixed replay memory to predict the Q-function, i.e., a reward of -1 for each step and a mapping from states to coordinates as $\mathbf{s}_t = [0, t]$. An action was again defined as the offset between the two states it connects and the policy was set as the one policy that always moves from one state to the next in the chain, i.e., $\forall t.\ \pi(\mathbf{s}_t) = \mathbf{s}_{t-1} - \mathbf{s}_t$. The Q-value for the transition from state $\mathbf{s}_t$ to $\mathbf{s}_{t-1}$ thus equals the state value which can be analytically computed as:

$$V(\mathbf{s}_t) = \sum_{j=0}^{t} \gamma^{t-j} \cdot (-1) = -\sum_{j=0}^{t} \gamma^{j} \tag{7.5}$$

The results shown in Figure 7.4 illustrate that the distance between the end state $\mathbf{s}'$ of a transition and the terminal state plays an important role: the longer the path to the anchor, the higher the variance in predictions.

7.6 Discussion

Based on the observation that even for continuous state and action spaces, model-free off-policy deep RL algorithms perform network updates on a finite set of transitions, I have used a graph perspective on the replay memory that allows closer analysis. Transitions can be characterized based

on their structural properties, such as the length of the path between the end state and its closest terminal state. Using four different types of graph structures, I have shown that these types are clearly linked to soft divergence: Loose ends, which often occur at the end of an episode, are relatively likely to lead to soft divergence. This is not surprising since the Q-value for such a transition necessarily depends on estimated Q-values for states that have never been visited. Therefore the correctness of all predictions almost exclusively depends on correct generalization of the Q-network. This insight is not new but has, in a different framing, also been mentioned in Kumar et al. (2020).

Also disconnected loop structures lead to soft divergence—in my example even more likely so than loose ends. This is surprising because the Q-value for loops can even be analytically computed in case the policy stays in this loop forever. An optimal policy may be able to do better than staying in the loop, but it cannot do worse. In this sense there is a theoretically justified lower bound and yet, I also observe divergence to negative infinity. In the next chapter, I will have a closer look at this idea of lower bounds.

7.7 Limitations and Open Questions

Since updates for the neural networks that are used to estimate Q-values in DQN and DDPG are only based on a finite replay memory, the data it contains can be represented as a graph. I have shown that the structure of this graph is correlated with soft divergence on two small examples for fixed datasets. It is not clear up to this point, whether the results from these examples transfer to real RL settings where the agent keeps collecting data and exploring while the networks are trained. Chapter 9 will touch upon this question in a system study with a real-world industrial insertion task.

The phenomenon of soft divergence in deep RL is still not fully understood: it remains unclear whether there are further factors, i.e., more precisely MDP properties in an environment, that impact convergence. For instance it remains unclear how the reward function is linked to soft divergence; and whether there are principles to follow for reward shaping to allow stable learning. If soft divergence occurs, it is also not understood in

which cases it diverges to plus and in which cases to minus infinity; and in which cases the weights to update spiral out to infinity as in Baird (1999)'s star example.

Also beyond reward shaping, it is open at this point whether the graph and its information can be used in a constructive way to improve convergence properties of deep off-policy model-free RL. This may contain smaller changes in the learning setup, such as a residual policy formulation, which can change the structure of the resulting data graph. Other ways to improve the graph structure may potentially be to explore in a directed fashion towards states or transitions which are anticipated to enhance the graph structure. This could be an automated exploration scheme as in Chapter 6 or based on expert demonstrations which could be queried for particular start or target states. While these questions are left for future work however, I will show in the next chapter that even for any dataset that has been collected without considering the data graph structure, the information contained in the data graph can be used constructively to improve the stability of learning.

Chapter 8

Stabilizing DDPG through Q-Graph-based Lower Bounds

In the last chapter I have taken a graph view on the data in the replay memory and linked the structure of this graph to soft divergence in model-free off-policy deep RL. While this work was insightful, it had a purely descriptive character. In this chapter, I will show that the graph structure can also be used in a constructive way: non-parametric information from the graph can be used to compute lower bounds to Q-values in the parametric network updates. Among other effects, this can prevent soft divergence and therefore on average increases sample efficiency in a series of experiments with DDPG and a simulated peg insertion task.

8.1 Related Work: Constrained Q-learning

Q-learning, which is prone to divergence as discussed in Section 7.3, can be stabilized by introducing constraints on the change in either target values or network parameters (Durugkar and Stone, 2018; Ohnishi et al., 2019). However, constraining change rates in a learning system may also limit the rate at which an agent can improve.

He et al. (2017) take an episode-wise trajectory perspective on the data an RL agent has collected and derive the following lower and upper bounds for the true Q-values Q^* and K steps of future and past experience on a trajectory:

$$\max_{\mathfrak{k}\in\{1,\dots,\mathrm{K}\}} L_{t,\mathfrak{k}} \leq Q^*(\mathbf{s}_t, \mathbf{a}_t) \leq \min_{\mathfrak{k}\in\{1,\dots,\mathrm{K}\}} U_{t,\mathfrak{k}} \tag{8.1}$$

where

$$U_{t,\mathfrak{k}} = \gamma^{-\mathfrak{k}-1} Q^*(\mathbf{s}_{t-\mathfrak{k}-1}, \mathbf{a}_{t-\mathfrak{k}-1}) - \sum_{i=0}^{\mathfrak{k}} \gamma^{i-\mathfrak{k}-1} r_{t-\mathfrak{k}-1+i}$$

$$L_{t,\mathfrak{k}} = \sum_{i=0}^{\mathfrak{k}} \gamma^i r_{t+i} + \gamma^{\mathfrak{k}+1} \max_{\mathbf{a}} Q^*(\mathbf{s}_{t+\mathfrak{k}+1}, \mathbf{a})$$

During training however, the optimal Q-value Q^* is not known yet. Therefore, the authors suggest to compute the lower and upper bounds using the current estimate for Q-values $\widehat{Q}$ instead. Thus, computing these bounds requires multiple additional forward passes in each update step and, more crucially, the resulting bounds need not be correct in general. In contrast, I will derive correct lower bounds for π^* in near-deterministic settings and show that incorrect empirical bounds can have adverse effects on the learning process.

Tang (2020) offers the intuition that lower bounds encourage the algorithm to focus on the best actions so far and thereby speed up learning. This idea is in line with Zhang et al. (2019) who introduce a separate replay buffer that only holds the best episodes and empirically improves learning performance on a range of simulated continuous control tasks.

8.2 Method

Let us again start from the data graph that represents the data an RL agent has collected. From the edges in the data graph the largest subset is selected such that it induces a well-defined MDP with finitely many discrete states and transitions: The subset of transitions from the data graph forms a new graph $\mathcal{G}$ with vertices $\mathcal{V}$ representing states and edges $\mathcal{E}$ that rep-

resent actions. The induced MDP $\mathscr{M}^{\mathcal{G}}$ is then a tuple $(\mathcal{E}, \mathcal{V}, \mathcal{R}, {\mathcal{P}_0}^{\mathcal{G}}, \mathcal{T}^{\mathcal{G}})$, where the action and state spaces are represented by the edges and vertices from the subgraph; the reward function is the same as for the original MDP $\mathscr{M}$ with continuous state and action spaces. The initial state probabilities ${\mathcal{P}_0}^{\mathcal{G}}$ are defined by the original probabilities $\mathcal{P}_0$ on those initial states from $\mathscr{M}$ that also occur in $\mathcal{G}$. To obtain a probability distribution, ${\mathcal{P}_0}^{\mathcal{G}}$ may need to be normalized. Since I assume deterministic transitions, all transition probabilities $\mathcal{T}^{\mathcal{G}}$ are one for a given state-action pair.

Selecting a subgraph such that $\mathscr{M}^{\mathcal{G}}$ is well-defined means that one particular type of transitions is excluded from the data graph: loose ends, i.e., transitions which end in a non-terminal state with no out-going edges. From such a non-terminal state no action could be taken and the target Q-value as in Eq. (2.6) is not defined, because the state contradicts the definition of "non-terminal".

On the finite MDP $\mathscr{M}^{\mathcal{G}}$ that was induced by a subgraph $\mathcal{G}$ without loose ends, exact Q-values $Q^{\mathcal{G}}$ can be computed analytically for all transitions. All transitions in subgraph $\mathcal{G}$ can thus be annotated with their Q-values $Q^{\mathcal{G}}$ from $\mathscr{M}^{\mathcal{G}}$. The full annotated subgraph is then referred to as *Q-Graph*.

8.2.1 Q-Graph Implementation

The Q-function $Q^{\mathcal{G}}$ for the finite MDP $\mathscr{M}^{\mathcal{G}}$ can be computed using tabular Q-iteration with guaranteed convergence due to its contraction property. The method I propose here is agnostic to the exact algorithm that is used, so for instance it is also possible to solve the linear equation system for a sparse transition matrix. In any case, the computational overhead to compute these Q-values depends on the number of transitions in the replay memory, but it is independent of the input dimensionality since the graph representation is non-parametric.

For my experiments, I implemented a Q-Graph such that it does not need to be re-computed for each usage but can instead be extended iteratively as more data comes in (see Algorithm 3).

Algorithm 3 Exemplar Q-Graph implementation for incrementally incoming data.

Initialization:
successors = {} ▷ initially empty mapping from state $\mathbf{s}$ to list of tuples ($\mathbf{s}'$, $\mathbf{a}$, r, LB$_Q$)
predecessors = {} ▷ initially empty mapping from state $\mathbf{s}'$ to list of tuples ($\mathbf{a}$, r, $\mathbf{s}$)
discount factor γ
zero action $\mathbf{a}_z$, if known
capacity $\mathcal{C}$ ▷ max. number of transitions to store

1: **procedure** ADDTRANSITION($\mathbf{s}$, $\mathbf{a}$, r, $\mathbf{s}'$)
2: add ($\mathbf{a}$, r, $\mathbf{s}$) to predecessors[$\mathbf{s}'$] unless already exists
3: LB=LBFORNEWTRANSITION($\mathbf{s}$, $\mathbf{a}$, r, $\mathbf{s}'$)
4: add ($\mathbf{s}'$, $\mathbf{a}$, r, LB) to successors[$\mathbf{s}$] unless already exists
5: **if** LB $> -\infty$ **then** ▷ if lower bound known
6: PROPAGATELB($\mathbf{s}$) ▷ Update predecessor bounds
7: **end if**
8: **if** capacity $\mathcal{C}$ reached **then**
9: remove transition ▷ e.g. first-in-first-out (FIFO)
10: **end if**
11: **if** Zero Action $\mathbf{a}_z$ known and $\mathbf{s}'$ is not terminal and $\mathbf{s} \neq \mathbf{s}'$ **then**
12: ADDTRANSITION($\mathbf{s}'$, $\mathbf{a}_z$, $\frac{r}{1-\gamma}$, $\mathbf{s}'$)
13: **end if**
14: **end procedure**

```
15: function LBFORNEWTRANSITION(s, a, r, s')
16:     LB = −∞                                          ▷ lower bound unknown so far
17:     if s' is terminal then
18:         LB = r
19:     end if
20:     if s = s' then                                   ▷ self-loop, e.g. zero action
21:         LB = max(LB, r/(1−γ))
22:     end if
23:     if larger loop with n transitions from s detected then
24:         LB = max(LB, r_L/(1−γ^n))                    ▷ see Eq. (7.4)
25:     end if
26:     if there is at least one successor transitions from s' with lower bound LB' then
27:         LB = max(LB, r + γ · max{lower bound LB' for transitions in successors[s']})
28:     end if
29:     return LB                                        ▷ tightest lower bound
30: end function

31: procedure PROPAGATELB(s_0)
32:     S = [s_0]                                        ▷ list of states to visit
33:     while states in S do
34:         s = S.pop(0)                                 ▷ remove and obtain first element in S
35:         if s has predecessors and successors then
                                                         ▷ if sequence of at least two transitions exists
36:             LB' = max{lower bounds LB'_Q for transitions in successors[s]}
37:             for (a_p, r_p, s_p) in predecessors[s] do   ▷ iterate predecessors of s
38:                 LB_2 = r_p + γ · LB'
39:                 if LB_2 > existing bound for s_p → s then
40:                     update LB_Q in successors[s_p] for the transition to s
41:                     S.add(s_p)                       ▷ propagate LB_Q to predecessors of s_p
42:                 end if
43:             end for
44:         end if
45:     end while
46: end procedure
```

8.2.2 Zero Actions

As discussed above, loose ends are discarded when a Q-Graph is constructed. It would therefore still be desirable to include information from loose end transitions into the Q-Graph however. In many settings, this is possible through *zero actions* $\mathbf{a}_z$: those are actions that do not change the agent's state, e.g., moving by zero units or applying zero force.

If those are applicable in all states, a self-loop can be added to every single node in the data graph (without actually executing an additional action). This effectively eliminates all loose ends and turns them into disconnected states. In other words it allows the Q-Graph $\mathcal{G}$ to contain all transitions from the data graph and compute their exact Q-values for the simplified MDP $\mathscr{M}^{\mathcal{G}}$.

8.2.3 Q-Graph Values as Lower Bounds

In general, the original MDP $\mathscr{M}$ contains more states or transitions than the Q-Graph-induced $\mathscr{M}^{\mathcal{G}}$. Then, the Q-values $Q^{\mathcal{G}}$ do not transfer to the original MDP as a correct solution but can be used as lower bounds for Q-values in the original MDP.

Assume w.l.o.g. that at least two transitions $(\mathbf{s}_0, \mathbf{a}_1, r_1, \mathbf{s}_1)$ and $(\mathbf{s}_1, \mathbf{a}_2, r_2, \mathbf{s}_2)$ are known and part of the Q-Graph $\mathcal{G}$. Since Q-values for all transitions in $\mathcal{G}$ can be computed exactly using Q-iteration, the Bellman optimality equation applies:

$$Q^{\mathcal{G}}(\mathbf{s}_0, \mathbf{a}_1) = r_1 + \max_{\mathbf{a} \in \mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}} Q^{\mathcal{G}}(\mathbf{s}_1, \mathbf{a}) \tag{8.2}$$

where $\mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}$ denotes all actions on out-going edges from $\mathbf{s}_1$.

In the original MDP $\mathscr{M}$ with potentially continuous state and action spaces, unseen states and transitions may exist. Still, for deterministic MDPs $\mathscr{M}$ and $\mathscr{M}^{\mathcal{G}}$, the Q-value is lower bounded due to the $\max$ operation and the fact that the available actions in the Q-Graph ($\mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}$) are a

subset of those in the continuous action space $\mathcal{A}$:

$$Q(\mathbf{s}_0, \mathbf{a}_1) = r_1 + \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}_1, \mathbf{a}) \tag{8.3}$$

$$\overset{\mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1} \subseteq \mathcal{A}}{\geq} r_1 + \max_{\mathbf{a} \in \mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}} Q(\mathbf{s}_1, \mathbf{a})$$

$$\overset{\mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1} \subseteq \mathcal{A}}{\geq} r_1 + \max_{\mathbf{a} \in \mathcal{E}^{\mathcal{G}}_{\mathbf{s}_1}} Q^{\mathcal{G}}(\mathbf{s}_1, \mathbf{a}) = Q^{\mathcal{G}}(\mathbf{s}_0, \mathbf{a}_1) \tag{8.4}$$

Thus, each Q-value for a transition in the Q-Graph $\mathcal{G}$ represents a lower bound of the Q-value for the same transition in the original MDP on continuous state and action spaces. In contrast to the prior work in He et al. (2017), these lower bounds do not depend on the current Q-estimate but hold for the optimal Q-value in general.

Note that the $\max$ operation in Eq. (8.4) operates on a discrete space and can thus be computed by a simple look-up and comparison of all known transitions from $\mathbf{s}_1$. The $\max$ operation in Eq. (8.3) does not need to be evaluated additionally.

For non-deterministic dynamics, potentially less tight bounds can be established under additional assumptions: If for any state and any series of actions $\mathfrak{A}$, the empirical return R that an agent can observe when following $\mathfrak{A}$ from $\mathbf{s}$ differs by at most δ, then all $Q^{\mathcal{G}}$-values apply as lower bounds with margin δ:

$$Q(\mathbf{s}, \mathbf{a}) \geq Q^{\mathcal{G}}(\mathbf{s}, \mathbf{a}) - \delta \tag{8.5}$$

Since non-deterministic environments are quite common and δ may not be known, I will additionally evaluate the empirical performance of my method under violation of the determinism assumption.

8.2.4 Q-Graph-bounded Q-learning

Bounds on Q-values, for instance those computed in Eq. (8.4), can be enforced in TD learning by clipping target values from Eq. (2.6) as follows:

$$Q_{\text{target}}(\mathbf{s}, \mathbf{a}) = r + \begin{cases} 0, & \text{if } \mathbf{s}' \text{ is terminal} \\ \gamma \cdot \widehat{Q}(\mathbf{s}', \pi(\mathbf{s}')), & \text{else} \end{cases}$$
$$Q^{\mathcal{G}}_{\text{target}}(\mathbf{s}, \mathbf{a}) = \max\left(\text{LB}, Q_{\text{target}}(\mathbf{s}, \mathbf{a})\right) \tag{8.6}$$

where LB can generally be any lower bound; in Q-Graph-bounded Q-learning I use the Q-value for the same transition in the Q-Graph LB $= Q^{\mathcal{G}}(\mathbf{s}, \mathbf{a})$. If another lower bound is known, e.g., based on a bounded reward as in Eq. (7.3), LB can be the maximum over all available lower bounds. Analogously, upper bounds UB could be enforced using the $\min$ operation.

He et al. (2017) have not applied clipping as I suggest here but encoded the constraints into the loss function from Eq. (2.9):

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{U}(\mathscr{D})}\Big[& \Big(Q_{\text{target}}(\mathbf{s}, \mathbf{a} \mid \boldsymbol{\theta}^-) - \widehat{Q}(\mathbf{s}, \mathbf{a} \mid \boldsymbol{\theta})\Big)^2 \\ & + \lambda (L^{\max} - \widehat{Q}(\mathbf{s}, \mathbf{a}))_+^2 \\ & + \lambda (\widehat{Q}(\mathbf{s}, \mathbf{a}) - U^{\min})_+^2 \Big] \end{aligned}$$

where $L^{\max}$ is the highest lower bound and $U^{\min}$ the lowest upper bound from Eq. (8.1). Empirically, I did not find a significant difference between the two variants and therefore used the clipping version which has less hyperparameters. Since our lower bounds are correct in general however, it also makes sense to enforce them strictly. The bounds by He et al. (2017) in comparison are heuristic and therefore may be better applied as soft constraints.

I refer to the suggested method of enforcing Q-values from $\mathcal{G}$ by clipping the target values for TD learning as *Q-Graph-bounded Q-learning*. When the Q-function is represented by a function approximator, e.g., a neural network in DDPG, it is defined for a continuous state and action space. While training however, the Q-targets are constrained by bounds derived from the Q-Graph-based $Q^{\mathcal{G}}$-values on a discrete domain.

If a state-action pair is not associated with a lower bound, i.e., loose ends, it can still be used as usual in Eq. (2.6) without clipping of their target value. If coincidentally no bounds are violated, the proposed method then reduces to vanilla DDPG. A full training step is illustrated as pseudocode in Algorithm 4. Note that the pseudocode also includes a priori bounds which were only introduced as baselines in the experimental sections below.

Algorithm 4 QG-bounded Deep Deterministic Policy Gradient (QG-DDPG)

procedure TRAINSTEP(
discount factor γ,
actor network π, ▷ mapping states to actions
critic network $\widehat{Q}$, ▷ predicting Q-values for state-action pairs
Q-Graph $\mathcal{G}$, ▷ see Algorithm 4
a priori lower bound LB^{AP}, ▷ $-\infty$ if unknown
a priori upper bound UB^{AP}) ▷ $+\infty$ if unknown

▷ sample minibatch with N transitions from Q-Graph:
$\mathscr{T}_i = (\mathbf{s}_i, \mathbf{a}_i, \mathbf{s'}_i, r_i, {Q^{\mathcal{G}}}_i)_{i=0}^{N} \sim \mathcal{G}$
▷ ${Q^{\mathcal{G}}}_i$ is set to $-\infty$ if unknown (e.g. loose end)

▷ compute classical target Q-values as in Eq. (8.6)
$\forall i. Q_{\text{target}}(\mathbf{s}_i, \mathbf{a}_i) = \begin{cases} r_i, & \text{if } \mathbf{s}' \text{ is terminal} \\ r_i + \gamma \cdot \widehat{Q}(\mathbf{s}', \pi(\mathbf{s}')), & \text{else} \end{cases}$
$\forall i. \text{LB}_i = \max({Q^{\mathcal{G}}}_i, \text{LB}^{\text{AP}})$ ▷ tightest lower bound
$Q^{\mathcal{G}}_{\text{target}}(\mathbf{s}_i, \mathbf{a}_i) = \min(\text{UB}^{\text{AP}}, \max(\text{LB}_i, Q_{\text{target}}(\mathbf{s}_i, \mathbf{a}_i)))$ ▷ clipping (Eq. (8.6))

$\mathcal{L}_{\widehat{Q}} = \frac{1}{N} \sum_{i=0}^{N} (Q^{\mathcal{G}}_{\text{target}}(\mathbf{s}_i, \mathbf{a}_i) - \widehat{Q}(\mathbf{s}_i, \mathbf{a}_i))^2$ ▷ DDPG critic loss
$\mathcal{L}_{\pi} = -\frac{1}{N} \sum_{i=0}^{N} \widehat{Q}(\mathbf{s}_i, \pi(\mathbf{s}_i))$ ▷ DDPG actor loss
Optimize both networks using $\mathcal{L}_{\pi}$ and $\mathcal{L}_{\widehat{Q}}$
end procedure

8.3 Experimental Results

I have evaluated the proposed method in a series of experiments: first on a classical example for divergence in Q-learning after Baird (1999) in Subsection 8.3.1, then on a simulated continuous control task which has been introduced in detail in Subsection 3.1.1.

8.3.1 Baird's Star Example

The 7-state star problem (Figure 8.1) was proposed by Baird (1999) to demonstrate convergence issues in value iteration with (linear) function approximation and often serves as a baseline tasks for approaches against divergence (e.g., Durugkar and Stone (2018)). The agent receives a reward of zero for each action and thus the correct solution to the problem is to set all weights to zero and obtain state-values of zero. If all weights are initially positive and w_0 larger than the others, this causes oscillatory behavior of both state values and weights. I reproduced the exact setting and result plots for Figure 4.2 in Baird (1999). Applying the proposed graph view to the problem, I can derive a lower bound of zero for V_7 because it has a self-loop with reward 0; and thus this lower bound recursively leads to a lower bound of $0 + \gamma V_7 = 0$ for all other states. These graph-based bounds can be applied in TD learning in analogy to Eq. (8.6) as $V_{\text{target}}(\mathrm{s}) = \max(\text{LB}, r + \gamma V(\mathrm{s}'))$. As a result, the proposed method converges to the correct state values rather than spiraling out to infinity as Figure 8.1 illustrates. This is a notable effect because the lower bound here not only stops divergence to minus infinity but an oscillatory behavior.

8.3.2 Simulated Clearance Fit Peg Insertion Task

To allow for extensive hyperparameter studies, I evaluated the proposed method on a simulated continuous control task in terms of sample efficiency and robustness to hyperparameters (subsubsection 8.3.2.2). In subsubsection 8.3.2.3, I verify that the outcome on the continuous control problem is in line with the insights about soft divergence from example I in Subsection 7.5.3. I further examined the impact of zero actions in subsubsection 8.3.2.4, different types of upper and lower bounds on Q-values (subsubsection 8.3.2.5) as well as the method's interaction with limited

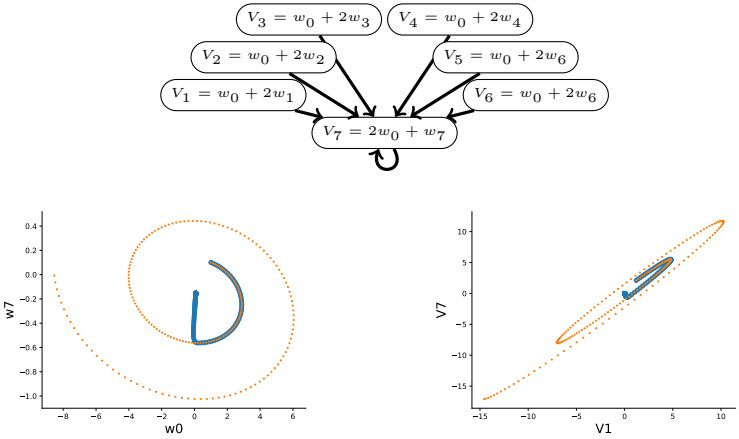


Figure 8.1: Graph-based bounds quickly lead to the correct solution (blue, solid) on the 7-state star problem after Baird (1999), for which states and weights spiral out to infinity under vanilla TD learning (orange, dotted).

replay memory capacity (subsubsection 8.3.2.6). Finally, I empirically assess the impact of non-deterministic transition dynamics in subsubsection 8.3.2.7.

8.3.2.1 Network Details

For all upcoming experiments I used the following instance of a standard DDPG architecture for learning: The critic network consists of three fully connected layers with 200 nodes each. For the inner layers, ReLU activations were used. The network was initialized with weights sampled from $\mathcal{N}(\mu = 0, \sigma = 0.001)$. The actor network also consists of three fully connected layers with 200 nodes each, but used tanh activations and was initialized from a He-uniform distribution (He et al., 2015). All neural networks were implemented using Tensorflow (Abadi et al., 2015) and optimized using the Adam optimizer (Kingma and Ba, 2014), with 50 training epochs after each episode (i.e., 200 agent steps) and up to 15 random mini batches of data per epoch. No target network was used, since those are known to prolong training and thereby postpone convergence issues but not solve them (Van Hasselt et al., 2018).

I tested vanilla DDPG for 300 episodes on a grid of learning rates for actor and critic in $\{10^{-2}, 10^{-3}, 10^{-4}\}$ and chose three sets of hyperparameters for the following experiments that are representative for the spectrum of DDPG performance, see Figure 8.2. Each experiment was repeated ten times with different random seeds. In our plots, all learning curves are summarized such that the solid line represents the mean performance over all runs and the shaded area highlights the standard deviation of the mean estimator, i.e., $\frac{\sigma}{\sqrt{n}}$ for n runs.

8.3.2.2 Sample Efficiency and Robustness to Hyperparameters

I hypothesized that Q-Graph-based lower bounds would correctly limit the range of Q-values which prevents some cases of soft divergence and thereby increases sample efficiency. I further hypothesized that explicit bounds would barely have any impact in cases when vanilla Q-learning works well, because my method as described in Eq. (8.6) reduces to standard TD learning when no bound is violated. In other words this implies that Q-Graph-bounded Q-learning should never decrease performance.
For a first overview, I compared learning curves of Q-Graph-bounded Q-learning ('Q-Graph (QG)') to those of vanilla DDPG; see the lower plot in Figure 8.2. As expected, Q-Graphs speed up learning for all examined learning rates. The effect size varies and is larger for those learning rates that lead to relatively poor performance in vanilla DDPG. This decreases the gap in performance between different learning rates and can therefore be interpreted as an indicator for increased robustness to hyperparameters.

8.3.2.3 Variance of Predictions

To assess if this increase in performance is due to similar effects as in the examples in Subsection 7.5.3, I evaluated the variance in predicted Q-values at the end of each experiment under the learning rate with largest effect size (10^{-4}). I covered the state space with a regular grid of 27 states and evaluated the learned Q-value for each of these states with a set of eleven given actions ('given') as well as with the action that the actor network suggests for each state ('pi').

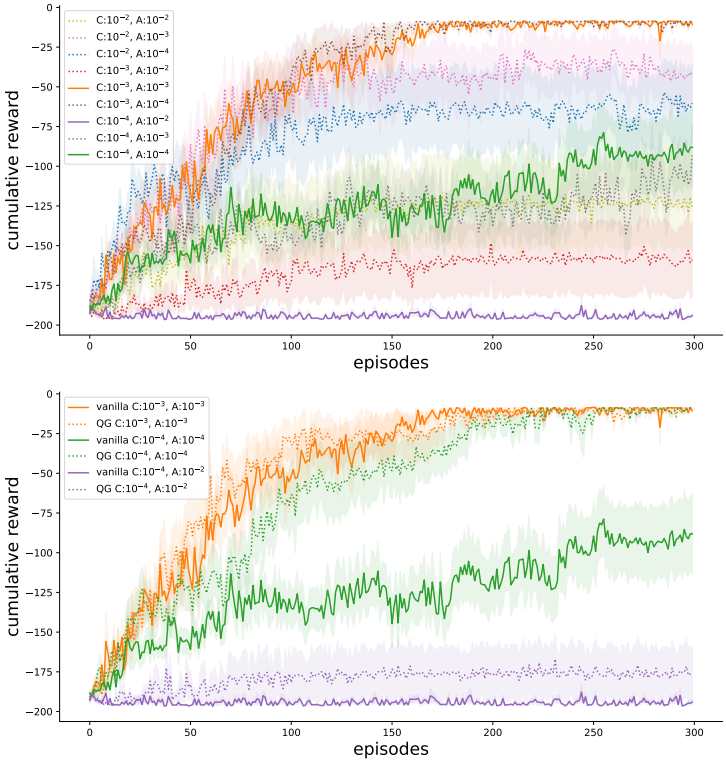


Figure 8.2: Performance of vanilla DDPG on the full grid of learning rates (top). Three representative parameters were identified (solid lines) and compared to the proposed method ('QG', dotted lines) on the bottom of the plot.

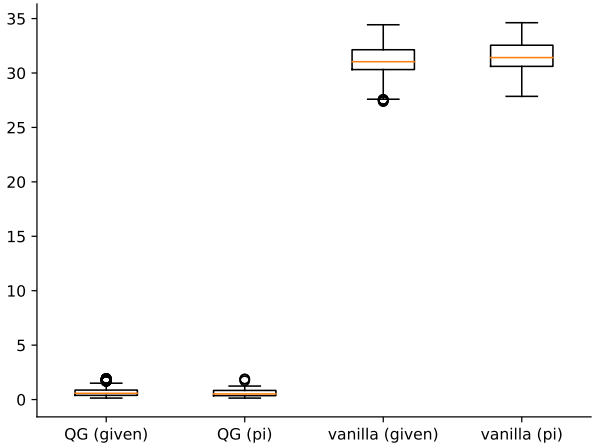


Figure 8.3: Standard deviation of predicted Q-values.

For the boxplot in Figure 8.3, I collected the standard deviations over the predicted Q-values for each state-action pair from 10 runs with different random seeds. The orange line indicates the median value, the box extends from the lower to the upper quartile value, the whiskers cover 1.5 times the inter quartile range and outliers are shown as circles. The results shows very clearly that Q-Graph-runs resulted in significantly less variance for predicted Q-values, indicating that Q-Graph-bounded Q-learning does indeed prevent cases of soft divergence as already suggested in Subsection 7.5.3.

8.3.2.4 Zero Actions

A zero action does not change the agent's state, in this case the offset in position by zero meters. I compare DDPG on the replay memory as is ('vanilla') to DDPG on enhanced data that was created by adding zero actions after each transition ('vanilla-ZA'). This improves the structure of the data graph by turning loose ends into disconnected transitions. The results as illustrated by the dotted lines in Figure 8.4 show that adding zero

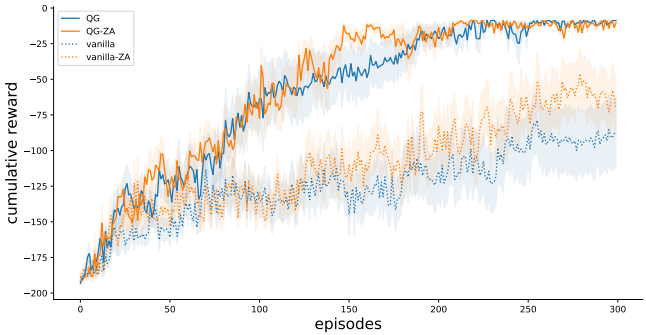


Figure 8.4: Performance of both vanilla DDPG and QG-DDPG with and without zero actions (Zero Action (ZA)) which effectively eliminate loose ends.

actions does lead to a slight improvement, even without any Q-Graph-bounded learning—emphasizing the importance of the data graph structure for Q-learning in general.

I also compared the proposed method ('QG') with and without zero actions (solid lines) which barely shows any effect. The largest gap in Figure 8.4 is clearly between vanilla-ZA and either version of QG. This indicates that while the data graph structure matters, the propagation of information through the Q-Graph and the integration of lower bounds into TD-learning are the main benefits of the proposed method.

8.3.2.5 Further Baselines

To prepare the evaluation of baselines, I first tuned hyperparameters for one of the main baselines: the bounds from He et al. (2017). All experiments were performed with the best set of learning rates I identified in Figure 8.2, i.e., both learning rates were fixed to 10^{-4}.

Using the same additional hyperparameters He et al. (2017) report in the paper ($\lambda = \mathrm{K} = 4$) did not perform well on my task and reward function. Instead, much smaller values for λ turned out useful for per-

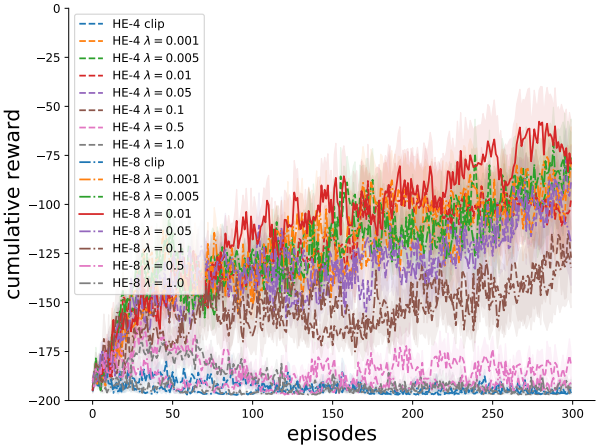


Figure 8.5: Performance of the He et al. (2017) baseline on the full grid of hyperparameters.

formance. Figure 8.5 shows the results of tuning the method with a grid of hyperparameters where $\lambda \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$ and $\mathrm{K} \in \{4, 8\}$.

Additionally, I experimented with clipping Q-targets as in Eq. (8.6) rather than using λ and a regularization term. Since the data in my replay memory was stored as a graph already, I sampled K steps in the future and past from this graph for each update step. It is theoretically possible that experience from multiple trajectories has been mixed if they shared a state.

I compared the proposed theoretically grounded Q-Graph-based lower bounds to various baselines:

1. I included the best case performance for the approach by He et al. (2017) ($\mathrm{K} = 8$, $\lambda = 0.01$) from the previous tuning procedure.

2. For a given range of possible rewards $[r_{\text{min}}, r_{\text{max}}]$, the range of possible Q-values can be determined using Eq. (7.3). Note that these bounds are the same for each transition though, whereas Q-Graph-based bounds are determined separately for each state-action pair.

For *a priori* bounds I used prior knowledge about the true r_{min} and r_{max} for the problem at hand, which leads to correct lower and upper bounds.

3. For *empirical* bounds, I determined r_{min} and r_{max} as the currently lowest and highest reward from the replay memory and then used Eq. (7.3) to derive lower and upper bounds. These bounds may not be correct in general.

Both a priori and empirical bounds can be used with vanilla DDPG: in this case, the target values are clipped by the given bounds. However, additional bounds can also be combined with Q-Graph-bounded Q-learning: then, the target values are clipped by the tightest available bounds, i.e., using the highest lower and the lowest upper bound. Adding further bounds to the Q-Graph-based method mainly adds upper bounds to the learning process, because the Q-Graph-derived lower bounds are available for all transitions except loose ends already. Note that the empirical bounds may be incorrect and could therefore clip the target value to wrong ranges.

I first compare unbounded DDPG ('vanilla') to DDPG with a priori and empirical bounds. As the dotted lines in Figure 8.6 show, there are only marginal differences but incorrect empirical bounds perform worst. Using bounds based on the current Q-estimates as in He et al. (2017) outperforms all tested variants of vanilla DDPG (red dashed line).

However, comparing the proposed Q-Graph-based bounds (solid lines) shows that Q-Graph-bounded Q-learning outperforms both trivial bounds only as well as He et al. (2017)'s bounds. Using incorrect empirical bounds has a significant adverse effect on QG; adding a priori bounds to the proposed method does not seem to have any significant effect. I hypothesize that this may be because mainly upper bounds are added but the behavior of a Q-learning system differs for under- and over-estimated states: while under-estimated states may be rarely visited, over-estimated states are likely to be visited using the currently estimated optimal policy. Therefore, lower bounds correcting under-estimated states may be more important than upper bounds which would correct over-estimated states. Overall, I conclude that the tight transition-specific lower bounds from the Q-Graph are key.

Figure 8.6: Baselines: Q-Graph-bounded Q-learning ('QG', solid lines) versus vanilla DDPG (Lillicrap et al. (2015), dotted), both combined with empirical and a priori bounds; as well as the bounds from He et al. (2017) (dashed) which are based on n-step returns and the currently predicted Q-values.

8.3.2.6 Limited Graph Capacity

In deep RL, the replay memory is typically a FIFO-buffer ('first in, first out'), i.e., those elements that were added first are overwritten first when the buffer is full. For a data graph, it is possible to delete single transitions but there are two possible effects: On the one hand, some information from deleted transitions can be implicitly contained in its predecessors' Q-values on the Q-Graph, which could imply that the proposed method is more robust to small memory capacities. On the other hand, cuts from deleted transitions can stop information propagation through the Q-Graph, which could in turn slow down further progress.

I therefore empirically compared the drop in performance for vanilla DDPG and Q-Graph-bounded Q-learning with graph capacities of 1000 and 5000 transitions. For comparison, the average unlimited graph contained roughly 30,000 unique transitions at the end of the 300 episode experiments. As Figure 8.7 illustrates, a Q-Graph-based method that is limited to only 1000 samples still performs on par with unlimited vanilla DDPG, while the vanilla DDPG performance decreases for a limit of 1000 transitions.

8.3.2.7 Non-Deterministic Transitions

As discussed in Section 8.2.3, the Q-Graph-derived lower bounds are based on the assumption that all transitions are deterministic. In case of non-deterministic transitions, correct lower bounds can be derived if for any state and any series of actions $\mathfrak{A}$, the empirical return R that an agent can observe when following $\mathfrak{A}$ from s differs by at most δ. In practice however, δ may not exist or be unknown. I therefore empirically compare the results from subsubsection 8.3.2.2 with increasing amounts of transition uncertainty. To obtain the results shown in Figure 8.7, each action was sampled from a Gaussian around the actor output with different σ: $\mathcal{N}(\pi(\mathrm{s}), \sigma)$. The results show that the performance generally drops with non-determinism for all methods, but the improvement of Q-Graph-bounded Q-learning over vanilla DDPG remains significant.

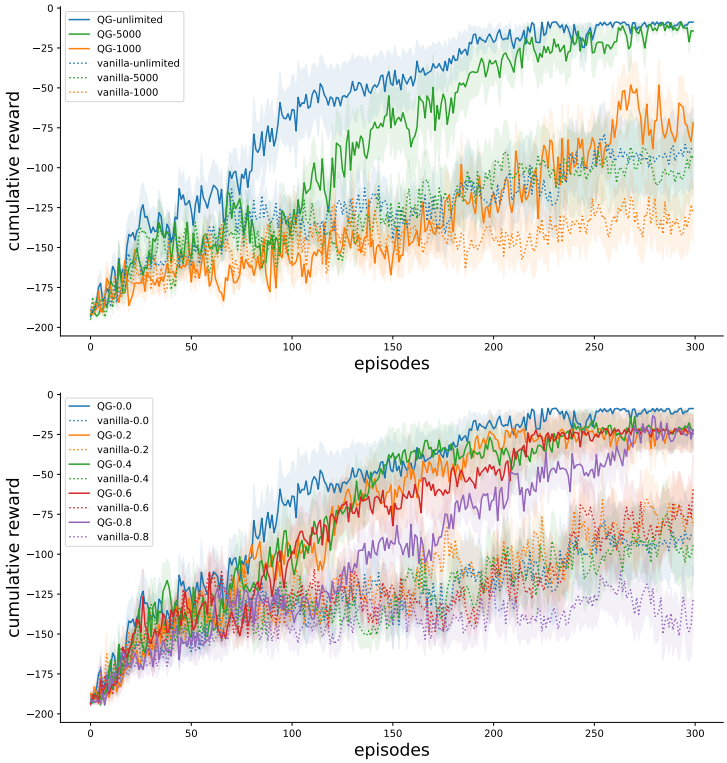


Figure 8.7: Performance with limited graph capacity (top) and increasingly non-deterministic transitions (bottom).

8.4 Discussion

The experiments in the previous chapter had shown that there is a link between data graph structure and soft divergence in Q-learning based deep RL methods. Loops without connections to a terminal state were more likely lead to soft divergence than other types of transitions although their Q-value can be computed analytically under the assumption that no further transitions exist. In this chapter, I have extended this by the insight that this analytically derived Q-value is a lower bound to the actual Q-value in the continuous MDP due to the $\max$ operation in TD learning (Eq. (8.4)).

The proposed method constructs a simplified MDP from a subgraph such that its exact Q-values can be computed by Q-iteration—resulting in a Q-Graph. This subgraph does not contain loose ends, but I introduce so-called *zero actions* which, if known, can be used to integrate loose ends into the Q-Graph as well. Q-values on the discrete simplified MDP associated with the Q-Graph represent lower bounds for the Q-values in the original continuous MDP. Enforcing these bounds in TD-learning empirically prevents cases of soft divergence on a continuous control task. Empirically I have shown that Q-Graph-bounded Q-learning indeed leads to less standard deviation in the distribution of predicted Q-values, which I have used as a proxy to measure soft divergence in the previous chapter.

Preventing soft divergence as the proposed method does, also increases sample efficiency on average and has its strongest impact under unfavorable hyperparameters; in other words the proposed method increases robustness to adverse hyperparameters. I have also demonstrated that the Q-Graph can serve as an additional implicit memory holding information from transitions that have already been overwritten in the replay memory and thus, the algorithm is able to cope better with restricted memory capacity. In contrast to prior work which has only derived either relatively loose bounds which are equal for each transition (Lee and Kim, 2015) or bounds that are based on the currently predicted Q-values (He et al., 2017), the bounds I have derived for Q-Graph-bounded Q-learning are correct in general. Empirically, the method also works in non-deterministic settings despite being derived under the assumption of deterministic transitions.

Taking a graph perspective on the data leads to a non-parametric representation which holds information about the graph structure. This representation is complementary to the parametric approach that is taken in function approximation, for instance in DDPG where each state is a multi-dimensional vector. The work presented in this chapter can therefore be seen as combining parametric information for function approximation with non-parametric information from the graph structure.

Relating to Chapter 5 and Chapter 6, the work in this chapter can also be seen as one more way to integrate prior (expert) knowledge into a model-free learning framework. Prior knowledge about the reward function can be used to derive a priori bounds as in Lee and Kim (2015) and prior knowledge about the action space can be used to define zero actions which can help to improve the structure of the data graph to learn from. In practice, these two ideas are very practical, because very often a MDP is not given from the start but instead, it is often an engineer who—searching for a good solution—describes the problem as a MDP and can therefore also actively choose the action representation and reward function such that they allow to apply a priori bounds and zero actions.

8.5 Limitations and Open Questions

I have shown that graph structures and the information contained in their structure can have a substantial impact on learning stability. Although the experiments in this chapter were conducted on a more realistic simulated control task than the small toy example from the last chapter, it remains open whether these effects could also be found on a real robotics system. In the upcoming chapter, I will thus present evidence from such a real-world systems study.

Moving on to real robotics systems, one of the main theoretical stepping stones is uncertainty. Most derivations in this chapter were based on the assumption of (near-)determinism and to this point it is unclear in how far the experiments on Gaussian non-determinism would transfer to robotics applications.

On a theoretical level, it remains unclear whether there are milder assumptions than the ones I have presented but that would allow to derive bounds for the expected Q-value also in non-deterministic settings. All the transition-specific bounds from a Q-Graph are lower bounds. It re-

mains open whether correct upper bounds can be derived, or alternatively which further useful heuristics can be defined. On the other hand, it is also unclear in which cases function approximation for Q-learning based model-free deep RL tends to overestimation, to underestimation or to a spiralling behavior as in Baird's star example (Baird, 1995). If there is a way to characterize these issues in function approximation, potentially one could also derive further guidelines on how to design a MDP such that learning is facilitated as far as possible.

Propagating information through a non-parametric graph structure is clearly related to trajectory-centric approaches such as eligibility traces (Precup et al., 2000). The bounds suggested by He et al. (2017) make use of these trajectories by explicitly incorporating n-step returns. One open question is therefore also whether it can be disentangled, to which degree the speed-up in learning for Q-Graph-bounded Q-learning is based on the fact that information is propagated along the full episode trajectory and how large the impact of bounds which prevent divergence is on sample efficiency.

Chapter 9

System Study: Q-Graph-bounded DDPG for Industrial Assembly

In the previous two chapters I have shown that the structure of the graph representing the data that an agent is supposed to learn from has a direct impact on learning stability—at least for Q-learning based deep RL methods, for deterministic environments and a small number of relatively simple example problems. In this chapter, I will therefore present results from extensive empirical evaluations of QG-bounded Deep Deterministic Policy Gradient (QG-DDPG) for a contact-rich task from industrial assembly for real robots and under substantial uncertainties (see `https://youtu.be/Z_GcNbCWE-E`).

9.1 Introduction & Motivation

This chapter aims to empirically address questions that were left open in the previous two chapters and additionally evaluate the practical usefulness of the proposed approach for industrial automation tasks. The following paragraphs will revisit these points in more detail.

In Chapter 7, I have demonstrated that there is a link between the structure of the data graph and the extent of soft divergence as measured by the standard deviation of predicted Q-values on small toy examples. In this chapter, I will corroborate that soft divergence is a problem not only on contrived examples but also for real-world challenges in RL. For improvements in sample efficiency with QG-DDPG, I will provide insights indicating that the observed improvements correlate strongly with the degree of soft divergence and therefore suggest a causal link between soft divergence and graph structures—even for complex real-world scenarios.

QG-DDPG was built on the assumption that transitions and rewards are deterministic. While I have shown that artificially introduced Gaussian noise on transitions probability only has mild effects on the algorithm's performance, it remained unclear whether this robustness against non-determinism would transfer to real-world settings. The task I will examine in this chapter involves a lot of friction and contact between different objects which the robot can only perceive through a force-torque sensor at the wrist. Additionally, the shaft to insert into the hole is freely grasped and thus can slip between the grippers. Both slippage and force-sensing introduce a great amount of uncertainty and non-determinism into this setting, but as I will show, Q-Graph-based lower bounds still improve learning stability for DDPG.

There is a strong interest from industry to economically automate contact-rich assembly tasks such as shaft insertion or cable plugging for small and medium lot sizes. Classical solutions based on compliant control require prohibitive effort in designing appropriate manipulation strategies and tuning, even though the latter might be attenuated with black-box optimization strategies (Johannsmeier et al., 2019). To assess the applicability of the proposed method in industrial automation, I applied the method on the task that was introduced in Figure 3.5. The following evaluation criteria were considered:

- Sample Efficiency:
  it is generally desirable to have a task formulation that allows the robot to quickly learn its task. I will evaluate the impact of Q-Graph-based lower bounds not only on learning stability but also on speed.

Additionally, I chose a residual policy formulation for this task that can be seen as incorporating prior knowledge about the task and is known to improve sample efficiency in learning (Silver et al., 2018; Johannink et al., 2019).

Another way to improve sample efficiency that I exploit is to design the action space such that model-based controllers can be used (Martín-Martín et al., 2019). This idea can be seen as an analogy to Chapter 6 where I used model information during exploration.

- Simplicity:
  Ideally, it is easy to set up a learning task even for engineers with relatively weak background in machine learning. Therefore the residual policy is not built around a complex manually designed insertion strategy. Instead only a coarse approximation to the desired movement is used: a constant force pushing in z direction of the EEF (see Figure 9.1).

  Another aspect of simplicity is the amount of expertise that is required to tune an algorithm. To lower the burden to get to good solutions, learning systems should therefore be as robust as possible against changes in hyperparameters and the reward function. I have therefore evaluated not only results for a variety of hyperparameters but also investigated the simple to set up sparse reward function as an alternative to the relatively intricate distance-based reward with position and rotation components that I used before.

- Computing Requirements:
  While all experiments in this chapter were conducted with a desktop machine next to the robot, one may want to consider robots with embedded computing resources in the future. As these resources will be scarcer than those on a separate machine, I have also assessed whether the results from Chapter 8 transfer to the real world and could show that QG-DDPG copes better with limited memory than vanilla DDPG.

- Generalization:
  For most insertion tasks, it is possible to engineer a solution which solves the task in a single specific setting, e.g., for one given peg and hole pose. However, it is typically much more difficult to design a general solution which can also be applied to further settings,

Figure 9.1: Close-up view on an e-Bike drive unit. The task was to insert the shaft into a tight ball bearing inside the drive unit's housing, which is turned upside down for images and the video to capture the relevant areas from a more natural angle. The colored arrows illustrate the EEF's coordinate system.

e.g., a slightly rotated hole. In production automation such tasks are particularly important though because the robot typically does not perform its task in isolation. It is more likely embedded into a production line and thus has to deal with all uncertainties that preceding production steps have introduced. For insertion tasks this often means for example that the target object may be placed in slightly different orientations. I therefore added an additional experiment showing that the solution I propose in this chapter transfers to different orientations of the target object. This is again possible because of the way I defined the MDP for the learning task: the states do not include any pose information. Instead the robot operates on force input only which naturally transfers much better to unseen target poses.

9.2 Industrial Shaft Fitting Task

In this work, I consider a realistic industrial manufacturing step taken from a Bosch eBike motor plant. The task is to achieve a tight fit between a shaft and a ball bearing in a motor housing. The insertion task considered in this work requires both high accuracy and significant force when solved using classical machinery: High accuracy is required to precisely align the bearing and the shaft. High force is required to overcome significant resistance of the fitting process originating from the mechanical specification and static friction effects. The accompanying video (`https://youtu.be/Z_GcNbCWE-E`) and Subsection A.3.1 show the insertion as performed by a human, which requires force greater than 10 N and a determined push to reach the final configuration. Note that the motor housing is turned upside down for the videos to obtain a better field of view around the ball bearing. In manufacturing lines, the fitting process is realized using a hydraulic press operated by a human, combined with high-accuracy alignment equipment designed to ensure perfect centering of the shaft. In the plant process, between 10 and 15 seconds of time are available for the fitting. The assembly of such a tight fit appears to be a particularly interesting benchmarking problem for robot learning approaches: While it is simple enough to be reproducible at any time, it comes with interesting challenges due to the stochasticity of the reaction forces.

My objective is not to investigate the entire process including grasping the shaft and positioning it in the vicinity of the ball-bearing—these steps are beyond the scope of the paper. Instead, I focus on performing the insertion step: it starts in loose, randomly oriented contact with the ball bearing, requires significant interaction force and ends with the shaft being completely inserted with some predefined accuracy.

This task can be considered a variant of classical peg-in-hole insertion, which belongs to the most extensively studied assembly problems in robotics. A complete review of peg-in-hole insertion methods is beyond the scope of this thesis, so I provide a coarse overview in the following section.

9.3 Related Work

The industrial shaft fitting task for evaluation in this chapter comes with significant friction forces. Finding a model accurate enough for such contact-rich manipulation tasks however is challenging, even for data-driven approaches (Fazeli et al., 2017). Residual networks, in which only an offset to an analytical model is learned, have been shown to be a particularly efficient solution (Kloss et al., 2017). Residual policies transfer this idea to RL (Silver et al., 2018; Johannink et al., 2019), and will also be applied in this work.

Especially in the context of industrial robotics, much effort is spent on providing hardware solutions for reliably executing insertion tasks. Successful examples are the active and controllable remote center compliance element from Rueb and Becker (2016) or the vibration device presented in Kilikevičius and Bakšys (2011). Torque-controlled robots as well as manipulators equipped with force/torque sensors can be used to implement force-controlled approaches to peg insertion. Often, an analytic point of view is adopted, trying to model and understand the contact physics and then deriving control strategies (Bruyninckx et al., 1995; Li, 1997). A crucial element of these methods is the accurate estimation of contact states, which is challenging but pivotal to the success of the insertion (Fei and Zhao, 2003). Once a contact has been established, compliant controllers are used to perform the insertion itself (Lefebvre et al., 2005). Most classical methods require the specification of a sequence of contact states and careful controller design. There have been efforts to lessen the manual engineering work using black-box optimization for controller tuning (Johannsmeier et al., 2019). Nevertheless, they are typically not robust to variations of model parameters like (static) friction or force limits. Also, they still require intricate manual strategy design and significant tuning effort to work for specific instances of the problem.

Only few model-free RL approaches have addressed industrially relevant tasks and often make additional assumptions such as the availability of CAD models (Schoettler et al., 2019; Thomas et al., 2018; Wirnshofer et al., 2018). In Inoue et al. (2017), a peg insertion task is learned from discrete actions in a Q-learning formulation using LSTMs. My approach uses a continuous action space and therefore also deals with a more intricate optimization process in DDPG.

9.4 Residual Formulation

One of the major drivers for sample complexity in many RL problems is exploration (Plappert et al., 2017). In general, adding prior (possibly domain-specific) knowledge, can speed up this phase as I discussed in Chapter 6. If the character of the task is known, e.g., in my case a shaft insertion with a dominant direction of force, this knowledge can be efficiently incorporated into the problem through residual policies (Silver et al., 2018; Johannink et al., 2019): the agent then does not learn the full behavior from scratch but an addition to a fixed policy. In this spirit, I model the insertion task as an MDP with the following definitions.

9.4.1 States

A state is defined as $(\widehat{\mathrm{f}}_x, \widehat{\mathrm{f}}_y, \widehat{\tau}_x, \widehat{\tau}_y)$, which are the *estimated* contact forces and torques in x and y direction in the EEF frame, c.f. Figure 9.1. Torques along the z axis correspond to a rotation around the shaft's symmetry axis and are omitted.

9.4.2 Actions

The actions are formulated as task-space wrenches and consist of a constant and a residual policy component from the MDP:

- the constant part of the policy exerts a force $\mathrm{f}_z = -15$ N in z-direction of the EEF frame, c.f. Figure 9.1.

- the residual actions as defined in the MDP consist of torques $[\tau_x,\ \tau_y]$ along the x and y axes of the EEF frame and are computed directly as the output of the actor network scaled to the interval $[-3, 3]$ Nm.

Combined, a feedforward wrench $\boldsymbol{\zeta} = [0\ 0\ \mathrm{f}_z\ \tau_x\ \tau_y\ 0]$ is obtained and set as reference to a hybrid task-space force/impedance controller (Anderson and Spong, 1988) together with the current EEF pose. The controller position gains are constantly set to 250 N/m in x and y direction and 0 N/m in z direction. The orientation gain is set to be 4.0 Nm/rad in all directions.

When executing the policy, an action is considered completed once the robot's EEF reaches a steady state with velocities below a predefined threshold. In essence, this leads to the robot applying constant force in z-direction of the shaft, while the residual policy allows to apply torque to the shaft and rotate the EEF in space.

Lastly, the controller allows to impose a limit onto the orientation of the tool. In this work, I limit the maximum tool tilt to be equal to $\frac{\pi}{4}$ w.r.t. the horizontal ground plane, which essentially allows the learning algorithm to safely explore all EEF orientations within a cone of $\frac{\pi}{2}$ opening angle w.r.t. the table.

9.4.3 Reward

I investigate two different reward functions:

- Sparse reward:
  in this setting, a reward of $r = -1$ is given for each transition and $r = 0$ if the terminal state is reached.
- Dense reward:
  the reward is considered proportional to the distance error between the current EEF pose and a target EEF pose (corresponding to a fully inserted peg). The distance error in position Δ^P is computed as the ℓ_2 norm of the Euclidean position difference vector. The distance error in orientation Δ^R is computed as the ℓ_2 norm of the angle-axis error in x and y rotation, since the insertion task is invariant to z orientation. The combined reward is then computed as

  $$r = r_P + r_R = \frac{1}{2}\left(\exp\left(-\frac{\Delta^R}{\sigma_R}\right) + \exp\left(-\frac{\Delta^P}{\sigma_P}\right)\right) - 1$$

  with manually tuned scaling factors $\sigma_P = 0.015$ and $\sigma_R = 0.7$. This formulation guarantees that r is always in $[-1, 0]$ for the given task.

9.4.4 QG-DDPG

Employing the residual policy, the agent reaches the goal in many episodes—so instead of loose ends or disconnected transitions, the predominant type of transitions in my problem are indirectly connected ones. In Chapter 7

I had shown that the degree of soft divergence that is introduced into DDPG by indirectly connected transitions largely depends on the length of the path between a state and the nearest terminal state. In my setting, the path length between initial- and target state can easily reach high numbers, up to 1,000. I therefore take the results from Chapter 7 as a motivation to apply QG-DDPG from Chapter 8 to my industrial task. More precisely, I apply the following items in the QG-DDPG conditions:

1. Q-Graph-based lower bounds, enforced in TD targets
2. a priori lower and upper bounds based on the minimum and maximum reward: $\frac{r_{\text{min}}=-1}{1-\gamma} \leq Q \leq \frac{r_{\text{max}}=0}{1-\gamma} = 0$,
3. zero actions which here correspond to zero residual torques. The constant force in z direction remains but since an action is executed until a steady state is reached, it will not have an effect without changes in the torque.

9.5 Experimental Setup

All experiments in Section 9.6 followed the same setup: The training phase consisted of 40 episodes. To start an episode, the EEF was manually set to one out of eight initial poses with different inclinations (see Figure 9.2). Note that these manual resets introduce additional noise on the initial poses and only the rough orientation of the shaft was fixed. The initial poses were alternated in a fixed order such that metrics like the average number of steps it takes the robot to reach the goal, can be averaged with a kernel width of eight to average out the impact of different initial poses. Each episode was stopped when either the target state or a maximum of 1000 steps was reached. I implemented a pose-based heuristic to verify whether the target state is reached, but always confirmed this by detailed inspection and manual feedback because of possible slippage between shaft and gripper fingers. The network was trained after every cycle consisting of 20 steps; the number of training iterations after each cycle is one of the hyperparameters investigated in Subsection 9.6.2.

The test phase consisted of eight more episodes also covering all initial positions. A test episode was stopped after 200 steps if the target is not reached, the remaining setup remained unchanged to the training phase.

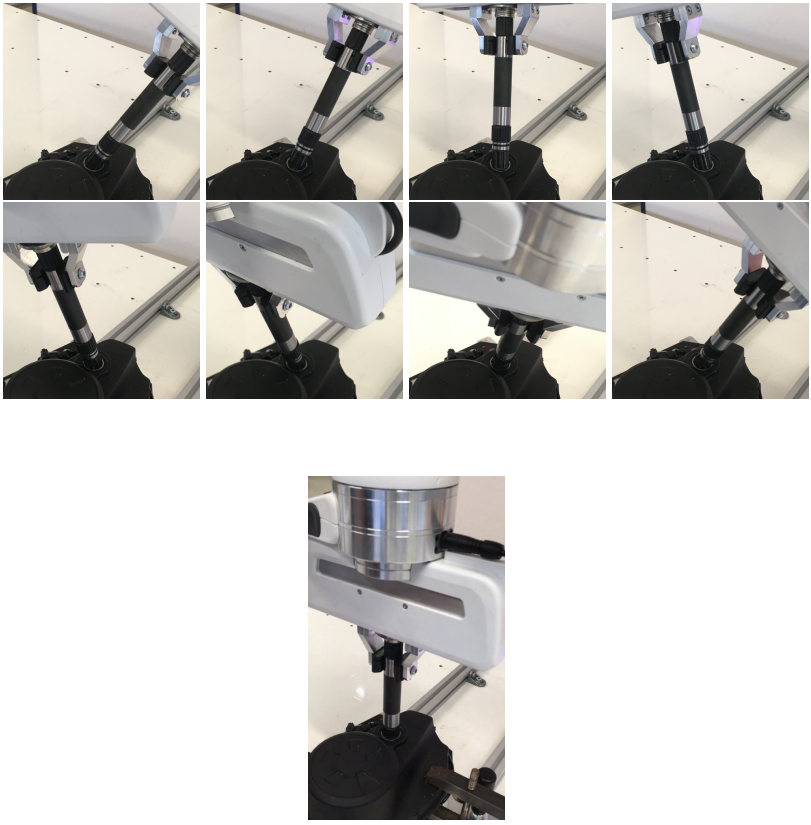


Figure 9.2: Top: Eight different orientations for initial poses that the robot is manually reset to in the beginning of each episode. Bottom: Target pose for comparison

9.5.1 Network Details

All networks were implemented in Tensorflow (Abadi et al., 2015). Both the actor and critic network consist of three fully connected layers, where the two hidden layers contain 100 nodes. The actor network has tanh activations on all layers and a two-dimensional output; all weights were initialized from a Glorot uniform distribution (Glorot and Bengio, 2010). The critic network has ReLU activations on the first two layers and no non-linearity on the one-dimensional output; the weights are initialized from a He uniform distribution (He et al., 2015). The forces and torques which served as state descriptors to the critic network were linearly scaled such that all values were in $[-1, +1]$. For optimization, the Adam optimizer (Kingma and Ba, 2014) was used—different learning rates and the number of training iterations per cycle were tuned on a grid of hyperparameters as described in Subsection 9.6.2. Following the argumentation in Van Hasselt et al. (2018), no target networks were used since they are known to delay but not prevent divergence.

9.5.2 Robot Control

All experiments were performed on a Franka Emika Panda CoBot, where I controlled the joint torques at 1 kHz using a custom control toolchain as introduced in Subsection 3.2.2. The gripper was controlled to grasp the shaft with a constant gripping force. In order to ensure a safe grasp, I used custom-printed finger tips shaped such that a variety of cylindrical objects can be centered and grasped robustly (see Figure 9.1).

9.6 Experimental Results

To validate the claims from Chapter 8 on the industrial force-fitting task, I evaluated the following aspects of the learning system: random baselines, robustness to hyperparameters, reproducibility of results, sparse rewards, limited replay memory capacity and generalization capability. Additionally, I evaluated the variance in predicted Q-values to close the loop to theoretical claims about soft divergence from Chapter 7.

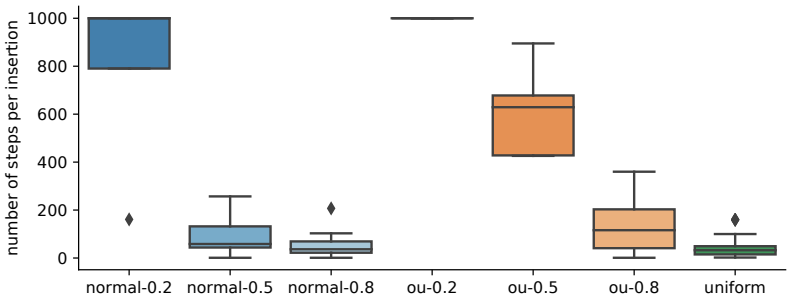


Figure 9.3: Random baseline performance: distribution of the number of steps per episode for different random actions. Each episode was stopped after 1000 steps if not successful and the experiment consisted of up to 3000 steps.

Due to the stochasticity of the experiments, I ran each experiment three times with different random seeds. All plots illustrating these results show the mean as a solid line surrounded by a shaded area representing the standard deviation of the mean estimator, i.e., $\frac{\sigma}{\sqrt{n=3}}$. One experiment took between 30 and 180 minutes, in total the results in the following sections sum up to approximately 60 hours of real world interaction.

9.6.1 Task Difficulty and Baselines

Significant force and precision is needed to fit the shaft into the ball bearing (see video attachment). However, the constant part of my residual policy already serves as a strong prior for this type of task. I therefore first evaluated a number of random baselines: instead of the actor net output, the residual policy consists of randomly sampled actions in the same output range. I compare uniform sampling and two of the standard noise processes for exploration in RL (Plappert et al., 2017), namely Gaussian noise ('normal') and Ornstein-Uhlenbeck ('ou'), both with different σ. For Ornstein-Uhlenbeck noise, I fixed $\theta = 1$ and dt $= 0.01$.

As Figure 9.3 shows, the uniformly sampled random actions show the best performance and solve the task in 32 steps on average. On first glance it may seem surprising that uniform sampling performs best but in

this case it is due to the residual formulation: uniform sampling leads to a wiggling kind of behavior with particularly large amplitudes which is the most successful trivial behavior for an insertion.

9.6.2 Sample Efficiency and Robustness to Hyperparameters

To obtain a broad overview of learning performance, I tuned those hyperparameters that are most related to sample efficiency on a grid: learning rates for actor and critic networks, as well as the number of training iterations per cycle.

I tested learning rates in $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$ for the critic and used one tenth of this learning rate for the actor. In prestudies a smaller learning rate for the actor seemed advantageous and confirms results from Chapter 8. Either 10 or 50 training iterations per cycle were used.

Each of these eight combinations of hyperparameters was tested with three random seeds and both algorithms, leading to robot interactions of approximately 48 hours for this particular experiment. Figure 9.4 shows the mean number of steps needed to successfully complete the task at test time for each combination of hyperparameters. Bearing in mind that the best baseline from Subsection 9.6.1 solved the task within 32 episodes, one can see that vanilla DDPG outperforms uniformly sampled action under only one particular set of hyperparameters. QG-DDPG, however, performs better than the best random baseline in six out of eight cases.

For closer inspection, Figure 9.5 depicts learning curves for the most favorable and unfavorable hyperparameters for both algorithms and plots the development of train and test performance. The shaded area represents the standard deviation of the mean estimator for performance during training episodes, the intervals on the right show the same confidence interval for test time results.

For the best case hyperparameters, both algorithms' test time performances are quite close, which is in line with the findings in Chapter 8. Interestingly, the variance during training is lower for QG-DDPG, potentially indicating higher reliability and reproducibility. For the worst case hyperparameters, one can observe that DDPG does not solve the task even once (at 200, the episodes were stopped if not successful). QG-DDPG also decreases in performance but still solves the task.

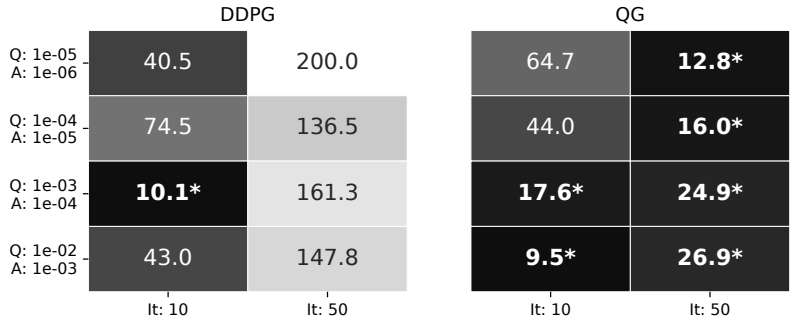


Figure 9.4: Performance comparison on full grid of hyperparameters, measured as steps needed to solve the task. Lower (darker) is better, entries beating all random baselines are highlighted by *. While classical DDPG outperforms the random baseline in just one out of eight cases, QG-DDPG achieves this in six cases.

The dotted green line illustrates anecdotal results from a single run of QG-DDPG in an extended setting where not only the initial shaft orientation was changed, but also the ball bearing orientation changed after four episodes during training and every second episode at test time. This evaluation is shown in the video attachment (`https://youtu.be/Z_GcNbCWE-E`). Key frames are also shown in Subsection A.3.2.

9.6.3 Relation to Soft Divergence

Since QG-DDPG resulted from observations about soft divergence that I empirically measured using the variance in predicted Q values, I also evaluated whether the differences in performance from the previous section correlate with variance in predicted Q-values. Figure 9.6 plots the predicted mean Q-value for each batch in training over time. Only for DDPG and unfavorable hyperparameters the Q-values diverge over time, while even bad trials of QG-DDPG do not lead to divergence.

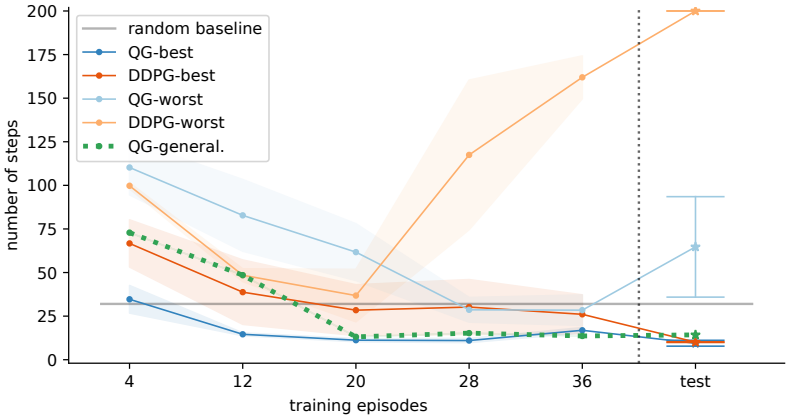


Figure 9.5: Best and worst case performance for vanilla DDPG and QG-DDPG (QG). The x-axis shows the number of episodes and at each tick, the performance of eight episodes has been averaged. The y-axis extends to 200, which is the worst possible test time performance in my setting. The green dotted line illustrates QG's performance on a more general and time-consuming task, where the orientation of the motor housing is changed for every fourth training episode and every second test episode.

9.6.4 Robustness to Sparse Rewards and Limited Memory

To successfully apply RL in practice, robustness is not only desirable w.r.t. hyperparameters but also regarding other design choices. Exemplarily, I here assess a drastic change in the reward function to sparse rewards, and a replay memory buffer that is limited to only 300 transitions.

Sparse rewards are a natural formulation for my setting because they reflect more precisely my evaluation criterion (the number of steps) and at the same time circumvent all issues related to reward shaping because the EEF pose of the robot only partially reflects the shaft pose. Limited memory availability is particularly interesting in industrial robotics as it creates a setting that is closer to the requirements of embedded AI.

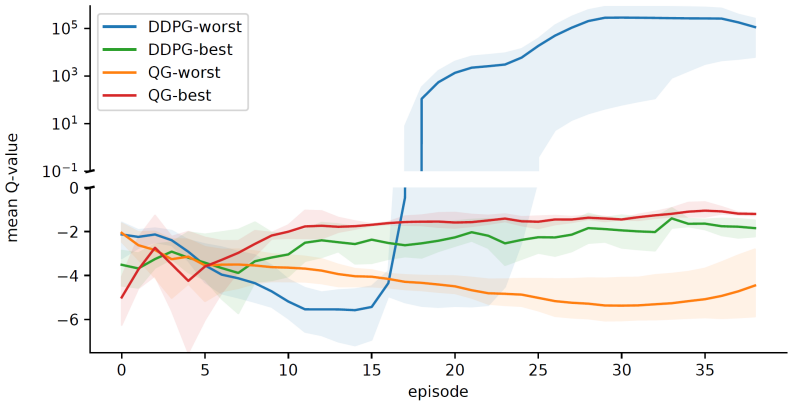


Figure 9.6: Evolution of mean Q-values over training episodes for the same hyperparameters as in Figure 9.5. Only DDPG diverges under bad hyperparameters while QG-DDPG is robust against those. The line represents the mean over all trials, the shaded area spans the full range between minimum and maximum.

Figure 9.7 summarizes the results for both sparse rewards and limited memory capacity under their respective best hyperparameter configurations. I can observe that QG-DDPG still performs better than random on average for both settings while DDPG does not. Additionally, QG-DDPG keeps the relatively low variance in performance, while the variance for DDPG increases significantly compared to its peak performance as in Figure 9.5.

9.7 Discussion

Targeting an industrial force fitting task, I have chosen an efficient formulation of a RL problem using residual policies. Applying classical DDPG in this setting however often leads to soft divergence. Partially, this can be explained with the particular structure of the data graph for residual policy formulations as discussed in Chapter 7. With more than 60 hours of real-world interaction, I have provided empirical evidence that QG-DDPG

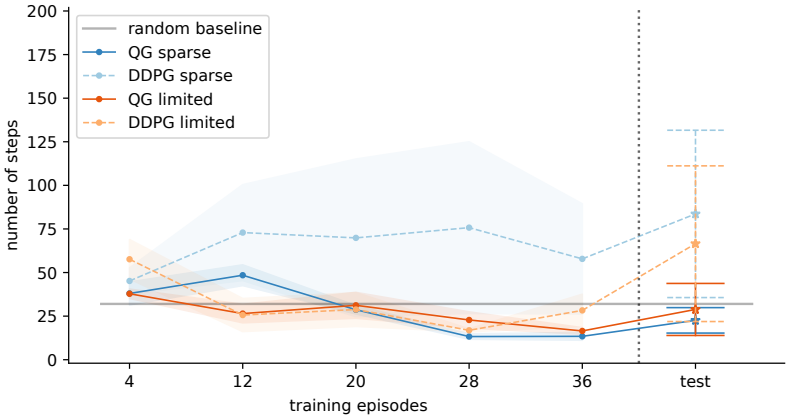


Figure 9.7: Robustness to changes in the learning setting: sparse rewards (blue) and limited memory capacity (orange) for vanilla DDPG (dashed) and QG-DDPG ('QG', solid lines). Both axes are scaled as in Figure 9.5 for comparison.

can indeed prevent this type of soft divergence: In trials in which DDPG happened to perform well, the predicted Q-values did not diverge. When DDPG performed worse, Q-values were consistently far beyond the range of true Q-values. In none of all trials with QG-DDPG however, such diverging behavior for Q-values has been observed. This indicates that not only the proposed method from QG-DDPG is effective in increasing stability of the learning process but also that the underlying reasoning and assumptions about causality transfer from toy examples as in Chapter 7 to real industrial tasks with substantial uncertainty.

Due to the reduced amount of soft divergence, the sample efficiency for QG-DDPG increases. On top, the proposed algorithm exhibits a series of further advantages: A grid search over hyperparameters has revealed that QG-DDPG reaches good performance six times more often than vanilla DDPG. Results for one given set of hyperparameters differ less for QG-DDPG, indicating improved reproducibility. QG-DDPG is also able to deal more gracefully with further changes in the learning setup, such as sparse rewards and limited replay memory. I showcase the gen-

eralization capabilities of my approach to varying ball bearing orientations, for which the algorithm autonomously discovers an appropriate insertion strategy.

9.8 Limitations and Open Questions

Although the results from Chapter 7 are interesting and have been confirmed for real-world applications in this chapter, our understanding of (soft) divergence in model-free off-policy deep RL remains limited. For instance, it remains unclear how other aspects of the learning formulation interact with divergence tendencies, for example hyperparameters, the choice of reward functions, action representation or state modalities in the specific MDP. The choice of state space is also linked to the question of whether QG-DDPG can benefit from state aggregation; and whether this could extend the approach to hierarchical learning.

The lack of theoretically grounded insights and instructions leads to increased tuning efforts for practitioners. To make sample efficient RL more easily accessible for industrial applications, several ideas from the previous chapters could be combined and evaluated in future work: First, better insertion strategies for the constant part of the residual policy formulation could be considered. For example, it may be possible to employ kinesthetic teaching or an intuitive user interfaces in which even untrained experts in production can sketch rudimentary strategies to replace the hard-coded push in z direction I have used in this chapter.

Second, an ideal autonomous robot that learns through RL can quickly be assigned new tasks and will help to increase flexibility in production. For this purpose however, it may need to learn how to solve tasks without custom hardware—in this case, without the 3D printed grippers. It would therefore be interesting and relevant to explore the limits of uncertainty from imprecise grasps or slippage that the system can learn to deal with.

Third, the shaft insertion for real-world industrial assembly processes may have to be integrated with other steps such as grasping the shaft or positioning the motor housing. On the one hand, such previous steps may increase the uncertainty due to imprecision which sums up over multiple steps. On the other hand, process information from previous manipulation steps could already provide valuable information. Combining information across tasks in an assembly process may be an additional way of improv-

ing sample efficiency once the individual subtasks have reached a good efficiency. Ultimately, such information from previously investigated tasks would be a very interesting type of information to integrate in model-free learning approaches.

Chapter 10

Conclusion

The overarching question for this doctoral thesis was how to improve sample-efficiency for model-free off-policy deep Reinforcement Learning (RL). This topic was driven by the emerging evidence that on the one hand, model-free RL promises to alleviate difficulties in obtaining accurate models for complex tasks such as contact-rich robotic manipulation; but on the other hand model-free RL is also known to be prohibitively data hungry. Improving sample-efficiency for model-free learning approaches therefore is one route towards unleashing the full potential of model-free learning also for domains with costly data such as real-world robotic manipulation. However, improving sample-efficiency typically comes at a cost—for example, it can be traded off against generalization capabilities.

In this thesis, I have proposed and used several types of information that can be infused into model-free learning approaches in various ways to achieve higher sample efficiency for use cases that were taken from or inspired by assembly processes in production automation. Table 10.1 lists these approaches from the previous chapters again and classifies the source of information as well as the method to infuse the information into the learning process. Note that there is no one-to-one mapping between information source and infusion approach, i.e., one source of information may be used in different places and vice versa.

The first source of information that model-free approaches can benefit from is the maybe most obvious one: environment dynamics or robot dynamics as used or approximated in model-based learning. In Chapter 6,

	source of information	infusion approach	
robot	coarse environment dynamics	exploration, e.g. via trajectory optimization (Chapter 6)	off-policy exploration
		action space using model-based control (Chapter 6, Chapter 9)	MDP design
	robot dynamics model	design action space to contain zero actions and complete the Q-Graph (Chapter 9)	
task	observability	choose modality for observation space (Chapter 5)	
	coarse solution strategy	residual formulation, e.g. movement in z direction (Chapter 5, Chapter 9)	
	success criteria	design reward function with a priori Q-learning bounds (Chapter 8)	
	additional information at train time, e.g. target pose	custom data collection (Chapter 5)	function approximation
available data	discontinuities	choose appropriate function approximator (Section 4.1)	
	relevance of information	network design, e.g. advantage stream (Section 4.2)	
	graph structure	Q-Graph-based lower bounds (Chapter 8)	

Table 10.1: Summary of pieces of information that I infused into model-free deep RL approaches to improve sample efficiency.

I have shown how a coarse steering function can be used to create more informed exploration data even in environments with unknown dynamics to speed up off-policy model-free RL. A precise model of the robot's dynamics can in addition be used when the MDP is designed, namely by defining actions not in joint space but in task space—exploiting model-based controllers for execution as in Chapter 6 and Chapter 9. Moreover, the action space can be designed such that the data the agent will collect has a favorable structure for learning. In Chapter 7 I have demonstrated how data can be represented by a graph whose structure can be linked to divergence in deep Q-learning-based RL approaches. For example, an action space that includes zero actions leads to more favorable graph structures for learning; and enables QG-DDPG to work with complete Q-Graphs—an approach I developed in Chapter 8 to prevent convergence issues in DDPG and thereby stabilize the learning process.

A second source of information is prior knowledge about the task at hand. For example, the observation space in an MDP can be designed such that the task becomes observable as in Chapter 5. Even for cases for which such information is not easily accessible, a coarse solution strategy may be intuitively available for humans: for instance, all successful peg insertion strategies involve a movement downwards in z direction. Such intuition can be captured by residual policy formulations as in Chapter 5 and Chapter 9, where the RL agent does not need to learn everything from scratch but only residuals which are added to a given base policy. Moreover, a human expert typically gets to design a reward function which is supposed to capture sensible criteria for successful task completion. This reward function can typically be chosen quite freely and therefore also such that it is bounded, e.g., if all rewards are negative then both empirical returns and the optimal Q-values are upper bounded by zero. Such a priori bounds for Q-values can be enforced in TD learning and also help to stabilize learning as shown in Chapter 7. If even more information is available at train time, e.g., the target pose, this can be used to design a custom data collection process as in Chapter 5.

The third and last source of information I identified is the data an agent collects during the learning process. Some information may be known in advance, e.g., from an expert view on the task. As I discussed in Section 4.1, contact-rich tasks are known to introduce discontinuities that different function approximators deal with more or less easily. The most intricate task is to tailor the function approximation directly to the task at hand.

I have demonstrated this in Chapter 6 by designing a network structure that structurally encodes the split of state-action values into state-value and advantage without losing their semantic meaning. This network architecture enabled me to build a custom exploration scheme which only relies on uncertainty from one of the two branches. Also the Q-Graph-based bounds that I introduced in Chapter 8 are an instance of such adaptations of the function approximation scheme.

All these instances constitute steps towards more efficient learning that utilizes flexible model-free RL methods, while sample efficiency is improved through targeted injection of single pieces of information that are available for certain tasks. The main novelty and contributions to the research community are the usage of a coarse steering function for trajectory optimization during exploration (Chapter 6), and all ideas related to the structure of a data graph and its application through Q-Graphs and zero actions (Chapter 7, Chapter 8). These approaches have been illustrated with toy examples, their effects were analyzed in detail on simulated control tasks first but the applicability of the proposed methods has finally been demonstrated on a real industrial assembly task with substantial uncertainties in Chapter 9.

Beyond the open questions that I discussed in the final sections of each of the previous chapters, some more high-level overarching questions arise or remain from the findings in this thesis.

First, instead of taking model-free approaches and infusing additional knowledge, one could also take the opposite approach and remove single pieces of information from model-based approaches. It would be an interesting task for future work to compare how far both paths lead and whether they converge to similar points on a spectrum between model-free and model-based methods.

Second, the list of information sources and infusion approaches in Table 10.1 is not exhaustive. Additional sources of information could be physical laws or affordances from sequencing, e.g., when multiple manipulation steps are supposed to be chained. Many items listed in the table could be extended and examined in more detail in the future. For example base policies for residual formulations can come from various sources and therefore differ greatly in maturity and optimality. Interesting directions for future research thus include simple user interfaces to define policies (whether this is by scribbling on a touch screen, using teleoperation or

physical teaching) as well as algorithms to derive policies from such input, as well as efficient methods to tune those policies by interacting with the environment as in RL.

Finally, I have demonstrated in this thesis how model-free approaches can be modified to achieve higher sample-efficiency and even become feasible for industry-scale assembly tasks. However, it remains unclear whether model-free approaches or RL in general are the best way to go for industrial assembly. On a more theoretical level it would be interesting to investigate whether there are theoretical limits in sample efficiency that specific RL methods can reach; and how these limits compare to those of alternative non-RL methods for automatic tuning of robotic manipulation strategies in contact-rich tasks.

Bibliography

Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Achiam, J., Knight, E., and Abbeel, P. (2019). Towards characterizing divergence in deep Q-learning. *arXiv preprint arXiv:1903.08894*.

Amiranashvili, A., Dosovitskiy, A., Koltun, V., and Brox, T. (2018). Analyzing the role of temporal differencing in deep reinforcement learning. In *International Conference on Learning Representations*.

Anderson, R. J. and Spong, M. W. (1988). Hybrid impedance control of robotic manipulators. *Journal on Robotics and Automation*, 4(5):549–556.

Anschel, O., Baram, N., and Shimkin, N. (2017). Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning*, pages 176–185.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier.

Baird, L. C. (1999). Reinforcement learning through gradient descent. *PhD thesis, Carnegie Mellon University*.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. (2016). Model-free episodic control. *arXiv preprint arXiv:1606.04460*.

Brafman, R. I. and Tennenholtz, M. (2002). R-max –a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231.

Bruyninckx, H., Dutre, S., and De Schutter, J. (1995). Peg-on-hole: a model based solution to peg and hole alignment. In *International Conference on Robotics and Automation*, volume 2, pages 1919–1924. IEEE.

Butler S., t. (1898). *The Iliad of Homer*. Longmans, Green, & Co.

Carpentier, J., Valenza, F., Mansard, N., et al. (2015–2019). Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio.

Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. (2017). Path integral guided policy search. In *international conference on robotics and automation*, pages 3381–3388. IEEE.

Chen, R. Y., Sidor, S., Abbeel, P., and Schulman, J. (2017). UCB and InfoGain exploration via Q-ensembles. *Computing Research Repository*, abs/1706.01502.

Chentanez, N., Barto, A. G., and Singh, S. P. (2005). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1281–1288.

Collins, J., Howard, D., and Leitner, J. (2019). Quantifying the reality gap in robotic manipulation tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6706–6712.

Corneil, D., Gerstner, W., and Brea, J. (2018). Efficient model-based deep reinforcement learning with variational state tabulation. In *International Conference on Machine Learning*, pages 1049–1058.

De Asis, K., Chan, A., Pitis, S., Sutton, R. S., and Graves, D. (2019). Fixed-horizon temporal difference methods for stable reinforcement learning. *arXiv preprint arXiv:1909.03906*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*. IEEE.

Durugkar, I. and Stone, P. (2018). TD learning with constrained gradients. https://openreview.net/forum?id=Bk-ofQZRb.

Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. (2013). Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174.

Englert, P. and Toussaint, M. (2018). Learning manipulation skills from a single demonstration. *The International Journal of Robotics Research*, 37(1):137–154.

Fazeli, N., Zapolsky, S., Drumwright, E., and Rodriguez, A. (2017). Learning data-efficient rigid-body contact models: Case study of planar impact. In *Conference on Robot Learning*, pages 388–397.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. (2020). Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*.

Fei, Y. and Zhao, X. (2003). An assembly process modeling and analysis for robotic multiple peg-in-hole. *Journal of Intelligent and Robotic Systems*, 36(2):175–189.

Fu, J., Kumar, A., Soh, M., and Levine, S. (2019). Diagnosing bottlenecks in deep Q-learning algorithms. In *International Conference on Machine Learning*, pages 2021–2030.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *Proceedings of Machine Learning Research*, 80:1587–1596.

Gal, Y. and Ghahramani, Z. (2016a). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Gal, Y. and Ghahramani, Z. (2016b). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*.

Giftthaler, M., Neunert, M., Stäuble, M., and Buchli, J. (2018). The control toolbox — an open-source c++ library for robotics, optimal and model predictive control. *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 123–129.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2001). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530.

Grześ, M. (2017). Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 565–573.

Gu, S. S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3846–3855.

He, F. S., Liu, Y., Schwing, A. G., and Peng, J. (2017). Learning to play in a day: Faster deep reinforcement learning by optimality tightening. In *International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *international conference on computer vision*, pages 1026–1034. IEEE.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hernandez-Garcia, J. F. and Sutton, R. S. (2019). Understanding multi-step deep reinforcement learning: A systematic study of the dqn target. *arXiv preprint arXiv:1901.07510*.

Hoppe, S., Giftthaler, M., Krug, R., and Toussaint, M. (2020). Sample-efficient learning for industrial assembly using Q-graph-bounded DDPG. In *International Conference on Intelligent Robots and Systems*, pages 9080–9087.

Hoppe, S., Giftthaler, M., Krug, R., and Toussaint, M. (2021). Stabilizing deep Q-learning with Q-Graph-based bounds. Under review for the International Journal for Robotics Research (IJRR).

Hoppe, S., Lou, Z., Hennes, D., and Toussaint, M. (2017). Deep learning for manipulation with visual and haptic feedback. IROS Workshop on Frontiers in Contact-Rich Robotic Interaction.

Hoppe, S., Lou, Z., Hennes, D., and Toussaint, M. (2019). Planning approximate exploration trajectories for model-free reinforcement learning in contact-rich manipulation. *IEEE Robotics and Automation Letters*, 4(4):4042–4047.

Hoppe, S. and Toussaint, M. (2020). Q-graph-bounded Q-learning: Stabilizing model-free off-policy deep reinforcement learning. *arXiv preprint arXiv:2007.0758*.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117.

Inoue, T., De Magistris, G., Munawar, A., Yokoya, T., and Tachibana, R. (2017). Deep reinforcement learning for high precision assembly tasks. In *International Conference on Intelligent Robots and Systems*, pages 819–825. IEEE.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710.

Jensen, A. R. (1989). The relationship between learning and intelligence. *Learning and individual differences*, 1(1):37–62.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Residual reinforcement learning for robot control. In *International Conference on Robotics and Automation*, pages 6023–6029. IEEE.

Johannsmeier, L., Gerchow, M., and Haddadin, S. (2019). A framework for robot manipulation: Skill formalism, meta learning and adaptive control. In *International Conference on Robotics and Automation*, pages 5844–5850. IEEE.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Kilikevičius, S. and Bakšys, B. (2011). Experimental investigation of vibratory peg-in-hole insertion for robotic assembly. In Náprstek, J., Horáček, J., Okrouhlík, M., Marvalová, B., Verhulst, F., and Sawicki, J. T., editors, *International Conference on Vibration Problems*, pages 621–627, Dordrecht. Springer Netherlands.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kloss, A., Schaal, S., and Bohg, J. (2017). Combining learned and analytical models for predicting action effects. *arXiv preprint arXiv:1710.04102*.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy Q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794.

Kumar, A., Gupta, A., and Levine, S. (2020). Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*.

Kussul, E., Baidyk, T., Kasatkina, L., and Lukovich, V. (2001). Rosenblatt perceptrons for handwritten digit recognition. In *International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 2, pages 1516–1520. IEEE.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y. et al. (1989). Generalization and network design strategies. In *Connectionism in perspective*, volume 19. Citeseer.

Lee, K. and Kim, K.-E. (2015). Tighter value function bounds for bayesian reinforcement learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Lee, M. A., Zhu, Y., Srinivasan, K., Shah, P., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2019a). Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *International Conference on Robotics and Automation*, pages 8943–8950. IEEE.

Lee, S. Y., Sungik, C., and Chung, S.-Y. (2019b). Sample-efficient deep reinforcement learning via episodic backward update. In *Advances in Neural Information Processing Systems*, pages 2112–2121.

Lefebvre, T., Xiao, J., Bruyninckx, H., and De Gersem, G. (2005). Active compliant motion: a survey. *Advanced Robotics*, 19(5):479–499.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016a). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016b). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Computing Research Repository*, abs/1603.02199.

Levine, S., Wagener, N., and Abbeel, P. (2015). Learning contact-rich manipulation skills with guided policy search. *Computing Research Repository*, abs/1501.05611.

Li, L., Littman, M. L., and Walsh, T. J. (2008). Knows what it knows: a framework for self-aware learning. In *International Conference on Machine Learning*, pages 568–575. ACM.

Li, Y. (1997). Hybrid control approach to the peg-in hole problem. *Robotics & Automation Magazine*, 4(2):52–60.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, pages 206–214.

Lowrey, K., Rajeswaran, A., Kakade, S. M., Todorov, E., and Mordatch, I. (2018). Plan online, learn offline: Efficient learning and exploration via model-based control. *Computing Research Repository*, abs/1811.01848.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*. Citeseer.

Martín-Martín, R., Lee, M. A., Gardner, R., Savarese, S., Bohg, J., and Garg, A. (2019). Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *International Conference on Intelligent Robots and Systems*, pages 1010–1017. IEEE.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Moravec, H. (1988). *Mind children: The future of robot and human intelligence*. Harvard University Press.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062.

Müller, C. and Kutzbach, N. (2019). World robotics 2019 – industrial robots. Technical report, International Federation of Robotics.

Ohnishi, S., Uchibe, E., Nakanishi, K., and Ishii, S. (2019). Constrained deep Q-learning gradually approaching ordinary Q-learning. *Frontiers in neurorobotics*, 13:103.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *Conference on computer vision and pattern recognition*, pages 1717–1724. IEEE.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Powell, M. J. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162.

Precup, D., Sutton, R. S., and Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rueb, A. and Becker, L. (US Patent US10480923B2, November 2016). Sensor apparatus and robot system having the sensor apparatus.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493.

Sayler, S. (2011). *Universelle Manipulationsstrategien für die industrielle Montage*. KIT Scientific Publishing.

Schaal, S. (1996). Learning from demonstration. In *Advances in Neural Information Processing Systems*.

Schoettler, G., Nair, A., Luo, J., Bahl, S., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. *arXiv preprint arXiv:1906.05841*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, page 103535.

Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. (2018). Residual policy learning. *arXiv preprint arXiv:1812.06298*.

Sousa, C. D. and Cortesão, R. (2019). Inertia tensor properties in robot dynamics identification: A linear matrix inequality approach. *Transactions on Mechatronics*, 24(1):406–411.

Spath, D., Müller, R., and Reinhart, G. (2013). *Zukunftsfähige Montagesysteme Wirtschaftlich, wandlungsfähig und rekonfigurierbar*. Fraunhofer Verlag, Stuttgart.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings*, pages 216–224. Elsevier.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Conference on computer vision and pattern recognition*, pages 1–9. IEEE.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017). # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2750–2759.

Tang, Y. (2020). Self-imitation learning via generalized lower bound Q-learning. *arXiv preprint arXiv:2006.07442*.

Thomas, G., Chien, M., Tamar, A., Ojea, J. A., and Abbeel, P. (2018). Learning robotic assembly from CAD. In *International Conference on Robotics and Automation*, pages 1–9. IEEE.

Touati, A., Zhang, A., Pineau, J., and Vincent, P. (2020). Stable policy optimization via off-policy divergence regularization. *arXiv preprint arXiv:2003.04108*.

van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*.

Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *AAAI*.

Van Hoof, H., Chen, N., Karl, M., van der Smagt, P., and Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ international conference on intelligent robots and systems*, pages 3928–3934. IEEE.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. A. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *Computing Research Repository*, abs/1707.08817.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. *PhD thesis*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR.

Wirnshofer, F., Schmitt, P. S., Feiten, W., Wichert, G. v., and Burgard, W. (2018). Robust, compliant assembly via optimal belief space planning. In *International Conference on Robotics and Automation*, pages 1–5. IEEE.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Trans. Evolutionary Computation*, 1:67–82.

Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. (2019). A mean field theory of batch normalization. In *International Conference on Learning Representations*.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *Computational Intelligence Magazine*, 13(3):55–75.

Yu, W., Tan, J., Liu, C. K., and Turk, G. (2017). Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*.

Zhang, Z., Chen, J., Chen, Z., and Li, W. (2019). Asynchronous episodic deep deterministic policy gradient: Toward continuous control in computationally complex environments. *Transactions on Cybernetics*.

Zhu, G., Lin, Z., Yang, G., and Zhang, C. (2019). Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations*.

Appendices

Appendix A

Visual Experiment Documentation

A.1 Visual Evaluation: Results from Chapter 5

This section contains key frames from a video that was taken during the experiments in Chapter 5 and therefore allows to qualitatively inspect positive and negative examples for each controller.

A.1.1 Controller A: Vision & Passive Compliance

Positive example:

The illustration of the other two controllers in the upcoming sections will be using failure cases of controller A. Therefore, I omit the illustration of failure here.

A.1.2 Controller B: Closed (Hard-Coded) Control Loop

Positive example (in a trial which would have been a failure with controller A):

Failure case:

A.1.3 Controller C: Vision & Compliance Feedback

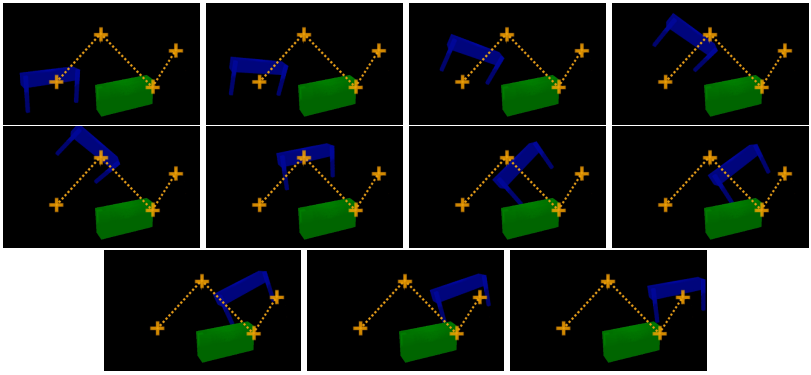
positive example:

From one hundred tests with randomly sampled initial positions, there were no failures in this condition.

A.2 Visual Evaluation: Results from Chapter 6

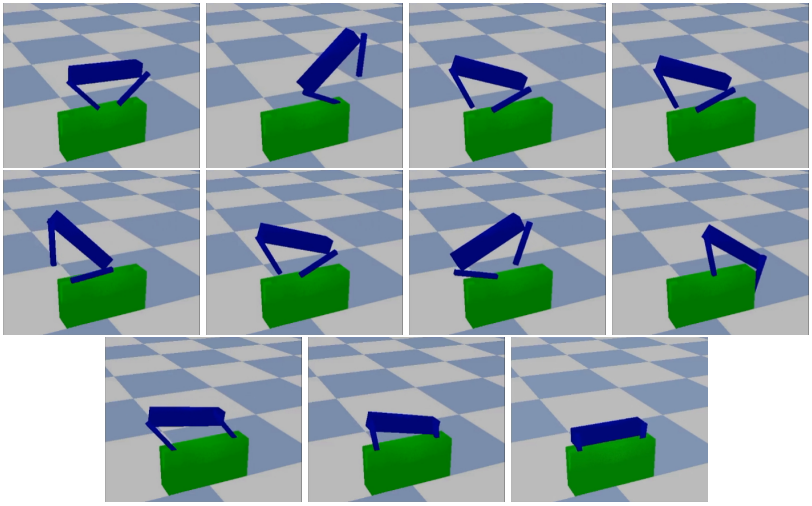
In this section, the main content from the video that illustrates learning progress in the empirical evaluation in Chapter 6 is summarized. The full video, which was published as supplementary material for Hoppe et al. (2019) is available online at `https://youtu.be/JTfeHhWSb0Y`.

A.2.1 Illustration of Steering Function



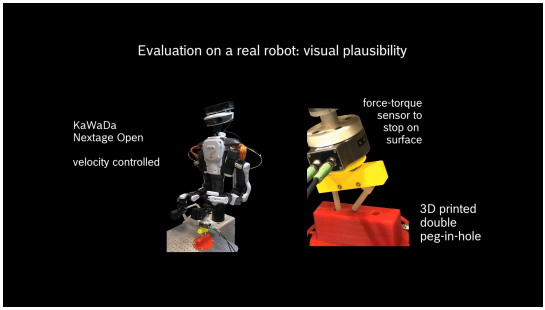
A.2.2 Task Illustration in Simulation

The steering function is only a coarse approximation of the dynamics and will fail whenever obstacles or contacts come into play. To move a long an optimal trajectory, the robot steers towards each waypoint but if it cannot get closer anymore, the waypoint is skipped.

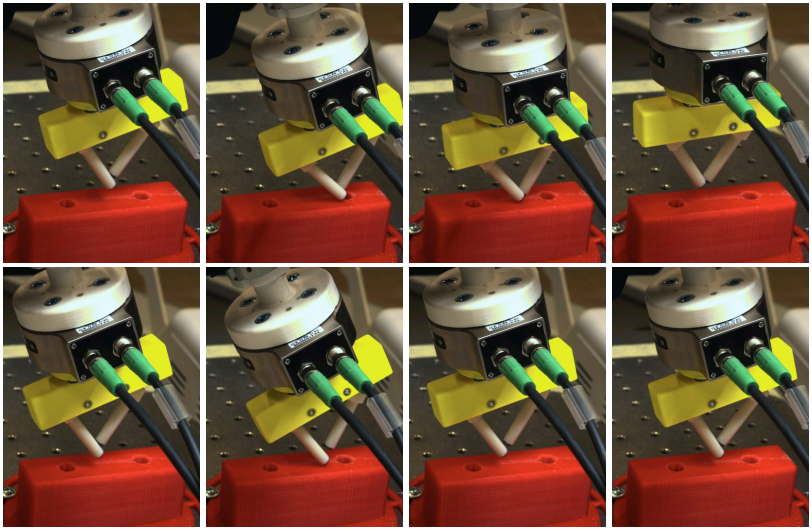


A.2.3 Learning Progress on the Real Robot

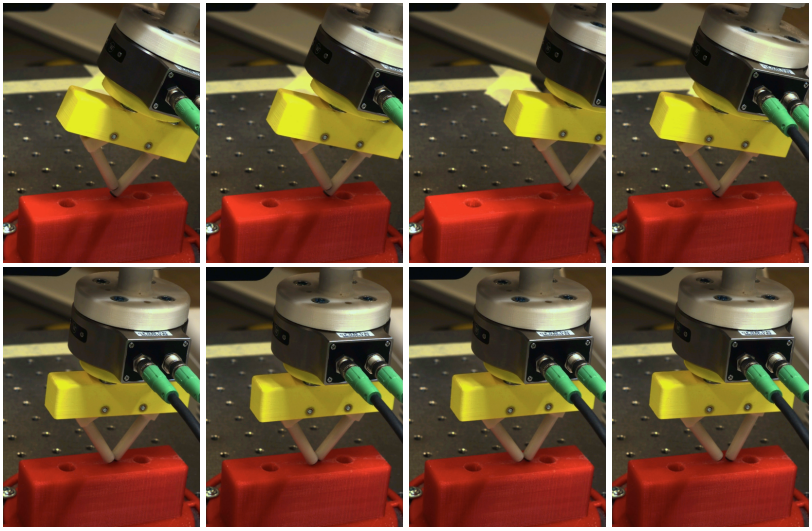
Setup:



First episode:
random movement based on actor network initialization and Gaussian noise:

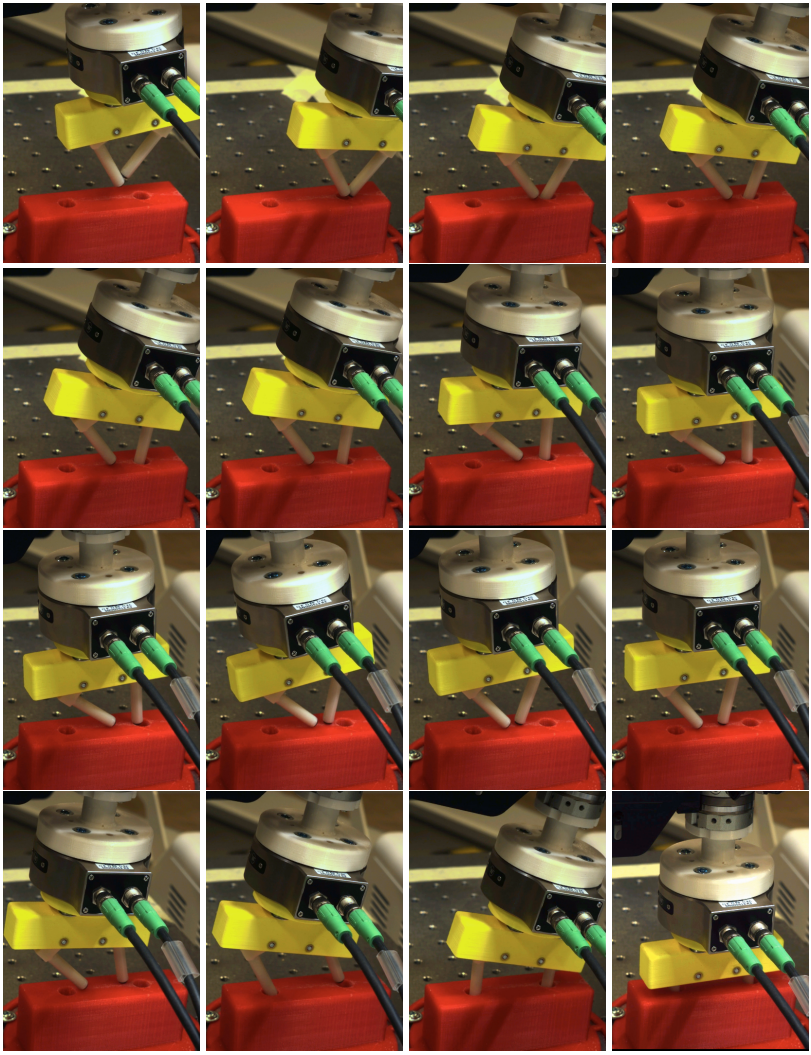


First episode with trajectory optimization:
small linear movements:



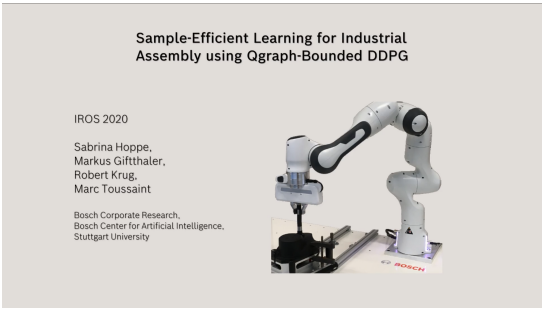
Focus on relevant areas:
local minimum in distance-based reward function (center between the two holes) and the holes themselves (in analogy to previous episodes which lead to high reward):

First solution found after less than 1200 steps:

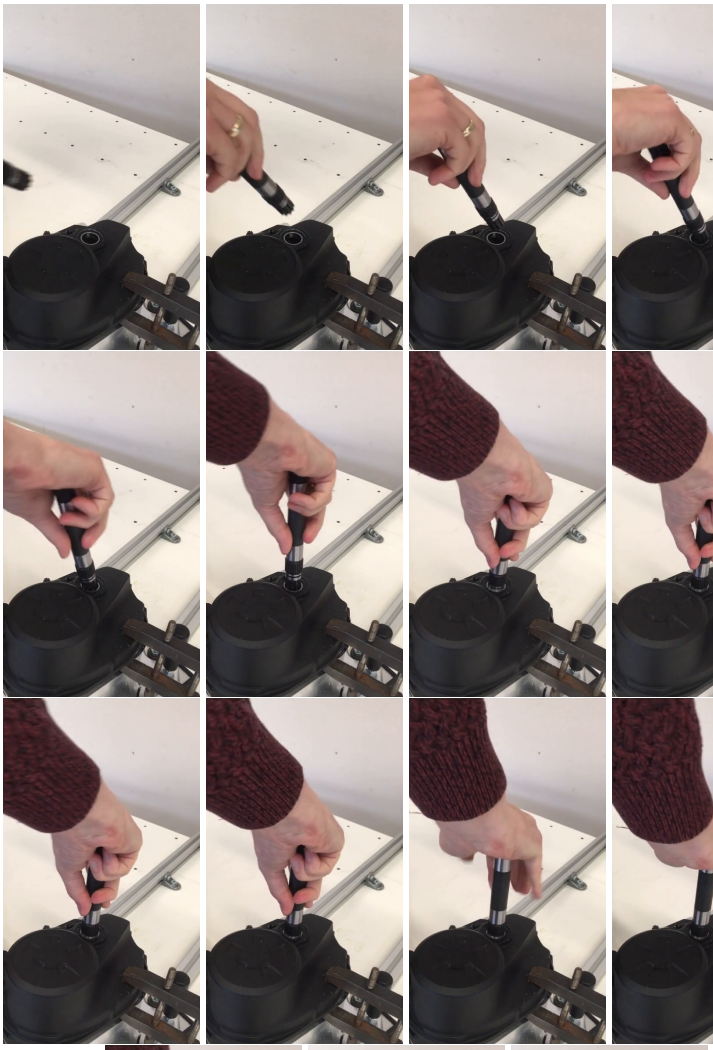


A.3 Visual Evaluation: Results from Chapter 9

In this section, the main content from the video that illustrates learning progress in the empirical evaluation in Chapter 9 is summarized. The full video, which was published as supplementary material for Hoppe et al. (2020) is available online at `https://youtu.be/Z_GcNbCWE-E`.

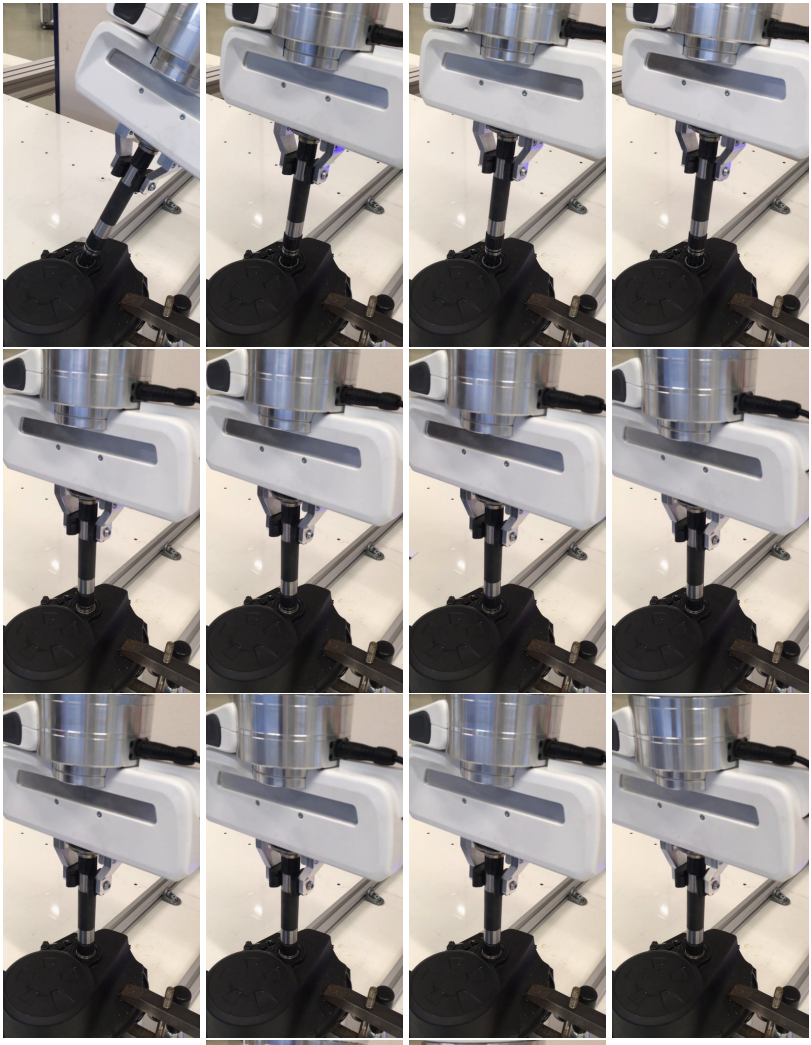


A.3.1 Human Baseline

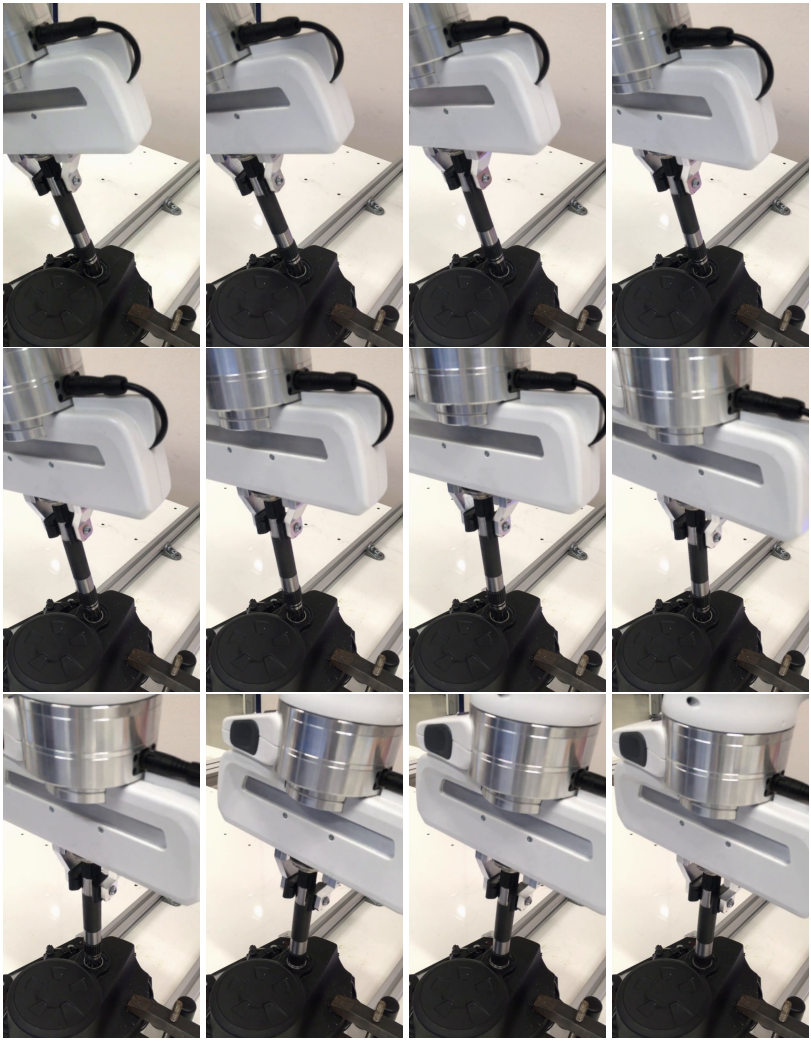


A.3.2 Learning Progress

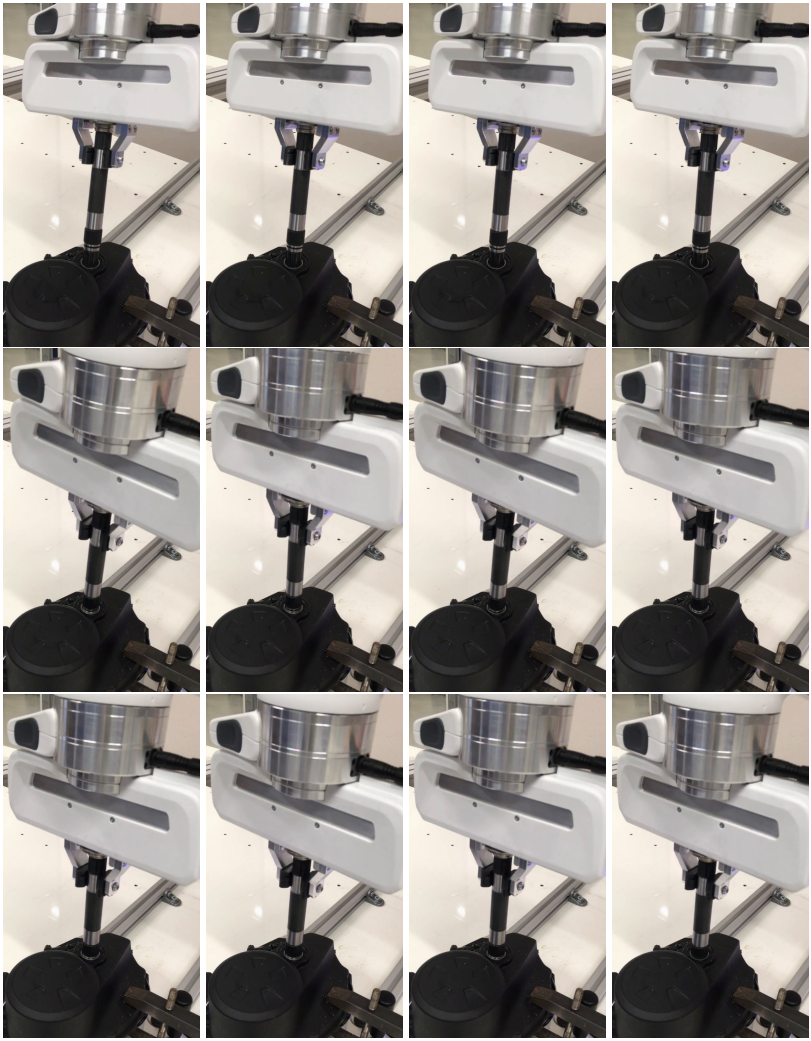
Episode 0:
(random movement, every 20th video frame)



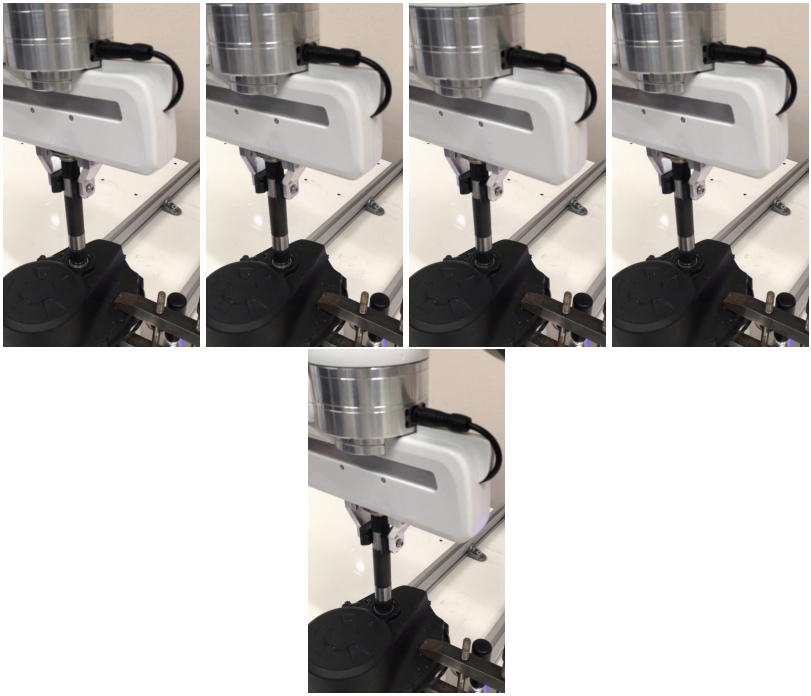
Episode 9 (every 3rd video frame):



Episode 18 (every 3rd video frame):

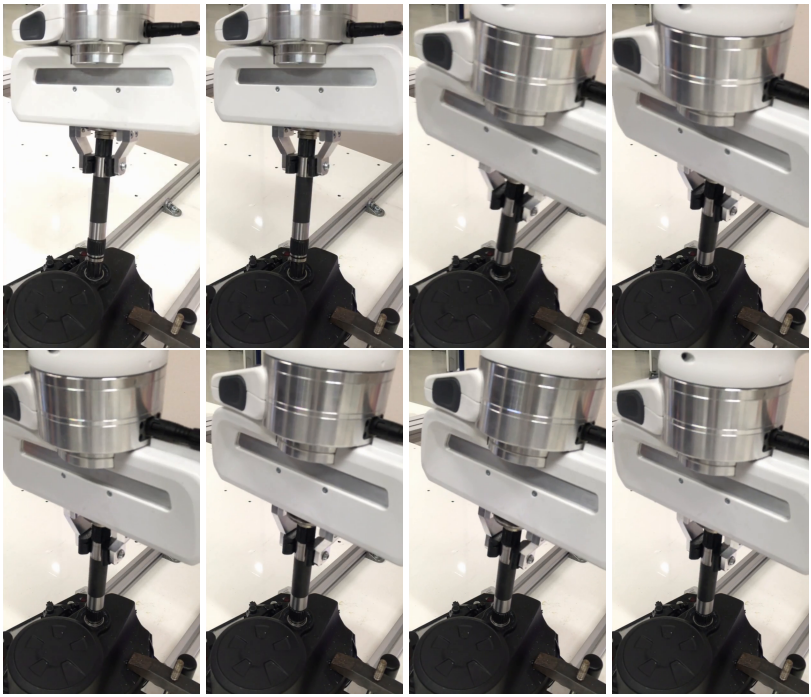


Episode 27 (every 3rd video frame):



Episode 32 (all available video frames):

Test Episodes:
without exploration noise. Every 3rd video frame.



Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation selbstständig und ausschließlich auf der Grundlage der in der Arbeit angegebenen Hilfsmittel und Hilfen verfasst habe.

Sabrina Hoppe
Stuttgart, den 19.04.2021