Understand-Compute-Adapt: Neural Networks for Intelligent Agents

Verstehen-Berechnen-Anpassen:
Neuronale Netzwerke für intelligente Agenten
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
vorgelegte Dissertation von Daniel Tanneberg aus Offenbach am Main
Tag der Einreichung: 22.10.2020, Tag der Prüfung: 03.12.2020

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Elmar Rückert
3. Gutachten: Prof. Dr. Martin Riedmiller
Darmstadt – D 17

TECHNISCHE UNIVERSITÄT DARMSTADT

Understand-Compute-Adapt:
Neural Networks for Intelligent Agents
Verstehen-Berechnen-Anpassen:
Neuronale Netzwerke für intelligente Agenten

Doctoral thesis by Daniel Tanneberg

1. Review: Prof. Dr. Jan Peters
2. Review: Prof. Dr. Elmar Rückert
3. Review: Prof. Dr. Martin Riedmiller

Date of submission: 22.10.2020
Date of thesis defense: 03.12.2020

Darmstadt – D 17

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-XXXX
URL: http://tuprints.ulb.tu-darmstadt.de/XXXX

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

For mankind.

Just kidding – For myself.

Erklärungen laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 22.10.2020

D. Tanneberg

Abstract

An artificial intelligent agent needs to be equipped with a multitude of abilities in order to interact in the world among us. These requirements for intelligent behaviour can roughly be separated into two main categories, *cognitive* abilities and *physical* skills. The cognitive abilities refer to cognition and problem solving, whereas the physical skills correspond to movements of an intelligent robot in the real world. In this thesis, we investigate three research questions tackling those different abilities. Precisely, how can new knowledge be taught to a robot in a natural way? How can neural networks learn abstract solution strategies that are independent of the task complexity, data representation and task domain? How can a robot efficiently adapt its movement during execution with a bio-inspired stochastic neural network? These questions span core requirements for intelligent autonomous agents, which we categorize as *Understand-Compute-Adapt* (UCA), in the style of the classical *Sense-Plan-Act* framework in robotics. To answer these questions, we investigate neural network based models on these cognitive and physical abilities.

In detail, the first question tackles the ability of cognition, which refers to an *understanding* of the world and is investigated by learning a set of skills from unlabelled demonstrations of full task executions. Therefore, we studied the task of trajectory segmentation and skill library learning. To provide a natural interface for teaching a robot new tasks, it is desirable to have the user only demonstrating the desired task, without worrying about all the skills that are required for the task and without manually annotating the demonstrations. Such an interface not only enables non-experts to teach robots, but also provides a cheaper approach to teaching robots, as demonstrating all individual skills or segmenting and labelling demonstrations by hand is time consuming and expensive. The approach proposed here learns to segment trajectories and the required skill library simultaneously from unlabelled demonstrations. In addition to this segmenting and skill discovery, the approach also learns the relations between individual skills, i.e., modelling how likely a certain skill follows after another skill. This additional knowledge, or understanding, can be used, for example, in human-robot-interaction scenarios by predicting the human behaviour and therefore enables a more intelligent adaptive behaviour of the robot. The

approach was successfully evaluated on multiple different trajectory datasets with varying complexities.

The second aforementioned required cognitive ability, problem solving, refers to the second question and the *Compute* step. In particular, we investigated the challenge of learning *algorithmic* solutions, i.e., learning abstract strategies that can easily be transferred to unfamiliar problem instantiations. This transferring of abstract knowledge and solution strategies into novel domains is another crucial feature of intelligent behaviour. Therefore, we investigated the learning of algorithmic solutions that are characterized by three requirements highlighting the abstract nature of the solution: scaling to arbitrary task configurations and complexities, and the independence of both the data representations as well as the task domain. For this purpose we developed a novel framework, the Neural Harvard Computer, that is based on memory-augmented neural networks and whose modular design is inspired by the von Neumann and Harvard architectures of modern computers. This framework enables the learning of abstract algorithmic solutions through its modular design and the separation of information flow into data and control signals. The algorithmic solution is learned in a reinforcement learning setting and solely operating on the control signal flow, enabling the independence of the data representation and task domain. We evaluated the framework's generalization and abstraction features by learning 11 different algorithms, where the approach was able to reliably learn algorithmic solutions with perfect generalization and abstraction, allowing to solve problems with complexities far beyond seen during training and by straight forward transfer to novel task representations and domains.

Ultimately an intelligent robot has to interact in the real world, giving rise to the third entry *Adapt*, the question of efficient online adaptation. In order to cope with the complex, dynamic and often unstructured real world, in addition to dealing with other agents and humans, the agent has to be able to adapt its models and movements while interacting. This online adaptation belongs to the mentioned physical skills that are required for intelligent behaviour. Moreover, this online adaptation has to be efficient in terms of number of physical interactions and be task-independent, as not every situation can be foreseen when constructing the agent or the method. In this thesis, we studied online adaptation within a bio-inspired spiking neural network that generates movements by simulating its inherent dynamics. The underlying stochastically spiking neurons mimic the behaviour of hippocampal place cells and their decoded activity represents the planned movement. Task-independent adaptation is achieved by using intrinsic motivation signals inspired by *cognitive dissonance* to guide the learning. These signals capture the discrepancy between the agents expectation of the world (the current model) and the observations of the world, and the online adaptation is triggered and steered through this mismatch. Sample-

efficiency is accomplished by using a mental replay strategy to intensify experienced situations and is implemented by using the inherent stochasticity of the framework. We evaluated this framework for online model adaptation and movement generation on an anthropomorphic KUKA LWR arm, where the robot has to adapt to unknown obstacles while performing a waypoint following task. The online adaptation happens within seconds and from few physical interactions while keeping interacting with the environment.

In summary, this thesis investigates three key aspects of intelligent behaviour with respect to cognitive and physical abilities. In more detail, we investigated how neural network based models can be used from *learning to understand* over *learning to compute* to *learning to adapt* to tackle the three raised research question. Each topic has its own requirements on the used neural network model and the learning mechanism. This modularity and diversity of subroutines is a crucial aspect for creating artificial intelligence.

Zusammenfassung

Ein künstlicher intelligenter Agent muss mit einer Vielzahl von Fertigkeiten ausgestattet sein, um unter uns in der Welt zu interagieren. Diese Voraussetzungen für intelligentes Verhalten können grob in zwei Hauptkategorien unterteilt werden, *kognitive* Fähigkeiten und *physische* Fertigkeiten. Die kognitiven Fähigkeiten beziehen sich auf das Verstehen und Problemlösen, wohingegen die physischen Fertigkeiten sich auf Bewegungen eines intelligenten Roboters in der echten Welt beziehen. In dieser Thesis untersuchen wir drei Forschungsfragen, die sich mit diesen verschiedenen Fertigkeiten beschäftigen. Konkret, wie kann einem Roboter neues Wissen auf eine natürliche Art beigebracht werden? Wie können Neuronale Netzwerke abstrakte Lösungsstrategien lernen, die unabhängig von der Komplexität der Aufgabe, der Datenrepräsentation und dem Aufgabengebiet sind? Wie kann ein Roboter seine Bewegungen effizient während ihrer Ausführung mit einem biologisch-inspiriertem stochastischem Neuronalen Netzwerk anpassen? Diese Fragen umfassen Kernanforderungen an intelligente autonome Agenten, welche wir als *Verstehen-Berechnen-Anpassen* kategorisieren, in Anlehnung an das klassische *Sense-Plan-Act* Modell in der Robotik. Um diese Fragen zu beantworten, untersuchen wir die Fähigkeiten von Neuronalen Netzwerken basierten Modellen im Bezug auf diese kognitiven und physischen Fertigkeiten.

Im Detail, die erste Frage beschäftigt sich mit der Fähigkeit des Erkennens, welche sich auf ein *Verstehen* der Welt bezieht und welche durch das Lernen eines Sets von Fertigkeiten aus unmarkierten Demonstrationen von vollständigen Aufgabenausführungen untersucht wird. Hierfür haben wir die Aufgabe der Segmentierung von Trajektorien und das Lernen einer Fertigkeiten Bibliothek studiert. Für eine natürliche Schnittstelle um einem Roboter neue Aufgaben beizubringen, ist es wünschenswert dass der Benutzer nur die vollständige Aufgabe demonstrieren muss, ohne sich über alle für die Aufgabe benötigten Fertigkeiten Gedanken machen zu müssen und ohne händisch die Demonstrationen markieren zu müssen. Solch eine Schnittstelle erlaubt es nicht nur Nicht-Experten das unterrichten von Robotern, sondern stellt auch eine günstigere Möglichkeit des Unterrichtens von Robotern da, denn das Demonstrieren von allen einzelnen Fertigkeiten oder das händische

Segmentieren und Markieren von Demonstrationen ist zeitaufwendig und teuer. Die hier vorgestellte Methode lernt gleichzeitig Trajektorien zu segmentieren und die benötigte Fertigkeiten Bibliothek von unmarkierten Demonstrationen. Zusätzlich zu dieser Segmentierung und Entdeckung von Fertigkeiten, lernt die Methode auch das Zusammenspiel zwischen einzelnen Fertigkeiten, d.h. sie modelliert wie wahrscheinlich eine bestimmte Fertigkeit auf eine andere folgt. Dieses zusätzliche Wissen, oder Verständnis, kann zum Beispiel dafür genutzt werden, das menschliche Verhalten vorherzusagen, um ein intelligenteres adaptives Verhalten des Roboters in Mensch-Roboter Szenarien zu ermöglichen. Die Methode wurde erfolgreich mit mehreren verschiedenen Trajektorien Datensets mit unterschiedlicher Komplexität evaluiert.

Die zweite vorher erwähnte benötigte kognitive Fähigkeit, Problemlösen, bezieht sich auf die zweite Fragen und damit den *Berechnen* Schritt. Im Detail haben wir die Herausforderung des Lernens von algorithmischen Lösungen untersucht, d.h., das Lernen von abstrakten Strategien, die einfach auf unbekannte Probleminstanzen übertragen werden können. Dieses Übertragen von abstraktem Wissen und Lösungsstrategien auf neue Aufgabengebiete ist eine weitere wichtige Eigenschaft von intelligentem Verhalten. Hierzu untersuchten wir das Lernen von algorithmischen Lösungen, welche durch drei Anforderungen charakterisiert sind, die die abstrakte Natur der Lösung hervorheben: das Skalieren auf beliebige Konfigurationen und Komplexitäten von Aufgaben, und die Unabhängigkeit von sowohl der Datenrepräsentation als auch des Aufgabengebiets. Zu diesem Zweck haben wir ein neues Modell entwickelt, den Neural Harvard Computer, das auf Neuronalen Netzwerken mit externen Speichern basiert und dessen modularer Aufbau von der von Neumann und Harvard Architektur von modernen Computern inspiriert ist. Dieses Modell ermöglicht das Lernen von abstrakten algorithmischen Lösungen durch seinen modularen Aufbau und die Trennung des Informationsflusses in Daten und Kontrollsignale. Die algorithmischen Lösungen werden in einem verstärkendem Lernen Szenario gelernt und operieren ausschließlich auf den Kontrollsignalen, was die Unabhängigkeit von der Datenrepräsentation und des Aufgabengebiets ermöglicht. Die Generalisierungs- und Abstraktionsfähigkeiten des Modells wurde durch das Lernen von 11 verschiedenen Algorithmen evaluiert, bei denen das Modell verlässlich algorithmische Lösungen mit perfekter Generalisierung und Abstraktion gelernt hat. Dies ermöglicht Probleme mit einer deutlich höheren Komplexität zu Lösen als während des Lernens und das Übertragen auf neue Repräsentationen und Aufgabengebiete.

Letztendlich muss ein intelligenter Roboter in der echten Welt interagieren, was sich auf den dritten Schritt *Anpassen* bezieht, die Frage nach effizienter Onlineanpassung. Um mit der komplexen, dynamischen und oft unstrukturierten echten Welt zurecht zu kommen, zusätzlich zum Beschäftigen mit anderen Agenten und Menschen, muss der

Agent die Fähigkeiten haben seine Modelle und Bewegungen während der Interaktion anzupassen. Diese Onlineanpassung gehört zu den erwähnten physischen Fertigkeiten, die für intelligentes Verhalten nötig sind. Zusätzlich muss diese Onlineanpassung effizient im Bezug auf die Anzahl der physischen Interaktionen und Aufgabenunabhängig sein, da nicht jede Situation vorhergesehen werden kann wenn der Agent oder das Modell konstruiert werden. In dieser Thesis studieren wir die Onlineanpassung mit einem biologisch inspiriertem feuerndem Neuronalem Netzwerk, welches Bewegungen generiert indem es seine inhärente Dynamik simuliert. Die zugrundeliegenden stochastisch feuernden Neuronen imitieren das Verhalten von Ortszellen im Hippocampus und ihre dekodierte Aktivität repräsentiert die geplante Bewegung. Aufgabenunabhängige Anpassung wird durch intrinsische Motivationssignale erreicht, welche durch *kognitive Dissonanz* inspiriert sind und die das Lernen steuern. Diese Signale messen den Unterschied zwischen der Erwartung des Agenten von der Welt (das aktuelle Modell) und die Beobachtung der tatsächlichen Welt, und die Onlineanpassung wird durch diese Ungleichheit ausgelöst und gesteuert. Sample-Effizienz wird durch eine mentale Wiederholungsstrategie erreicht, die widerfahrende Situationen verstärkt, und welche durch die inhärente Stochastizität des Modells implementiert ist. Wir haben das Modell zur Onlineanpassung und Bewegungsgenerierung mit einem anthropomorphen KUKA LWR Roboterarm evaluiert, wobei der Roboter sich an unbekannte Hindernisse anpassen musste während er die Aufgabe hatte Wegpunkten zu folgen. Die Onlineanpassung erfolgte innerhalb von Sekunden und von wenigen physischen Interaktionen, während der durchgehenden Interaktion mit der Umgebung.

Zusammengefasst untersucht diese Thesis drei Schlüsselaspekte von intelligentem Verhalten im Bezug auf kognitive und physische Fähigkeiten. Im Detail untersuchten wir wie auf Neuronalen Netzwerken basierte Modelle benutzt werden können, um die vom *Lernen zu Verstehen*, über *Lernen zu Berechnen*, bis hin zu *Lernen Anzupassen* gestellten Forschungsfragen zu untersuchen. Dabei hat jedes Thema seine eigenen Anforderungen an das Neuronale Netzwerk und die benutzten Lernmethoden. Diese Modularität und Diversität von Teilroutinen ist ein entscheidender Aspekt, um künstliche Intelligenz zu erschaffen.

Acknowledgements

The work leading to this thesis has only been possible due to the support of several people. First of all, I'd like to thank my two supervisors: thank you Jan for your support and for giving me the opportunity to do the research that lead to this thesis. Likewise, thank you Elmar for your additional guidance and support, the refreshing conversations, and for not killing some of my crazy ideas too early.

Next, a big thanks to all the colleagues and staff at the Intelligent Autonomous Systems group, where everybody creates a supportive and creative atmosphere, with inspiring discussions and just a great time, inside and outside of the office. I am happy and grateful to have been a member of this family over the years. Thank you Veronika, Marion, Nanette and Sabine for your help and care about bureaucratic and technical issues. A special thanks also to my office mates over the years, who contributed to a fantastic work atmosphere with great conversations (on and off topic) and their open ears: Herke, Chris, Gregor, Hany, Doro, Pascal, Joe and Georgia, with a very special thanks to Svenja, for not only always pointing out the smallest mistakes I've overlooked and fruitful discussions, but for always having the time and patience to listen to my nagging.

I would like to thank Prof. Riedmiller for accepting the invitation as a reviewer for this thesis, as well as the Professors Reuter, Koch and Weihe from the TU Darmstadt for being part of my thesis committee. Thank you for your work, time and valuable feedback.

Last but not least, I want to thank my family and friends for the support over the years, and for much-needed distractions from time to time. Cheers!

Contents

Figures and Tables

List of Figures

List of Tables

1. Introduction

The integration of artificial intelligence (AI) technologies in our life, in personal and social environments as well as at work, already had an immense impact and currently it seems as there is no end of that development [1–3]. From personal assistants that help to organize daily routines or work flows, over smart devices for intelligent homes, to less observable applications like automated loan decisions and surveillance, artificial intelligence technology is part of the daily life. Notably though, there are two main limitations of current AI systems: the targeted applications are usually restricted and specialized with individual solutions, and the seemingly intelligent behaviour is mostly carried out by software agents without physical interaction.

One major current barrier underlying those limitations is that an intelligent robot has to interact in the real world, in contrast to aforementioned smart decision systems. This physical component raises multiple challenges in addition, ranging from the perception and cognition of the complex environment, over the calculation of a solution to the queried task, up to the point of executing the appropriate movements. Therefore, intelligent autonomous agents and especially physical robots require a number of different skills and abilities, both *cognitive* and physical, to interact in the real world. In this thesis we therefore investigated the following three research questions:

1) How can tasks and their structure be taught to a robot in a natural way?

2) How can neural networks learn abstract solution strategies that are independent of the task complexity, data representation and task domain?

3) How can a robot efficiently adapt its movement during execution with a bio-inspired stochastic neural network?

These research questions tackle core requirements of an intelligent autonomous agent and we categorize them into *Understand-Compute-Adapt* (UCA), in the style of the classical concept of *Sense-Plan-Act* (SPA) in robotics. The topics are not intended to implement the

Figure 1.1.: *Illustration depicting the thesis title 'Understand-Compute-Adapt: Neural Networks for Intelligent Agents' as the three investigated research questions of cognition, problem solving and online adaptation linked together by the Understand-Compute-Adapt framework and the underlying neural network based models.*

SPA framework directly, but the UCA framework rather serves as both the separation as well as the junction of the topics spanned by these questions.

The first limitation of restricted and specialized tasks refers to cognitive skills of the system and is tackled by the first two research questions. These questions investigate cognitive abilities related to intelligent behaviour, precisely, the ability of understanding and abstraction – a crucial feature for intelligent agents [4, 5]. When talking about intelligent behaviour, we need to clarify the term intelligent. Defining if an artificial system is intelligent is a fundamental research question, dating back to Alan Turing's 'turing test' [6] and John Searle's 'chinese room' experiment [7], and this thesis is not aiming at solving this question. Rather, the question investigated here is how AI systems can learn more *intelligent* behaviour, where intelligent here refers to the ability *to make sense* of the perceived input and to learn more *general* solutions.

In more detail, the agent needs to have an ability of cognition in order to make sense of its perceived input and to learn a transferable solution. Both of these features are characterized by the concept of *abstraction*.

Starting with the first question, an intelligent agent should be equipped with mechanisms that allow to teach it structured tasks in a natural way. In detail, when teaching a robot

a new task, the user should be able to demonstrate the full task to the robot, rather than being forced to show all individual skills that may occur within the whole task. An intelligent robot learns simultaneously to segment these raw demonstrations into reoccurring patterns and a skill library to reproduce those. By showing only the full task execution, without worrying about individual skills, the teaching interface is not only more natural and usable by non-experts, it is also cheaper in terms of data acquisition and required time. Moreover, the agent can additionally learn relations between skills, like which skill is likely to follow another skill. This task structure learning is beneficial for using the learned skill library in novel situations by, for example, reducing the search space for planning algorithms. Additionally, this understanding may also be used in human-robot-interaction scenarios to predict the human behaviour and allowing the robot a more intelligent adaptive behaviour.

The described cognitive *understanding*, i.e., the abstract knowledge inferred from the raw data, helps the agent to *think* about possible actions, with their effects, for solving a given task. This leads to the second question, on how abstract solution strategies can be learned and links to the next step within our UCA frame, namely the *Compute* part. Here, *Compute* refers to the ability of cognitive planning of abstract solutions, a form of *thinking* or *computing*, i.e., learning abstract structured solutions that can be directly transferred to novel problem instantiations. In more detail, the agent is equipped with mechanisms that allow to learn algorithmic solutions with strong abstraction features. The abstract nature of the algorithmic solutions is characterized by the ability of the solution to scale to arbitrary task configurations and complexities, as well as the independence from both the data representation and the task domain. Having learned such general routines for solving certain problem categories, enables the agent to compute solutions efficiently and reliably – a step towards more human-like learning and intelligence [5].

After understanding the input and computing a solution to a given problem, a physical agent ultimately has to interact in the real world to solve the task. As the real world is dynamic, complex and in general not exactly as the agent believes it is, meaning the agent's reception of the real world differs from its cognitive models of the world, online adaptation, to be able to alter its movements and models while interacting in the world is another crucial ability – investigated by the third research question and refers to the third UCA part *Adapt*. Here, online adaptation focuses on altering a probabilistic model for motion planning to adapt to changes in the environment represented as unknown obstacles. This online adaptation is task-independent and utilizes an intrinsic motivation signal for that purpose. In combination with a mental replay strategy, to intensify experienced situations, the adaptation happens within few seconds and from few physical interactions – another important feature for intelligent robots equipped with lifelong learning capacities.

In summary, this thesis investigates three topics tackling core requirements of an intelligent autonomous agent – skill discovery, learning abstract solutions, and online adaptation – and explores how these can be implemented with neural network based models.

1.1. Overview

In this section, the three main topics of this thesis, spanned by the three investigated research questions, are summarized and presented in the UCA-based categorization. In particular, the main findings on how the aforementioned abilities for intelligent behaviour can be implemented and learned in neural network based models are presented.

1.1.1. *Understand* - Unsupervised Skill Discovery

Integrating robots in complex everyday environments requires a multitude of problems to be solved. One crucial feature among those is to equip robots with a mechanism for teaching them a new task in an easy and natural way. When teaching tasks that involve sequences of different skills, with varying order and number of these skills, it is desirable to only demonstrate full task executions instead of all individual skills. For this purpose, we propose a novel approach that simultaneously learns to segment trajectories into individual skills and these skills to reconstruct the data from unlabelled demonstrations without further supervision. Moreover, the approach learns a skill conditioning that can be used to understand possible sequences of skills, a practical mechanism to be used in, for example, human-robot-interactions for a more intelligent and adaptive robot behaviour. The Bayesian and variational inference based approach is evaluated on synthetic and real human demonstrations with varying complexities and dimensionality, showing the successful learning of segmentations and skill libraries from unlabelled data.

This topic is described in Chapter 2 and is based on our work presented in [8].

1.1.2. *Compute* - Learning Algorithmic Solutions

A key feature of intelligent behaviour is the ability to learn abstract strategies that scale and transfer to unfamiliar problems. An abstract strategy solves every sample from a problem class, no matter its representation or complexity – like algorithms in computer

science. Neural networks are powerful models for processing sensory data, discovering hidden patterns, and learning complex functions, but they struggle to learn such iterative, sequential or hierarchical algorithmic strategies. Extending neural networks with external memories has increased their capacities in learning such strategies, but they are still prone to data variations, struggle to learn scalable and transferable solutions, and require massive training data. We present the Neural Harvard Computer (NHC), a memory-augmented network based architecture, that employs abstraction by decoupling algorithmic operations from data manipulations, realized by splitting the information flow and separated modules. This abstraction mechanism and evolutionary training enable the learning of robust and scalable algorithmic solutions. On a diverse set of 11 algorithms with varying complexities, we show that the NHC reliably learns algorithmic solutions with strong generalization and abstraction: perfect generalization and scaling to arbitrary task configurations and complexities far beyond seen during training, and being independent of the data representation and the task domain.

This topic is described in Chapter 3 and is based on our work presented in [9, 10].

1.1.3. *Adapt* - Efficient Online Adaptation

Autonomous robots need to interact with unknown, unstructured and changing environments, constantly facing novel challenges. Therefore, continuous online adaptation for lifelong-learning and the need of sample-efficient mechanisms to adapt to changes in the environment, the constraints, the tasks, or the robot itself are crucial. In this work, we propose a novel framework for probabilistic online motion planning with online adaptation based on a bio-inspired stochastic recurrent neural network. By using learning signals which mimic the intrinsic motivation signal *cognitive dissonance* in addition with a mental replay strategy to intensify experiences, the stochastic recurrent network can learn from few physical interactions and adapts to novel environments in seconds. We evaluate our online planning and adaptation framework on an anthropomorphic KUKA LWR arm. The rapid online adaptation is shown by learning unknown workspace constraints sample-efficiently from few physical interactions while following given way points.

This topic is described in Chapter 4 and is based on our work presented in [11–14].

2. Skill Discovery from Raw Trajectories

In this chapter we are going to investigate the first research question related to the topic *Understand*, i.e., the questions of how new tasks and their structure can be taught to a robot in a natural way. Therefore, we investigate the challenge of simultaneously learning to segment unlabelled trajectories and the required skills to reproduce them.

2.1. Introduction

While pattern recognition, the ability to find order and regularities in noisy observations, is a necessary skill for intelligent behaviour, learning to abstract and transfer that information is a crucial ability that goes beyond pattern recognition [5]. Such cognition abilities are also helpful to employ intelligent robots into our everyday life, by lowering the barrier to program the desired robotic behaviour. While often trained experts can program robots with complex behaviour, this process requires a lot of expert knowledge in different domains, is cost intensive and often limited to special tasks or domains. Instead, it is easier to teach the robot the desired behaviour instead of programming it and is the main motivation behind the learning from demonstrations paradigm [15]. Teaching the robot by showing the desired behaviour is not only a more natural and intuitive way of programming the robot, but it also drastically reduces the required expert knowledge of the system and time.

Typically demonstrations consist of single tasks, and one skill, or policy, per such task is learned from these demonstrations. Fore more complex tasks, that typically consist of multiple subtasks, the task is often either broken down into demonstrated subtasks or the demonstrations are segmented and labelled afterwards. Both methods again require specific knowledge, are time consuming, and scale poorly with the number of (sub)tasks and their arrangements. Subtasks can appear within a demonstration as well as across

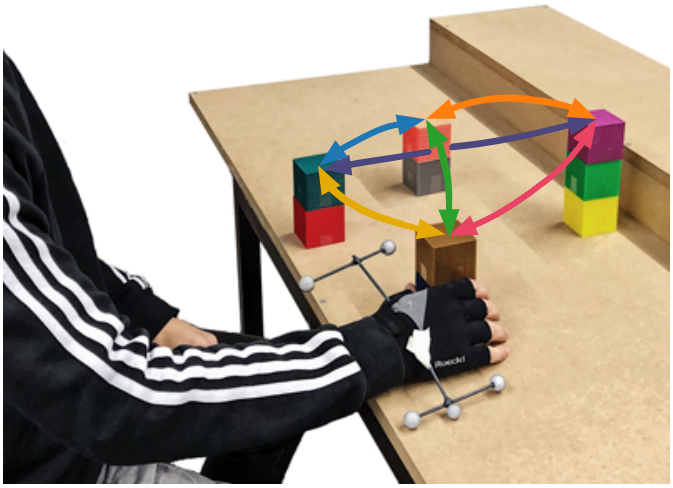
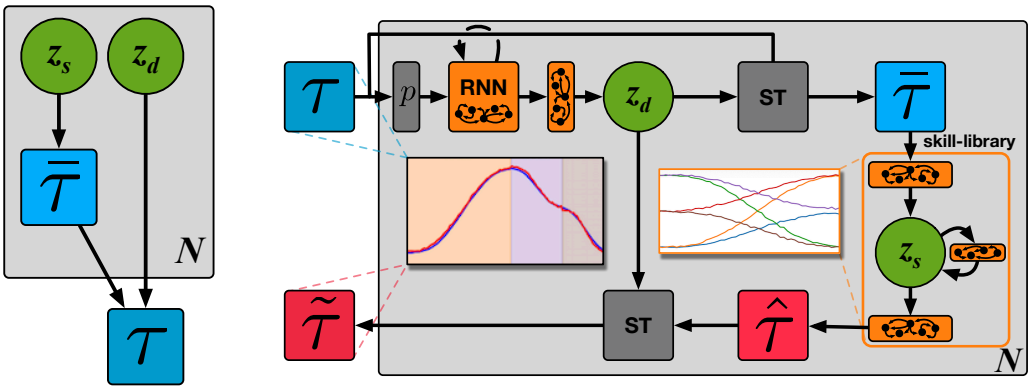


Figure 2.1.: *Snapshot from the data collection process from the human teaching experiment. The human was instructed to move* 1-3 *cubes from one position to another, while the hand position was tracked. These unlabelled hand position trajectories consisting of a variable number of moved cubes are used to learn to segment these into skills and the skills simultaneously.*

demonstrations and it can be challenging to decide where to cut or which segments can be considered the same subtask.

A more natural approach is to just demonstrate the full tasks, and let the system automatically learn to segment the full demonstration into subtasks [16, 17]. Such decompositions of movements into different phases were also detected in the primary motor cortex of monkeys performing reaching tasks [18]. When equipping robots with such an automatic segmentation technique, teaching tasks which consists of a sequence of skills becomes easier for non-experts. Furthermore, by learning the segmentation, the robot can *understand* the demonstrated tasks by, for example, learning important (sub)goals or which skills are likely to follow each other. This approach can therefore also be used in human-robot interaction [19] by, for example, learning the sequence of movements of a human, and using this knowledge to predict the humans future behaviour, which allows for a more intelligent and adaptive robot behaviour.



(a) Graphical model. (b) SKID overview.

Figure 2.2.: *(a) Graphical model representation of the relationship between the N-dimensional latent trajectory description $z = \{z_d, z_s\}$, the sub-trajectory $\bar{\tau}$ and the full trajectory τ. (b) Sketch of the SKID implementation with the learned neural networks in orange. The recurrent neural network (RNN) and the output layer learn the distribution of z_d given the, potentially p transformed, trajectory τ. Then z_d and the raw trajectory are used in a spatial transformer (ST) to extract the sub-trajectory $\bar{\tau}$. This sub-trajectory is used to learn the skill library and the approximated sub-trajectory $\hat{\tau}$ is added to the reconstructed trajectory $\tilde{\tau}$ with a spatial transformer. This process is repeated N times, iteratively segmenting the full trajectory into sub-trajectories utilizing the simultaneously learned skill library. The two inlays show examples from the 1D synthetic task, where the left inlay shows the original trajectory τ in blue and the reconstructed trajectory $\tilde{\tau}$ in red. The coloured background indicates the learned skill z_s used to approximate this segment of size z_d. The right inlay shows the learned skill library.*

2.1.1. Contribution

Here, we propose a novel approach for learning simultaneously to segment trajectories into skills and these skill from unlabelled trajectories without supervision. Our approach is based on the variational autoencoder (VAE) [20] framework and the iterative idea of AIR [21] to explain sub-parts of the given data per iteration. In contrast to related approaches [22, 23] that learn skills with VAEs from single-skill trajectories, our approach operates on trajectories with a varying and unknown number of skills per demonstration and learns to segment the trajectory and the skill library simultaneously.

Furthermore, the proposed method requires less initial knowledge like heuristic cutting points [16], expert domain knowledge [24], segmented and labelled demonstrations [25],

pretraining [26], and segmentation and skill library are trained simultaneously instead of subsequently [17, 27, 28] to use the skill knowledge for segmentation.

We show that our approach can be used to learn skills from unlabelled demonstrations of full task executions involving a varying and unknown number of skills per demonstration, and that the learned model can also be used to understand the relation between skills and, hence, predict possible future skills for adaptive behaviour.

2.2. Discovering Skills from Raw Trajectories

The proposed skill discovery (SKID) approach is inspired by the AIR [21] model for images. AIR is a generative model that learns to reconstruct visual scenes by learning the properties of individual objects to render the scene. We take this approach as inspiration to construct an AIR-like generative model for trajectories, that learns to segment raw trajectories into skills and the individual skills to reconstruct the demonstrations simultaneously and without supervision.

We assume that a trajectory τ is composed of N sub-trajectories $\bar{\tau}$ that we call skills. Like AIR [21] did for images, we take a Bayesian perspective of trajectory understanding and treat it as inference in a generative model. In general, for a given trajectory τ, the model parametrized by θ is given by $p^{\tau}_{\theta}(\tau|z)p^{z}_{\theta}(z)$, where the prior $p^{z}_{\theta}(z)$ over the latents z captures the trajectory assumptions, and the likelihood $p^{\tau}_{\theta}(\tau|z)$ describes how the latent trajectory description is composed into the full trajectory. We are interested in recovering the trajectory description z, given by computing the posterior

$$p(z|\tau) = p^{\tau}_{\theta}(\tau|z)p^{z}_{\theta}(z)/p(\tau) \ . \tag{2.1}$$

As we assume a trajectory τ consists of N sub-trajectories or skills, the trajectory description is structured into a sequence of z^i. Each z^i is a structure that describes the properties of one skill, here, duration and skill type, i.e., $z = \{z_d, z_s\}$. The duration of a skill z_d is used for the segmentation and the skill type z_s is a discrete identifier for each skill. Figure 2.2(a) shows this relation between τ and z^i in a graphical model. The generative model is given as

$$p_{\theta}(\tau) = \sum_{n=1}^{N} \int p^{z}_{\theta}(z^n|z^{n-1})p^{\tau}_{\theta}(\tau|z)dz \ , \tag{2.2}$$

where N is given by the sequence of z_d^n summed up until a given threshold is reached, i.e., $\sum_{n=1}^{N} z_d^n \geq t_\epsilon$. The condition of z^n on z^{n-1} allows to learn a conditional sequencing of the skills, i.e., learning which skills are likely to follow after each other. This conditioning is necessary to sample valid trajectories from the generative model, i.e., trajectories without big jumps between the individual skills. Additionally, by learning this conditioning, the model can be used to *understand* the presented task trajectories, e.g., like predicting which skills are likely to follow each other for adaptive robot behaviour, or be used in planning algorithms when solving new tasks with the learned skills.

2.2.1. SKID Instantiation

We use an amortized variational approximation of the true posterior $p(z|\tau)$ which learns a parametrized distribution $q_\phi(z|\tau)$ by minimizing the Kullback–Leibler divergence between $KL(q_\phi(z|\tau)||p(z|\tau))$. The inference model $q_\phi(z|\tau)$, parametrized by ϕ is realized as a recurrent neural network to take previous z_d's into account, i.e., allowing the model to remember which part of the trajectory have been explained already. Before the trajectory τ is fed into the recurrent network, it is preprocessed by a function p. In our experiments, we tested with p as the identity function and p outputting the mean velocity of τ. The mean velocity performed slightly better and was used in all evaluations. The skill duration z_d is modelled with a Gaussian distribution with a given prior, i.e., $z_d \sim \mathcal{N}(\mu_d, \sigma_d^2)$, fed into a `sigmoid` activation to get the duration as the fraction of the trajectory length.

To extract the part of the trajectory τ indicated by z_d in a differentiable way, a spatial transformer (ST) [29] is used. Each skill duration z_d is used starting with the remaining part of the trajectory, i.e., the part of the trajectory that has not been explained by all previous z_d. This extracted sub-trajectory $\bar{\tau}$ is then used as the input for learning the skill library, the generative model parametrized by θ. Here, we use discrete β-variational autoencoder (VAE) [20, 30, 31] with skip-connections [32] using the continuous gumbel-softmax/concrete approximation [33, 34] for the discrete skill type z_s. This realization of the skill library is very general, which allows to capture any kind of trajectories and is trainable end-to-end within the SKID framework. Similar VAE based approaches but for single skill trajectories were successfully used in [22, 23]. The condition of z^n on z^{n-1} is realized as an additional neural layer, with z^{n-1} as input and the output is added to the logits of the encoder network before sampling z^n. The output of the skill library is the sub-trajectory $\hat{\tau}$ approximated with one activated skill, which is then added to the overall approximated trajectory $\tilde{\tau}$ by a spatial transformer. This iterative process is repeated N

times until the sum of z_d^n reaches a given threshold t_ϵ, reconstructing a fraction z_d of the full trajectory τ per step. The full framework is shown in Figure 2.2(b).

2.2.2. SKID Learning

Learning is done by jointly optimizing the generative model and the inference network, i.e., the parameters θ and ϕ, to maximize the evidence lower bound (ELBO) given as

$$\begin{aligned}\mathcal{L}(\tau;\theta,\phi) &= \mathbb{E}_{q_\phi(z|\tau)}[\log\frac{p_\theta(\tau,z)}{q_\phi(z|\tau)}] \\ &= \mathbb{E}_{q_\phi(z|\tau)}[\log p_\theta(\tau|z)] - \mathrm{KL}(q_\phi(z||\tau)|p(z)) \ , \end{aligned} \tag{2.3}$$

with the parametrized likelihood $p_\theta(\tau|z)$, the parameterized inference model $q_\phi(z|\tau)$ and the latent prior $p(z)$. The first term aims at reconstructing the data while the KL-divergence forces the model to stay close to a given prior. Due to the reparametrization trick for the Gaussian distributed z_d and the gumbel-softmax/concrete distributed z_s, the ϕ parametrized inference network and the θ parametrized generative model, can be learned jointly via stochastic gradient descent.

To enforce disentangled representations, the β-VAE [30] was introduced, which weights the KL term by the hyperparameter β. This balancing was further refined by adding capacity terms to the KL [31, 35], which can be seen as a slack variable allowing some distance in the KL, and which is increased during training. Adding these to Equation 2.3 and separating the different latent variables [35], we get our learning objective as

$$\begin{aligned}\mathcal{L}(\tau;\theta,\phi) = \mathbb{E}_{q_\phi(z|\tau)}[\log p_\theta(\tau|z)] \\ - \gamma_d|\mathrm{KL}(q_\phi(z_d||\tau)|p(z_d)) - C_d| \\ - \gamma_s|\mathrm{KL}(q_\phi(z_s||\tau)|p(z_s)) - C_s| \ , \end{aligned} \tag{2.4}$$

with γ_d, γ_s constant scaling factors, and C_d, C_s the information capacities, i.e, the allowed slack.

To tighten the lower bound estimate and get a more complex implicit distribution, we use the importance weighted autoencoders (IWAE) [36] objective. This formulation uses K samples of the latent variables and weights the according gradients by their relative importance.

2.3. Experiments

In all experiments, SKID is presented unlabelled trajectories and learns to segment those into skills and the required skills simultaneously. The different datasets vary in their complexity in multiple ways: artificially created or real world recorded trajectories, the dimensionality of the trajectories, the number of subsequent skills N, or the total number of different skills. With these different datasets, we test the capacity of SKID to uncover the underlying latent representation from various complex data.

2.3.1. 1D Synthetic

This dataset consists of 1-dimensional trajectories with up to $N = 3$ skills per trajectory, and with a 6 different skills. Here skills refer to reaching specific locations. Data was generated by sampling 10.000 trajectories for each sequence length. The individual locations per trajectory are sampled uniformly and Gaussian noise is added to each skill location. The resulting trajectory is generated by creating a minimum jerk trajectory connecting all the locations. To create a diverse dataset, we adapt a trajectory augmentation method [22] to generate the trajectory τ as:

$$\tau = \mathcal{N}(\tau_o, aB^\dagger) \,, \tag{2.5}$$

with the original trajectory τ_o, the constant a and $B^\dagger$ the Moore-Penrose pseudo-inverse of M, where M is set to

$$M = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 2 & -1 & 0 & \vdots \\ 0 & -1 & 2 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \tag{2.6}$$

to smoothly propagate the disturbances along the trajectory.

2.3.2. 3D Synthetic

This dataset consists of 3-dimensional trajectories with up to $N = 3$ skills per trajectory, with a 12 different skills. Thus, the dataset is more complex in contrast to the 1-dimensional not only in the dimensionality of the trajectories, but also in the number of skills that need to be learned. Generating the dataset was done similar to the 1-dimensional dataset.

2.3.3. 2D Human-Robot-Interaction

This dataset is taken from a human-robot-interaction (HRI) scenario presented in [37], where we use the tracked hand movements of the human to learn the skills used by the human. As SKID also learns the conditioning or sequencing of the skills, this can be used to predict the human behaviour for a more intelligent robot adaptation in such collaborative scenarios. The dataset was created by taking 1000 random cuts at the four known goal locations, such that each trajectory consists of 2 to 5 of those locations, and use the described trajectory augmentation method to create 30.000 trajectories in total. The augmented dataset consists of two-dimensional x, y trajectories with up to $N = 4$ subsequent skills, and ideally 5 different skills to learn. Due to the heuristic cuts and human variation, the actual dataset includes samples with $N > 4$, and has the biggest variability within skills and the biggest noise across the datasets.

2.3.4. 2D Teaching

For this dataset, we recorded the wrist position of a human during an object manipulation task, shown in Figure 2.1. The human was instructed to move between 1 and 3 boxes per demonstration between the four locations. SKID learns the required skills for such manipulation tasks, than can be used for planning by sequencing the discovered skills, and to directly teach the skills to the robot as task space trajectories. The learned conditioning can be used, for example, to reduce the search complexity of planning algorithms. In total we recorded 223 demonstrations and used the trajectory augmentation method described in the synthetic datasets to create a dataset of 5000 trajectories for each number of moved boxes. Thus, the resulting dataset consists of two-dimensional x, y trajectories with up to $N = 5$ subsequent skills and 12 different skills to learn.

2.3.5. Experimental Setup

Learning is done by optimizing Equation 2.5 with the Adam optimizer [38] with learning rate $\alpha = 1e^{-3}$, weight decay λ [39] and variance normalization is applied. The capacities C_d, C_s are linearly increased during training. The mini-batch size is set to 64 and for the IWAE importance sampling $k = 20$ samples are used. The recurrent neural network that learns z_d is a vanilla recurrent network with 64 neurons, and the threshold $t_\epsilon = 0.9$. The skill library VAE encoder and decoder consist of two hidden layers with $[30, 15]$ neurons with `tanh` activations and skip connections within the decoder. For the gumbel-softmax encoded z_s the temperature is decreased from 1.5 to 0.15 over 30.000 iterations with a cosine schedule, using the continuous approximation for training but hard one-hot vectors for evaluation. Trajectories are normalized to 200 steps and the sub trajectories extracted by the spatial transformer consist of 50 steps. The remaining best performing parameters for all datasets are given in Table 2.1.

2.3.6. Results

The goal of SKID is to simultaneously learn to segment trajectories into a varying number of sub-trajectories (skills) and these skills from unlabelled trajectories. Additionally, as part of the skill library, a skill conditioning is learned, that encodes how likely a certain skill follows after another skill. In Figures 2.3 & 2.4 the results on the different datasets are summarized, showing the evaluation of the test model on held-out data and using hard one-hot samples of z_s.

Table 2.1.: *Hyperparameters used in the Experiments.*

	1D syn.	**3D syn.**	**2D HRI**	**2D teaching**
λ	$1e^{-1}$	$5e^{-1}$	$2e^{-1}$	$7e^{-1}$
C_d	$(0, 2, 30k)$	$(1, 5, 30k)$	$(0, 3, 30k)$	$(1, 5, 30k)$
C_s	$(0, 1, 30k)$	$(0, 2, 30k)$	$(0, 1, 30k)$	$(0, 1, 30k)$
γ_d	30	50	30	50
γ_s	10	50	10	30
μ_d	0	0	0.5	0
σ_d	1	1	1	1
VAE σ	0.02	0.02	0.03	0.02
p_θ σ	0.1	0.02	0.1	0.02

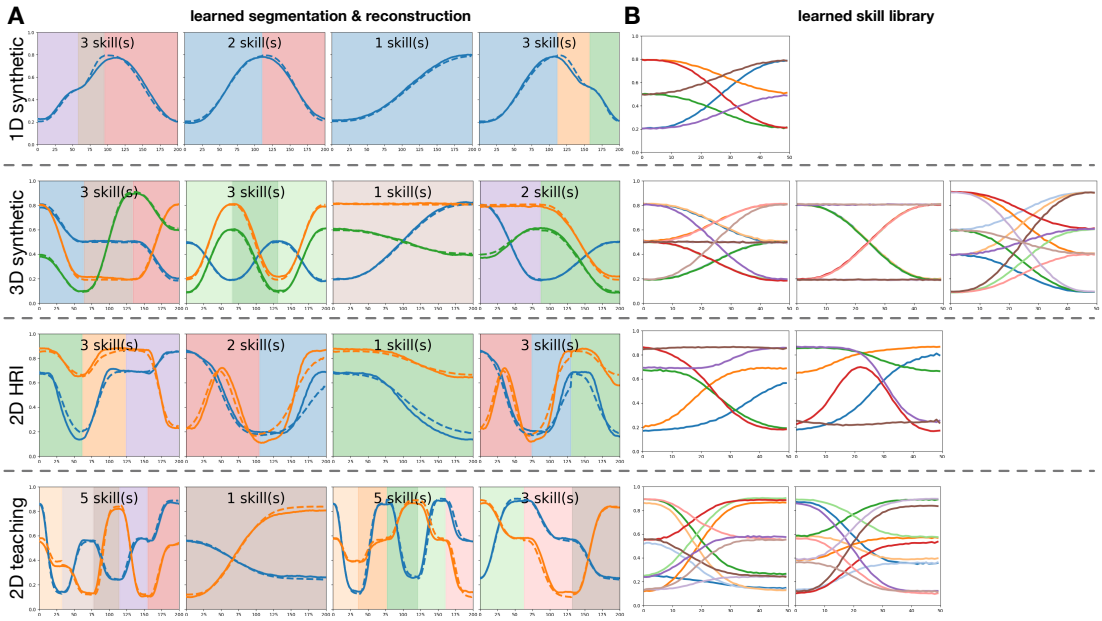


Figure 2.3.: *Showing the learned segmentations and skill libraries by SKID for the four datasets.* ***A*** *shows segmented and reconstructed sample trajectories. Line colours indicate trajectory dimensions, where solid lines show the original data and dashed lines show the reconstructions. The coloured area highlights the segmentation and the colour indicates which skill is used for that segment.* ***B*** *shows the learned skill library with one panel per trajectory dimension and one colour for each skill.*

Segmentation & Skill Library

The segmentation and skill library learning results for all four datasets are shown in Figure 2.3, where each row corresponds to one dataset. The plot shows one exemplary run for each dataset. In the left part of the figure (A), different random trajectories are shown along with their reconstruction and segmentation. Line colours indicate the different dimensions of the trajectories, where solid lines show the input trajectory τ and dashed lines show the reconstructed trajectory $\tilde{\tau}$. Shaded areas indicate the skill used for that segment, where the according learned skills are shown in the right part of the figure (B), with one panel per trajectory dimensions and one coloured line per skill.

For all datasets SKID is able to learn to segment given trajectories into skills and the required skill library simultaneously without supervision. Learning the underlying skills and their segmentation from full plan executions without additional knowledge and

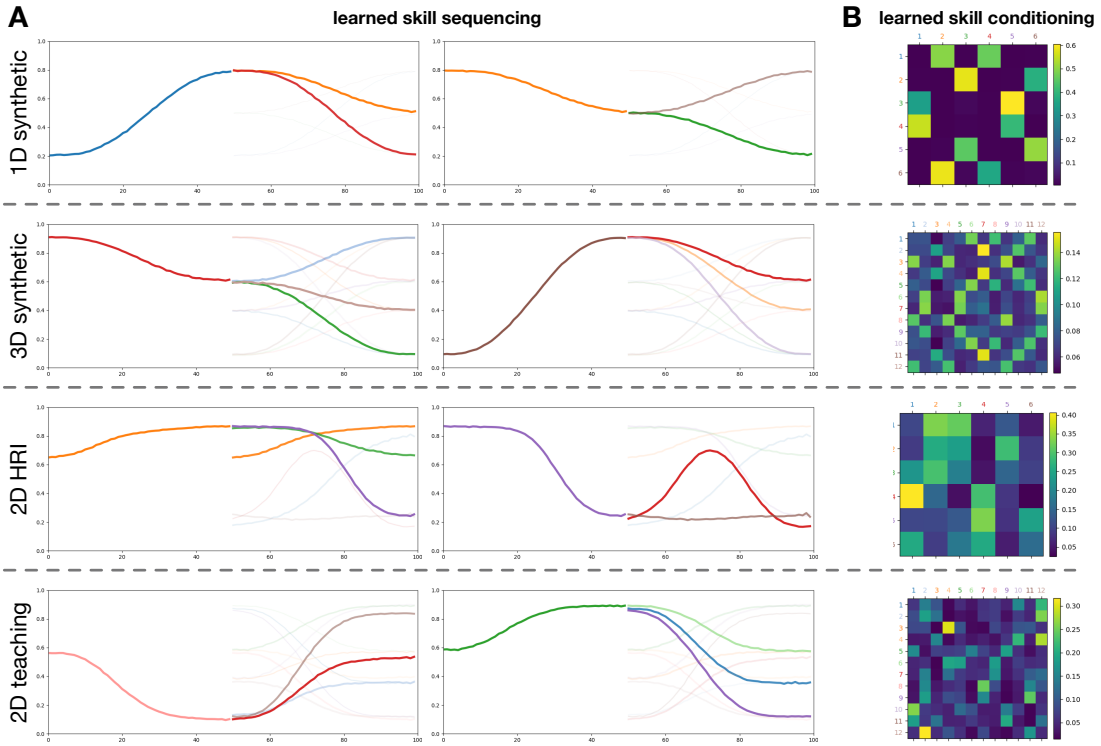


Figure 2.4.: *Showing the learned skill conditioning by SKID for the four datasets.* ***A*** *shows two examples of learned skills and the sequencing of the subsequent skills in one dimension, where the strength of the plotted skill indicates how likely that skill follows the first one.* ***B*** *shows the learned conditioning for all skills as a matrix heatmap, encoded as how likely the skill in column j follows the skill in row i.*

feedback, enables users to teach a robot sequential tasks in a natural way by just executing the full task, without having to specify and demonstrate the individual skills. This learned knowledge can then be used by the robot to, for example, solve new task instantiations by planning with the learned skills.

Skill Conditioning

In addition to learning the segmentation and skill library, SKID also learns a skill conditioning. This learned conditioning and sequencing is shown for all datasets in Figure 2.4, with each dataset in one row. In the left part (A), two examples are shown in the two

panels. In each panel, first one learned skill is plotted followed by all learned skills, but with the line intensity scaled by how likely this skill follows the first one. This sequencing shows that SKID is able to learn which skills are likely to follow each other, and that, for example, jumps between subsequent skills are unlikely. The right part of the figure (B) shows this learned skill conditioning for the whole skill library with a matrix heatmap, where the values indicate how likely the skill in column j follows the skill in row i.

SKID is able to learn this skill conditioning, i.e., task structures, for all datasets. This information can additionally be used, for example, to support the planning for new tasks with the learned skills by reducing the planning space. Another possibility for human-robot-interaction scenarios, the robot can learn to *understand* the movements of the human, and use the learned structure to predict the future behaviour of the human in order to adapt its own behaviour.

2.3.7. Limitations

While SKID is able to learn the segmentation, conditioning and skill library from different datasets with varying complexities, the stochastic variational inference setting is challenging. Not all runs achieve perfect results, where the major problem that occurs is that the discrete VAE used for the skill library sometimes *misses* one or a few skills, while the segmentation is still perfect. Due to the continuous approximation during training, the training model can mix multiple skills to achieve good performance, the test model with hard one-hot vectors for the skills type z_s can only use one skill, and thus, this results in a sub-perfect performance. While using lower temperatures for the gumbel-softmax during training can help with this issue, the lower temperatures create higher variance in the gradients.

As the number of skills has to be specified for the skill library, another possibility is to use more skills than required, such that unused latent dimensions matter less. This strategy was successful in some experiments, but with more potential skills, the segmentation becomes worse, as the model tends to learn too complex skills, i.e., combining multiple skills into one, as the higher number of available skills reduces the necessity of finding the simplest individual skills.

2.4. Conclusion

Here we proposed a novel Bayesian approach to learn trajectory segmentation from unlabelled raw trajectories. The SKID framework builds on a hierarchical VAE structure and learns simultaneously to segment trajectories into reoccurring skills, the required skills, and the temporal relation between these skills. These features were successfully shown on multiple datasets with varying complexities, including two datasets with motions tracked from a human teacher.

Such automatic skill discovery can be used as a natural interface for teaching a robot complex tasks consisting of multiple skills. In addition with the learned skill conditioning, the framework can also be used to analyse and predict the behaviour of a human, or another robot, for a more intelligent adaptive behaviour of the agent.

Moreover, the framework is not limited to robotic or human movement trajectories, but can be applied to any kind of trajectories, that consist of reoccurring patterns and opens interesting future research. Another promising direction could tackle the limitation of the offline setting, i.e., a full trajectories are used for segmentation and learning, by integrating online change point detection [40, 41] into the SKID framework and feeding the trajectory step by step to allow the processing of longer tasks.

3. Algorithmic Generalization of Neural Computers

The following chapter investigates the second research question, namely, the question how abstract strategies in form of algorithmic solutions can be learned, relating to topic *Compute*. Abstraction here refers to the ability to find solutions that scale to arbitrary task complexities, and that are independent from the data representation and task domain. Such abstract solutions are beneficial for an intelligent agent in order to cope with unforeseen situations.

3.1. Introduction

A crucial ability for intelligent behaviour is to transfer strategies from one problem to another, studied, for example, in the fields of lifelong and transfer learning [42–45]. Learning and especially deep learning systems have been shown to learn a variety of complex specialized tasks [46–51], but extracting the underlying structure of the solution for effective transfer is an open research question [44].

The key for effective transfer, and a main pillar of (human) intelligence, is the concept of structure and abstraction [4, 5, 52]. To study the learning of such abstract strategies, the concept of *algorithms* like in computer science [53] is an ideal example for such transferable, abstract and structured solution strategies.

An algorithm is a sequence of instructions, which often represent solutions to smaller subproblems. This sequence of instructions solves a given problem when executed, independent of the specific instantiation of the problem. For example, consider the task of sorting a set of objects. The algorithmic solution, specified as the sequence of instructions, is able to sort any number of arbitrary classes of objects in any order, e.g., toys by colour,

waste by type, or numbers by value, by using *the same sequence of instructions*, as long as the features and comparison operators defining the order are specified.

Learning such structured, abstract strategies enables the effective transfer to new domains and representations as the abstract solution is independent of both. In contrast, transfer learning usually focuses on improving learning speed on a new task by leveraging knowledge from previously learned tasks, whereas algorithmic solutions do not need to (re)learn at all, only the data specific operations need to adapt. In other words, the sequence of instructions does not need to be adapted, only the instructions, i.e., the solutions to smaller subproblems. Moreover, such structured abstract strategies have built-in generalization capabilities to new task configurations and complexities, and can be interpreted better than, for example, common blackbox models like deep end-to-end networks.

3.1.1. The Problem of Learning Algorithmic Solutions

To study the learning of such abstract and structured strategies, we investigate the problem of learning algorithmic solutions which we characterize by three requirements:

- **R1** – generalization and scaling to different and unseen task configurations and complexities
- **R2** – independence of the data representation
- **R3** – independence of the task domain

Picking up the sorting algorithm example again, R1 represents the generalization and scaling properties, which allow to sort lists of arbitrary length and initial order, while R2 and R3 represent the abstract nature of the solution. This abstraction enables the algorithm, for example, to sort a list of binary numbers while being trained only on hexadecimal numbers (R2). Furthermore, the algorithm trained on numbers is able to sort lists of strings (R3). If R1 – R3 are fulfilled, the algorithmic solution does not need to be retrained or adapted to solve unforeseen task instantiations – only the data specific operations need to be adjusted.

Earlier research on solving algorithmic problems has been done, for example, in grammar learning [54–56], and is becoming a more and more active field in recent years outside of it [57–71], with a typical focus on identifying algorithmic generated patterns or solving *algorithmic problems* in an end-to-end setup [57–66], and less on finding *algorithmic*

solutions [67–71] that consider the three discussed requirements R1 – R3 for generalization, scaling and abstraction.

While R1 is typically tackled in some (relaxed) form, as it represents the overall goal of generalization in machine learning, the abstraction abilities R2 and R3 are missing. Additionally, most algorithms require a form of feedback, using computed intermediate results from one computational step in subsequent steps, and a variable number of computational steps to solve a problem instance. Thus, it is necessary to be able to cope with varying numbers of steps and determining when to stop, in contrast to using a fixed number of steps [60, 72], and to be able to re-use intermediate results, i.e., feeding back the models output as its input. These features make the learning problem even more challenging.

3.2. The Neural Harvard Computer

The proposed Neural Harvard Computer (NHC) is a modular architecture that is based on memory-augmented neural networks [54–60, 62, 63, 66–68, 72–78] and inspired by modern computer architectures (see Figure 3.1 for a sketch of the NHC). Memory-augmented networks add an external memory to a neural network, that allows to separate computation and memorization – in classical neural networks both is encoded in the synaptic weights.

The external memory can be realized differently, e.g., as a memory matrix [59], tape [62], or stack [58], and the so called controller network can write and read information through a defined interface, that controls the memory access, e.g., moving the head one step to the right for a tape memory, or pop the top information in a stack memory. In the NHC the external memory is realized as a matrix and interaction with the memory is done via write and read heads, similar to the Differential Neural Computer [59]. These heads interact with the memory by writing or reading information into or from the memory matrix, where each row corresponds to a memory location with a specified word size, i.e., length of the information vector.

Information Split Learning algorithmic solutions requires the decoupling of algorithmic computations from data dependent manipulations and domain. Therefore, an abstraction level is introduced by dividing the information flow into two streams, data d and control stream c. Like the introduction of external memories to neural networks helps to separate

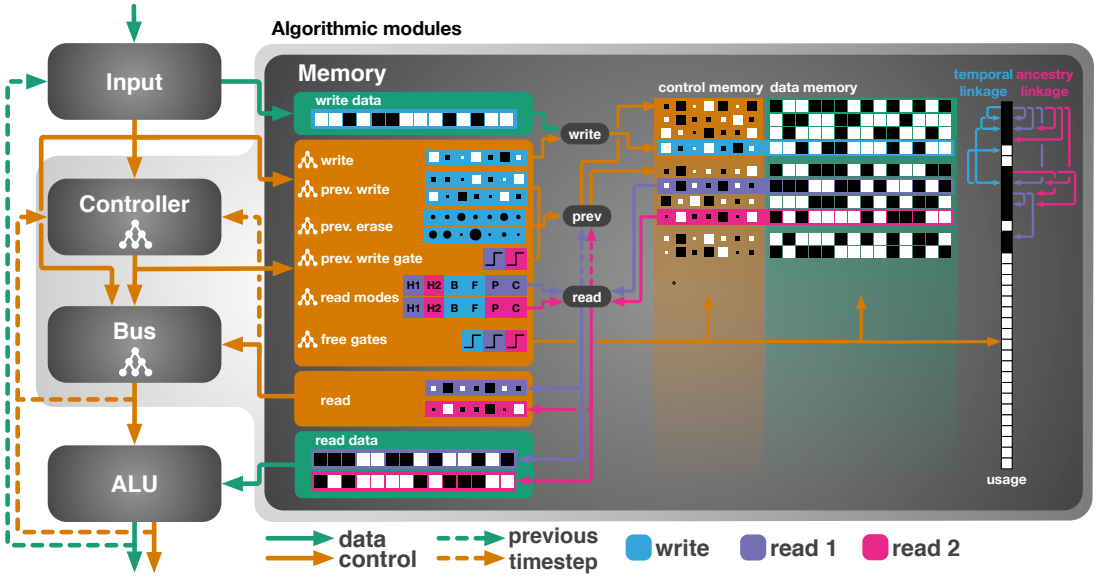


Figure 3.1.: ***The neural harvard computer architecture.*** *Information flow is divided into data (green) and control (orange) streams. The modules inside the light grey area – the controller, the memory and the bus – are learning the algorithmic solution on the control stream, whereas the data modules are either learned beforehand or hand-designed. The algorithmic solution operates solely on the control stream to steer the data access and manipulation, whereas the learning signal can be provided on any connection in the architecture (data or control) due to the evolution based training. Inside the memory module the learnable interfaces which control the data access to the two memory matrices are shown. Sign and magnitude of vectors are shown as the colour and size of the boxes and circles.*

computation and memorization, the information flow split helps to separate algorithmic computations and data specific manipulations.

This information split induces two major features of the NHC: (1) the split into data modules that operate on the data stream d and algorithmic modules that operate on the control stream c, and (2) the introduction of two coupled memories. The algorithmic modules operate on the control stream c, i.e., $AlgModule(c) \longrightarrow c$, whereas the data modules *Input* and *ALU* are operating on the data stream d, i.e., $DataModule(d, c) \longrightarrow d, c$. They create an abstract interface and *separation* between algorithmic computation and data specific processing. While the *Input* module receives the external data and provides

algorithm specific control signals, the *ALU* receives data and control information to manipulate the data to create new data – hence the name *arithmetic logic unit* – that is fed back to *Input* to be available in the next computation step. These two modules are data specific and need to be adapted for a new data representation or domain.

The Algorithmic Modules consists of the *Controller*, *Memory* and *Bus*. These modules form the core of the NHC (see Figure 3.1) and are responsible for encoding and learning the algorithmic solution based on the control stream c.

The Controller receives the control signals from the *Input* and the information read from the memory in the previous step – depending on the task to learn, it can also receive feedback from the *Bus* and *ALU*. It learns an internal representation of the algorithm state that is send to the *Memory* and *Bus* modules.

The Memory module uses this representation in addition to the control signals from the *Input* to learn a set of interfaces for interacting with the memory matrices. These learned interfaces control the write and read heads and hence, what information is accessed. First the locations read in the previous step are potentially updated (prev), then new information is written via the write heads (write), and finally the read heads read information (read) that is send back to controller and the *Bus*. Write and read heads are using hard decisions, i.e., each head interacts with one memory location.

The abstraction introduced by the information split also creates the necessity to store data and control information separately. Therefore, the *Memory* module uses two memory matrices $M^c : N \times C$ and $M^d : N \times D$ to store the control and data information respectively, with N locations, C and D the word size accordingly. The two memories are coupled such that in each step the same locations are accessed. This allows to store algorithmic control information alongside the data information. New information is written via the write heads to unused memory locations, and locations can be freed by free gates to be reused. For reading information from the memory, there are several read modes to steer the read heads.

The *HALT* modes move the read head to the previously read location of the associated head. For example, with two read heads, each head can use two modes *H1* and *H2* to move to the location previously read by the corresponding head.

To read data in the order of which it was written, the DNC introduced a *temporal linkage* mechanism that keeps track of the order of written locations. The NHC uses a simplified version of this temporal linkage. This temporal linkage provides two read modes, one to move the read head *forward* to the location that was written next (F), and one to move the head *backwards* to the location that was written before (B).

Algorithms often require hierarchical data structures or dependencies. To provide such dependencies, the NHC employs an *ancestry linkage* mechanism. This mechanism keeps track for each written location, which location was read before. Therefore, this mechanisms provides two read modes, one to move the read head to the *parent* (P), the location that was read before, and one to move the read head to the *child* (C), the location that was written after.

The Bus combines the representation learned by the *Controller* with the information read out from the *Memory* to produce the control signal that is send to the *ALU*, indicating which operation to apply on the read data. By using the information read from memory, the *Bus* can incorporate this new information in the same computational step.

3.3. Learning Algorithmic Solutions

For evaluating the proposed NHC on the three algorithmic requirements R1 – R3, a diverse set of algorithms was learned and the solutions were tested on their generalization, scaling and abstraction abilities.

The 11 learned algorithms solve search, plan, addition, sorting, evaluating arithmetic expressions, and sequence retrievals problems. In Figure 3.2 all 11 algorithms are sketched with their pseudocode and examples (more details can be found in Section 3.6).

Learning is done in a curriculum learning setup [79], where the complexity of presented samples increases with each curriculum level. During learning, samples up to curriculum level 10 are considered, with an additional level 11 that samples from all previous levels. Generalization and scaling is tested on complexities up to level 1000. The direct transfer is tested by transferring the learned solutions to novel problem representations.

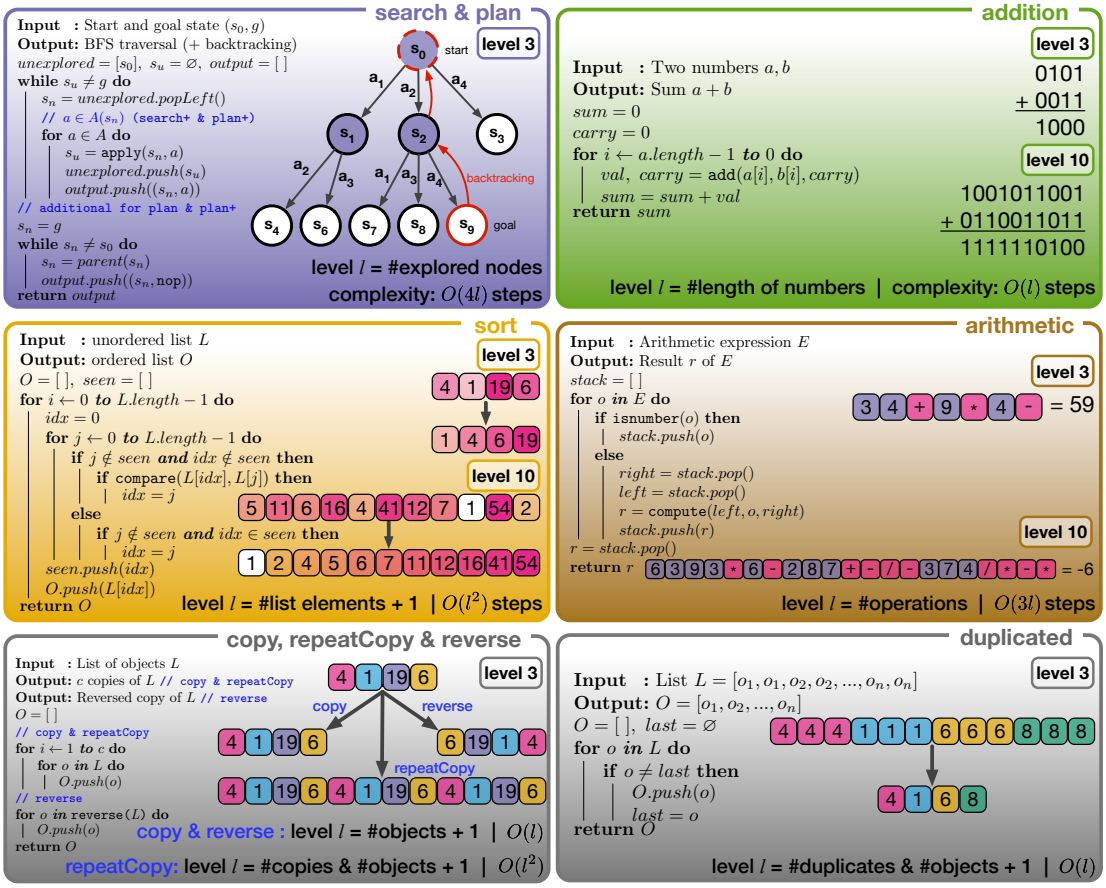


Figure 3.2.: ***Overview of the learned algorithms.*** *All considered algorithms to learn are shown with their pseudocode, how their curriculum level complexity is defined, how the step complexity scales with the level, and examples from indicated levels. Note, the step complexity indicates the runtime complexity and only considers the steps after the input data is shown, neither taking the complexity of the data manipulation into account, nor the structure learning required while the input data is presented.*

3.3.1. Learning Procedure Overview

The algorithmic modules, encoding the algorithmic solution, are learned via Natural Evolution Strategies (NES) [80]. In each iteration t, a population of P offspring (altered parameters θ_t^o) is generated, and the parameters are updated in the direction of the best

performing offspring. Parameters are updated based on their fitness, a measurement that scores how well the offspring perform. Such optimizers do not require differentiable models, giving more freedom to the model design, e.g., using non-differentiable hard memory decisions [63] and instantiating the modules freely and flexible.

An update at iteration $t+1$ of the parameters θ with learning rate α and search distribution variance σ^2 is performed as $\theta_{t+1} = \theta_t + \alpha\nabla_{\theta_t}$ with the sampled NES gradient given as

$$\nabla_{\theta_t}\mathbb{E}_{\epsilon\sim N(0,I)}\left[f(\theta_t+\sigma\epsilon)\right] \approx \frac{1}{P\sigma}\sum_{o=1}^{P} f(\theta_t^o)\epsilon_i \ .$$

Hence, parameters are updated based on a performance-weighted sum of the offspring. Here, the fitness function $f(\cdot)$ scores how many algorithmic steps were done correctly – if the correct data was manipulated in the correct way at the correct step. These binary signals for each step are averaged over all steps and all samples in the minibatch to get a scalar fitness value. This results in a coarse feedback signal and harder learning problem in contrast to gradient based training, where the error backpropagation gives localized feedback to each parameter.

For all algorithms, generalization and scaling (R1) was tested in two ways. First, testing for scaling to more complex configurations is integrated into our learning procedure, and second, the solutions were tested on complexities far beyond those seen during training.

A curriculum level is considered solved after a defined number of subsequent iterations with maximum fitness, i.e., with perfect solutions where every bit in every step is correct. When a new level is unlocked, samples with higher complexity are presented and hence, if the fitness stays at maximum, the acquired solution scaled to that new complexity. Learning is only performed in iterations which do not have maximum fitness.

In addition to this built-in generalization evaluation, the learned solutions were tested on complexities far beyond seen during training, i.e., corresponding to curriculum levels 100, 500 and 1000, while being trained only up to level 10.

3.3.2. Learning Results

Learning results on all 11 algorithms are presented in Figure 3.3, Table 3.1 and the Extended Data Figure 3.6, where all results are obtained over 15 runs of each configuration, and Figure 3.5 illustrates the learned algorithmic behaviour for four algorithms.

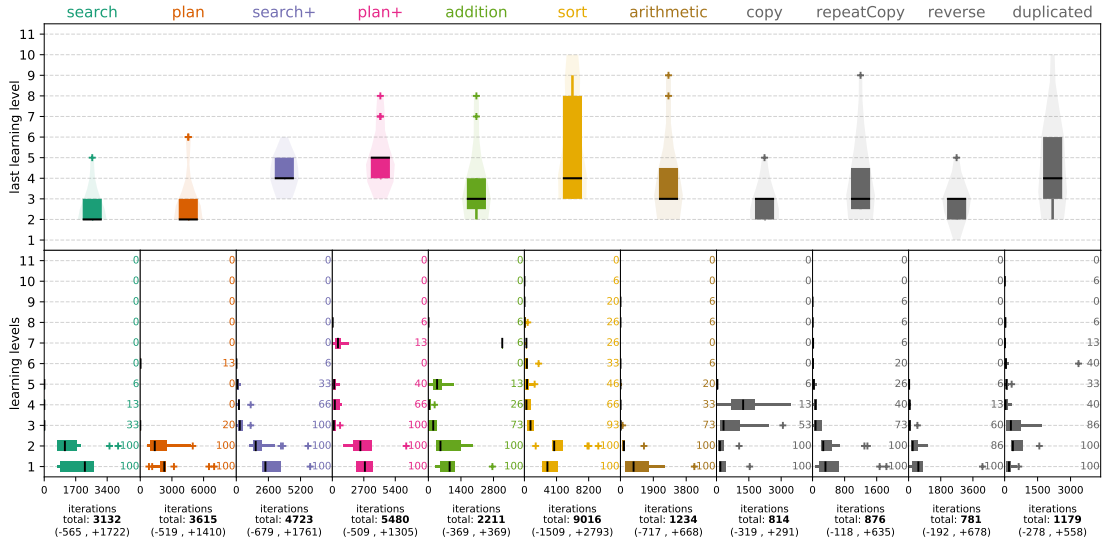


Figure 3.3.: ***Learning overview of all 11 learned algorithms.*** *(Top) The last curriculum level that triggered learning, i.e., where the last mistake occurred, is shown as the median (black line), the interquartile range (box) and outliers (plus). All algorithms are learned within the first few levels and the solution generalizes to higher levels. (Bottom) The number of training iterations per curriculum level is shown. The coloured numbers indicate the percentage of runs that triggered learning in each level. Learning occurs in the first levels, mostly within the first two, and subsequent levels only need a small amount of iterations to adapt, if at all. The total iterations show median and distances to the interquartile range of the total number of learning iterations. Results are obtained over* 15 *runs for each algorithm.*

In Figure 3.3, we illustrate in which curriculum levels learning was triggered. The top row shows the last level in which training was triggered, the last level with an error, indicating that learning only occurs in the first levels and solutions generalize to subsequent levels, i.e, to higher complexities, which can be observed for all 11 algorithms reliably over all runs. In the bottom row, we investigated in how many iterations learning was observed in each level and in total. This highlights the fact that most training happens in the first levels, and subsequent levels only need a few iterations to adapt, if at all. The total number of learning iterations highlights the efficient training in terms of samples. This measure provides an indicator for the task complexity, i.e., for sorting 9016 iterations caused network updates, whereas copying is less challenging and only required 814 iterations of network updates.

In Table 3.1 more details of the learning, the generalization and scaling evaluation for R1,

and comparison methods are shown. The *last level* entry shows the last level triggering learning alongside the last level that was solved successfully, highlighting that all runs for all algorithms were able to solve all 11 training levels while triggering learning only in the first levels. Next, the table shows the results on testing the solutions on complexities far beyond those seen during training. Each run was presented 50 samples from the associated level (20 samples for sort levels 500 and 1000 due to runtime scaling).

For the majority of algorithms, all runs generalized perfectly to complexities up to level 1000. In the harder tasks, like sorting, some runs fail for perfect generalization, still performing well and the majority of runs also shows perfect generalization. Note that, a sample from level 1000 in the sort task requires over $1\ million$ perfect computational steps to be considered solved. The performances below 100% for some runs can be explained with the mechanisms of the previous write head. The model has to learn if the previously read location should be updated and with which information, without explicit feedback on these signals. Thus, an update mechanism that learned to slightly update the previous location works fine on shorter sequences (like seen during training), but the small changes accumulate on longer sequences and may result in wrong behaviour. A possible solution would be to add feedback to these signals during training if it can be provided.

Overall, the results summarized in Figure 3.3 and Table 3.1 show that the solutions learned by the NHC fulfil the algorithmic requirement R1 of generalization.

Comparison For comparison we trained four additional models. First, the DNC [59] model as a state-of-the-art memory-augmented neural. This model is trained in a supervised setting with backpropagation, i.e., having a much richer and localized learning signal. It was able to learn some of the baseline algorithms up to level 5, like addition, copy and reverse, but failed in earlier levels in the remaining tasks, despite being trained for $500k$ iterations. Notably, the DNC struggled with those tasks requiring to reuse intermediate results or iterating over the data multiple times.

Second, we integrated the DNC into the NHC architecture by replacing the algorithmic modules of the NHC – controller, memory, bus – with the original DNC. This DNC+is+ha model uses the same data modules and is trained like the NHC with NES. It performs notably better than the DNC, indicating the help of the proposed abstraction mechanisms and the evolutionary training. Nevertheless, it still is not able to generalize comparable to the NHC and struggles with the same algorithms as the DNC. More details on these comparisons and their learning are given in Section 3.7.

			search		plan		search+		plan+		addition		sort		arithmetic		copy		repeatCopy		reverse		duplicated	
NHC	train	iterations	3132 $^{+1722}_{-565}$		3615 $^{+1410}_{-519}$		4723 $^{+1761}_{-679}$		5480 $^{+1305}_{-509}$		2211 $^{+369}_{-369}$		9016 $^{+2793}_{-1509}$		1234 $^{+668}_{-717}$		814 $^{+291}_{-319}$		876 $^{+635}_{-118}$		781 $^{+678}_{-192}$		1179 $^{+558}_{-278}$	
		last level	2 $^{+1}_{-0}$	11 $^{+0}_{-0}$	2 $^{+1}_{-0}$	11 $^{+0}_{-0}$	4 $^{+1}_{-0}$	11 $^{+0}_{-0}$	5 $^{+0}_{-1}$	11 $^{+0}_{-0}$	3 $^{+1}_{-0.5}$	11 $^{+0}_{-0}$	4 $^{+4}_{-1}$	11 $^{+0}_{-0}$	3 $^{+1.5}_{-0}$	11 $^{+0}_{-0}$	3 $^{+0}_{-1}$	11 $^{+0}_{-0}$	3 $^{+1.5}_{-0.5}$	11 $^{+0}_{-0}$	3 $^{+0}_{-1}$	11 $^{+0}_{-0}$	4 $^{+2}_{-1}$	11 $^{+0}_{-0}$
		level 10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	test	level 100	100%	100%	100%	100%	100%	100%	100%	100%	80%	80%	73.3%	86.1%	100%	100%	93.3%	93.3%	80%	80%	100%	100%	100%	100%
		level 500	100%	100%	100%	100%	100%	100%	100%	100%	80%	80%	73.3%	87.7%	100%	100%	93.3%	93.3%	80%	80%	100%	100%	100%	100%
		level 1000	100%	100%	100%	100%	100%	100%	100%	100%	80%	80%	73.3%	86.1%	100%	100%	93.3%	93.3%	80%	80%	100%	100%	100%	100%
NHC-anc	train	iterations	4319 $^{+448}_{-888}$		–		–		–		1586 $^{+1410}_{-488}$		5677 $^{+892}_{-1311}$		2467 $^{+494}_{-1041}$		1051 $^{+841}_{-159}$		3496 $^{+648}_{-697}$		848 $^{+717}_{-558}$		2465 $^{+607}_{-132}$	
		last level	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	–		–		–		3 $^{+1}_{-1}$	11 $^{+0}_{-0}$	2 $^{+0}_{-1}$	1 $^{+0}_{-1}$	3 $^{+1.5}_{-1}$	11 $^{+0}_{-0}$	2 $^{+0}_{-0}$	11 $^{+0}_{-0}$	2 $^{+0}_{-0}$	1 $^{+0}_{-1}$	3 $^{+1}_{-1}$	11 $^{+0}_{-0}$	2 $^{+0}_{-0}$	1 $^{+0}_{-0}$
		level 10	0%	0%	–		–		–		100%	100%	0%	0%	86.7%	86.7%	86.7%	86.7%	0%	1.9%	100%	100%	0%	2.86%
	test	level 100	–		–		–		–		93.9%	93.3%	–		80%	86.3%	86.7%	86.7%	0%	0%	93.3%	93.3%	0%	0%
		level 500	–		–		–		–		93.9%	93.3%	–		80%	82.3%	86.7%	86.7%	–		93.3%	93.3%	–	
		level 1000	–		–		–		–		93.9%	93.3%	–		80%	80.1%	86.7%	86.7%	–		93.3%	93.3%	–	
NHC-prev	train	iterations	3913 $^{+937}_{-749}$		–		–		–		2722 $^{+765}_{-868}$		3784 $^{+1002}_{-564}$		2391 $^{+1432}_{-824}$		362 $^{+163}_{-123}$		1118 $^{+267}_{-500}$		322 $^{+762}_{-118}$		2398 $^{+235}_{-275}$	
		last level	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	–		–		–		3 $^{+0}_{-1}$	11 $^{+0}_{-0}$	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	6 $^{+0}_{-0.5}$	11 $^{+0}_{-0}$	2 $^{+1.5}_{-0}$	11 $^{+0}_{-0}$	3 $^{+0}_{-1}$	11 $^{+0}_{-0}$	3 $^{+0.5}_{-0.5}$	11 $^{+0}_{-0}$	2 $^{+0}_{-0.5}$	1 $^{+0}_{-0.5}$
		level 10	0%	0%	–		–		–		100%	100%	0%	0%	93.3%	93.3%	100%	100%	100%	100%	86.7%	86.7%	0%	0.7%
	test	level 100	–		–		–		–		100%	100%	–		93.3%	93.3%	100%	100%	93.3%	93.3%	86.7%	86.7%	0%	0%
		level 500	–		–		–		–		100%	100%	–		86.7%	93.2%	100%	100%	93.3%	93.3%	86.7%	86.7%	–	
		level 1000	–		–		–		–		100%	100%	–		86.7%	93.2%	100%	100%	93.3%	93.3%	86.7%	86.7%	–	
DNC+is+ha	train	iterations	3159 $^{+937}_{-749}$		–		–		–		4315 $^{+1584}_{-365}$		5539 $^{+2653}_{-905}$		4360 $^{+1842}_{-845}$		4355 $^{+945}_{-1121}$		4163 $^{+475}_{-999}$		2126 $^{+766}_{-123}$		3756 $^{+2025}_{-1046}$	
		last level	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	–		–		–		10 $^{+0}_{-0}$	11 $^{+0}_{-0}$	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	2 $^{+1}_{-0}$	1 $^{+0}_{-1}$	10 $^{+0}_{-1}$	11 $^{+0}_{-0}$	10 $^{+0}_{-0}$	11 $^{+0}_{-0}$	10 $^{+0}_{-1}$	11 $^{+0}_{-0}$	2 $^{+1}_{-0}$	1 $^{+0}_{-1}$
		level 10	0%	0%	–		–		–		20%	65.2%	0%	0%	0%	0%	100%	100%	20%	92%	100%	100%	0%	5.7%
	test	level 100	–		–		–		–		0%	0%	–		–		0%	0%	13.3%	13.3%	0%	0%	0%	0%
		level 500	–		–		–		–		–		–		–		–		13.3%	13.3%	–		–	
		level 1000	–		–		–		–		–		–		–		–		13.3%	13.3%	–		–	
DNC[59]	train	iterations	500k		–		–		–		500k		500k		500k		500k		500k		500k		500k	
		last level	1 $^{+0}_{-0}$	0 $^{+0}_{-0}$	–		–		–		6 $^{+0}_{-0}$	5 $^{+0}_{-0}$	2 $^{+0}_{-0}$	1 $^{+0}_{-0}$	2 $^{+0}_{-0}$	1 $^{+0}_{-0}$	6 $^{+0}_{-0}$	5 $^{+0}_{-0}$	2 $^{+0}_{-0}$	1 $^{+0}_{-0}$	6 $^{+0}_{-0}$	5 $^{+0}_{-0}$	5 $^{+0}_{-0}$	4 $^{+0}_{-0}$
		level 10	0%	0%	–		–		–		0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	13.3%	13.3%	0%	0%
	test	level 100	–		–		–		–		–		–		–		–		–		0%	0%	–	
		level 500	–		–		–		–		–		–		–		–		–		–		–	
		level 1000	–		–		–		–		–		–		–		–		–		–		–	

Table 3.1.: ***Evaluation and Comparison.*** *Shown are results over* 15 *runs for each algorithm and model. Triplets like* 5 $^{+3}_{-2}$ *show median and distances to the interquartile range. Iterations refers to the number of learning iterations. The two last level triplets show the last learning level (left) and the last solved level (right). The two percentages in level X indicate the amount of perfect runs (left), i.e., runs that solved all presented samples, and the amount of solved samples over all runs (right). All NHC variants and the DNC+is+ha model are using the information split and data modules, and are trained with NES. The original DNC is trained in a supervised setting with backpropagation.*

Next, we removed the proposed ancestry linkage (NHC-anc) and the previous location update (NHC-prev) to evaluate their influence. To counter the removed update head, the NHC-prev model uses two write heads, enabling it to learn a similar update mechanism. Both models perform better than the DNC+is+ha and are able to learn the majority of algorithms and even achieve perfect generalization in some, strengthening the importance of the evolutionary training and highlighting the influence of the proposed mechanisms. The performance of the two ablation models depends on the algorithm to learn, i.e.,

whether the algorithm requires the hierarchical knowledge provided by the ancestry linkage or the updating of previously read locations. Notably, both mechanisms are required to learn the search and plan algorithms.

These results suggest that the evolutionary training with the proposed abstraction mechanisms and the new memory module are key ingredients for reliably learning algorithmic solutions that generalize and scale, and hence, fulfilling R1.

3.3.3. Transfer of the Learned Algorithmic Solutions

Next, we evaluated the ability to generalize the learned solutions to new problem instantiations, testing the requirements R2 (independence of the data representation) and R3 (independence of the task domain). Therefore, the algorithmic solutions were tested on unseen data representations and task domains. For these transfers, the learned algorithmic modules were used with adapted data modules for the new setups.

In Figure 3.4 all transfers are illustrated, showing the training setup and the successful transfers. The transferred solutions solved all 11 curriculum levels in the new setup without triggering learning once, i.e., no single error occurred.

For search and plan, we investigated if the strategy learned in sokoban could be transferred to bigger environments, to a different data representation, to a sliding puzzle problem, and to a robot manipulation task. The solutions were learned in 6×6 environments, and could perfectly solve 8×8 environments and a changed encoding of the environment, e.g., the penguin represents a wall instead of the agent (see Figure 3.4). In the 3×3 sliding puzzles, the white space represents empty space onto which adjacent tiles can be moved. In the robotic setup, the task is to rearrange the four stacks of boxes from one configuration into another.

Addition, sort and the baseline algorithms – copy, repeatCopy, reverse, duplicated – were trained on binary numbers and were successfully transferred to decimal numbers.

The arithmetic algorithm was trained on decimal arithmetic and was transferred to a boolean algebra. As the atomic operations $[+, -, *, /]$ are part of the data input sequence, the solution is independent from the number of atomic operations, shown by having only two atomic operations `AND` & `OR` in the boolean algebra setup.

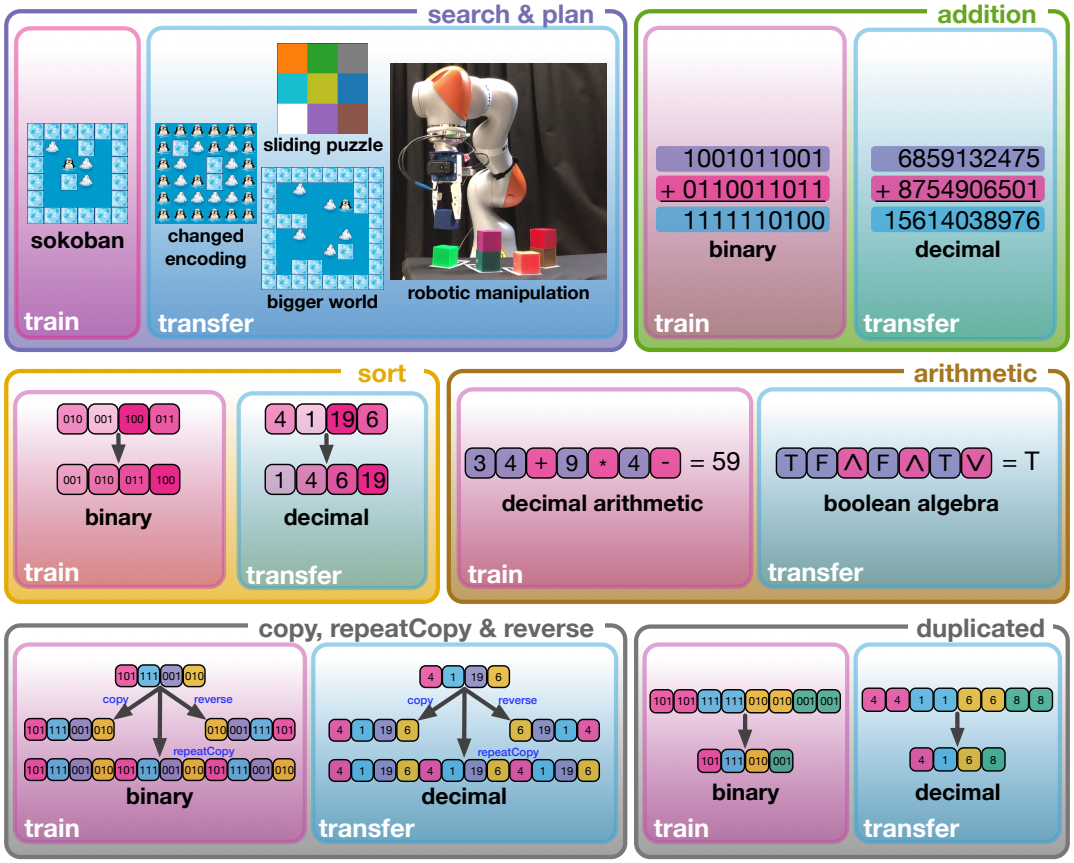


Figure 3.4.: ***Overview of the transfers of the learned algorithms.*** *To show the abstract nature of the learned algorithms, each learned algorithm was transferred and tested on at least one different data representation or domain. All transfers were successful, i.e., the learned algorithm solved all samples in the new domain without triggering learning of the algorithmic modules, indicating the fulfilment of R2-R3.*

Limitations and Assumptions In our transfer experiments, we assumed the same number of operations available for the *ALU* and adapted data modules. The number of operations needs to be the same as these, together with the control signals from the *Input*, form the abstraction interface between data and algorithmic modules. This can be relaxed either by including the domain specific operations into the data sequence, as shown with the arithmetic transfer, or by extending the interface between *Bus* and *ALU*. The learned algorithmic solution is represented by the *Controller*, *Memory*, and *Bus*, which encode

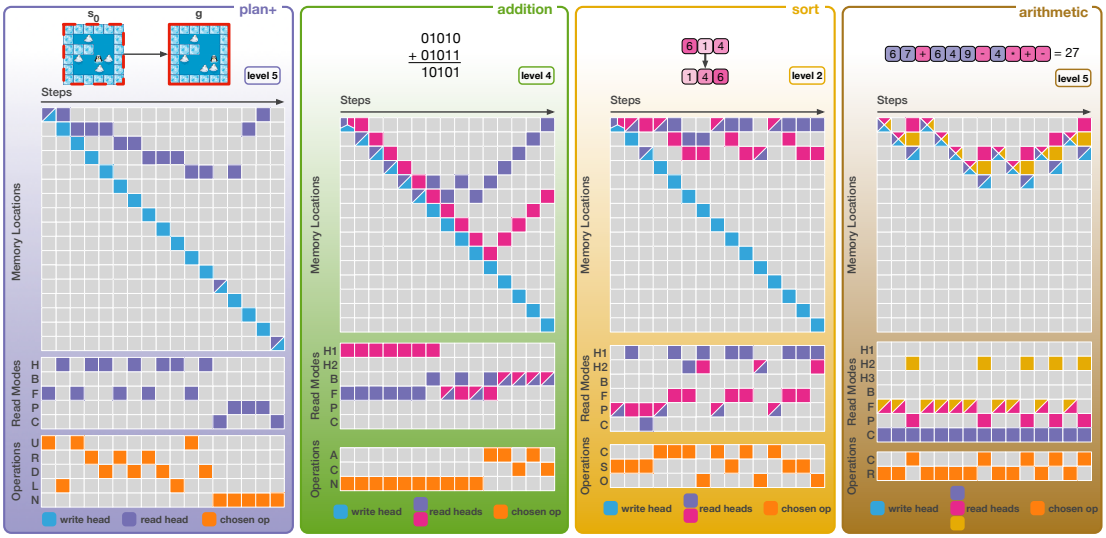


Figure 3.5.: ***Learned algorithmic behaviour of the NHC.*** *The learned behaviour for four algorithms solving the examples shown at the top is illustrated. Shown are the written and read memory locations, the used read mode for each read head, and the operation signal sent to the ALU. First an example from the (plan+) task. The algorithm first builds the search tree by applying all applicable operations in a state and then shifts reading to the next state until the goal is found, then it backtracks the solution. Next an example from the (addition) task. First, the two numbers to be added are presented after each other and are just stored. Then the two numbers are traversed from the low to the high end in parallel, adding the corresponding digits including possible carry bits. Next an example from the (sort) task. After reading the unsorted list, the algorithm iterates over the list, finding and outputting the smallest element in each iteration. Lastly an example from the (arithmetic) task. In the arithmetic task, the free gates were activated and the model learned to reuse memory locations in order to emulate the behaviour of a stack. The read heads always keep track of the head of the stack and when an arithmetic operation should be applied, it pops the two top elements from the stack, which are then combined by the ALU according to the read operation.*

the abstract strategies fulfilling R1-R3, building on the data modules implementing the abstract interface. As the data modules are domain and representation dependent, they need to be relearned or handcrafted for new setups. Typically learning these modules is less complex than learning a new algorithm as they solve smaller subproblems (and often can be hardcoded), and is a benefit of the modular architecture with its abstraction mechanism and the evolutionary training.

3.4. Conclusion

A major challenge for intelligent artificial agents is to learn strategies that scale to higher complexities and that can be transferred to new problem instantiations. We presented a modular architecture for representing and learning such algorithmic solutions that fulfil the three introduced algorithmic requirements: generalization and scaling to arbitrary task configurations and complexities (R1), as well as independence from both, the data representation (R2) and the task domain (R3). Algorithmic solutions fulfilling R1 – R3 represent strategies that generalize, scale, and can be transferred to novel problem instantiations, providing a promising building block for intelligent behaviour.

On a diverse set of 11 algorithms with varying complexities, the proposed NHC was able to reliably learn such algorithmic solutions. These solutions were successfully tested on complexities far beyond seen during training, involving up to over $1\ million$ recurrent computational steps without a single bit error, and were transferred to novel data representations and task domains. Experimental results highlight the importance of the employed abstraction mechanisms, supporting the ablation study results of prior work [9], providing a potential building block for intelligent agents to be incorporated in other models.

Discussion The modular structure and the information flow of the NHC enable the learning and transfer of algorithmic solutions, and the incorporation of prior knowledge. Using NES for learning removes constraints on the modules, allowing for arbitrary instantiations and combinations, and the beneficial use of non-differentiable memories [63]. As the complexity and structure of the algorithmic modules need to be specified, it is an interesting road for future work to learn these in addition, utilizing recent ideas [63, 77]. To speed up computation, parallel models like the neural GPU [61] may be incorporated into the NHC architecture.

The presented work showed how algorithmic solutions with R1 – R3 can be represented and learned. Based on this foundation, a challenging and interesting research question is how such algorithms can be learned with less feedback. The usage of NES allows to provide different kinds of feedback on any connection in the architecture, and on different timescales. This opens the opportunity to discover new and unexpected strategies, novel algorithms, and may be achieved by incorporating intrinsic motivation [81, 82] to explore the space of hidden algorithmic solutions in the model.

3.5. Methods

In this section a detailed description of the NHC architecture and its modules is given, the learning procedure is described, the task specific data module instantiations are discussed, and details about the comparison methods are given.

All modules are described with their formal functionality, i.e., the input signals they receive and the output signals they produce, in the form of $Module(\text{inputs}) \longrightarrow \text{output}$. The information flow is split into data and control signals, denoted with d and c respectively. In addition to this high level description, details how the output signals are generated are given for each module.

3.5.1. The Algorithmic Modules

The algorithmic modules consists of the *Controller*, *Memory* and *Bus* module and form the core of the NHC architecture. These modules are learning the algorithmic solution on the control stream and are responsible for the data management in the memory and steer the data manipulation done by the ALU module. They share similarities with the original DNC [59], like the temporal linkage and usage vector, but with major changes, e.g., hard decisions for the heads and read modes, two coupled memories, simplified and additional attention mechanisms and more, described in detail next.

Controller

The controller module receives input from the Input and the signals read from Memory from the previous step. Additionally feedback signals from the Bus and ALU from the previous step can be activated if desired. It produces one output signal going to the Memory and Bus modules. Formally given by

$$Ctr(c_t^{i\to c},\ c_{t-1}^b,\ c_{t-1}^a,\ c_{t-1}^m) \longrightarrow c_t^c\ .$$

Here we use a single layer of size L_C to learn $c_t^c \in (-1,1)^{L_C}$ at step t, given by

$$c_t^c = \tanh(W_c x_c + b_c)\ ,$$

with $x_c = [c_t^{i\to c};\ c_{t-1}^b;\ c_{t-1}^a;\ c_{t-1}^m]$. Depending on the task to learn, the feedback signals c_{t-1}^b and c_{t-1}^a can be activated, and more complex instantiations can be used for the controller, like more layers or recurrent networks.

Memory

The memory module receives signals from the Input and the Controller and is responsible for storing and retrieving information from the two memories. Therefore it produces two output signals, a data and a control signal, give by

$$Mem(d_t^i,\ c_t^{i\to m},\ c_t^c) \longrightarrow (d_t^m, c_t^m)\ .$$

The memory module has two coupled control and data memories, M^c and M^d, which are matrices of size $N \times C$ and $N \times D$ with N locations, C the control memory word size and D the data memory word size. Multiple write and read heads can be used, where the number of write and read heads is set task dependently to h_w and h_r respectively.

Learnable Interfaces As input for all learned layers only the concatenated control signals are used, i.e., $x_m = [c_t^{i\to m};\ c_t^c]$, and the weight matrices W and biases b are the parameters that are learned.

The **write vectors** $v_t^i \in \Re^C$ at step t are the control signals that are stored in M^c via the write heads and given by

$$v_t = W_v x_m + b_v\ ,$$

with v_t split into $\{v_t^i \mid \forall i : h_w\}$ for each write head.

The **previous write vectors** $\hat{v}_t^j \in \Re^C$ at step t are the control signals that are used to update M^c and given by

$$\hat{v}_t = W_{\hat{v}} x_m + b_{\hat{v}}\ ,$$

with $\hat{v}_t$ split into $\{\hat{v}_t^j \mid \forall j : h_r\}$ for each read head.

The **previous erase vectors** $\hat{e}_t^j \in (0,1)^C$ at step t are the control signals used to erase values in M^c and given by

$$\hat{e}_t = \sigma(W_{\hat{e}} x_m + b_{\hat{e}})\ ,$$

where $\sigma(\cdot)$ is the logistic sigmoid function and $\hat{e}_t$ is split into $\{\hat{e}_t^j \mid \forall j : h_r\}$ for each read head.

The **previous write gate** $\hat{g}_t \in \{0,1\}^{h_r}$ at step t determines if the memory M^c is updated with $\hat{v}_t^j$ and $\hat{e}_t^j$, given by

$$\hat{g}_t = \mathrm{H}(W_{\hat{g}} x_m + b_{\hat{g}}) \ ,$$

where $H(\cdot)$ is the heavyside step function.

The **read modes** $m_t^j \in \{0,1\}^{h_r+4h_w}$ at step t are the control signals that determine which attention mechanism is used to read from the memory, and is given by

$$m_t = W_m x_m + b_m \ ,$$

with m_t split into $\{\text{onehot}(m_t^j) \mid \forall j : h_r\}$ and $\text{onehot}(x) = \{x_k = 1 \text{ if } x_k = \max(x) \ , \ x_k = 0 \text{ else}\}$.

The **free gates** $f_t^w \in \{0,1\}^{h_w}$ and $f_t^r \in \{0,1\}^{h_r}$ at step t determine if locations written to and read from can be freed after interaction, and are given by

$$f_t^w = \mathrm{H}(W_{f^w} x_m + b_{f^w}) \quad \text{and} \quad f_t^r = \mathrm{H}(W_{f^r} x_m + b_{f^r}) \ .$$

These are all learned parameters of the memory module that define the interfaces to manipulate the memory.

Writing and Reading Given the learned interface described before and the write w_t^i and read r_t^j head locations, information is stored and retrieved from memory as follows.

Writing v_t^i to location w_t^i in M^c at step t is done via

$$M_t^c = M_{t-1}^c \circ (E - w_t^i 1^\top) + w_t^i {v_t^i}^\top \ ,$$

where $\circ(\cdot)$ denotes element-wise multiplication and E is a matrix of ones of the same size as M^c.

Writing d_t^i to location w_t^i in M^d at step t is done via

$$M_t^d = M_{t-1}^d \circ (E - w_t^i 1^\top) + w_t^i {d_t^i}^\top \ ,$$

here E is a matrix of ones of the same size as M^d. Note that the same write location w_t^i is used to couple the control and data memories.

Updating the previously read location r_{t-1}^j in M^c is done via

$$M_t^c = M_{t-1}^c \circ (E - \hat{g}_t^j r_{t-1}^j \hat{e}_t{}^{j\top}) + \hat{g}_t^j r_{t-1}^j \hat{v}_t{}^{j\top} \ ,$$

where E is a matrix of ones of the same size as M^c. If the previous write gate $\hat{g}_t^j = 0$ no update is performed, and with $\hat{g}_t^j = 1$ the previously read location r_{t-1}^j is erased with $\hat{e}_t{}^j$ and $\hat{v}_t{}^j$ is written to it.

Reading from memory is done via the read locations r_t^j used on both memories to obtain the data and control output of the memory module via

$$d_t^{m,j} = M_t^{d\top} r_t^j \quad \text{and} \quad c_t^{m,j} = M_t^{c\top} r_t^j \ ,$$

and are concatenated for the final memory module output $d_t^m = [d_t^{m,1}; \ldots; d_t^{m,h_r}]$ and $c_t^m = [c_t^{m,1}; \ldots; c_t^{m,h_r}]$.

Next, how to obtain the head locations is described in detail.

Head Locations The write and read heads locations, $w_t^i \in \{0,1\}^N$ and $r_t^j \in \{0,1\}^N$, are hard decisions, i.e., onehot encoded vectors, where exactly one location is written to or read from respectively. To determine the write head locations w_t^i, i.e., the memory locations for writing to, a simplified dynamic memory allocation scheme from the DNC is used. It is based on a *free list* memory allocation scheme, where a linked list is used to maintain the available memory locations. Here, a usage vector $u_t \in \{0,1\}^N$ indicates which memory locations are currently used, with $u_0 = 0$ and updated in each step with the written w_t^i and read r_t^j locations via

$$u_t = u_{t-1} + (1 - f_t^{w,i}) w_t^i \quad \text{and} \quad u_t = u_{t-1}(1 - f_t^{r,j} r_t^j) \ ,$$

with the free gates $f_t^{w,i}$ and $f_t^{r,j}$ determining if the write location is marked as used and if the read location can be freed respectively. Due to this dynamic allocation scheme, the model is independent from the size of the memory, i.e., can be trained and later used with different sized memories. To obtain the write location w_t^i, the memory locations are ordered by their usage u_t, and w_t^i is set to the first entry in this list – the first unused location is used to write to.

Read head locations r_t^j are determined by the active read mode given by $m_t^j \in \{0,1\}^{h_r+4h_w}$, i.e, only one mode can be active. There are three main attentions implemented for reading from memory, *HALT*, *temporal linkage* and *ancestry linkage*. The total number of available

read modes is $h_r + 4h_w$ as *HALT* is depended on the number of read heads and both linkages can be used in two directions for each write head.

The *HALT* attentions are used to read the previously read locations again. When multiple read heads are used, each head can read its own last location or the locations from the other read heads, e.g., with three read heads, each head has three *HALT* attentions (*H1,H2,H3*).

The *temporal linkage* attention is used to read locations in the order they were written, either in forward or backward direction. This mechanism enables the architecture to retrieve sequences, or parts of sequences, in the order they were presented, or in reversed order. Here, we use a simplified version of the mechanism from the DNC. As our architecture uses hard decisions for the heads locations, the linkages can be stored more efficiently in N-dimensional vectors, in contrast to a $N \times N$ matrices in the DNC. Each temporal linkage vector $L^{T,i}$ stores the order of write locations for one write head, updated at step t via

$$L_t^{T,i} = L_{t-1}^{T,i} \circ (1 - w_{t-1}^i) + \tilde{w}_t^i w_{t-1}^i \ ,$$

where $\tilde{w}_t^i = \text{argmax}(w_t^i)$. The temporal linkage mechanism can be used in two directions. Either move the read head in the order of which the locations were written, or in reversed order – resulting in two read modes, backward (B) and forward (F), per write head for each read head, given by

$$\begin{aligned} B : r_t^j &= \text{I}(L_t^{T,i} \ , \ \tilde{r}_{t-1}^j) \quad \text{and} \\ F : r_t^j &= \text{onehot}(L_t^{T,i} \circ r_{t-1}^j) \ , \end{aligned}$$

where r_{t-1}^i is the previously read location, $\tilde{r}_{t-1}^j = \text{argmax}(r_{t-1}^i)$ and $\text{I}(x, y) = \{x_k = 1 \text{ if } x_k = y \ , \ x_k = 0 \text{ else}\}$. When a location is freed through the free gates, the location is removed from the linkage such that it remains a linked list.

The *ancestry linkage* also uses N-dimensional vectors to store relations between memory locations. While the temporal linkage stores information about the order of which locations were written to, the ancestry linkage stores information about which memory locations were read before a location was written – captures a form of *usage* or hierarchical relation instead of temporal relation. Each ancestry linkage vector $L^{A,i,j}$ stores which location r_{t-1}^j was read before location w_t^i was written, and is updated at step t via

$$L_t^{A,i,j} = L_{t-1}^{A,i,j} \circ (1 - w_t^i) + \tilde{r}_{t-1}^j w_t^i \ ,$$

where r_{t-1}^j is the previously read location and $\tilde{r}_{t-1}^j = \text{argmax}(r_{t-1}^j)$. The ancestry linkage mechanism can also be used in two directions, to either move the read head to *parent* (P) location or the *child* (C) location. This results in two modes per write head for each read head, given by

$$P : r_t^j = \text{onehot}(L_t^{A,i,j} \circ r_{t-1}^j) \quad \text{and}$$
$$C : r_t^j = \text{onehot}(\text{I}(L_t^{A,i,j}\ ,\ \tilde{r}_{t-1}^j) \circ \text{h}_t)\ ,$$

where h_t is a N-dimensional vector storing for each location the step t when it was written. A location can be read multiple times, thus it can have multiple *children*. But as we need a single location to read, the C mode returns the location that was written to the latest when r_{t-1}^j was read, i.e., the newest child. This is implemented with the history vector h_t. When a location is freed through the free gates, the location is removed from the linkage and its children are attached to its parent.

Bus

The Bus module is responsible to generate the control signal that indicates how the ALU module should manipulate the data stream, i.e., which action or operation to perform. Therefore, it receives the control signal from the Controller and the Input as well as the output from the memory signal, given by

$$Bus(c_t^{i\to b},\ c_t^c,\ c_t^m) \longrightarrow c_t^b\ .$$

Here we use a single layer of size L_B to learn $c_t^b \in \{0,1\}^{L_B}$ at step t, given by

$$c_t^b = \text{onehot}(W_b x_b + b_b)\ ,$$

with $x_b = [c_t^{i\to b};\ c_t^c;\ c_t^m]$.

3.5.2. Learning Procedure

Learning the algorithmic modules, and hence the algorithmic solution, is done using Natural Evolution Strategies (NES) [80]. NES is a blackbox optimizer that does not require differentiable models, giving more freedom to the model design, e.g., the hard attention mechanisms are not differentiable and the data modules can be instantiated arbitrarily.

Recent research showed that NES and related approaches like Random Search [83] or NEAT [84] are powerful alternatives to gradient based optimization in reinforcement learning. They are easier to implement and scale, perform better with sparse rewards and credit assignment over long time scales, have fewer hyperparameters [85] and were used to train memory-augmented networks [63, 77, 78].

NES updates a search distribution of the parameters to be learned by following the natural gradient towards regions of higher fitness using a population P of offspring o (altered parameters) for exploration. The performance of an offspring o is measured with one scalar value summarized over all samples N in the mini-batch and over all computational steps T_{max} of each sample, with sparse binary signals for each step – framing a challenging learning problem albeit given the sequence. Let θ be the parameters to be learned – the weight matrices and biases in the three algorithmic modules $\theta = [W_c; b_c; W_v; b_v; W_{\tilde{v}}; b_{\tilde{v}}; W_{\tilde{e}}; b_{\tilde{e}}; W_{\tilde{g}}; b_{\tilde{g}}; W_m; b_m; W_{f^w}; b_{f^w}; W_{f^r};$
$b_{f^r}; W_b; b_b]$ – and using an isotropic multivariate Gaussian search distribution with fixed variance σ^2, the stochastic natural gradient at iteration t is given by

$$\nabla_{\theta_t} \mathbb{E}_{\epsilon \sim N(0,I)} \left[u(\theta_t + \sigma\epsilon) \right] \approx \frac{1}{P\sigma} \sum_{o=1}^{P} u(\theta_t^o)\epsilon_i \ ,$$

where P is the population size and $u(\cdot)$ is the rank transformed fitness $f(\cdot)$ [80]. With a learning rate α, the parameters are updated at iteration t by

$$\theta_{t+1} = \theta_t + \frac{\alpha}{P\sigma} \sum_{o=1}^{P} u(\theta_t^o)\epsilon_i \ .$$

For all experiments the fitness function is defined for N samples as $f(\theta_t^o) = 1/N \sum_s^N f_s(\theta_t^o)$ with

$$f_s(\theta_t^o) = \frac{1}{T_{\text{max}}} \sum_{k=1}^{T_e} \delta(d_k^m - \bar{d}_k^m) + \delta(c_k^b - \bar{c}_k^b) m(c_k^b)$$

to evaluate the offspring parameters θ_t^o on one sample s. Here, $\delta(x) = \{1 \text{ if } x = 0, 0 \text{ else}\}$ gives sparse binary reward if the two signals are equal or not, where d_k^m is the data output from the memory, c_k^b the control output from the Bus, and $\bar{d}_k^m$ and $\bar{c}_k^b$ the true values respectively. Thus, reward is given for choosing the correct data and operation for the ALU in each step. Note, there is no feedback on memory access, only on the output, i.e., where, when and how to write and read has to be learned without explicit feedback. The stepwise signals are summed up until the first mistake occurs (T_e) or until the maximum

length of the sample T_{max}, and is normalized with $1/T_{\text{max}}$, i.e., $f(\theta_t^o)$ measures the fraction of subsequently correct algorithmic steps. To encourage strong operation choices, the operation reward is multiplied with the margin penalty

$$m(c_k^b) = clip\Big(\frac{\tilde{c}^1/\tilde{c}^2 - 1}{m_{\text{max}}}\ ,\ 0\ ,\ 1\Big)\ ,$$

where $\tilde{c}^1, \tilde{c}^2$ are first and second largest values of c_k^b, i.e., the chosen operation and the runner up, and m_{max} is a chosen percentage indicating how much bigger the chosen action should be. Note that this penalty is only considered if the operation is already correct.

For robustness and learning efficiency, weight decay for regularization [86] and automatic restarts of runs stuck in local optima are used [80]. This restarting can be seen as another level of evolution, where some lineages die out. Another way of dealing with early converged or stuck lineages is to add intrinsic motivation signals like novelty, that help to get attracted by another local optima, as in NSRA-ES [87]. In the experiments however, we found that within our setting, restarting – or having an additional *survival of the fittest* on the lineages – was more effective in terms of training time.

The algorithmic solutions are learned in a curriculum learning setup [79] with sampling from old lessons to prevent unlearning and to foster generalization. Furthermore, we created *bad memories*, a learning from mistakes strategy, similar to the idea of AdaBoost [88], which samples previously failed samples to encourage focusing on the hard cases. This can also be seen as a form of experience replay [47, 89], but only using the initial input data to the model, not the full generated sequences. Bad memories were initially developed for training the data-dependent modules to ensure their robustness and 100% accuracy, which is crucial to learn algorithmic solutions. If the individual modules do not have 100% accuracy, no stable algorithmic solution can be learned even if the algorithmic modules are doing the correct computations. For example, if one module has an accuracy of 99%, the 1% error prevents learning an algorithmic solution that works *always*. This problem is even reinforced as the proposed model is an output-input architecture that works over multiple computation steps using its own output as the new input – meaning the overall accuracy drops to 36.6% for 100 computation steps. Therefore using the bad memories strategy, and thus focusing on the mistakes, helps significantly in achieving robust results when learning the modules, enabling the learning of algorithmic solutions.

Experimental Setup In all experiments, the hyperparameters were set to: batch size $N = 32$, population size $P = 20$, learning rate $\alpha = 0.01$, search distribution exploration

$\sigma = 0.1$, weight decay $\lambda = 0.9995$, action margin $m_{\max} = 0.1$, max iterations = 20.000, restart iterations = 2.000. In each batch, 33% of the samples were drawn from previous levels and another 33% were drawn from the bad memories buffer, which stores the last 200 mistakes. A curriculum level is considered solved when 750 subsequent iteration are perfectly solved, i.e., no single mistake in any sample, any step, any bit, that is 24.000 perfectly solved samples. In levels were training was triggered, the required subsequent perfect iterations are doubled, i.e., 48.000 perfectly solved samples. Whenever an iteration achieves maximum fitness, no learning is triggered, i.e., no parameter update is performed.

3.5.3. The Modules Instantiations

The preceding sections described the design and functionality of the algorithmic modules in general. Here, the used instantiations and parameters for the experiments are presented as well as the data modules and their task-dependent instantiations.

Algorithmic Modules In all experiments, the Controller size was set to $L_C = 6$ and the control memory word size $C = 4$. All tasks use one write head, $h_w = 1$, and the number of read heads is $h_r = 1$ for the four search & plan, and the four copy tasks, $h_r = 2$ for addition and sort and $h_r = 3$ for the arithmetic task. The data memory word size D and the Bus size L_B are set by the task, as each task has a different data representation (D) and a different amount of available operations for the ALU (L_B). In all tasks the ALU to Controller feedback (c^a_{t-1}) was activated, except for the four copy tasks as the ALU has no functionality there. The free gates were activated for the four copy tasks and the arithmetic task. In total, depending on the algorithm to learn, this results in 300-650 trainable parameters in the algorithmic modules.

Input The first data dependent module is the Input module. That is the interface to receive data and provide control signals. It receives the data input from the outside d^{in}_t as well as the data output from the ALU from the previous computational step d^{out}_{t-1}, formally given as

$$In(d^{in}_t,\ d^{out}_{t-1}) \longrightarrow (d^{i}_t,\ c^{i\to c}_t,\ c^{i\to m}_t,\ c^{i\to b}_t)\ .$$

The main functionality is to generate task related control signals, data preprocessing if applicable and determining to stop. The control signals $c^{i\to c}_t, c^{i\to m}_t, c^{i\to b}_t$ can be different to provide different signals to the Controller, Memory and Bus, but can also share the

same information. The data d_t^i is forwarded to the memory module with or without preprocessing, depending on the task.

ALU The arithmetic logic unit (ALU) module is responsible for data manipulation. It receives the read data from the memory and the operation to apply on these from the Bus to produce the next data output alongside control signals via

$$ALU(d_t^m,\ c_t^b) \longrightarrow (d_t^{out}, c_t^a)\ .$$

This module implements elemental operations for each task such that the algorithmic solution can be learned by applying the correct operation on the correct data in the correct step.

Both data modules can be instantiated arbitrarily due to the NES approach for learning the algorithmic solution. They can also be trained from data beforehand or be hardcoded if possible. In the experiments, we tested both variations and details for each algorithm are given in the next section.

3.6. Algorithms to Learn

In this section additional information for the different algorithms that are learned and the modules instantiations are given.

3.6.1. Search & Plan

In the Search & Plan tasks, the goal is to reach a given goal from a starting state (search), and to generate a path between them utilizing the search result (plan). Therefore, a breadth-first-search algorithm with additionally backtracking is learned. Given a start and goal state (s_0 and g), the model has to learn to implicitly build a search tree by applying all available actions on a state and then move on to the next unexplored state until the goal state is reached. For the planning tasks, after reaching the goal state, the model has to output the sequence of states from goal to start, encoding the sequence of actions to solve the given planning problem. For the extended versions (search+ and plan+), the available actions for each state are state dependent such that only actions that have an effect in this state should be applied. This increases the complexity of the learned

algorithm, but the resulting search trees after learning are smaller and, hence, search+ and plan+ more efficient.

As training domain, the gridworld game sokoban is used, where an agent can move in four directions – move up (U), right (R), down (D), left (L) – and an additional `nop` operation(N) that leaves the data unchanged, resulting in 5 operations. The world consists of empty spaces that can be entered, walls that block movement and boxes that can be pushed onto adjacent spaces. Figure 3.5 shows a learned solution, where the agent is visualized as a penguin, empty space is water, boxes are icebergs and walls are ice floes.

The curriculum level complexity is the number of fully explored nodes, i.e., for level 3, three nodes have to be explored. See Figure 3.2 for an example for the extended version from level 3 alongside the pseudocode of the algorithm to learn.

For learning, we use a sokoban world of size 6×6 that is enclosed by walls. A world is represented with binary vectors and four-dimensional one-hot encodings for each position, resulting in 144-dimensional data words, and thus $D = 144$. The configuration of each world – inner walls, boxes and agent position – is sampled randomly. Each world is generated by sampling uniformly the number of inner walls from $[0, 2]$ and boxes from $[1, 5]$. The positions of these walls, boxes and the position of the agent are sampled uniformly from the empty spaces.

Input The Input module produces control signals indicating in which phase the algorithm is – building the search tree, goal reached and backtracking the solution. The signal is created as $c_t^p = [c_t^{[1]}; c_t^{[2]}; c_t^{[3]}; c_t^{[4]}]$, with

$$
\begin{aligned}
c_t^{[1]} &= \max(0, (1 - E_t) - c_{t-1}^{[2]}) \ , \\
c_t^{[2]} &= \min(1, E_t + c_{t-1}^{[2]}) \ , \\
c_t^{[3]} &= (1 - E_t) p_t^{[2]} \ , \\
c_t^{[4]} &= E_t p_{t-1}^{[2]} \ .
\end{aligned}
$$

The signal E_t is a learned equality function using differential rectifier units as inductive bias [90] and consists of a feedforward network with 10 hidden units and `leaky-ReLU` activation, trained in a supervised setting with cross-entropy loss. This signal indicates if two given states are equal and is used to identify the goal and initial state. The four signals $c_t^{[x]}$ are created using this information and represent the state of the algorithms, where $c^{[1]} = 1$ during building the search tree, $c^{[2]} = 1$ when the goal was found, $c^{[3]} = 1$

during the backtracking, and $c^{[4]} = 1$ when backtracking reached the initial state. The algorithm stops if $c_t^{[4]} = 1$ for the planning tasks, and if $c_t^{[2]} = 1$ for the search tasks, as search only uses the two first signals.

For the extended search and plan tasks, the Input module additionally learned an action mask a. This binary action mask indicates which operations are applicable in a given state and which not, and is learned with a feedforward network with 256 hidden units and `leaky-ReLU` activation.

Using these learned signals, the outputs of the Input module are given by

$$\begin{aligned} d_t^i &= d_t^{in} \quad \text{if} \quad t = 0 \ , \quad d_{t-1}^{out} \quad \text{else} \ , \\ c_t^{i \to c} &= c_t^p \ , \\ c_t^{i \to b} &= \ c_t^{i \to c}, \\ c_t^{i \to m} &= c_t^{i \to c} \quad \text{or} \quad c_t^{i \to m} = [a; c_t^p] \ \text{(extended)} \ . \end{aligned}$$

ALU The ALU learns to apply the available operations on the data, i.e., it encodes an action model A by learning preconditions and effects, and outputs the new data together. There are four available operations in sokoban and the sliding block puzzle domain, i.e., move up (U), right (R), down (D), left (L), and a fifth `nop` operation is added that does not change the data (N). In the robotic manipulation domain, the available actions are the four locations on which objects can be stacked, e.g., the action pos1 encodes to move the gripper to the position and place the grasped object on top, or to pick up the top object if no object is grasped. The maximum stacking height is 3 boxes, resulting in a discrete representation of the object configuration with a 3×4 grid. Thus, the ALU for the search and plan algorithms has five operations and, hence, $L_B = 5$.

For this task, the ALU internally uses three submodules – one does dimensional reduction on the data, one applies the operation to manipulate the lower-dimensional data, and one combines the manipulated and original data for the final output. Each submodule consists of feedforward networks with $1 : 500$, $2 : ([128, 64], [64, 64])$, $3 : ([500, 500], [500, 250])$ hidden units to produce the data and control stream respectively. All networks use `leaky-ReLU` activation and are trained with cross-entropy loss in a supervised setting. The output of the ALU is therefore defined as

$$\begin{aligned} d_t^{out} &= A(d_t^m, \ c_t^b) \ , \\ c_t^a &= [l, 1 - l, n] \ , \end{aligned}$$

where d_t^m is the read state, c_t^b the operation to apply, $l = 1$ if the applied operation was the last for this state, and $n = 1$ if the nop operation was used.

A more detailed description of these submodules and the search & plan specific data modules can be found in the predecessor model [9] used for symbolic planning tasks.

3.6.2. Addition

In the addition task, two numbers have to be added. The two numbers a, b are presented subsequently in big-endian order. To solve the task, the model has to learn to read the two numbers correctly aligned, add the corresponding bits and remember the carry for the next step. Therefore, 3 operations are available, adding two given bits without (A) and with carry bit (C), and a nop operation (N).

Curriculum level complexities are defined as the bit length of the numbers a, b, i.e., in level 3 numbers of length 3 have to be added, and a, b have an additional leading 0. For training, a, b were randomly generated binary numbers and thus $D = 1$.

Input The Input module produces control signals indicating if a or b is presented, or if addition should be done. Thus, the module output is given by

$$d_t^i = d_t^{in} \quad \text{if} \quad t \leq 2T_l \ , \quad 0 \quad \text{else} \ ,$$
$$c_t^{i \to c} = c_t^{i \to m} = c_t^{i \to b} = [c_a; c_b; c_s] \ ,$$

where T_l is the the length of a and b, $c_a = 1$ and $c_b = 1$ when a or b are presented respectively, and $c_s = 1$ during the addition phase. The algorithm stops after T_l steps with $c_s = 1$.

ALU There are 3 operations for the ALU module here, adding two given bits without (A) and with carry bit (C), and a nop operation (N), and hence, $L_B = 3$. The data input consists of two bits $d_t^m = [v_1; v_2]$, read from the memory with the two read heads. The outputs of the ALU are given by

$$d_t^{out} = v_1 + v_2 + carry \ ,$$
$$c_t^a = [c, 1 - c, n] \ ,$$

where $carry = 1$ if the operation C is chosen, $c = 1$ if the current operation produced a carry bit, and $n = 1$ if the nop operation was used.

3.6.3. Sort

In the sort task the model is given an unordered list of objects and has to output the objects in order. There are 3 operations, comparing two objects (C), skipping the current objects (S), and outputting one object (O). To solve the task, the model has to iterate over the sequence of objects and output the *smallest* object in each iteration. Note, it does not need to be the smallest, if the sequence should be ordered in descending order for example, it outputs the largest. The order is defined by the compare operation implemented in the ALU module, the learned algorithm can therefore order any sequence in any order.

Curriculum level complexities are defined as the length of the sequence to order, e.g., in level 3 sequences of length 4 have to be sorted. For training, the sequences consist of randomly generated 8 bit binary numbers ($D = 8$) and the ALU uses $lessEqual$ as $compare$ function.

Input The Input module generates control signals indicating if the unordered sequence is presented or if sorting should be done. Thus, the module output is given by

$$d_t^i = d_t^{in} \quad \text{if} \quad t \leq T_s \ , \quad 0 \quad \text{else} \ ,$$
$$c_t^{i \to c} = c_t^{i \to m} = c_t^{i \to b} = [c_f; c_e; c_l; c_s] \ ,$$

where T_s is the length of the sequence, $c_f = 1$ if the first object in the sequence is presented, $c_l = 1$ for the last object, $c_e = 1$ for all other objects, and $c_s = 1$ during the sort phase. The algorithm stops if the output operation was used T_s times during the sort phase.

ALU The ALU has 3 operations, a compare operation (C) that compares two objects, a skip operation (S), and a output operation (O) to mark one object for output, so $L_B = 3$. The data input consists of two objects $d_t^m = [o_1; o_2]$, read from the memory with the two read heads. The outputs of the ALU are given by

$$d_t^{out} = o_1 \ ,$$
$$c_t^a = [c, 1 - c, s, o] \ ,$$

where $c = 1$ if $compare(o_1, o_2) = true$, $s = 1$ if the skip operation was used, and $o = 1$ if the output operation was used. For training, sequences consisted of binary numbers and $compare(o_1, o_2) = o_1 \leq o_2$, i.e., sorting numbers in ascending order.

3.6.4. Arithmetic

In the arithmetic task the model is given a sequence that encodes an arithmetic expression in postfix notation, e.g., $3 + 5 * 6$ is presented as $5\ 6 * 3+$. There are 2 operations, a calculation operation (C) that calculates a given atomic operation, and a read operation (R). To solve the task, the model has to essentially learn to emulate a stack. It iterates over the input sequence and learns that numbers need to be stored on the stack, and if an atomic operation is presented, takes the two most recent numbers from the stack to combines them accordingly until the input sequence has finished. As the atomic operations $[+, -, *, /]$ are part of the input sequence, the solution is independent from the amount of different atomic operations, and the ALU has the two described operations.

For training, the sequences consist of arithmetic expression with the four atomic operations $[+, -, *, /]$, where a modulo 10.000 operation was applied to atomic results for numeric stability, and numbers in the input sequence were drawn from $[1, 10]$, and hence $D = 1$. Training sequences are generated randomly, with uniformly sampled atomic operations and numbers. Curriculum level complexities are defined as the number of atomic operations, i.e., $5\ 6 * 3+$ is an example for level 2.

Input The Input module provides controls signals indicating if the current data word is a value or an atomic operation. Therefore, the module outputs are given by

$$\begin{aligned} d_t^i &= d_t^{in} \quad \text{if} \quad c_{t-1}^{i\to c} = [1, 0]\ , \quad d_{t-1}^{out} \quad \text{else}\ , \\ c_t^{i\to c} &= c_t^{i\to m} = c_t^{i\to b} = [c_v; c_a]\ , \end{aligned}$$

where $c_v = 1$ when d_t^i is a value (e.g., a number), and $c_a = 1$ is d_t^i is an atomic operation. The algorithm stops if the last atomic operation was presented.

ALU The ALU module has 2 operations, a calculation operation (C) that calculates a given atomic operation, and a read operation (R), and hence $L_B = 2$. The data input consists of three values $d_t^m = [d_1; d_2; d_3]$, read from the memory with the three read heads. The module outputs are then given by

$$\begin{aligned} d_t^{out} &= d_1(d_2, d_3) \quad \text{if} \quad C\ , \quad d_2 \quad \text{else}\ , \\ c_t^a &= c_t^b\ , \end{aligned}$$

where c_t^b is the signal coming from the Bus module, indicating which operation to apply, i.e., here $c_t^b = [C; R]$ with $C = 1$ is an atomic operation should be used, or $R = 1$ if not. If $C = 1$, the first data word d_1 read from the memory is interpreted as the atomic operation and is applied on d_2 and d_3, e.g., if $d_t^m = [+; 4; 5]$ then $d_t^{out} = d_1(d_2, d_3) = 4 + 5 = 9$. The four atomic operations $[+, -, *, /]$ are encoded as $[-1, -2, -3, -4]$ in the input, as the input sequence can only contain positive numbers in the used setup – the learned algorithm is independent from that choice.

3.6.5. Copy, RepeatCopy, Reverse, Duplicated

In the four baseline tasks – Copy, RepeatCopy, Reverse, Duplicated – there is no data manipulation. For solving these tasks, the model has to learn the proper data management. There are 2 ALU operations to *mark* the data (O and M), which do not alter the data.

In the *copy* task, the model is presented a sequence of objects L and has to output the same sequence of objects, i.e., $L = [x_1, \ldots, x_n] \longrightarrow [x_1, \ldots, x_n]$. Therefore, it needs to learn to iterate over the data in the presented order.

In the *repeatCopy* task, the model is also presented a sequence of objects L and has to output the sequence c times, i.e., $L = [x_1, \ldots, x_n], c \longrightarrow [x_1, \ldots, x_n, x_1, \ldots, x_n, \ldots]$. Here, it needs to learn to iterate multiple times over the data, requiring to jump back to the start of the sequence.

In the *reverse* task, the model is presented a sequence of objects L and has to output the sequence in reversed order, i.e., $L = [x_1, \ldots, x_n] \longrightarrow [x_n, \ldots, x_1]$. Solving requires to learn to iterate over the data in reversed order of presentation.

In a remove duplicates (*duplicated*) task, the model is presented a sequence of objects L with duplicates of each object and has to output the sequence without these duplicates, i.e., $L = [x_1, x_1, x_1, x_2, x_2, x_2 \ldots, x_n, x_n, x_n] \longrightarrow [x_1, \ldots, x_n]$. In order to solve this task, the model has to learn during the presentation of the input sequence which data *to ignore*, while getting only feedback for outputting the sequence without duplicates.

Curriculum level complexities are defined as the number of objects in the sequence, and as the number of copies or duplicates for the repeatCopy and duplicated tasks respectively. For training, objects were random binary vectors of length 6 ($D = 6$).

Input The Input module provides control signals indicating if objects are presented or output should be done. For the copy, repeatCopy and reverse tasks, the modules outputs are given by

$$d_t^i = d_t^{in} \quad \text{if} \quad t \leq T_L \ , \quad 0 \quad \text{else} \ ,$$
$$c_t^{i \to c} = c_t^{i \to m} = c_t^{i \to b} = [c_f; c_i; c_l; c_o] \ ,$$

where T_L is the length of the sequence L, $c_f = 1$ if the first object is presented, $c_l = 1$ for the last object, $c_i = 1$ for the remaining objects, and $c_o = 1$ indicating the output phase. The algorithm stops, if the M action was used cT_l times in the output phase, with $c = 1$ for copy and reverse.

For the duplicated task, the outputs are given by

$$d_t^i = d_t^{in} \quad \text{if} \quad t \leq T_L \ , \quad 0 \quad \text{else} \ ,$$
$$c_t^{i \to c} = c_t^{i \to m} = c_t^{i \to b} = [c_f; c_i; c_o] \ ,$$

, where T_L is the length of the sequence L including the duplicates, $c_f = 1$ when an object is presented the first time, $c_i = 1$ for the remaining times, and $c_o = 1$ indicating the output phase. The algorithm stops, if the M action was used in the output phase.

ALU The ALU has 2 operations, an output operation (O) to mark data for output, and a mark operation (M), to mark an output as the last, thus $L_B = 2$. In these four baseline task, these operations signals are only indicating what the algorithm is currently doing, but the ALU has no functionality, i.e., there is no data manipulation. Hence, the output is just the forwarded input, given by

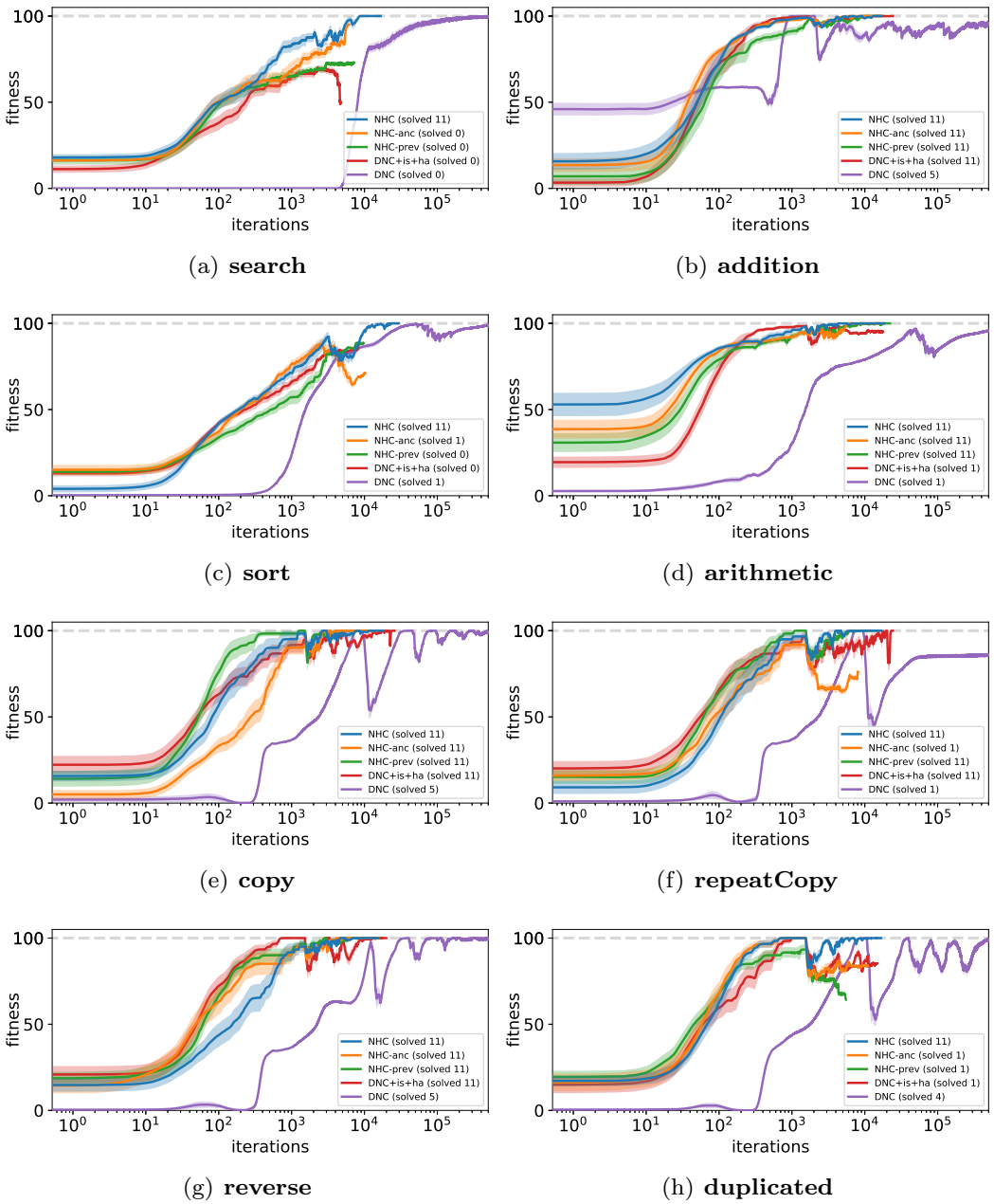
$$d_t^{out} = d_t^m \ ,$$
$$c_t^a = c_t^b \ .$$

3.7. Details to the Comparison Methods

For comparison we used the original Differential Neural Computer [59] (DNC), trained in a supervised setting with gradient descent using the Adam [38] optimizer and cross-entropy loss for each step. The loss is computed based on the correct algorithmic sequences created by the linear layer in the DNC, similar like the fitness function for our architecture, but

the cross-entropy loss provides a much richer and localized learning signal. The DNC is trained for considerable more iterations ($500k$) to counter the pretrained data modules. It also receives its own output as Input and the data input is managed equally as in our model for each task (see Input module descriptions). For the repeatCopy task, as in the original implementation, the number of copies c is normalized in the input. The controller network is a LSTM [91] network with 64 hidden units except for the Search task, where it has 256 hidden units to counter the additional data modules. In the arithmetic task, the modulo 10.000 operator for intermediate results is replaced with a modulo 10 operator and the numbers are binary encoded with 4 bits. This is done as dealing with decimal input adds an additional challenge and the reduction of the range of the numbers lets the DNC focus on the algorithmic structure of the task, instead of data encoding related issues. The same bad memories strategy and curriculum schedule as for the NHC are used. The memory size as well as the number of write and read heads is set to the same values as in the NHC for each task.

As second comparison method, we integrated the DNC into the NHC architecture, named DNC+is+ha. Therefore, we replaced the NHC algorithmic modules – Controller, Memory and Bus – with the original DNC. To enable this, the information split including the second memory was added to the DNC (+is), and the memory access was changed to hard attentions (+ha), i.e., each head writes and reads one memory location in each step, in contrast to the soft attention and weighted averaged readouts in the DNC. This is enabled by transforming the computed soft attention heads from the DNC just before memory access into hard attention vectors. For training this DNC+is+ha model, the exact same learning procedure and parameters were used as for the NHC model.



(a) **search** (b) **addition**

(c) **sort** (d) **arithmetic**

(e) **copy** (f) **repeatCopy**

(g) **reverse** (h) **duplicated**

Figure 3.6.: ***Learning curves comparison.*** *Shown are the mean and the standard error of the fitness during learning over 15 runs. Note the log-scale of the x-axis. Solved X in the legend indicates the median solved level. The full NHC is the only model that successfully learns all algorithms reliably. More details on these evaluations are given in Table 3.1.*

4. Efficient Online Adaptation in Stochastic Recurrent Networks

As a physical intelligent autonomous agent has to interact in a complex environment, the real world, it has to have physical adaptation skills. While the previous two chapters mainly focused on cognitive abilities for intelligent behaviour, this chapter tackles the question of physical interaction, related to the topic *Adapt*. In detail, how can a robot efficiently adapt its movement during execution with a bio-inspired stochastic network?

4.1. Introduction

One of the major challenges in robotics is the concept of developmental robots [92–94], i.e., robots that develop and adapt autonomously through lifelong-learning [95–97]. Although a lot of research has been done for learning tasks autonomously in recent years, experts with domain knowledge are still required in many setups to define and guide the learning problem, e.g., for reward shaping, for providing demonstrations or for defining the tasks that should be learned. In a fully autonomous self-adaptive robot however, these procedures should be carried out by the robot itself. In other words, the robot and especially its development should not be limited by the learning task specified by the expert, but should rather be able to develop *on its own*. Thus, the robot should be equipped with mechanisms enabling autonomous development to *understand* and decide when, what, and how to learn [98, 99].

Furthermore, as almost all robotic tasks involve movements and therefore movement planning, this developing process should be continuous. In particular, planning a movement, executing it, and learning from the results should be integrated in a continuous online framework. This idea is investigated in iterative learning control approaches [100, 101], which can be seen as a simple adaptation mechanism that learns to track given

repetitive reference trajectories. More complex adaptation strategies are investigated in model-predictive control approaches [102–105] that simultaneously plan, execute and re-plan motor commands. However, the used models are fixed and cannot adapt straightforwardly to new challenges.

Online learning with real robots was investigated in [106], where multiple models were learned online for reaching tasks. Online learning of push recovery actions during walking in a humanoid robot was shown in [107], and in [108] a mechanism for online learning of the body structure of a humanoid robot was discussed. Recurrent neural networks were used to learn body mappings in a humanoid robot [109], and for efficient online learning of feedback controllers [110]. However, in all these online learning settings, the learning problem was designed and specified a priori by a human expert, providing extrinsic reward.

From autonomous mental development in humans however, it is known that intrinsic motivation is a strong factor for learning [111, 112]. Furthermore, intrinsically motivated behavior is crucial for gaining the competence, i.e., a set of reusable skills, to enable autonomy [113]. Therefore, the abstract concept of intrinsically motivated learning has inspired many studies in artificial and robotic systems, e.g. [114–116], which investigate intrinsically motivated learning in the reinforcement learning framework [117]. Typically, such systems learn the consequences of actions and choose the action that maximizes a novelty or prediction related reward signal [118–120].

Intrinsic motivation is used for self-generating reward signals that are able to guide the learning process without an extrinsic reward that has to be manually defined and provided by an expert. For the concept of lifelong-learning, intrinsic motivation signals are typically used for incremental learning within hierarchical reinforcement learning [121] and the options framework [122]. Starting with a developmental phase, the robots learn incrementally more complex tasks utilizing the previously and autonomously learned skills. Furthermore, the majority of related work on intrinsically motivated learning focuses on concepts and simulations, and only few applications to real robotic systems exist, for example [123, 124].

Contribution The contribution of this work is a neural-based framework for robot control that enables efficient online adaptation during motion planning tasks. A novel intrinsically motivated *local* learning signal is derived and combined with an experience replay strategy to enable efficient online adaptation. We implement the adaptation approach into a biologically inspired stochastic recurrent neural network for motion planning [11, 125]. This

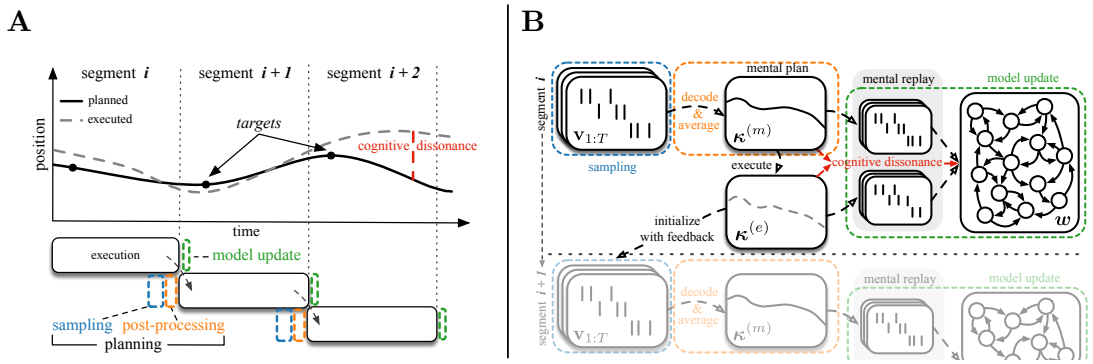


Figure 4.1.: ***Conceptual sketch of the framework.*** ***A*** *shows the online planning and adaptation concept of using short segments. On the upper part the idea of cognitive dissonance is illustrated with a planned and executed trajectory. The steps sampling and post-processing for a segment are timed such that they are performed during the end of the execution of the previous segment, whereas model adaptation is performed at the beginning of the segment execution.* ***B*** *shows the process with two segments in detail, including sampling of movements, decoding and averaging for creating the mental plan and the model update. The executed segment provides feedback for planning the next segment and the matching mental and executed trajectory pairs are used for updating the model based on their cognitive dissonance.*

work builds on recent prior studies where a *global* learning signal was investigated [12, 13]. These global and local learning signals enable efficient task-independent online adaptation without an explicit specified objective or learning task. In robotic experiments we evaluate and compare these global and local learning signals and discuss their properties. This study shows that our framework is suitable for model based robot control tasks where adaptation of the state transition model to dynamically changing environmental conditions is necessary.

The task-independent online adaptation is done by updating the recurrent synaptic weights encoding the state transition model. The proposed learning principle, therefore, can be applied to model-based (control) approaches with internal (transition) models, like, for example, (stochastic) optimal control [126–128] and model-predictive control [102–105]. Furthermore, the method is embedded into a novel framework for continuous online motion planning and learning that combines the scheduling concept of model-predictive control with the adaptation idea of iterative learning control.

The online model adaptation mechanism uses a supervised learning approach and is modulated by intrinsic motivation signals that are inspired by cognitive dissonance [129, 130]. We use a knowledge-based model of intrinsic motivation [131] that describes the divergence of the expectation to the observation. This intrinsic motivation signal tells the agent where its model is *incorrect* and guides the adaptation of the model with this mismatch. In our experiments, this dissonance signal relates to a tracking error, however, the proposed method is more general and can be used with various modalities like vision or touch. We derive two different mechanisms to compute the dissonance, a *global* learning signal that captures the *distance* between mental and executed trajectory, and a *local* learning signal that takes the neurons *responsibilities* for encoding these trajectories into account. These learning signals trigger the online adaptation when *necessary* and guide the strength of the update.

Additionally, to intensify the effect of the experience, we use a mental replay mechanism, what has been proposed to be a fundamental concept in human learning [132]. This mental replay is implemented by exploiting the stochastic nature of the spiking neural network model and its spike encodings of trajectories to generate multiple sample encodings for every experienced situation.

We will show that the stochastic recurrent network can adapt efficiently to novel environments without specifying a learning task within seconds from few interactions by using the proposed intrinsic motivation signals and a mental replay strategy on a simulated and real robotic system (shown in Figure 4.2).

4.1.1. Related Work on Intrinsically Motivated Learning

In this subsection we discuss the related work for intrinsically motivated learning from practical and theoretical perspectives.

Early work on intrinsically motivated learning not using the typically reinforcement learning framework used the prediction error of sensory inputs for self-localization tasks [133]. In an online setup, the system explored novel and interesting stimuli to learn a representation of the environment. By using this intrinsic motivation signal, the system developed structures for perception, representation and actions in a neural network model. Actions were chosen such that the expected increase of knowledge was maximized. The approach was evaluated in a gridworld domain and on a simple mobile robot platform.

Intrinsic motivation signals prediction, familiarity (in terms of frequency of state transitions) and stability (in terms of sensor signals to its average) were investigated in [134] in task-independent online visual exploration problems in simulation and on a simple robot.

By using the hierarchical reinforcement learning framework and utilizing the intrinsic motivation signal novelty, autonomous learning of a hierarchical skill collection in a playroom simulation was shown in [114]. The novelty signal directed the agent to novel situations when it got *bored*. As already learned skills can be used as actions in new policies, the approach implements an incremental learning setup.

A similar approach was investigated in [124], were a framework for lifelong-learning was proposed. This framework learns hierarchical polices and has similarities to the options framework. By implementing a motivation signal based on affordance discovery[1], a repertoire of movement primitives for object detection and manipulation was learned on a platform with two robotic arms. The authors also showed that these primitives can be sequenced and generalized to enable more complex and robust behavior.

Another approach for lifelong-learning based on hierarchical reinforcement learning and the options framework is shown in [135]. The authors learn incrementally a collection of reusable skills in simulations, by implementing the motivation signals novelty for learning new skills and prediction error for updating existing skills.

A different approach based on competence improvement with hierarchical reinforcement learning is discussed in [136]. The agent is given a set of skills, or options as in the options framework, and needs to choose which skill to improve. The used motivation signal competence is implemented as the expected return of a skill to achieve a certain

[1] Affordance refers to the possibility of applying actions to objects or the environment.

goal. Rewards are generated based on this competence progress and the approach is evaluated in a gridworld domain.

In [123], the *intelligent adaptive curiosity* system is introduced and used to lead a robot to maximize its learning progress, i.e., guiding the robot to situations, that are neither too predictable nor too unpredictable. The reinforcement learning problem is simplified to only trying to maximize the expected reward at the next timestep and a positive reward is generated when the error of an internal predictive model decreases. Thus, the agent focuses on exploring situations whose complexity matches its current abilities. The mechanism is used on a robot that learns to manipulate objects. The idea is to equip agents with mechanisms *computing* the degree of novelty, surprise, complexity or challenge from the robots point of view and use these signals for guiding the learning.

In [120] different prediction based signals are investigated within a reinforcement learning framework on a simulated robot arm learning reaching movements. The framework uses multiple expert neural networks, one for each task, and a selection mechanism that determines which expert to train. The motivation signals are implemented with learned predictors with varying input that learn to predict the achievement of the selected task. Predicting the achievement of the task once in the beginning of a trial produced the best results.

Recently, open-ended learning systems based on intrinsic motivation increasingly give importance to explicit goals – known from the idea of goal babbling for learning inverse kinematics [137] – for autonomous learning of skills to manipulate the robots environment [138].

Beside the aforementioned more practical research, also work on theoretical aspects of intrinsic motivated learning exists. For example, a coherent theory and fundamental investigation of using intrinsic motivation in machine learning over two decades is discussed in [139]. The authors state that the improvement of prediction errors can be used as an intrinsic reinforcement for efficient learning.

Another comprehensive overview of intrinsically motivated learning systems is given in [116]. The authors introduce three classes for clustering intrinsic motivation mechanisms. In particular, they divide these mechanisms into prediction based, novelty based and competence based approaches, and discuss their features in detail. Furthermore, that prediction based and novelty based intrinsic motivations are subject to distinct mechanisms was shown in [140].

In [131] a psychological view on intrinsic motivation is discussed and a formal typology of computational approaches for studying such learning systems is presented.

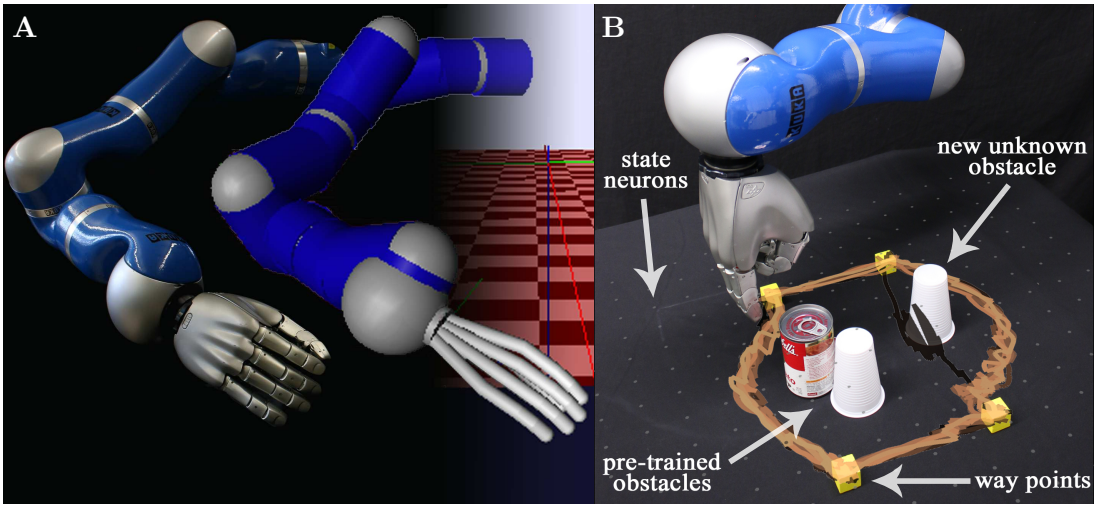


Figure 4.2.: ***Experimental setup.*** ***A*** *shows the KUKA LWR arm (left) and its realistic dynamic simulation (right).* ***B*** *shows the setup for online learning on the real robot. The model was initialized with one trial from the simulation of the robot (1st trial in Figure 4.4) and the new obstacle is learned additionally online on the real system. The overlay shows the mental plan over one trial of about* 5:30 *minutes. See Figure 4.5 for more details.*

Typically intrinsic motivation signals have been used for incremental task learning, acquiring skill libraries, learning perceptual patterns and for object manipulation. For the goal of fully autonomous robots however, the ability to focus and guide learning independently from tasks, specified rewards and human input is crucial. The robot should be able to learn without *knowing* what it is supposed to learn in the beginning. Furthermore, the robot should detect on its own if it needs to learn something new or adapt an existing ability if its internal model differs from the perceived reality. To achieve this, we equip the robot with a mechanism for task-independent online adaptation utilizing intrinsic motivation signals inspired by cognitive dissonance. For rapid online adaptation within seconds, we additionally employ a mental replay strategy to intensify experienced situations. Adaptation is done by updating the synaptic weights in the recurrent layer of the network that encodes the state transition model, and this learning is guided by the cognitive dissonance inspired signals.

4.2. Materials and Methods

In this section, we first summarize the challenge and goal we want to address with this paper. Afterwards, we describe the functionality and principles of the underlying bio-inspired stochastic recurrent neural network model, that samples movement trajectories by simulating its inherent dynamics. Next we introduce our novel framework, which enables this model to plan movements online and show how the model can adapt online utilizing intrinsic motivation signals within a supervised learning rule and a mental replay strategy.

4.2.1. The Challenge of (Efficient) Online Adaptation in Stochastic Recurrent Networks

The main goal of the paper is to show that efficient online adaptation of stochastic recurrent networks can be achieved by using intrinsic motivation signals and mental replay. Efficiency is measured as the number of updates triggered, which is equal to the number of required samples, e.g., here the number of physical interactions of the robot with the environment. Additionally, we will show that using adaptive learning signals and only trigger learning when necessary are crucial mechanisms for updating such sensitive stochastic networks.

4.2.2. Motion Planning with Stochastic Recurrent Neural Networks

The proposed framework builds on the model recently presented in [125], where it was shown that stochastic spiking networks can solve motion planning tasks optimally. Furthermore, in [11] an approach to scale these models to higher dimensional spaces by introducing a factorized population coding and that the model can be trained from demonstrations was shown.

Inspired by neuroscientific findings on the mental path planning of rodents [141], the model mimics the behavior of hippocampal place cells. It was shown that the neural activity of these cells is correlated not only with actual movements, but also with future mental plans. This bio-inspired motion planner consists of stochastic spiking neurons forming a multi-layer recurrent neural network. It was shown that spiking networks can encode arbitrary complex distributions [142] and learn temporal sequences [143, 144]. We utilize these properties for motion planning and learning as well as to encode

multi-modal trajectory distributions that can represent multiple solutions to planning problems.

The basis model consists of two different types of neuron populations: a layer of K *state* neurons and a layer of N *context* neurons. The state neurons form a fully connected recurrent layer with synaptic weights $w_{i,k}$, while the context neurons provide feedforward input via synaptic weights $\theta_{j,k}$, with $j \in N$ and $k, i \in K$ with $N \ll K$. There are no lateral connections between context neurons. Each constraint or any task-related information is modeled by a population of context neurons. While the state neurons are uniformly spaced within the modeled state space, the *task-dependent* context neurons are Gaussian distributed *locally* around the corresponding location they encode, i.e., there are only context neurons around the specific constraint they encode.

The state neurons can be seen as an abstract and simplified version of place cells and encode a cognitive map of the environment [145]. They are modeled by stochastic neurons which build up a membrane potential based on the weighted neural input. Context neurons have no afferent connections and spike with a fixed time-dependent probability. Operating in discrete time and using a fixed refractory period of τ timesteps that decays linearly, the neurons spike in each time step with a probability based on their membrane potential. All spikes from presynaptic neurons get weighted by the corresponding synaptic weight and are integrated to an overall postsynaptic potential (PSP). Assuming linear dendritic dynamics, the membrane potential of the state neurons is given by

$$u_{t,k} = \sum_{i=1}^{K} w_{i,k} \tilde{v}_i(t) + \sum_{j=1}^{N} \theta_{j,k} \tilde{y}_j(t) \ , \tag{4.1}$$

where $\tilde{v}_i(t)$ and $\tilde{y}_j(t)$ denote the presynaptic input injected from neurons $i \in K$ and $j \in N$ at time t respectively. Depending on the used PSP kernel for integrating over time, this injected input can include spikes from multiple previous timesteps. This definition implements a simple stochastic spike response model [146]. Using this membrane potential, the probability to spike for the state neurons can be defined by $\rho_{t,k} = p(v_{t,k} = 1) = f(u_{t,k})$, where $f(\cdot)$ denotes the activation function, that is required to be differentiable. The binary activity of the state neurons is denoted by $\mathbf{v}_t = (v_{t,1}, .., v_{t,K})$, where $v_{t,k} = 1$ if neuron k spikes at time t and $v_{t,k} = 0$ otherwise. Analogously, $\mathbf{y}_t$ describes the activity of the context neurons. The synaptic weights $\boldsymbol{\theta}$ which connect context neurons to state neurons provide task related information. By injecting this task related information, the context neurons modulate the random walk behavior of the state neurons towards goal directed movements. This input from the context neurons can also be learned [125] or can be used to, for example, include known dynamic constraints in the planning process [11].

We compared setting the feedfoward context neuron input weights $\boldsymbol{\theta}$ as in [11] – proportional to the euclidean distance – to using Student's t-distributions and generalized error distributions, where the latter produced the best results and was used in the experiments. At each context neuron position such a distribution is located and the weights to the state neurons are drawn from this distribution using the distance between the connected neurons as input. For way points, these context neurons install a *gradient* towards the associated position such that the random walk samples are biased towards the active locations.

For planning, the stochastic network encodes a distribution

$$q(\mathbf{v}_{1:T}|\boldsymbol{\theta}) = p(\mathbf{v}_0) \prod_{t=1}^{T} \mathcal{T}(\mathbf{v}_t|\mathbf{v}_{t-1}) \phi_t(\mathbf{v}_t|\boldsymbol{\theta})$$

over state sequences ($\mathbf{v}_{1:T}$) of T timesteps, where $\mathcal{T}(\mathbf{v}_t|\mathbf{v}_{t-1})$ denotes the transition model and $\phi_t(\mathbf{v}_t|\boldsymbol{\theta})$ the task related input provided by the context neurons. Using the definition of the membrane potential from Equation (4.1), the state transition model is given by

$$\mathcal{T}(v_{t,i}|\mathbf{v}_{t-1}) = f\left(\sum_{k=1}^{K} w_{k,i} \tilde{v}_k(t) v_{t,i}\right) , \tag{4.2}$$

where a PSP kernel that covers multiple time steps includes information provided by spikes from multiple previous time steps. In particular, we use a rectangular PSP kernel of τ timesteps, given by

$$\tilde{v}_k(t) = \begin{cases} 1 \text{ if } \exists l \in [t-\tau, t-1] : v_{l,k} = 1 \\ 0 \text{ otherwise} \end{cases} ,$$

such that, if neuron k has spiked within the last τ timesteps, the presynaptic input $\tilde{v}_k(t)$ is set to 1. Movement trajectories can be sampled by simulating the dynamics of the stochastic recurrent network [142] where multiple samples are used to generate smooth trajectories.

Encoding continuous domains with binary neurons All neurons have a preferred position in a specified coordinate system and encode binary random variables (spike $= 1$/no spike $= 0$). Thus, the solution sampled from the model for a planning problem is the spiketrain of the state neurons, i.e., a sequence of binary activity vectors. These binary

neural activities encode the continuous system state $\mathbf{x}_t$, e.g., end-effector position or joint angle values, using the decoding scheme

$$\mathbf{x}_t = \frac{1}{|\hat{\mathbf{v}}_t|} \sum_{k=1}^{K} \hat{v}_{t,k} \mathbf{p}_k \quad \text{with} \quad |\hat{\mathbf{v}}_t| = \sum_{k=1}^{K} \hat{v}_{t,k} \ ,$$

where $\mathbf{p}_k$ denotes the preferred position of neuron k and $\hat{v}_{t,k}$ is the continuous activity of neuron k at time t calculated by filtering the binary activity $v_{t,k}$ with a Gaussian window filter. Together with the dynamics of the network, that allows multiple state neurons being active at each timestep, this encoding enables the model to work in continuous domains. To find a movement trajectory from position $\mathbf{a}$ to a target position $\mathbf{b}$, the model generates a sequence of states encoding a task fulfilling trajectory.

4.2.3. Online Motion Planning Framework

For efficient online adaptation, the model should be able to react during the execution of a planned trajectory. Therefore, we consider a short time horizon instead of planning complete movement trajectories over a long time horizon. This short time horizon sub-trajectory is called a *segment*. A trajectory $\boldsymbol{\kappa}$ from position $\mathbf{a}$ to position $\mathbf{b}$ can thus consist of multiple segments. This movement planning segmentation has two major advantages. First, it enables the network to consider feedback of the movement execution in the planning process and, second, the network can react to changing contexts, e.g., a changing target position. Furthermore, it allows the network to update itself during planning, providing a mechanism for online model learning and adaptation to changing environments or constraints. The general idea of how we enable the model to plan and adapt online is illustrated in Figure 4.1.

To ensure a continuous execution of segments, the planning phase of the next segment needs to be finished before the execution of the current segment finished. On the other hand, planning of the next segment should be started as late as possible to incorporate the most up-to-date feedback into the process. Thus, for estimating the starting point for planning the next segment, we calculate a running average over the required planning time and use the three sigma confidence interval compared to the expected execution time. The expected execution time is calculated from the distance the planned trajectory covers and a manually set velocity. The learning part can be done right after a segment execution is finished. The alignment of these processes are visualized in Figure 4.1A.

As the recurrent network consists of stochastic spiking neurons, the network models a distribution over movement trajectories rather than a single solution. In order to create a smooth movement trajectory, we average over multiple samples drawn from the model when planning each segment. Before the final mental movement trajectory is created by averaging over the drawn samples, we added a sample rejection mechanism. As spiking networks can encode arbitrary complex functions, the model can encode multi-modal movement distributions. Imagine that the model faces a known obstacle that can be avoided by going around either left or right. Drawn movement samples can contain both solutions and when averaging over the samples, the robot would crash into the obstacle. Thus, only samples that encode the same solution should be considered for averaging.

Clustering of samples could solve this problem, but as our framework has to run online, this approach is too expensive. Therefore, we implemented a heuristic based approach that uses the angle between approximated movement directions as distance. First a reference movement sample is chosen such that its average distance to the majority of the population is minimal, i.e., the sample that has the minimal mean distance to 90% of the population is chosen as the reference. Subsequently only movement samples with an approximated movement direction close to the reference sample are considered for averaging. As threshold for rejecting a sample, the three-sigma interval of the average distances of the reference sample to the closest 90% of the population is chosen.

The feedback provided by the executed movement is incorporated before planning the next segment in two steps. First, the actual position of the robot is used to initialize the sampling of the next segment such that planning starts from where the robot actually is, not where the previous mental plan indicates, i.e., the refractory state of the state neurons is set accordingly. Second, the executed movement is used for updating the model based on the cognitive dissonance signal it generated. In Figure 4.1B this planning and adaptation process is sketched.

4.2.4. Online Adaptation of the Recurrent Layer

The online update of the spiking network model is based on the contrastive divergence (CD) [147] based learning rules derived recently in [11]. CD draws multiple samples from the current model and uses them to approximate the likelihood gradient. The general CD update rule for learning parameters Θ of some function $f(x;\Theta)$ is given by

$$\Delta\Theta = \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^0} - \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^1} , \tag{4.3}$$

where $\mathbf{X}^0$ and $\mathbf{X}^1$ denote the state of the Markov chain after 0 and 1 cycles respectively, i.e., the data and the model distribution. We want to update the state transition function $\mathcal{T}(\mathbf{v}_t|\mathbf{v}_{t-1})$, which is encoded in the synaptic weights $\mathbf{w}$ between the state neurons (see Equation (4.2)). Thus, learning or adapting the transition model means to change these synaptic connections. The update rule for the synaptic connection $w_{k,i}$ between neuron k and i is therefore given by

$$w_{k,i} \leftarrow w_{k,i} + \alpha \Delta w_{k,i} \tag{4.4}$$
$$\text{with} \quad \Delta w_{k,i} = \tilde{v}_{t-1,k}\tilde{v}_{t,i} - \tilde{v}_{t-1,k}v_{t,i} \ ,$$

where $\tilde{v}$ denotes the spike encoding of the training data, v the sampled spiking activity, t the discrete timestep and α is the learning rate. Here, we consider a resetting rectangular PSP kernel of one time step ($\tilde{v}_{t-1,k}$), a PSP kernel of τ time steps follows the same derivation and is used in the experiments. In summary, this learning rules changes the model distribution slowly towards the presented training data distribution. For a more detailed description of this spiking contrastive divergence learning rule, we refer to [11]. This learning scheme works for offline model learning when the previously gathered training data is replayed to an inhibitory initialized model.

For using the derived model learning rule in the online scenario, we need to make several changes. In the original work, the model was initialized with inhibitory connections. Thus, no movement can be sampled from the model for exploration until the learning process has converged. This is not suitable in the online learning scenario, as a *working* model for exploration is required, i.e., the model needs to be able to generate movements at any time. Therefore, we initialize the synaptic weights between the state neurons using Gaussian distributions [148], i.e., a Gaussian is placed at the preferred position of each state neuron and the synaptic weights are drawn from these distributions with an additional additive negative offset term that enables inhibitory connections. The synaptic weights are limited within $[-1, 1]$.

This process initializes the transition model with an uniform prior, where for each position, transitions in all directions are equally likely. The variance of these basis functions and the offset term are chosen such that only close neighbors get excitatory connections, while distant neighbors get inhibitory connections, ensuring only small state changes within one timestep. i.e, a movement cannot *jump* to the target immediately.

Furthermore, the learning rule has to be adapted as we do not learn with an empty model from a given set of demonstrations but rather update a working model with online feedback. Therefore, we treat the perceived feedback in form of the executed trajectory as

a sample from the training data distribution and the mental trajectory as a sample from the model distribution in the supervised learning scheme presented in Equation (4.3).

Spike Encoding of Trajectories For encoding the mental and executed trajectories into spiketrains, Poisson processes with normalized Gaussian responsibilities of the state neurons at each timestep as time-varying input are used as in [11]. These responsibilities are calculated using the same Gaussian basis functions, centered at the state neurons preferred positions, as used for initializing the synaptic weights. More details on these responsibilities are given in Subsection 4.2.6 as they are also used for the local adaptation signals. To transform these continuous responsibilities of the state neurons into binary spiketrains, they are scaled by a factor of 100, limited into $[0, 10]$ and used as mean input to a Poisson distribution for each neuron. The drawn samples for each neuron from these Poisson distributions for each timestep are compared to a threshold of 4 and the neurons spike at time t if this threshold is reached and the neuron has not spiked within its refractory period before. The used parameters were chosen as they produced similar spiketrains as the ones sampled from the model.

4.2.5. Global Intrinsically Motivated Adaptation Signal

For online learning, the learning rate typically needs to be small to account for the noisy updates, inducing a long learning horizon, and thus requires a large amount of samples. Especially, for learning with robots this is a crucial limitation as the number of experiments is limited. Furthermore, the model should only be updated if *necessary*. Therefore, we introduce a time-varying learning rate α_t that controls the update step. This dynamic rate can for example encode uncertainty to update only reliable regions, can be used to emphasize updates in certain workspace or joint space areas, or to encode intrinsic motivation signals.

In this work, we use an intrinsic motivation signal for α_t that is motivated by cognitive dissonance [129, 130]. Concretely, the dissonance between the mental movement trajectory generated by the stochastic network and the actual executed movement is used. Thus, if the executed movement is similar to the generated mental movement, the update is small, while a stronger dissonance leads to a larger update. In other words, learning is guided by the mismatch between expectation and observation.

This cognitive dissonance signal is implemented by the timestep-wise distance between the mental movement plan $\boldsymbol{\kappa}^{(m)}$ and the executed movement $\boldsymbol{\kappa}^{(e)}$. Thus, the resulting

learning factor is generated globally and is the same for all neurons. As distance metric we chose the squared L^2 norm but other metrics could be used as well depending on, for example, the modeled spaces or environment specific features. Thus, for updating the synaptic connection $w_{k,i}$ at time t, we change Equation (4.4) to

$$\begin{aligned} w_{k,i} &\leftarrow w_{k,i} + \alpha_t \Delta w_{k,i} \\ \text{with} \quad \alpha_t &= \|\boldsymbol{\kappa}_t^{(m)} - \boldsymbol{\kappa}_t^{(e)}\|_2^2 \\ \text{and} \quad \Delta w_{k,i} &= \tilde{v}_{t-1,k}\tilde{v}_{t,i} - \tilde{v}_{t-1,k}v_{t,i} \quad , \end{aligned} \tag{4.5}$$

where $\tilde{v}_t$ is the spike encoding generated from the actual executed movement trajectory $\boldsymbol{\kappa}_t^{(e)}$ and v_t the encoding from the mental trajectory $\boldsymbol{\kappa}_t^{(m)}$ using the previously described Poisson process approach.

To stabilize the learning progress and for safety on the real system, we limit α_t in our experiments to $\alpha_t \in [0, 0.3]$ and use a learning threshold of 0.02. Thereby, the model update is only triggered when the cognitive dissonance is larger than this threshold, avoiding unnecessary computational resources, being more robust against noisy observations. Note that during the experiments, α_t did not reach the safety limit and, therefore, the limit had no influence on the learning. With this intrinsic motivated learning factor and the threshold that triggers adaptation, the update is regulated according to the model error and *invalid* parts of the model are updated accordingly.

4.2.6. Local Intrinsically Motivated Adaptation Signals

In the previous subsection we discussed a mechanism for determining the cognitive dissonance signal that relies on the distance between the mental and the executed plan. Thus, the resulting α_t is the same for all neurons at each timestep t, i.e., resulting in a *global* adaptation signal. Furthermore, the adaptation signal is calculated without taking the model into account. To generate the adaptation signal incorporating the model, we need a different mechanism which is already inherent to the model. Furthermore, we want to have individual learning signals for each neuron leading to a more focused and flexible adaptation mechanism. Thus, the resulting learning signal should be *local* and generated using the model. To fulfill these properties, we utilize the mechanism that is already used in the model to encode trajectories into spiketrains – the responsibilities of each neuron for a trajectory. Inserting these individual learning signals into the update

rule from Equation (4.5) alters the update rule to

$$\begin{aligned} w_{k,i} &\leftarrow w_{k,i} + \alpha_{t,i}\Delta w_{k,i} \\ \text{with} \quad \alpha_{t,i} &= c(\omega_{t,i}^{(m)} - \omega_{t,i}^{(e)})^2 \\ \text{and} \quad \Delta w_{k,i} &= \tilde{v}_{t-1,k}\tilde{v}_{t,i} - \tilde{v}_{t-1,k}v_{t,i} \quad , \end{aligned} \tag{4.6}$$

with an additional constant scaling factor c. For each neuron i, $\alpha_{t,i}$ encodes the time dependent adaptation signal. These local adaptation signals are calculated as the squared difference between the responsibilities $\omega_{t,i}^{(m)}$ and $\omega_{t,i}^{(e)}$ for each neuron i for the mental and the executed trajectory respectively. These responsibilities emerge from the Gaussian basis functions centered at the state neurons positions that are also used for initializing the state transition model and the spike encoding of trajectories. Therefore, the responsibilities are given by $\omega_{t,i}^{(m)} = b_i(\boldsymbol{\kappa}_t^{(m)})$ and $\omega_{t,i}^{(e)} = b_i(\boldsymbol{\kappa}_t^{(e)})$ with

$$b_i(\mathbf{x}) = \exp\left(\frac{1}{2}(\mathbf{x} - \boldsymbol{p}_i)^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{p}_i)\right) \quad ,$$

where $\mathbf{p}_i$ is the preferred position of neuron i. In the experiment we set $c = 3$, the learning threshold for $\alpha_{t,i}$ that triggers learning for each neuron to 0.05 and limit the signal like in the global adaptation signal setting to $\alpha_{t,i} \in [0, 0.3]$. Note, as in the global adaptation experiments, this limit was never reached in the local experiments and thus, had no influence on the results.

4.2.7. Using Mental Replay Strategies to Intensify Experienced Situations

As the encoding of trajectories into spiketrains using Poisson processes is a stochastic operation, we can obtain a whole population of encodings from a single trajectory. Therefore, populations of training and model data pairs can be generated from one experience and used for learning. We utilize this feature to implement a mental replay strategy that intensifies experienced situations to speed up adaptation. In particular, we draw 20 trajectory encoding samples per observation in the adaptation experiments, where each sample is a different spike encoding of the trajectory, i.e., a mental replay of the experienced situation. Thus, by using such a mental replay approach, we can apply multiple updates from a single interaction with the environment. The two mechanisms, using intrinsic motivation signals for guiding the updates and mental replay strategies to intensify experiences, lower the required number of experienced situations, which is a crucial requirement for learning with real robotic systems.

4.3. Results

We conducted four experiments to evaluate the proposed framework for online planning and learning based on intrinsic motivation and mental replay. In all experiments the framework had to follow a path given by way points that are activated successively one after each other. Each way point remains active until it is reached by the robot. In the first two experiments a realistic dynamic simulation of the KUKA LWR arm was used. First, the proposed framework had to adapt to an unknown obstacle that blocks the direct path between two way points using the global adaptation signal and, second, by using the local adaptation signals and, third, by using constant learning rates (in combination with the global adaptation signal for triggering learning). In the fourth experiment, we used a pre-trained model from the simulation in a real robot experiment to show that it is possible to transfer knowledge from simulation to the real system. Additionally, the model had to adapt online to a new unknown obstacle, again using the local adaptation signals, to highlight online learning on the real system.

4.3.1. Experimental Setup

For the simulation experiments, we used a realistic dynamic simulation of the KUKA LWR arm with a cartesian tracking controller to follow the reference trajectories generated by our model. The tracking controller is equipped with a safety controller that stops the tracking when an obstacle is hit. The task was to follow a given sequence of way points, where obstacles block the direct path between two way points in the adaptation experiments. In the real robot experiment, the same tracking and safety controllers were used. Figure 4.2 shows the simulated and real robot as well as the experimental setup.

By activating the way points successively one after each other as target positions using appropriate context neurons, the model generates online a trajectory tracking the given shape. The model has no knowledge about the task or the constraint, i.e., the target way points, their activation pattern and the obstacle. We considered a two-dimensional workspace that spans $[-1, 1]$ for both dimensions – the neuron's coordinate system – encoding the 60×60 cm operational space of the robot. Each dimension is encoded by 15 state neurons, which results in 225 state neurons using full population coding as in [11]. The refractory period is set to $\tau = 10$, mimicking biological realistic spiking behavior and introducing additional noise in the sampling process. The transition model is initialized by Gaussian basis functions centered at the preferred positions of the neurons (see *Materials*

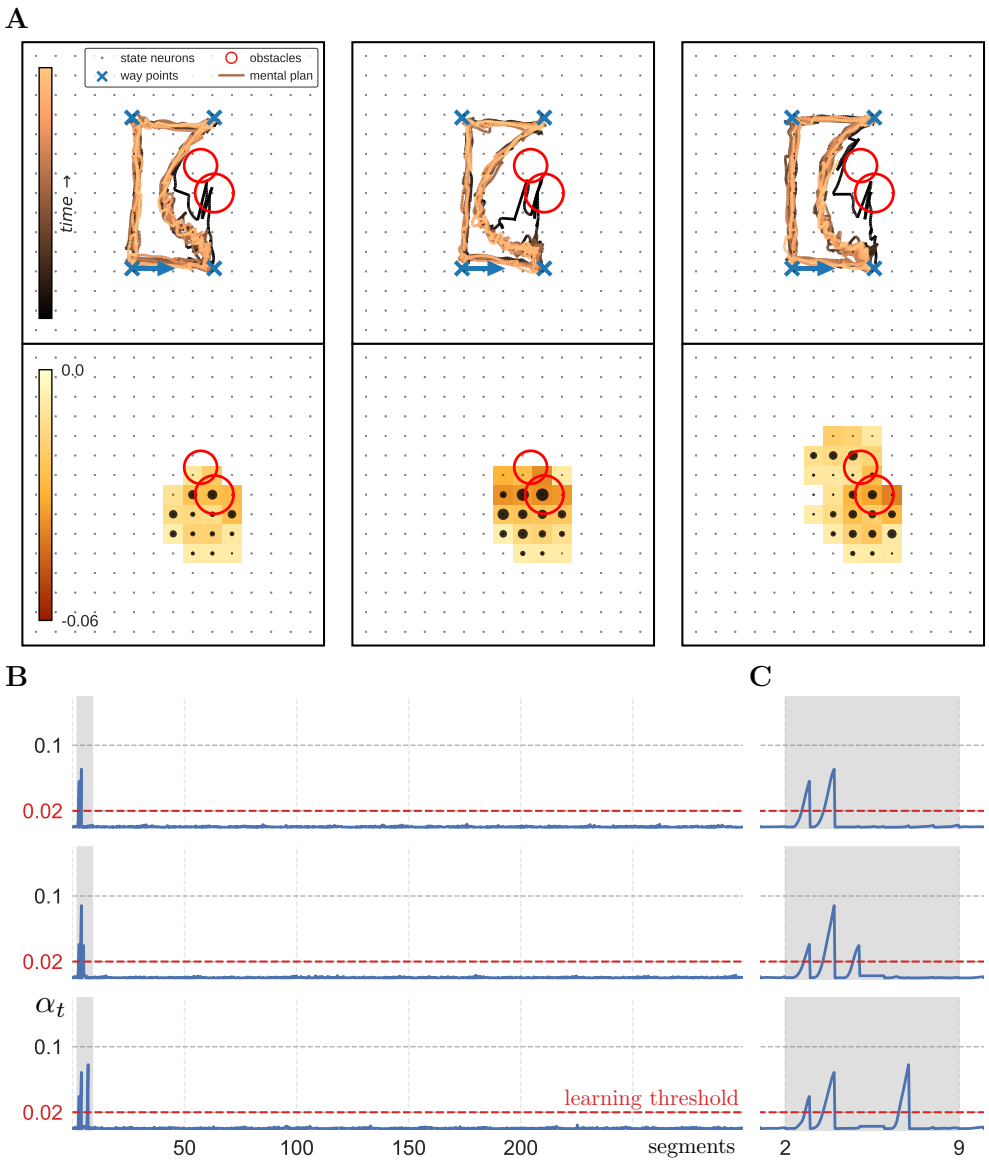


Figure 4.3.: ***Adaptation results for three trials with the global learning signal.*** *Each column in* ***A*** *shows one trial of the online adaptation with the global learning signal, where the upper row shows the mental plan over time and the lower row depicts the adapted model. This change in the model is depicted with the heatmap showing the average change of synaptic input each neuron receives. Similarly the average change of synaptic output each neuron sends is shown with the scaled neuron sizes.* ***B*** *and* ***C*** *show the global learning signals α_t for the three trials over the planned segments.*

and Methods for more details). For the mental replay we used 20 iterations, i.e., 20 pairs of training data were generated for each executed movement. All adaptation experiments were 300 segments long, where 40 trajectory samples were drawn for each segment and 10 trials were conducted for each experimental setting.

	updates triggered (⇓)	update time (⇓)	planning time (⇓)	exec. time (⇑)	target reached (⇑)
global trigger					
$\alpha = 0.001$	7.3 ± 1.1	$42.6 \pm 4.2\ ms$	$0.238 \pm 0.047\ s$	$0.898 \pm 0.656\ s$	10.5 ± 0.5
$\alpha = 0.01$	2.4 ± 0.5	$43.7 \pm 4.2\ ms$	$0.237 \pm 0.046\ s$	$0.806 \pm 0.652\ s$	8.5 ± 3.2
$\alpha = 0.1$	$\mathbf{2.0 \pm 0.0}$	$46.7 \pm 5.7\ ms$	$0.234 \pm 0.043\ s$	$0.771 \pm 0.670\ s$	7.9 ± 4.1
global α_t	2.8 ± 0.9	$43.3 \pm 4.1\ ms$	$0.241 \pm 0.044\ s$	$0.928 \pm 0.658\ s$	10.4 ± 0.8
local $\alpha_{t,i}$	8.6 ± 2.8	$52.6 \pm 7.5\ ms$	$0.235 \pm 0.042\ s$	$\mathbf{1.11 \pm 0.679\ s}$	$\mathbf{13.7 \pm 1.4}$

Table 4.1.: *Evaluation of the adaptation experiments for* 10 *trials with each the global, the local and constant learning signals in simulation. The values denote the number of times learning was triggered by a segment (updates triggered = required samples = physical interactions), the time required per triggered update including the mental replay strategy (update time), the planned execution time per segment (exec. time), the required time for planning a segment including sampling and post-processing (planning time), and the number of times the blocked target was reached within the budget of* 300 *segments (target reached), i.e., number of times all way points were visited. All values denote mean and standard deviation. The* $\Downarrow$ *and* $\Uparrow$ *symbols denote if a lower or higher value is better respectively. Note that the constant* α *settings use the global adaptation signal* α_t *for triggering learning.*

4.3.2. Rapid Online Model Adaptation using Global and Local Signals

In this experiment, we want to show the model's ability to adapt continuously during the execution of the planned trajectory. A direct path between two successively activated way points is blocked by an unknown non-symmetric obstacle, which results in a discrepancy between the planned and executed trajectory due to the interrupted movement.

Constant Learning Rates and the Importance of the Learning Threshold The main and starting motivation of the project was to enable online adaptation in the proposed stochastic recurrent network. Therefore, we first created the framework for online planning (and adaptation – see Figure 4.1). Afterwards we started experiments with online adaptation using the original learning rule (see Equation (4.4)) and a constant learning

rate α. We were not able to find a constant α for which the online learning was successful and stable, i.e., learning to avoid the obstacles and generating valid movements throughout the whole experiment. With small learning rates, learning to avoid obstacles was successful, however, as the model is updated *permanently* and in areas that are not affected by the environmental change, the model became unstable over time, resulting in a model, that was not able to produce valid movements anymore. The effect on the transition model using different constant learning rate is shown in Figure 4.8, which shows the *unlearned* transition model that cannot produce valid movements (compare to Figures 4.3 & 4.4).

These insights gave rise to the idea of using adaptive learning signals in combination with a learning threshold to trigger learning only when an unexpected change is perceived. With these mechanisms, successful and stable online adaptation of the stochastic recurrent network was possible.

Most closely related to our work are potential fields methods for motion planning and extensions to dynamic obstacle avoidance (see [149–153] for example). All these approaches are deterministic models that consider obstacles through fixed heuristics of repelling potential fields. In contrast, in our work we learn to avoid obstacles online through interaction by using the unexpected perceived feedback. In addition to the gradient based method in [149], we can learn to avoid obstacles with unknown shapes through the interactive online approach and static obstacles do not need to be known a priori. To evaluate the benefit of the dynamic online adaptation signals, we additionally compare to a baseline of our model using constant learning rates (with the adaptive global signal as learning trigger). This model can be seen as an extension of [149] using stochastic neurons with the ability to adapt the potential field whenever an obstacle is hit.

Online Adaptation Experimental Results The effect of the online learning process using intrinsically motivated signals is shown in Figure 4.3 and Figure 4.4 for the global and the local signals respectively, where the mental movement trajectories, the adapted models and the adaptation signals α_t and $\alpha_{t,i}$ for three trials are shown. Additionally we compare to using different constant learning rates α, which use the global adaptation signal and its learning threshold to trigger learning (see the previous paragraph for why this is important), but using the constant α for the update.

With the proposed intrinsically motivated online learning, the model initially tries to enter the *invalid* area but *recognizes*, due to the perceived feedback of the interrupted movement encoded in the cognitive dissonance signals, the unexpected obstacle. As a result the model adapts successfully and avoids the obstacle. This adaptation happens efficiently

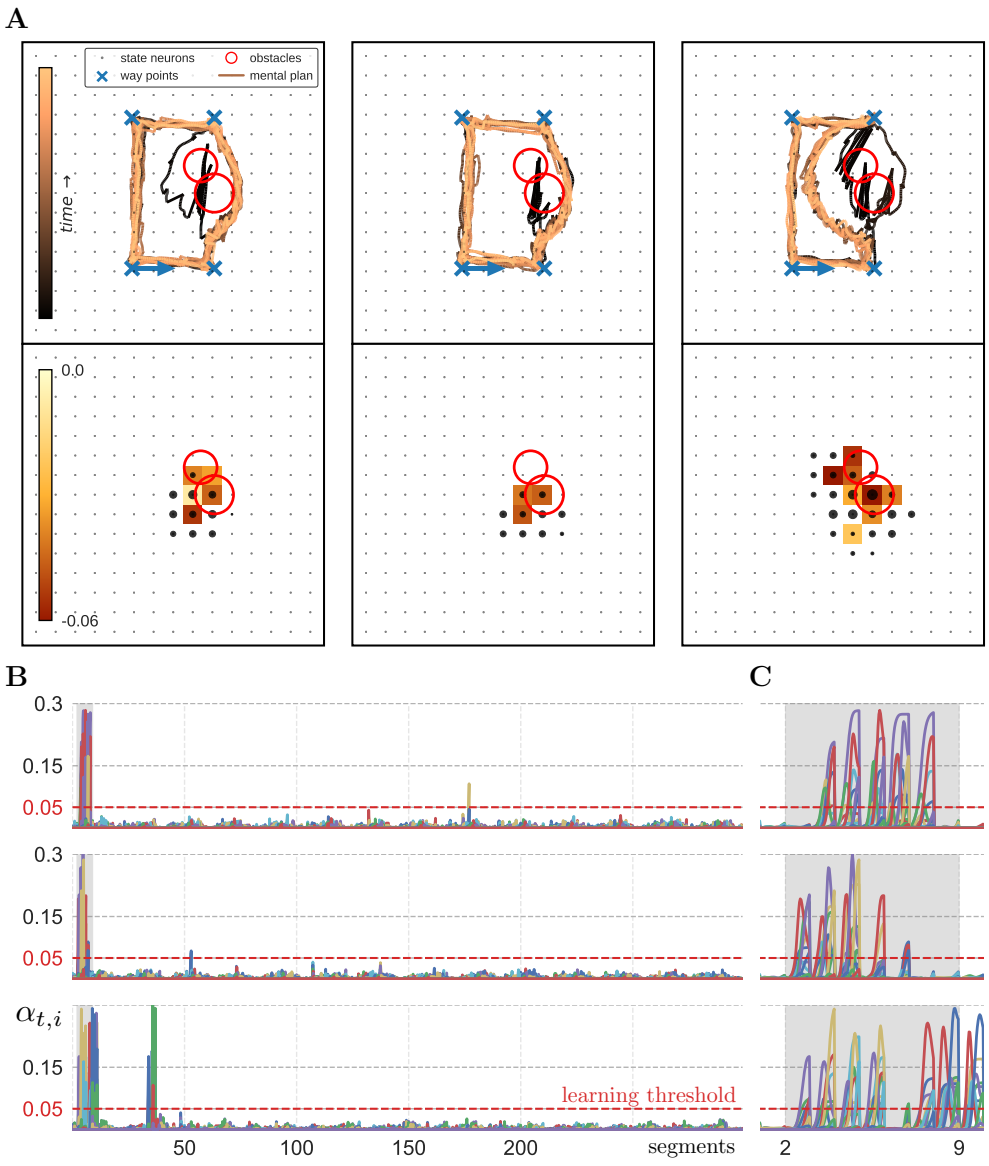


Figure 4.4.: ***Adaptation results for three trials with the local learning signals.*** *Each column in* ***A*** *shows one trial of the online adaptation with the local learning signals, where the upper row shows the mental plan over time and the lower row depicts the changed model. This change in the model is depicted with the heatmap showing the average change of synaptic input each neuron receives. Similarly the average change of synaptic output each neuron sends is shown with the scaled neuron sizes.* ***B*** *and* ***C*** *show the local learning signals* $\alpha_{t,i}$ *for the three trials over the planned segments. Each color indicates the learning signal for one neuron.*

from only 2.8 ± 0.9 physical interactions – planned segments that hit the obstacle, which is equal to the number of samples required for learning – with the global learning signal, where the planned execution time of one segment is 0.928 ± 0.658 seconds. Moreover, the learning phase including the mental replay strategy takes only 43.3 ± 4.1 milliseconds per triggered segment.

Update and planning time with the local learning signals are similar, but adaptation is triggered 8.6 ± 2.8 times and the planned execution time is 1.11 ± 0.679 seconds. The increase of triggered updates is induced by the higher variability and noise in the individual learning signals, enabling more precise but also more *costly* adaptation. Still, the required samples – triggered updates – for successful adaptation reflect a sample efficient adaptation mechanism for a complex stochastic recurrent network. The longer execution time indicates that the local learning signals generate more efficient solutions, as every segment covers a larger part of the trajectory, i.e., less segments are required resulting in a higher number for reaching the blocked target. The local adaptation signals reached the blocked target 13.7 ± 1.4 times, which outperforms the other adaptation signals. These results are summarized in Table 4.1. Thus, during the adaptation the global learning signals need fewer interactions, but the resulting solutions afterwards are less efficient. The different effects of the global and local learning signals are discussed in more detail in Section 4.4.

The results when using constant learning rates are summarized in Table 4.1 as well. The best result was achieved with a learning rate of $\alpha = 0.001$, resulting in similar number of reached targets like the global adaptation signal (see also Figure 4.6), but required almost as much updates – i.e., samples – as the local adaptation signals. In addition to tuning this additional parameter, i.e., the constant learning rate, an adaptive signal for triggering learning is still required for successful and robust adaptation. Moreover, when using the higher constant learning rates, the learning was unstable in some trials even with the adaptive trigger signals, i.e., after adaptation no valid movements could be sampled anymore.

By adapting online to the perceived cognitive dissonances, the model generates new valid solutions avoiding the obstacle within seconds from few physical interactions (samples) with both learning signals.

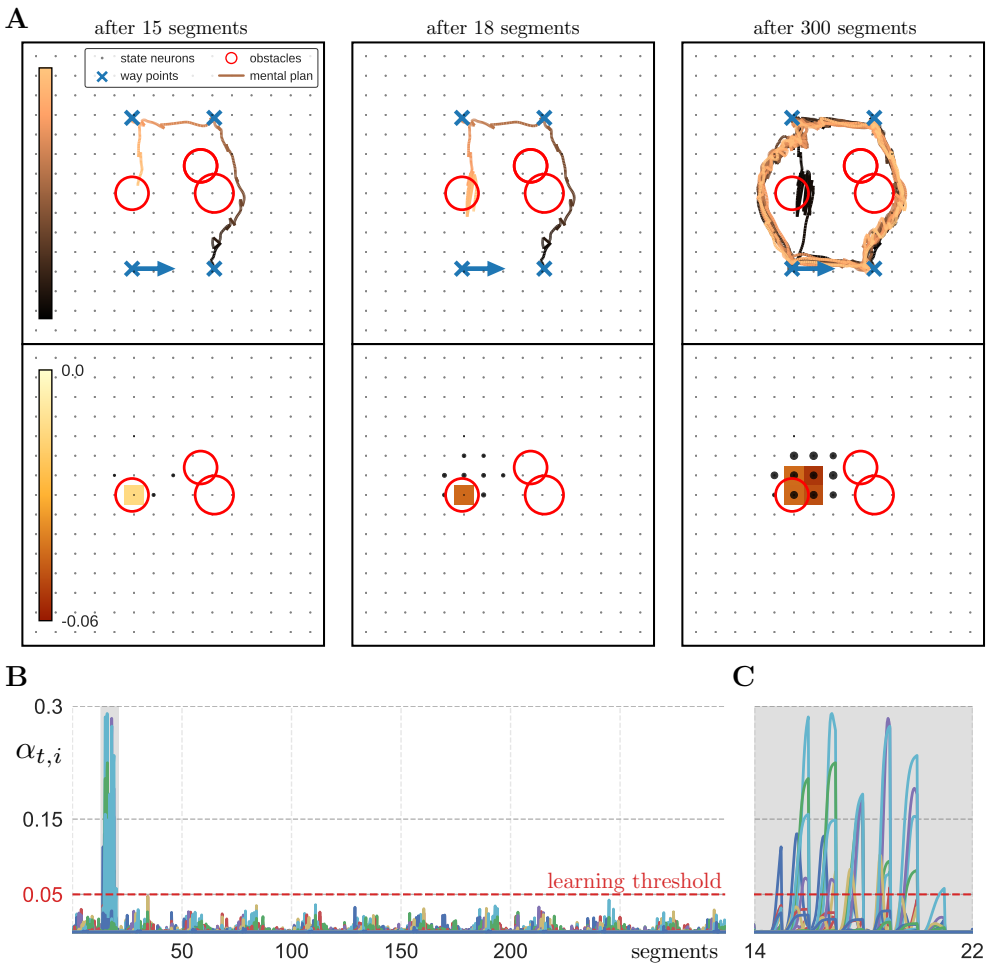


Figure 4.5.: ***Adaptation results on the real robot.*** *Online adaptation with the real KUKA LWR arm using the local learning signals initialized with simulation results, i.e., the right obstacles are already learned. The left obstacle is added to the real environment (see Figure 4.2B). Each column in* ***A*** *shows the mental plan and the model for the indicated time. The change in the model is depicted with the heatmap showing the average change of synaptic input each neuron receives compared to the pre-trained model. Similarly the average change of synaptic output each neuron sends is shown with the scaled neuron sizes. The mental plan demonstrates the rapid adaptation, as only a few interactions of the robot are necessary to adapt to the new environment. This efficiency is further highlighted in* ***B*** *and* ***C****, where the local learning signals* $\alpha_{t,i}$ *are shown over the execution time. Each color indicates the learning signal for one neuron.*

4.3.3. Transfer to and Learning on the Real Robot

With this experiment we show that the models learned in simulation can be transferred directly onto the real system and, furthermore, that the efficient online adaptation can be done on a real robotic system. Therefore, we adapted the simulated task of following the four given way points. Additionally to the obstacles that were already present in simulation, we added a new unknown obstacle to the real environment. The setup is shown in Figure 4.2B. The framework parameters were the same as in the simulation experiment, except that the recurrent weights of the neural network were initialized with one trial of the simulation. For updating the model the local learning signals were used and therefore the model was initialized with the 1st trial of the local signals simulation experiments (1st column in Figure 4.4A). On average, an experimental trial on the real robot took about $5{:}30$ minutes (same as in simulation) and Figure 4.5 shows the execution and adaptation over time.

As we started with the network trained in simulation, the robot successfully avoids the first obstacles right away and no adaptation is triggered before approaching the new obstacle (Fig. 4.5 first column).

After 15 segments, the robot collides with the new obstacle and adapts to it within 7 interactions (Figure 4.5 second and third column). The mismatch between the mental plan and the executed trajectory is above the learning threshold and the online adaptation is triggered and scaled with $\alpha_{t,i}$ (Figure 4.5B).

To highlight the efficient adaptation on the real system, we depicted the mental plan after 15, 18 and 300 segments in Figure 4.5A. For the corresponding segments, the cognitive dissonance signals show a significant mismatch that leads to the fast adaptation, illustrated in Figure 4.5B-C. After the successful avoidance of the new obstacle, the robot performs the following task while avoiding both obstacles and no further updates are triggered.

4.4. Discussion

In this section we evaluate and compare the learning signals, the resulting models after the adaptation process, and the generated movements of the local and the global learning signals.

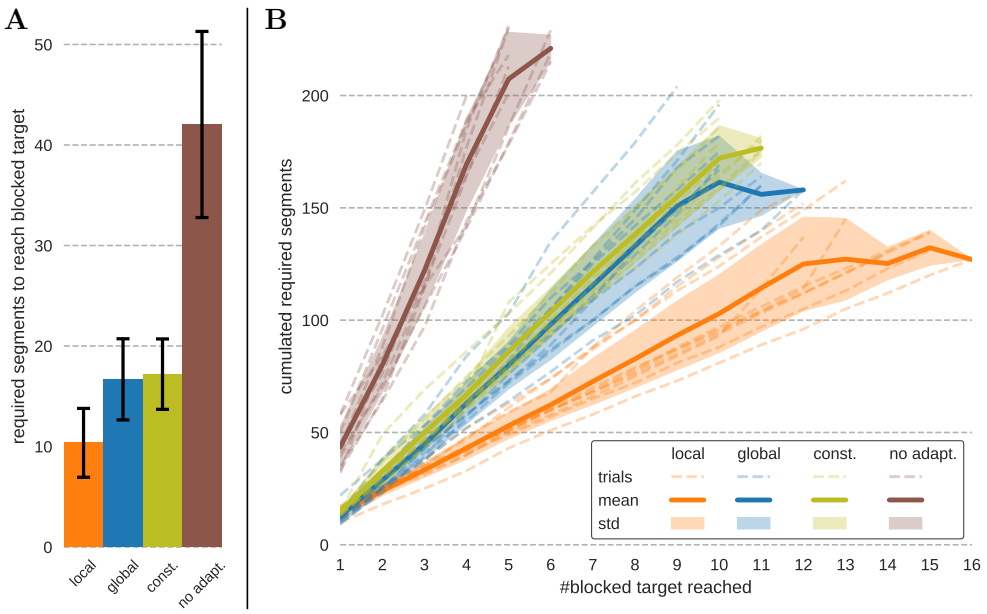


Figure 4.6.: ***Efficiency of the learned solutions.*** ***A*** *shows the mean and standard deviation of the number of required segments to reach the blocked target over all* 10 *trials for each setting.* ***B*** *shows the cumulated required segments for reaching the blocked target for each trial and setting together with the mean and standard deviation of the trials of one setting. Note that due to the limit of planning* 300 *segments in each trial, the number of times the blocked target is reached differs across trials. The constant learning rate* ($\alpha = 0.001$) *still uses the global adaptation signal for triggering learning.*

4.4.1. Efficiency of the Learned Solutions

Comparing the generated movements in Figure 4.4A to the movements generated with the global adaptation signal in Figure 4.3A, the model using the local learning signals anticipates the learned obstacle earlier resulting in more natural evasive movements, i.e., more efficient solutions. Here we define efficiency as the number of segments required to reach the blocked target. As shown in Figure 4.4B-C, each neuron has a different learning signal $\alpha_{t,i}$ and therefore a different timing and scale for the adaptation, i.e., the neurons adapt independently in contrast to the global signal. These individual updates enable a more flexible and finer adaptation, resulting in more efficient solutions. As a result, when using the local adaptation signals, the model favors the more efficient solution on the right and chooses the left solution only in some trials at all after adaptation. In contrast, this behavior never occurred in all ten trials with the global signal.

This efficiency can also be seen in Figure 4.6, where the required segments to reach the blocked target are shown for the local signals, the global signal, a constant learning rate $\alpha = 0.001$, and without any adaptation. Note that due to the stochasticity in the movement generation, the model can reach the block target without adaptation as well. However, without adaption the obstacle is only avoided occasionally through the stochasticity in sampling the movements.

In Figure 4.6A the mean and standard deviation of the required segments for reaching the blocked target are shown for 10 trials with each setting over the complete 300 segments in each trial. All adaptation mechanisms outperform the model without adaptation, whereas the local signals perform better than the global signal and the constant learning rate. Similar, in Figure 4.6B the cumulated required segments for reaching the blocked target consecutively are shown for each trial together with the mean and standard deviation over the trials. Note that, as all trials were limited to 300 segments, the number of times the blocked target was reached differs in the different settings and trials (see Table 4.1), depending on the efficiency of the generated movements, i.e., the amount of segments used.

4.4.2. Comparison of the Learning Signals

To investigate the difference in the generated movements when using the global or local signals, we analyzed the corresponding learning signals α_t and $\alpha_{t,i}$. This evaluation is shown in Figure 4.7, where the magnitudes of the generated learning signals are plotted

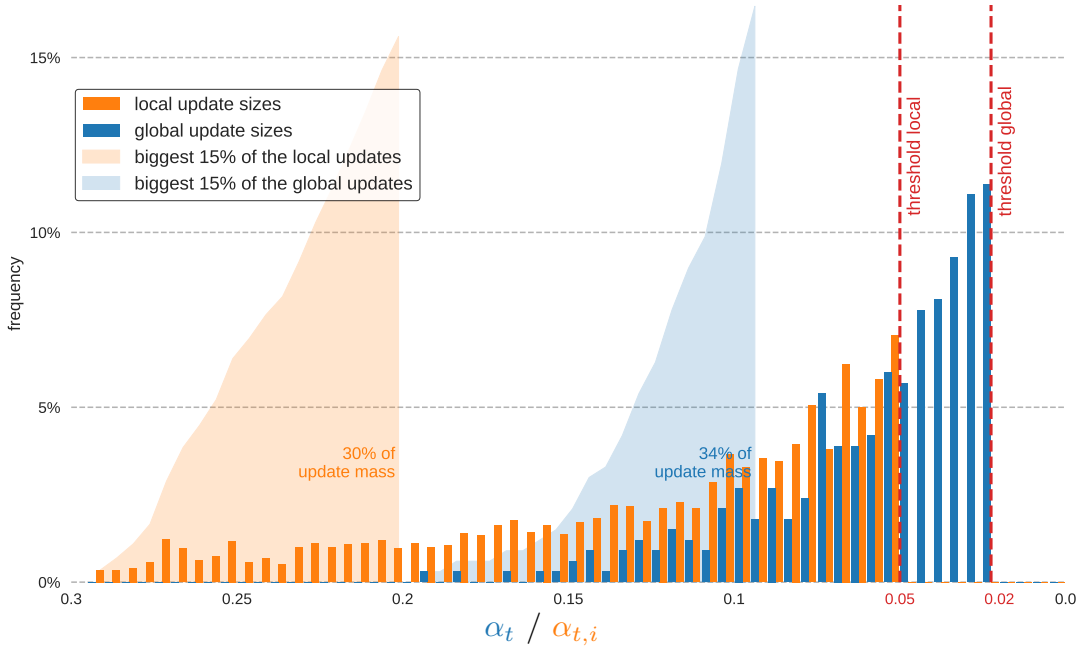


Figure 4.7.: ***Comparison of the learning signals.*** *The magnitude of the generated learning signals over all* 10 *trials for each of the global and local mechanisms are shown with their respective frequencies. Update mass refers to the sum of all generated learning signals weighted by their frequencies.*

with their occurring frequency. When looking at the right half of the histograms – the α_t and $\alpha_{t,i}$ with lower magnitude –, both learning mechanisms produce similar distributions of the learning signals magnitude. The main difference is the range of the generated signals, i.e., the local mechanism is able to generate stronger learning signals. Even though the frequency of these bigger updates is low – about 15% of the total updates –, they cover 30% of the total update mass, where update mass is calculated as the sum over all generated learning signals weighted by their frequencies. In contrast, the biggest 15% of the global learning signals cover 34% of the update mass and are all smaller than the biggest 15% of the local signals.

The ability to generate stronger learning signals in addition to the flexibility of individual signals, enables the local adaptation mechanism to learn models which generate more efficient solutions. The importance of the flexibility enabled by the individual learning signals is further discussed in the subsequent section.

4.4.3. Spatial Adaptation

Investigating the structure of the changes induced by the different learning signals, reveals a difference in the spatial adaptation and especially in the strength of the changes. In the lower rows of Figure 4.3A and Figure 4.4A the changes in the models are visualized with heatmaps showing the *average change* of *synaptic input* each state neuron receives, e.g., a value of -0.03 indicates that the corresponding neuron receives more inhibitory signals after adaptation. Additionally, the *average change* of *synaptic output* of each state neuron is depicted by the scaled neuron sizes.

When the model adapts with the global signal (Figure 4.3), the incoming synaptic weights of neurons with preferred positions around the blocked area are decreased – the model only adapts in these areas. The neurons around the constraint are inhibited after adaptation and, therefore, state transitions to these neurons get less likely. This inhibition hinders the network to sample mental movements in affected areas, i.e., the model has learned to avoid these areas. Due to the global signal, the learning is coarse and the affected area is spread larger than the actual obstacle.

In contrast, when adapting using the local signals (Figure 4.4), the structure of the changes in the model are more focused. The strongest inhibition is still around the obstacle – and stronger than with the global signal –, but much less changes can be found *in front* of the obstacle. This concentration of the adaptation can also be seen when comparing the changes in the synaptic input and output. Both learning mechanisms produce a similar

change in the output, but very different changes in the input, i.e., the neurons adapted with the local signals *learned to focus* their output more precisely.

These stronger and more focused adaptations seem to enable the models updated with the local learning signals to generate more efficient solutions and favor the simpler path.

4.4.4. Learning Multiple Solutions

Even though during the adaptation phase the model only experienced one successful strategy to avoid the obstacle, it is able to generate different solutions, i.e., bypassing the obstacle left or right, with both adaptation mechanisms. Depending on the individual adaptation in each trial, however, the ratio between the generation of the different solutions differs. Especially when using the local signals, the frequency of the more efficient solution is higher, reflecting the efficiency comparison in Figure 4.6.

The feature of generating different solutions is enabled by the model's intrinsic stochasticity, the ability of spiking neural networks to encode arbitrary complex functions, the planning as inference approach and the task-independent adaptation of the state transition model.

4.5. Conclusion

In this work, we introduced a novel framework for probabilistic online motion planning with an efficient online adaptation mechanism. This framework is based on a recent bio-inspired stochastic recurrent neural network that mimics the behavior of hippocampal place cells [11, 125]. The online adaptation is modulated by intrinsic motivation signals inspired by *cognitive dissonance* which encode the mismatch between mental expectation and observation. Based on our prior work on the global intrinsic motivation signal [12, 13], we developed in this work a more flexible local intrinsic motivation signal for guiding the online adaptation. Additionally we compared and discussed the properties of these two intrinsically motivated learning signals. By combining these learning signals with a mental replay strategy to intensify experienced situations, sample-efficient online adaptation within seconds is achieved. This rapid adaptation is highlighted in simulated and real robotic experiments, where the model adapts to an unknown environment within seconds from few interactions with unknown obstacles without a specified learning task or other human input. Although requiring a few interactions more, the local learning signals learn

more focused and are able to generate more efficient solutions – less segments to reach the blocked target – due to the high flexibility of individual learning signals.

In contrast to [125], where the *task-dependent context neuron input* was learned in a reinforcement learning setup, we update the state transition model, encoded in the recurrent state neurons connections, to adapt *task-independently* with a supervised learning approach. This sample-efficient and task-independent adaptation lowers the required expert knowledge and makes the approach promising for learning on robotic systems, for reusability and for adding online adaptation to (motion) planning methods.

Learning to avoid unknown obstacles by updating the state transition model encoded in the recurrent synaptic weights is a step towards the goal of recovering from failures. One limitation to overcome before that, is the curse of dimensionality of the full population coding used by the uniformly distributed state neurons to scale the model to higher dimensional spaces. In future work therefore, we want to combine this approach with the factorized population coding from [11] – where the model's ability to scale to higher dimensional spaces and settings with different modalities was shown – and learning the state neuron population [150], in order to apply the framework to recover from failure tasks with broken joints [154, 155], investigating an intrinsic motivation signal mimicking the avoidance of arthritic pain [156, 157].

With the presented intrinsic motivation signals, the agent can adapt to novel environments by reacting to the perceived feedback. For active exploration, and thereby *forgetting* or finding novel solutions after failures, we plan to investigate intrinsic motivation signals mimicking curiosity [158] in addition.

As robots should not be limited in their development by the learning tasks specified by the human experts, equipping robots with such task-independent adaptation mechanisms is an important step towards autonomously developing and lifelong-learning systems.

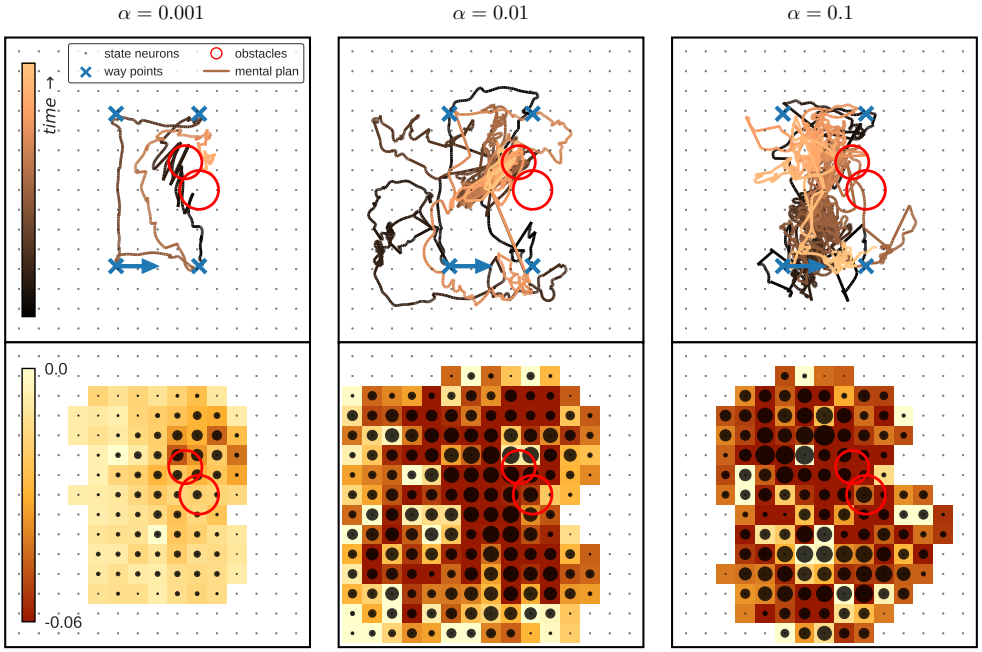


Figure 4.8.: ***Adaptation results with constant learning rates.*** *Each column shows one trial of the online adaptation with different constant learning rates α, where the upper row shows the mental plan over time and the lower row depicts the adapted model. This change in the model is depicted with the heatmap showing the average change of synaptic input each neuron receives. Similarly the average change of synaptic output each neuron sends is shown with the scaled neuron sizes. With constant learning rates and no adaptive signal for triggering learning, the model is updated constantly, by what the state transition model gets destroyed and no correct movement can be sampled anymore.*

5. Conclusion

For integrating intelligent agents into real world scenarios and our daily lives, we identified two main limitations of current AI systems: the usually restricted targeted applications with specialized solutions, and the seemingly intelligent behaviour *caged* in software agents without physical interaction. To contribute in overcoming these limitations, we investigated the learning of cognitive and physical abilities which were summarized in the tackled research questions:

1) How can tasks and their structure be taught to a robot in a natural way?

2) How can neural networks learn abstract solution strategies that are independent of the task complexity, data representation and task domain?

3) How can a robot efficiently adapt its movement during execution with a bio-inspired stochastic neural network?

These questions resulted in the three investigated topics arranged in our *Understand-Compute-Adapt* framework, in which we proposed novel approaches for natural skill teaching and task understanding, for learning of transferable abstract strategies, and for the ability of efficient online adaptation during physical interaction.

5.1. Summary

Understand - **Unsupervised Skill Discovery** Chapter 2 tackled the first question, for which we proposed SKID, a probabilistic framework that can be used as a natural interface for demonstrating new tasks to a robot. The framework learns simultaneously to segment trajectories into reoccurring patterns and a skill library to reproduce those patterns without supervision from unlabelled raw trajectories. In addition to these features which enable a natural teaching interface, the framework also learns the structure of the tasks as

relations between skills, i.e., it learns for each skill how likely it follows another skill. This structural knowledge can be used, for example, to reduce the search space of a planning algorithm that uses the learned skills to solve a new task. By equipping a robot with such an *understanding* mechanism, not only teaching the robot becomes feasible for non-experts, but the robot can leverage the learned task structure for predicting, for example, the human behaviour and hence, can incorporate this knowledge in a more intelligent adaptive behaviour.

Compute **- Learning Algorithmic Solutions** In Chapter 3 the second question has been investigated, which deals with the challenge of learning abstract solutions. To characterize abstract strategies, we introduced three requirements for such algorithmic solutions: generalization to novel task instantiations and complexities, the independence from the data representation, and the independence of the task domain. For representing and learning algorithmic solutions fulfilling these criteria, we proposed the Neural Harvard Computer architecture, a modular framework based on memory-augmented networks. This architecture learns abstract solutions in form of algorithmic solutions, that generalize to task complexities far beyond seen during training, and that can be transferred to novel data representations and task domains. Such mechanisms enable robots and intelligent agents to transfer learned strategies to *compute* solutions for unfamiliar situations.

Adapt **- Efficient Online Adaptation** After the mainly cognitive related skills investigated before, Chapter 4 focused on the physical interaction of the robot in the real world. Even with perfectly learned skills and a computed abstract strategy out of those, during execution in the dynamic and complex physical environment, the world, unforeseen situations may occur. To cope with such situations, the robot needs to be able to adapt its motion during the execution and integrate the freshly gained knowledge into its model. For this purpose, we proposed a task-independent efficient online adaptation mechanism based on intrinsic motivation and mental replay, that enables a bio-inspired stochastic neural network to incorporate new knowledge gained during motion execution. With such an *adaptation* mechanism, the robot can deal with unforeseen circumstances like new obstacles, by updating its motion planning model during motion execution, a crucial step towards employing intelligent physical agents into our daily lives.

Altogether, the presented approaches provide potential stepping stones towards intelligent agents operating among us to improve our daily lives. From a natural interface to teach the robot how to tidy up a workshop, to learning transferable strategies to clean up the living room as well, up to efficient online motion adaptation to incorporate the newly

arranged kitchen, the proposed mechanisms show the potential and need of modular and diverse learning approaches to provide human-supporting robots.

5.2. Future Work

The presented approaches tackled crucial features of AI systems that are required for their employment in everyday life, to support humans either in workplaces or at home. Each approach investigated one of three key aspects: the natural teaching of robots, the computation of efficient solutions, and the flexibility through motion adaptation. Yet these challenges are not fully solved and interesting and necessary future research questions arise from the presented results.

The natural teaching interface provided by SKID allows to discover individual skills and their relations from observations of full task executions. While it was evaluated on different datasets including demonstrations recorded from humans, one major challenge is to scale the approach to more complex and realistic scenarios. For example, using vision as input to the system in order to eliminate the need of a tracking system, as vision would be a much more flexible data source in non laboratory environments. Such a vision module inside of SKID could either preprocess the image input into trajectories, or SKID could be extended to handle vision input directly. Another point of potential extension is the choice of the skill (library) representation. The VAE based approach of SKID offers a very general and flexible representation of the skills, other movement primitives may be beneficial for certain tasks. For example, a library consisting of skill encoded as dynamic [159] or probabilistic [160] movement primitives may allow more adaptive individual skills. The challenge is to integrate such representations into the probabilistic model of SKID to allow the simultaneous end-to-end learning. Another possible and interesting extension of SKID could be to change from an *offline* to an *online* setting, i.e., instead of presenting a full trajectory for segmentation and skill learning, the trajectory could be presented iteratively to the model. This could be enabled by integrating online change point detection algorithms and may allow to process longer tasks. As SKID is not restricted to robot (or human) motion data, it is an interesting direction to apply the framework to other data that consists of reoccurring patterns within time series data like, for example, network traffic or audio signals.

Representing and learning of algorithmic solution, i.e., strategies that generalize to any problem instantiation and complexity, and that are transferable to novel data representation and task domains due to their abstraction features, was investigated and realized

by the proposed Neural Harvard Computer (NHC) architecture. The potentially most interesting and promising next step with the NHC could be to investigate the learning of such algorithmic solution from less feedback. By providing only sparse signals on the final output of the learned strategy without constraining intermediate computation steps, the NHC may be able to discover new algorithms. To explore the space of algorithmic solutions in the model with such sparse rewards, intrinsic motivation based exploration may play a key role [161, 162]. Besides investigating the exploration of algorithmic solutions, the structure of the NHC itself is a potential starting point for future research. Instead of fixing the structures of the neural networks encoding the algorithmic modules, these could be learned in addition, minimizing the model complexity by learning the best suited structure for each algorithm and may be incorporated by adding ideas from neural architecture search [163, 164].

Learning to detect and incorporate unforeseen environmental changes like obstacles during motion planning and execution is a key requirement for robots operating among us and was investigated with a bio-inspired stochastic neural network. The neural network that plans the motion was extended with an intrinsic motivation based update mechanism that guides the network to alter its world model according to the perceived feedback, while the efficiency of this update was realized through a mental replay strategy. While the proposed approach is able to adapt the model and the planned motions to incorporate new obstacles in the environment, the currently used signal that guides this update is limited to adapt to the perceived feedback. This means that it can learn to avoid an obstacle, but if the obstacle is removed again, the model is not able to *forget* it. For this purpose, exploration is necessary to actively check the environment for changes and may be achieved by additional exploration focused intrinsic motivation signals [81, 165]. Such active exploration may also be an important step towards online adaptation to failures of the robotic system like broken joints. Another open challenge is to scale the online motion planning and adaptation to higher dimensional spaces by incorporating insights from the model used in higher dimensional spaces in an offline setting without adaptation [11].

Bibliography

[1] Vincent C Müller and Nick Bostrom. “Future progress in artificial intelligence: A survey of expert opinion”. In: *Fundamental issues of artificial intelligence*. Springer, 2016, pp. 555–572.

[2] Spyros Makridakis. “The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms”. In: *Futures* 90 (2017), pp. 46–60.

[3] Ross Gruetzemacher, David Paradice, and Kang Bok Lee. “Forecasting Transformative AI: An Expert Survey”. In: *arXiv preprint arXiv:1901.08579* (2019).

[4] Joshua B Tenenbaum et al. “How to grow a mind: Statistics, structure, and abstraction”. In: *Science* 331.6022 (2011), pp. 1279–1285.

[5] Brenden M Lake et al. “Building machines that learn and think like people”. In: *Behavioral and Brain Sciences* 40 (2017).

[6] Alan M Turing. “Computing machinery and intelligence”. In: *Parsing the Turing Test*. Springer, 2009, pp. 23–65.

[7] John R Searle. “Minds, brains, and programs”. In: *Behavioral and brain sciences* 3.3 (1980), pp. 417–424.

[8] Daniel Tanneberg, Elmar Rueckert, and Jan Peters. “SKID RAW: Skill Discovery from Raw Trajectories”. In: *IEEE Robotics and Automation Letters* (in preparation).

[9] Daniel Tanneberg, Elmar Rueckert, and Jan Peters. “Learning Algorithmic Solutions to Symbolic Planning Tasks with a Neural Computer Architecture”. In: *arXiv* (2019).

[10] Daniel Tanneberg, Elmar Rueckert, and Jan Peters. “Evolutionary Training and Abstraction Yields Algorithmic Generalization of Neural Computers”. In: *Nature Machine Intelligence* (in press).

[11] Daniel Tanneberg et al. “Deep spiking networks for model-based planning in humanoids”. In: *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. 2016.

[12] Daniel Tanneberg, Jan Peters, and Elmar Rueckert. “Online Learning with Stochastic Recurrent Neural Networks using Intrinsic Motivation Signals”. In: *Conference on Robot Learning*. 2017.

[13] Daniel Tanneberg, Jan Peters, and Elmar Rueckert. “Efficient Online Adaptation with Stochastic Recurrent Neural Networks”. In: *IEEE-RAS 17th International Conference on Humanoid Robots (Humanoids)*. 2017.

[14] Daniel Tanneberg, Jan Peters, and Elmar Rueckert. “Intrinsic motivation and mental replay enable efficient online adaptation in stochastic recurrent networks”. In: *Neural Networks* 109 (2019), pp. 67–80.

[15] Brenna D Argall et al. “A survey of robot learning from demonstration”. In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[16] Rudolf Lioutikov et al. “Learning movement primitive libraries through probabilistic segmentation”. In: *The International Journal of Robotics Research* 36.8 (2017), pp. 879–894.

[17] Scott Niekum et al. “Learning and generalization of complex tasks from unstructured demonstrations”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5239–5246.

[18] Naama Kadmon Harpaz et al. “Movement decomposition in the primary motor cortex”. In: *Cerebral Cortex* 29.4 (2019), pp. 1619–1633.

[19] Michael A Goodrich and Alan C Schultz. *Human-robot interaction: a survey*. Now Publishers Inc, 2008.

[20] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *International Conference on Learning Representations*. 2014.

[21] SM Ali Eslami et al. “Attend, infer, repeat: Fast scene understanding with generative models”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3225–3233.

[22] Takayuki Osa and Shuhei Ikemoto. “Variational Autoencoder Trajectory Primitives with Continuous and Discrete Latent Codes”. In: *arXiv preprint arXiv:1912.04063* (2019).

[23] Michael Noseworthy et al. “Task-Conditioned Variational Autoencoders for Learning Movement Primitives”. In: *Conference on Robot Learning*. 2020, pp. 933–944.

[24] Franz Steinmetz, Verena Nitsch, and Freek Stulp. “Intuitive Task-Level Programming by Demonstration Through Semantic Skill Recognition”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3742–3749.

[25] Simon Manschitz et al. “Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations”. In: *Robotics and Autonomous Systems* 74 (2015), pp. 97–107.

[26] Tanmay Shankar and Abhinav Gupta. “Learning Robot Skills with Temporal Variational Inference”. In: *International Conference on Machine Learning*. 2020.

[27] Zhe Su et al. “Learning manipulation graphs from demonstrations using multimodal sensory signals”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2758–2765.

[28] Huiwen Zhang et al. “Complex sequential tasks learning with Bayesian inference and Gaussian mixture model”. In: *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2018, pp. 1927–1934.

[29] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial transformer networks”. In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[30] Irina Higgins et al. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”. In: *International Conference on Learning Representations*. 2017.

[31] Christopher P Burgess et al. “Understanding disentangling in beta-VAE”. In: *arXiv preprint arXiv:1804.03599* (2018).

[32] Adji B Dieng et al. “Avoiding latent variable collapse with generative skip models”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. 2019, pp. 2397–2405.

[33] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *International Conference on Learning Representations*. 2017.

[34] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The concrete distribution: A continuous relaxation of discrete random variables”. In: *International Conference on Learning Representations*. 2017.

[35] Emilien Dupont. “Learning disentangled joint continuous and discrete representations”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 710–720.

[36] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. “Importance weighted autoencoders”. In: *International Conference on Learning Representations*. 2016.

[37] Dorothea Koert et al. “Learning Intention Aware Online Adaptation of Movement Primitives”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3719–3726.

[38] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations* (2015).

[39] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *International Conference on Learning Representations* (2019).

[40] Ryan Prescott Adams and David JC MacKay. “Bayesian online changepoint detection”. In: *arXiv preprint arXiv:0710.3742* (2007).

[41] Diego Agudelo-España et al. “Bayesian Online Prediction of Change Points”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 320–329.

[42] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey”. In: *Journal of Machine Learning Research* 10.Jul (2009), pp. 1633–1685.

[43] Daniel L Silver, Qiang Yang, and Lianghao Li. “Lifelong Machine Learning Systems: Beyond Learning Algorithms.” In: *AAAI Spring Symposium: Lifelong Machine Learning*. Vol. 13. 2013, p. 05.

[44] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big data* 3.1 (2016), p. 9.

[45] German I Parisi et al. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* (2019).

[46] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

[47] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[48] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), p. 484.

[49] Hassan Ismail Fawaz et al. “Deep learning for time series classification: a review”. In: *Data Mining and Knowledge Discovery* 33.4 (2019), pp. 917–963.

[50] Mathew Botvinick et al. “Reinforcement learning, fast and slow”. In: *Trends in cognitive sciences* (2019).

[51] Li Liu et al. “Deep learning for generic object detection: A survey”. In: *International journal of computer vision* 128.2 (2020), pp. 261–318.

[52] George Konidaris. “On the necessity of abstraction”. In: *Current opinion in behavioral sciences* 29 (2019), pp. 1–7.

[53] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[54] Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. “Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory”. In: *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*. 1992, p. 14.

[55] Michael C Mozer and Sreerupa Das. “A connectionist symbol manipulator that discovers the structure of context-free languages”. In: *Advances in neural information processing systems*. 1993, pp. 863–870.

[56] Zheng Zeng, Rodney M Goodman, and Padhraic Smyth. “Discrete recurrent neural networks for grammatical inference”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 320–330.

[57] Alex Graves, Greg Wayne, and Ivo Danihelka. “Neural turing machines”. In: *arXiv preprint arXiv:1410.5401* (2014).

[58] Armand Joulin and Tomas Mikolov. “Inferring algorithmic patterns with stack-augmented recurrent nets”. In: *Advances in neural information processing systems*. 2015, pp. 190–198.

[59] Alex Graves et al. “Hybrid computing using a neural network with dynamic external memory”. In: *Nature* 538.7626 (2016), p. 471.

[60] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. “Neural programmer: Inducing latent programs with gradient descent”. In: *International Conference on Learning Representations* (2016).

[61] Łukasz Kaiser and Ilya Sutskever. “Neural GPUs learn algorithms”. In: *International Conference on Learning Representations* (2016).

[62] Wojciech Zaremba et al. “Learning simple algorithms from examples”. In: *International Conference on Machine Learning*. 2016, pp. 421–429.

[63] Rasmus Boll Greve, Emil Juul Jacobsen, and Sebastian Risi. “Evolving neural turing machines for reward-based learning”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM. 2016, pp. 117–124.

[64] Andrew Trask et al. “Neural arithmetic logic units”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 8035–8044.

[65] Andreas Madsen and Alexander Rosenberg Johansen. “Neural Arithmetic Units”. In: *International Conference on Learning Representations*. 2020.

[66] Hung Le, Truyen Tran, and Svetha Venkatesh. “Neural Stored-program Memory”. In: *International Conference on Learning Representations*. 2020.

[67] Scott Reed and Nando De Freitas. “Neural programmer-interpreters”. In: *International Conference on Learning Representations* (2016).

[68] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. “Neural random-access machines”. In: *International Conference on Learning Representations* (2016).

[69] Jonathon Cai, Richard Shin, and Dawn Song. “Making neural programming architectures generalize via recursion”. In: *International Conference on Learning Representations* (2017).

[70] Honghua Dong et al. “Neural Logic Machines”. In: *International Conference on Learning Representations*. 2019.

[71] Petar Velickovic et al. “Neural execution of graph algorithms”. In: *International Conference on Learning Representations*. 2020.

[72] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. “End-to-end memory networks”. In: *Advances in neural information processing systems*. 2015, pp. 2440–2448.

[73] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory Networks”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[74] Edward Grefenstette et al. “Learning to transduce with unbounded memory”. In: *Advances in neural information processing systems*. 2015, pp. 1828–1836.

[75] Ankit Kumar et al. “Ask me anything: Dynamic memory networks for natural language processing”. In: *International Conference on Machine Learning*. 2016, pp. 1378–1387.

[76] Greg Wayne et al. “Unsupervised Predictive Memory in a Goal-Directed Agent”. In: *arXiv preprint arXiv:1803.10760* (2018).

[77] Jakob Merrild, Mikkel Angaju Rasmussen, and Sebastian Risi. “HyperNTM: evolving scalable neural Turing machines through HyperNEAT”. In: *International Conference on the Applications of Evolutionary Computation*. Springer. 2018, pp. 750–766.

[78] Shauharda Khadka, Jen Jen Chung, and Kagan Tumer. “Neuroevolution of a modular memory-augmented neural network for deep memory problems”. In: *Evolutionary computation* 27.4 (2019), pp. 639–664.

[79] Yoshua Bengio et al. “Curriculum learning”. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[80] Daan Wierstra et al. “Natural evolution strategies.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 949–980.

[81] Pierre-Yves Oudeyer and Frederic Kaplan. “What is intrinsic motivation? A typology of computational approaches”. In: *Frontiers in neurorobotics* 1 (2009), p. 6.

[82] Gianluca Baldassarre and Marco Mirolli. “Intrinsically motivated learning systems: an overview”. In: *Intrinsically motivated learning in natural and artificial systems*. Springer, 2013, pp. 1–14.

[83] Horia Mania, Aurelia Guy, and Benjamin Recht. “Simple random search of static linear policies is competitive for reinforcement learning”. In: *Advances in Neural Information Processing Systems 31*. 2018, pp. 1803–1812.

[84] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[85] Tim Salimans et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017).

[86] Anders Krogh and John A Hertz. “A simple weight decay can improve generalization”. In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[87] Edoardo Conti et al. “Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 5027–5038.

[88] Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[89] Long-Ji Lin. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine learning* 8.3-4 (1992), pp. 293–321.

[90] Tillman Weyde and Radha Manisha Kopparti. “Feed-Forward Neural Networks Need Inductive Bias to Learn Equality Relations”. In: *arXiv preprint arXiv:1812.01662* (2018).

[91] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[92] Max Lungarella et al. “Developmental robotics: a survey”. In: *Connection Science* 15.4 (2003), pp. 151–190.

[93] Jürgen Schmidhuber. “Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts”. In: *Connection Science* 18.2 (2006), pp. 173–187.

[94] Minoru Asada et al. “Cognitive developmental robotics: A survey”. In: *IEEE Transactions on Autonomous Mental Development* 1.1 (2009), pp. 12–34.

[95] Sebastian Thrun and Tom M Mitchell. “Lifelong robot learning”. In: *Robotics and autonomous systems* 15.1-2 (1995), pp. 25–46.

[96] Mark B Ring. “CHILD: A first step towards continual learning”. In: *Machine Learning* 28.1 (1997), pp. 77–104.

[97] Paul Ruvolo and Eric Eaton. “ELLA: An efficient lifelong learning algorithm”. In: *International Conference on Machine Learning*. 2013, pp. 507–515.

[98] Juyang Weng et al. “Autonomous mental development by robots and animals”. In: *Science* 291.5504 (2001), pp. 599–600.

[99] Juyang Weng. “Developmental robotics: Theory and experiments”. In: *International Journal of Humanoid Robotics* 1.02 (2004), pp. 199–236.

[100] Abdelhamid Tayebi. “Adaptive iterative learning control for robot manipulators”. In: *Automatica* 40.7 (2004), pp. 1195–1203.

[101] Douglas A Bristow, Marina Tharayil, and Andrew G Alleyne. “A survey of iterative learning control”. In: *IEEE Control Systems* 26.3 (2006), pp. 96–114.

[102] Dongbing Gu and Huosheng Hu. “Neural predictive control for a car-like mobile robot”. In: *Robotics and Autonomous Systems* 39.2 (2002), pp. 73–86.

[103] Manuel Krause et al. “Stabilization of the capture point dynamics for bipedal walking based on model predictive control”. In: *IFAC Proceedings Volumes* 45.22 (2012), pp. 165–171.

[104] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[105] Aurelien Ibanez, Philippe Bidaud, and Vincent Padois. “Emergence of humanoid walking behaviors from mixed-integer model predictive control”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014, pp. 4014–4021.

[106] Lorenzo Jamone et al. “Autonomous online learning of reaching behavior in a humanoid robot”. In: *International Journal of Humanoid Robotics* 9.03 (2012), p. 1250017.

[107] Seung-Joon Yi et al. “Online learning of a full body push recovery controller for omnidirectional walking”. In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE. 2011, pp. 1–6.

[108] Micha Hersch, Eric Sauser, and Aude Billard. “Online learning of the body schema”. In: *International Journal of Humanoid Robotics* 5.02 (2008), pp. 161–181.

[109] René Felix Reinhart and Jochen Jakob Steil. “Neural learning and dynamical selection of redundant solutions for inverse kinematic control”. In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference On*. IEEE. 2011, pp. 564–569.

[110] Tim Waegeman, Francis Wyffels, and Benjamin Schrauwen. “Feedback control by online learning an inverse model”. In: *IEEE transactions on neural networks and learning systems* 23.10 (2012), pp. 1637–1648.

[111] Richard M Ryan and Edward L Deci. “Intrinsic and extrinsic motivations: Classic definitions and new directions”. In: *Contemporary educational psychology* 25.1 (2000), pp. 54–67.

[112] Richard M Ryan and Edward L Deci. “Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being.” In: *American psychologist* 55.1 (2000), p. 68.

[113] Robert W White. “Motivation reconsidered: The concept of competence.” In: *Psychological review* 66.5 (1959), p. 297.

[114] Andrew G Barto, Satinder Singh, and Nuttapong Chentanez. “Intrinsically motivated learning of hierarchical collections of skills”. In: *Proceedings of the 3rd International Conference on Development and Learning*. 2004, pp. 112–19.

[115] Gianluca Baldassarre. “What are intrinsic motivations? A biological perspective”. In: *Proceedings of the International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob-2011)*. Ed. by Angelo Cangelosi et al. Frankfurt am Main, Germany, 24–27/08/11. New York, NY: IEEE, 2011, E1–8.

[116] Gianluca Baldassarre and Marco Mirolli, eds. *Intrinsically motivated learning in natural and artificial systems*. Berlin: Springer, 2013. ISBN: 978-3-642-32375-1 978-3-642-32374-4. DOI: 10.1007/978-3-642-32375-1.

[117] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge, 1998.

[118] Andrew Stout, George D Konidaris, and Andrew G Barto. *Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning*. Tech. rep. MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE, 2005.

[119] Ulrich Nehmzow et al. “Novelty detection as an intrinsic motivation for cumulative learning robots”. In: *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer, 2013, pp. 185–207.

[120] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli. “Intrinsic motivation signals for driving the acquisition of multiple tasks: a simulated robotic study”. In: *Proceedings of the 12th International Conference on Cognitive Modelling (ICCM)*. 2013, pp. 1–6.

[121] Andrew G Barto and Sridhar Mahadevan. “Recent advances in hierarchical reinforcement learning”. In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379.

[122] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[123] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. “Intrinsic motivation systems for autonomous mental development”. In: *IEEE transactions on evolutionary computation* 11.2 (2007), pp. 265–286.

[124] Stephen Hart and Roderic Grupen. “Learning generalizable control programs”. In: *IEEE Transactions on Autonomous Mental Development* 3.3 (2011), pp. 216–231.

[125] Elmar Rueckert et al. “Recurrent spiking networks solve planning tasks”. In: *Scientific reports* 6 (2016), p. 21142.

[126] Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. “Optimal control as a graphical model inference problem”. In: *Machine learning* 87.2 (2012), pp. 159–182.

[127] Matthew Botvinick and Marc Toussaint. “Planning as inference”. In: *Trends in cognitive sciences* 16.10 (2012), pp. 485–488.

[128] Elmar Rueckert et al. “Learned Graphical Models for Probabilistic Planning Provide a New Class of Movement Primitives”. In: *Frontiers in Computational Neuroscience* 6.97 (2012).

[129] Leon Festinger. “Cognitive dissonance.” In: *Scientific American* (1962).

[130] Jerome Kagan. “Motives and development.” In: *Journal of personality and social psychology* 22.1 (1972), p. 51.

[131] Pierre-Yves Oudeyer and Frederic Kaplan. “What is intrinsic motivation? A typology of computational approaches”. In: *Frontiers in neurorobotics* 1 (2007).

[132] David J Foster and Matthew A Wilson. “Reverse replay of behavioural sequences in hippocampal place cells during the awake state”. In: *Nature* 440.7084 (2006), p. 680.

[133] J Michael Herrmann, Klaus Pawelzik, and Theo Geisel. “Learning predictive representations”. In: *Neurocomputing* 32 (2000), pp. 785–791.

[134] Frederic Kaplan and Pierre-Yves Oudeyer. “Motivational principles for visual know-how development”. In: *Proceedings of the Third International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*. 2003, pp. 73–80.

[135] Jan H Metzen and Frank Kirchner. “Incremental learning of skill collections based on intrinsic motivation”. In: *Frontiers in neurorobotics* 7 (2013).

[136] Andrew Stout and Andrew G Barto. “Competence progress intrinsic motivation”. In: *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*. IEEE. 2010, pp. 257–262.

[137] Matthias Rolf, Jochen J. Steil, and Michael Gienger. “Goal babbling permits direct learning of inverse kinematics”. In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010), pp. 216–229.

[138] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli. “GRAIL: a Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning”. In: *IEEE Transactions on Cognitive and Developmental Systems* 8.3 (2016), pp. 214–231. DOI: 10.1109/TCDS.2016.2538961. URL: http://ieeexplore.ieee.org/document/7470616/.

[139] Jürgen Schmidhuber. “Formal theory of creativity, fun, and intrinsic motivation (1990–2010)”. In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010), pp. 230–247.

[140] A. Barto, M. Mirolli, and G. Baldassarre. “Novelty or surprise?” In: *Frontiers in Psychology – Cognitive Science* 4.907 (2013), e1–15. DOI: 10.3389/fpsyg.2013.00907.

[141] Brad E Pfeiffer and David J Foster. “Hippocampal place cell sequences depict future paths to remembered goals”. In: *Nature* 497.7447 (2013), p. 74.

[142] Lars Buesing et al. “Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons”. In: *PLoS computational biology* 7.11 (2011), e1002211.

[143] Johanni Brea, Walter Senn, and Jean-Pascal Pfister. “Sequence learning with hidden units in spiking neural networks”. In: *Advances in neural information processing systems*. 2011, pp. 1422–1430.

[144] David Kappel, Bernhard Nessler, and Wolfgang Maass. “STDP installs in winner-take-all circuits an online approximation to hidden Markov model learning”. In: *PLoS computational biology* 10.3 (2014), e1003511.

[145] Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. “Design principles of the hippocampal cognitive map”. In: *Advances in neural information processing systems*. 2014, pp. 2528–2536.

[146] Wulfram Gerstner and Werner M Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[147] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[148] SM Stringer et al. “Self-organizing continuous attractor networks and path integration: one-dimensional models of head direction cells”. In: *Network: Computation in Neural Systems* 13.2 (2002), pp. 217–242.

[149] Hao-Tien Chiang et al. “Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2347–2354.

[150] Uğur M Erdem and Michael Hasselmo. “A goal-directed spatial navigation model using forward trajectory planning based on grid cells”. In: *European Journal of Neuroscience* 35.6 (2012), pp. 916–931.

[151] Min Cheol Lee and Min Gyu Park. “Artificial potential field based path planning for mobile robots using a virtual obstacle concept”. In: *2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Vol. 2. IEEE. 2003, pp. 735–740.

[152] Jerome Barraquand, Bruno Langlois, and J-C Latombe. “Numerical potential field techniques for robot path planning”. In: *IEEE transactions on systems, man, and cybernetics* 22.2 (1992), pp. 224–241.

[153] Shuzhi Sam Ge and Yun J Cui. “Dynamic motion planning for mobile robots using potential field method”. In: *Autonomous robots* 13.3 (2002), pp. 207–222.

[154] David Johan Christensen, Ulrik Pagh Schultz, and Kasper Stoy. “A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots”. In: *Robotics and Autonomous Systems* 61.9 (2013), pp. 1021–1035.

[155] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), pp. 503–507.

[156] Bhavna Kulkarni et al. “Arthritic pain is processed in brain areas concerned with emotions and fear”. In: *Arthritis & Rheumatology* 56.4 (2007), pp. 1345–1354.

[157] Maaike Leeuw et al. “The fear-avoidance model of musculoskeletal pain: current state of scientific evidence”. In: *Journal of behavioral medicine* 30.1 (2007), pp. 77–94.

[158] P-Y Oudeyer, Jacqueline Gottlieb, and Manuel Lopes. “Intrinsic motivation, curiosity, and learning: Theory and applications in educational technologies”. In: *Progress in brain research* 229 (2016), pp. 257–284.

[159] Auke Jan Ijspeert et al. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.

[160] Alexandros Paraschos et al. “Probabilistic movement primitives”. In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[161] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. “Intrinsically motivated reinforcement learning”. In: *Advances in neural information processing systems*. 2005, pp. 1281–1288.

[162] Andrew G Barto. “Intrinsic motivation and reinforcement learning”. In: *Intrinsically motivated learning in natural and artificial systems*. Springer, 2013, pp. 17–47.

[163] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20 (2019), pp. 1–21.

[164] Kenneth O Stanley et al. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (2019), pp. 24–35.

[165] Guido Schillaci et al. “Intrinsic motivation and episodic memories for robot exploration of high-dimensional sensory spaces”. In: *Adaptive Behavior* (2020).

A. Publication List

Journal Papers

1. **Tanneberg, D.**; Rueckert, E.; Peters, J. (in preparation). **SKID RAW: Skill Discovery from Raw Trajectories**, *IEEE Robotics and Automation Letters*
2. **Tanneberg, D.**; Rueckert, E.; Peters, J. (in press). **Evolutionary Training and Abstraction Yields Algorithmic Generalization of Neural Computers**, *Nature Machine Intelligence*
3. **Tanneberg, D.**; Peters, J.; Rueckert, E. (2019). **Intrinsic Motivation and Mental Replay enable Efficient Online Adaptation in Stochastic Recurrent Networks**, *Neural Networks*, 109, pp.67-80.
4. van Hoof, H.; **Tanneberg, D.**; Peters, J. (2017). **Generalized Exploration in Policy Search**, *Machine Learning (MLJ)*, 106, 9-10, pp.1705-1724.
5. Rueckert, E.; Kappel, D.; **Tanneberg, D.**; Pecevski, D; Peters, J. (2016). **Recurrent Spiking Networks Solve Planning Tasks**, *Nature PG: Scientific Reports*, 6, 21142, Nature PG.

Conference Papers

1. Keller, L.; **Tanneberg, D.**; Stark, S.; Peters, J. (2020). **Model-Based Quality-Diversity Search for Efficient Robot Learning**, *International Conference on Intelligent Robots and Systems (IROS)*.
2. **Tanneberg, D.**; Rueckert, E.; Peters, J. (*2019*). **Learning Algorithmic Solutions to Symbolic Planning Tasks with a Neural Computer Architecture**, *arXiv*.

3. **Tanneberg, D.**; Peters, J.; Rueckert, E. (2017). **Online Learning with Stochastic Recurrent Neural Networks using Intrinsic Motivation Signals**, *Proceedings of the Conference on Robot Learning (CoRL)*.
4. **Tanneberg, D.**; Peters, J.; Rueckert, E. (2017). **Efficient Online Adaptation with Stochastic Recurrent Neural Networks**, *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*.
5. **Tanneberg, D.**; Paraschos, A.; Peters, J.; Rueckert, E. (2016). **Deep Spiking Networks for Model-based Planning in Humanoids**, *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*.

Workshop & Symposium Papers

1. Delfosse, Q.; Stark, S.; **Tanneberg, D.**; Santucci, V.; Peters, J. (2019). **Open-Ended Learning of Grasp Strategies using Intrinsically Motivated Self-Supervision**, *Workshop at the International Conference on Intelligent Robots and Systems (IROS)*.
2. Thiem, S.; Stark, S.; **Tanneberg, D.**; Peters, J.; Rueckert, E. (2017). **Simulation of the underactuated Sake Robotics Gripper in V-REP**, *Workshop at the International Conference on Humanoid Robots (HUMANOIDS)*.
3. Sharma, D.; **Tanneberg, D.**; Grosse-Wentrup, M.; Peters, J.; Rueckert, E. (2016). **Adaptive Training Strategies for BCIs**, *Cybathlon Symposium*.
4. Friess, T.; Fiebig, K.H.; Sharma, D.; Faber, N.; Hesse, T.; **Tanneberg, D.**; Peters, J.; Grosse-Wentrup, M. (2016). **Personalized Brain-Computer Interfaces for Non-Laboratory Environments**, *Cybathlon Symposium*.

B. Curriculum Vitae

Education

Ph.D. Computer Science (ongoing) — **TU Darmstadt**
Machine Learning & Robotics @ Intelligent Autonomous Systems Lab — *since Oct 2015*

Master of Science (with honors) — **TU Darmstadt**
Computer Science, Focus: Machine Learning | Minor: Biological Psychology — *2013 – 2015*

Bachelor of Science — **TU Darmstadt**
Computer Science — *2008 – 2013*

University qualification (german: Abitur) — **BS Gelnhausen**
Data Processing Technology — *2007*

Honors and Awards

Hanns-Voith-Stiftungspreis Award 2017 (Master Thesis)

Teaching

Machine Learning I Lecture, Teaching Assistant, TU Darmstadt (*Summer 2018*)

Robot Learning Lecture, Teaching Assistant, TU Darmstadt (*Winter 2017/2018*)

Robot Learning Project Class, Teaching Assistant, TU Darmstadt (*Summer 2017*)

Student Supervision

2020 Student Project; Keller, L.; (joint supervision with Svenja Stark)
Model-Based Quality-Diversity Search for Efficient Robot Learning

2019 Honor Thesis; Hussing M.;
Exploration through Intrinsic Motivation in Stochastic Recurrent Networks

2019 Master Thesis; Delfosse, Q.; (joint supervision with Svenja Stark)
Grasping Objects Using a Goal-Discovering Architecture for Intrinsically-Motivated Learning

2019 Student Project; Keller, L.; (joint supervision with Svenja Stark)
Learning object manipulation skills through novelty search and local adaptation

2019 Student Project; Hussing, M.;
Online Adaptation, Exploration through Forgetting in Stochastic Recurrent Networks

2019 Master Thesis; Wölker A.;
Local Pixel Manipulation Detection with Deep Neural Networks

2018 Student Project; Hu, Z.; Lölkes, C.; Yang, H.; (joint supervision with Svenja Stark)
Learning Symbolic Representations for Abstract High-Level Planning

2017 Master Thesis; Sharma D.; (joint supervision with Elmar Rückert)
Adaptive Training Strategies for Brain Computer Interfaces

2017 Bachelor Thesis; Polat, H.; (joint supervision with Elmar Rückert)
Nonparametric deep neural networks for movement planning

2016 Bachelor Thesis; Plage, L. M.;
Reinforcement Learning for tactile-based finger gaiting

Reviewing

2016 International Joint Conference on Artificial Intelligence (**IJCAI**)
2016 Frontiers in Computational Neuroscience
2016 Conference on Neural Information Processing Systems (**NIPS**)

2017 PLOS Computational Biology
2018 Neural Computation
2018 Robotics: Science and Systems (**R:SS**)
2018 Conference on Robot Learning (**CoRL**)
2019 IEEE Transactions on Neural Networks and Learning Systems (**TNNLS**)
2019 International Conference on Machine Learning (**ICML**)
2019 IEEE Robotics and Automation Letters (**RA-L**)
2020 IEEE International Conference on Robotics and Automation (**ICRA**)
2020 IEEE International Conference on Intelligent Robots and Systems (**IROS**)