Deep Learning Based Approaches for Imitation Learning

Ahmed Hussein

A report submitted as part of the requirements for the degree of PhD in Computing: Artificial Intelligence at the School of Computing
Robert Gordon University
Aberdeen, Scotland

May 2018

Supervisor Dr. Eyad Elyan

Abstract

Imitation learning refers to an agent's ability to mimic a desired behaviour by learning from observations. The field is rapidly gaining attention due to recent advances in computational and communication capabilities as well as rising demand for intelligent applications. The goal of imitation learning is to describe the desired behaviour by providing demonstrations rather than instructions. This enables agents to learn complex behaviours with general learning methods that require minimal task specific information. However, imitation learning faces many challenges. The objective of this thesis is to advance the state of the art in imitation learning by adopting deep learning methods to address two major challenges of learning from demonstrations.

Firstly, representing the demonstrations in a manner that is adequate for learning. We propose novel Convolutional Neural Networks (CNN) based methods to automatically extract feature representations from raw visual demonstrations and learn to replicate the demonstrated behaviour. This alleviates the need for task specific feature extraction and provides a general learning process that is adequate for multiple problems. The second challenge is generalizing a policy over unseen situations in the training demonstrations. This is a common problem because demonstrations typically show the best way to perform a task and don't offer any information about recovering from sub-optimal actions. Several methods are investigated to improve the agent's generalization ability based on its initial performance. Our contributions in this area are three fold. Firstly, we propose an active data aggregation method that queries the demonstrator in situations of low confidence. Secondly, we investigate combining learning from demonstrations and reinforcement learning. A deep reward shaping method is proposed that learns a potential reward function from demonstrations. Finally, memory architectures in deep neural networks are investigated to provide context to the agent when taking actions. Using recurrent neural networks addresses the dependency between the state-action sequences taken by the agent.

The experiments are conducted in simulated environments on 2D and 3D navigation tasks that are learned from raw visual data, as well as a 2D soccer simulator. The proposed methods are compared to state of the art deep reinforcement learning methods. The results show that deep learning architectures can learn suitable representations from raw visual data and effectively map them to atomic actions. The proposed methods for addressing generalization show improvements over using supervised learning and reinforcement learning alone. The results are thoroughly analysed to identify the benefits of each approach and situations in which it is most suitable.

Acknowledgements

Firstly, I would like to express my sincere gratitude to Dr. Eyad Elyan for his mentorship, motivation and support throughout my PhD. I would also like to thank the rest of my supervisory team Dr. Mohamed Medhat Gaber and Prof. Chrisina Jayne for their continuous guidance and enthusiastic support. I thank my fellow labmates for their stimulating discussions, moral support and for the long and fun nights we shared at the lab. I would like to particularly thank Jeremy Close and Pamela Johnston for their kind support during this thesis. Last but not least, I would like to thank my parents whose guidance and love have helped me throughout the PhD as they do in life.

Declaration

I confirm that the work contained in this PhD project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed .... Date ....

Ahmed Hussein

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In recent years, the demand for intelligent agents capable of mimicking human behaviour has grown substantially. Advancement in robotics and communication technology have given rise to many potential applications that need artificial intelligence that can not only make predictions, but is able to perform sequences of actions that reflect robust, intelligent behaviour. Many future directions in technology rely on the ability of artificial intelligence agents to behave as a human would when presented with the same situation. Examples of such fields are self-driving vehicles, assistive robots and human computer interaction

Most applications are dynamic and involve many variables and are therefore not suitable for manually designed policies. For examples, self driving cars should be able to robustly act in real urban conditions, where the roads are shared with other vehicles and pedestrians that can behave unexpectedly. It is not feasible to take into account all the possibilities that may face the autonomous vehicle when manually programming a policy. It is also difficult to break down and articulate how humans perform tasks in order to program intelligent agents to replicate this behaviour. Sticking with the autonomous vehicle example, it is hard for an experienced driver to describe to another human how to drive well. A more intuitive and effective method of imparting this knowledge is to show the student examples of good driving.

Imitation learning, also called learning from demonstrations, is a paradigm where an intelligent agent is taught to mimic human behaviour by supplying the agent with demonstrations provided by a teacher rather than instructions. By learning from demonstration, the agent does not require explicit knowledge about the task or the environment such as objectives or constraints. Rather, a general learning process is advocated where all needed information is inferred from the provided demonstrations. For example, an autonomous vehicle doesn't require knowledge of how the vehicle works, how the other agents in the environment are modelled or that it shouldn't drive over the pavement. Instead The policy learns to operate the vehicles controls as demonstrated by the teacher without knowing the underlying mechanics of these controls or the rules of the road; and learns to adapt to the dynamic environment by observing the interaction between the teacher and the environment through the demonstrated behaviour. This thesis explores deep learning approaches for training intelligent agents from demonstrations.

1.1 Motivation

Learning by imitation has gained a lot of attention due to its relevance to a wide range of potential applications where machines are required to behave intelligently (like a human would). Inspired by neuroscience, this field aims to mimic the process of learning by imitation in humans and animals [1] and is regarded from early stage as an integral part of machine intelligence [2]. Several other approaches exist for learning intelligent behaviour, commonly formulating the desired task as an optimization problem. These are goal oriented methods where the agent is required to achieve a certain target behaviour without any information about how to do it. However, it has become widely accepted that having prior knowledge provided by an expert is more effective and efficient than searching for a solution from scratch [2, 3, 4, 1].

While Learning from experience can produce robust policies that generalize to dynamic scenarios, finding a solution through trial and error may take too long. Especially in problems that require performing long trajectories of actions with delayed rewards, which is common in many real life applications. In such cases it may be extremely difficult to reach rewards by chance; and the time to learn a policy to maximize the rewards exponentially increases. For example in a game of football, the target is to score more goals than your opponent. However, scoring a goal requires a complicated sequence of actions, taking into account the other agent's in the environment. It is therefore extremely unlikely that a goal will be scored using trial and error alone. Demonstrations of how the game is played provides invaluable information for the agent in this case. Such information can be imparted in other ways than demonstrations, such as rules or sub goals but may lack the generalization and simplicity to generate of demonstrations. Moreover, learning through trial and error may result in a policy that solves the problem differently to how a human would. Performing a task in a manner that is intuitive to a human observer may be crucial in applications where humans and intelligent agents interact together in an environment [5]. Nass et al [6] suggest that humans view computers interacting with them as social agents and that humans interact with them in a manner derived from their experiences interacting with other humans. Therefore, even with the conscious knowledge that an agent is not a human, interaction is improved when the agent behaves in a way that is familiar to its human counterpart. This also emphasises that it is intuitive for humans to provide demonstrations to teach intelligent agents.

On the other hand learning from demonstrations may result in faster learning and produce a policy that follows the teacher's way of solving the task. However, learning a direct mapping between observation and action can commonly result in a policy that generalizes poorly to unseen scenarios. The supervised policy only learns to deal with situations covered in the demonstrations. Since demonstrations only cover the optimal trajectory, if the agent deviates even slightly from that trajectory at any point, it finds itself in an unseen situation not covered by the training data [7]. For example, if an autonomous vehicle trained to drive in the middle of the lane veers slightly towards the lane line (which is expected in a machine learning application), the agent has no information on how to recover this error. A more pronounced case is unexpected behaviour on the road such as an accident up ahead or any of the countless cases that may not have been included in the demonstrations. The agent has no information on how to behave in such situations. So essentially the policy is trained using samples from a distribution that is different to the one it is evaluated on. Therefore, in many cases, policies need to be refined based on the performance of the initially learned policy. Moreover, supervised learning needs a sufficient number of demonstrations; which for deep network architectures may be large. These cases highlight the limitations of imitation learning; in that it depends on the demonstrations. The demonstrations may not be sufficient or contain faults. The pros and cons of learning from experience and demonstrations motivate combining both approaches; which is not dissimilar to how humans and animals partly learn [3].

Both learning from demonstrations and experience approaches aim to create a general learning process that requires minimal task specific information. In contrast to hand crafted methods, ideally all the information needed by the learner would be incorporated in the demonstrations or the reward function. One aspect of learning that usually requires knowledge of the task is creating the feature representations used for learning. Features are engineered to best represent a specific task or environment. Manually designing features is time consuming and can be in efficient if we need to apply the agent in different environments. Deep learning methods have shown great success in learning from high dimensional raw data in a variety of applications. Automatically extracting feature representations can greatly facilitate creating general learning processes for learning from demonstration. Thus, the same network architecture can extract relevant features for different situations depending on the provided demonstrations.

1.2 Objectives

The main objective of this thesis is to expand the state of the art in autonomous agents through novel imitation learning methods. To achieve this target, sub-objectives are specified that dictate the directions taken in this body of work.

- The first objective is to review imitation learning methods for autonomous agents. Imitation learning is a multi-disciplinary field and the literature is filled with diverse approaches that consider the challenges of intelligent autonomous agents from different points of view. This survey aims to categories learning methods that utilize demonstrations and bridge gap between the various methods that tackle the same problem from different perspectives. Reviewing related literature helps identify important research areas and promising directions.

- Representing demonstrations is a key challenge in imitation learning. We aim to create general methods for creating feature representations from raw sensory information without requiring task specific knowledge. In particular we target demonstrations from visual information and sequences.

- Generalization is another important challenge as the agent is required to adapt beyond the provided demonstrations. This thesis investigates several approaches to improve generalization. The target is to create autonomous agents that demonstrate robust behaviour in dynamic environments by using the agent's performance in the environment to refine its policy and allow it to generalize to situation not covered in the demonstrations.

- Deep reinforcement learning is gaining a lot of attention as an approach for teaching autonomous agents. At this point in AI research we find it is important to perform comparisons between imitation learning and reinforcement learning approaches to highlight the advantages and challenges of each approach. Such studies can help identify when these approaches are most suitable; as well as evaluate the benefits of combining learning from demonstrations and experience compared to using only one approach.

- Reproducibility is an integral part in emerging research fields. The easy reproduction of results and comparison of state of the art methods can substantially expedite the rate of research advancement. I this thesis we aim to conduct all experiments on open-source benchmarks that can be used for both imitation and reinforcement learning. The applications should have clear evaluation criteria and the challenges and the relevance of each task should be explicitly stated. The experiments are implemented in a general design that allows interchanging learning methods easily. The implementations are uploaded on Github to allow other researchers to easily reuse, compare and expand our proposed methods.

1.3 Problem Definition

The process of imitation learning is one by which an *agent* uses demonstrations of performed actions to learn a *policy* that replicates the demonstrated behaviour.

Definition 1 *An agent is defined as an entity that autonomously interacts within an environment towards achieving or optimizing a goal [8]. An agent can be thought of as a software robot; it receives information from the environment by sensing or communication and acts upon the environment using a set of actuators.*

Definition 2 *A policy is a function that maps a state to an action. It is what the agent uses to decide which action to execute in any given situation.*

Definition 3 *A state is a representation of world at a given point in time. The state describes details about the agent such as pose, position and internal parameters as well as its surrounding environment. States are feature representations and should include all features relevant to learning the task at hand. This information can be captured by the agent's observations or can be provided from any source.*

Definition 4 *An action is a decision made by the agent to interact with the environment. The set of actions available to the agent may be discrete, continuous or parametrized (a set of discrete actions where each action may have one or more continuous parameters).*

An action $u(t)$ can often be represented by a vector rather than a single value. This means that the action is comprised of more than one decision executed simultaneously; such as pressing multiple buttons on a controller or moving multiple joints in a robot. Actions can also represent different levels of autonomous control such as low level actions and high level macro actions [9].

1.4 Challenges

Imitation learning problems pose a variety of challenges at different stages of the training process.

- Capturing demonstrations whether from sensors on the learner or the teacher, or using visual information, can be challenging. A discrepancy between the agent's perspective and the perspective from which the demonstrations are captured is challenging to deal with. Therefore acquiring demonstrations should be designed to minimize such discrepancies. Moreover, the captured signals are prone to noise, synchronization and sensor errors.

- Similar problems arise during execution when the agent is sensing the environment. Noisy or unreliable sensing will result in erroneous behaviour even if the model is adequately trained.

- A key issue concerning demonstration is the correspondence problem. Correspondence is the matching of the learner's capabilities, skeleton and degrees of freedom to that of the teacher. Any discrepancies in the size or structure between the teacher and learner need to be compensated for during training. Often in this case a learner can learn the shape of the movement from demonstrations, then refine the model through trial and error to achieve its goal.

- A related challenge is the problem of observability where the kinematics of the teacher are not known to learner. If the demonstrations are not provided by a designated teacher, the learner may not be aware of the capabilities and possible actions of the teacher; it can only observe the effects of the teacher's actions and attempt to replicate them using its own kinematics.

- The quality of the demonstrations can also greatly affect the performance of the policy. Sub-optimal demonstrations will degrade the learned policy as they are the only source of information. Moreover, in many tasks behaviour can be subjective and using a limited number of teachers can restrict the ability of the policy to execute the given task. Another common issue is that in most cases only good demonstrations can be provided, while in machine learning having negative and positive examples can be very beneficial.

- The learning algorithms also face challenges as traditional machine learning techniques do not scale well to high degrees of freedom. Imitation learning methods typically need to deal with high dimensional sensory input as well as high dimensional multivariate predictions.

- Unlike typical machine learning applications that assume independent and identically distributed (i.i.d.) samples, motor behaviours consist of sequences of dependent decisions. This makes autonomous agents prone to accumulation of error. Therefore, the learned policy must be able to adapt its behaviour based on previous actions and make corrections if necessary. The policy must also be able to adapt to variations in the task and the surrounding environment. The complex nature of imitation learning applications dictate that agents must be able to reliably perform the task even under circumstances that vary from the training demonstration.

The scope of this thesis focuses on the last two challenges that directly concern learning a policy, while maintaining real-time execution of the policy. The sensor-based challenges are bypassed by utilizing simulators for our experiments. The simulators provide tasks where we can control observability, perspective and sensor noise. We now elaborate on two major challenges that are directly addressed in this thesis: 1- Creating adequate feature representations for learning from high dimensional data. 2- Learning a policy that generalizes to unseen situations. Feature representations are required to encode the demonstrations in a way that the agent can learn from and also to represent how the agent perceives its environment from its sensory data. The representations must be adequate for learning as well as be suitable for real time processing. Manually designing suitable features for imitation learning is an arduous task as different representations must be tailored for each task or environment. Especially in dynamic settings where the representations must be robust against various scenarios. Generalization to unseen scenarios is also a challenge because of the dynamic nature of the tasks. This is a common problem because demonstrations typically show the best way to perform a task and do not offer any information about recovering from sub-optimal actions. Therefore approaches are required that can generalize beyond demonstrated behaviour without extensive feedback from a teacher or the environment.

1.5 Contributions

The aim of this thesis is to expand the state of the art by proposing novel deep learning approaches to leverage imitation learning methods.

- The first contribution of this thesis is to survey and categories the imitation learning literature. Imitation learning is a relatively new field, however it has been contemplated from early stages of AI research and is approached from various perspectives. Therefore, a modern survey is needed categorize the learning methods for imitation problems and perform in-depth analysis to compare the different approaches and highlight their similarities and differences.

Following, we mainly address two aspects of imitation learning: representing the state of the agent using adequate feature representation and improving the agent's ability to generalize to unseen situation.

- To create feature representations, we propose the use of a deep convolutional neural network (CNN) to automatically learn features suitable to the task from the provided demonstrations and directly map them to actions. The proposed deep network is modular and is used in several experiments throughout this thesis in conjunction with other contributions.

To address the generalization challenge in imitation learning we propose the following contributions:

- Firstly, A deep active data aggregation method is proposed to improve the adaptability of the agent. Based on an initial policy trained using supervised learning, the agent queries the teacher for additional demonstrations in situations of uncertainty. Active data aggregation significantly improves the generalization of the agent using relatively few additional samples.

- Secondly, different methods for utilizing demonstrations to guide reinforcement learning are investigated. A comprehensive comparison and analysis is carried out between the proposed methods, state of the art deep reinforcement learning and our deep active imitation method.

- Thirdly, learning from demonstrations is used to shape a reward function for reinforcement learning. Both the supervised and reinforcement learning policies learn using the proposed learning representations approach. Shaping the reward from demonstrations provides a more general alternative to the standard method of tailoring shaped rewards and provides a significant performance improvement over using reinforcement learning alone.

- Finally, proposes a method for imitation learning from sequences in a multi-agent setting. The demonstrations are represented as sequences of state action pairs which keep the temporal relation between the decisions taken by the teacher. A stacked long short term memory network is used to learn a policy that takes past experiences into consideration when making a decision at each time step. The network automatically maps raw sensory data from the agent's perspective to multivariate parametrized atomic actions.

1.5.1 List of publications

The contributions of this thesis have resulted in the following publications.

- Hussein, A., Gaber, M. M., & Elyan, E. (2016, September). Deep active learning for autonomous navigation. In International Conference on Engineering Applications of Neural Networks (pp. 3-17). Springer, Cham. [10].

- Hussein A, Gaber MM, Elyan E, Jayne C. Imitation Learning: A Survey of Learning Methods. ACM Comput Surv. 2017 Apr;50(2):21:121:35. Available from: http://doi.acm.org/10.1145/3054912. [11].

- Hussein A, Elyan E, Gaber MM, Jayne C. Deep imitation learning for 3D navigation tasks. Neural computing and applications. 2017;p. 116. [12].

- Hussein A, Elyan E, Gaber MM, Jayne C. Deep reward shaping from demonstrations. In: Neural Networks (IJCNN), 2017 International Joint Conference on. IEEE; 2017. p. 510-517. [13].

- Hussein A, Elyan E, Jayne C. Deep Imitation learning with memory for Robocup Soccer Simulation. Submitted for review to the International Conference on Engineering Applications of Neural Networks (EANN 2018).

1.6 Used Frameworks

The proposed approaches in this thesis are demonstrated on a number of applications that pose various key challenges. Firstly a simple 2D navigation task is designed to evaluate learning navigation from raw visual data. The task is formulated as a grid navigation problem. The agent makes a decision given an image representing its current state. Although this task makes a number of simplifications, it poses a number of challenges common in real navigation applications. Specifically, learning from raw visual observations and performing long trajectories without intermediate feedback. This application is used to evaluate the ability of learning from demonstrations and experience approaches to deal with these challenges. Figure 1.1 illustrates the task on a 5×5 grid. The visualization shows the starting cell of the agent marked by the blue cursor, and the target cell marked by the green cursor. The agent is able to move in the four cardinal directions one cell at a time. For each cell, the state is represented by an image that shows the number of the cell as illustrated in the figure. This task can be performed with a grid of any arbitrary size.

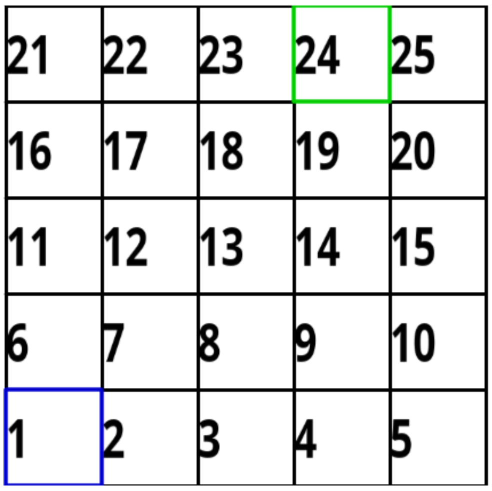


Figure 1.1: Illustration of the Grid Navigation Task

Moreover a 3D simulator, MASH simulator [14], is also used to evaluate navigation from visual data. The 3D environment is more realistic and poses extra challenges such as randomizing the shape and lighting conditions of the agent's surroundings as well as the positions of relevant objects. Another challenge that is shared with real applications is that the observations are captured from the point of view of the agent. In a 3D setting, a small movement from the agent can dramatically change the visual observations and so it is harder to keep track of relevant information. MASH simulator is used to evaluate navigation on 4 different tasks. All the approaches evaluated learn from raw pixels and therefore do not need any task specific information to learn the different tasks other than the rewards provided by the environment or demonstrations provided by a teacher.

Finally, we use a benchmark soccer simulator to evaluate multi-agent scenarios. Robocup [15] is a popular benchmark and is used in competitions for hand crafted agents as well as machine learning research. In this application, the agent needs to perform a long trajectory of actions while at the same time reacting to the actions of other agents. The observations are presented to the agent in the lowest level possible and can depend on the agent's perception. Moreover, the actions available to the agent are atomic parametric actions. This means that at the lowest level of decision making the agent need to generate continuous value as parameters along with the chosen action (such as kicking angle and force). The delayed terminal rewards inherent in a game of soccer are especially challenging for reinforcement learning agents. Figure 1.2 show a visualization of the robocup simulator. The figure shows a football field with the team names and scores illustrated. The visualizer also has playback controls to rewind, pause and speed up the the action. The figure shows a 2 vs 1 match being played in half the field.

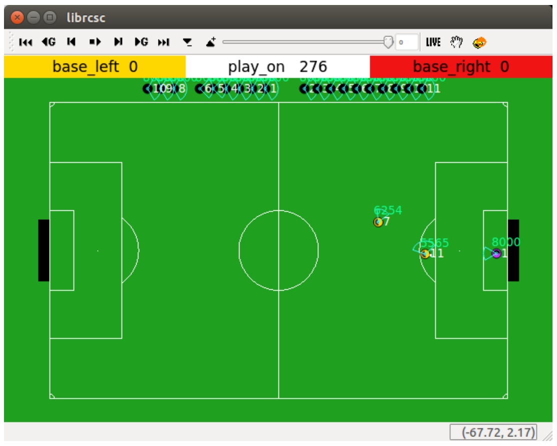


Figure 1.2: Illustration of the Soccer simulator Task

The approaches used throughout this thesis are designed to be as general as possible and require minimal task specific information. They are by no means limited to the demonstrated applications. This is shown within the thesis to some extent by using the same methods for various tasks within one application, but they can be used for a variety of other applications as well.

1.7 Chapter List

The remainder of the thesis consists of the following chapters.

Chapter 2 This chapter provides background information that is necessary for the remainder of the thesis. Problem formulation and notations for imitation learning are defined, and an introduction to deep learning is given.

Chapter 3 This chapter surveys related work in the literature. It provides a review and categorization of imitation learning methods.

Chapter 4 This chapter addresses learning state representations from raw sensory data using deep learning. It details the processes of acquiring the demonstrations, formulating the data for training, and the training model used to learn from the data. This chapter focuses on the generality of the learning method while maintaining the effectiveness of the learned representations.

Chapter 5 This chapter presents the proposed method for deep active data aggregation. Active learning is employed improve a learned policy's performance, especially it's generalization ability. Active learning is combined with the deep learning of the policy by iteratively aggregating additional training samples based on the confidence of the policy.

Chapter 6 This chapter addresses combining learning from demonstrations and experience. Two methods are proposed to train a deep neural network using both demonstrations and reinforcement learning. The proposed methods are compared to state of the art reinforcement learning methods as well as direct imitation and the active data aggregation method proposed in chapter 5.

Chapter 7 This chapter proposes a novel reward shaping method to learn reward functions from demonstrations. The shaped reward is learned from raw demonstrations using a deep neural network and added to the reward signal to train a reinforcement learning policy. Experiments demonstrate that the augmented reward signal provides a significant improvement in performance over state of the art deep reinforcement learning.

Chapter 8 This chapter presents the proposed method to learn sequences of actions by utilizing memory in deep neural networks. Temporal dependencies in the task are captured by representing demonstrations as a sequence of state-action pairs and using a long short-term memory network to learn a policy from demonstrations. The results show that utilizing memory while learning significantly improves the performance and generalization of the agent and can provide a stationary policy.

Chapter 9 This chapter concludes the thesis. It summarizes the contributions and findings of this work and discusses these conclusions in the context of the state of the art in autonomous agents. Future direction are and recommendation are discussed to highlight potential next steps.

1.8 Summary

This chapter provides and introduction to this thesis, by defining the problem of imitation learning and briefly stating the purpose of this thesis. The challenges and problems addressed are identified and motivation is provided for the proposed methods. The contributions of this thesis are listed and the chapters describing the contributions are outlined. Finally, we describe the problems to which the proposed methods are applied.

Chapter 2

Background

This chapter provides a background to the topics discussed in this thesis. Namely the process of training autonomous agents is formally described and an introduction to deep neural networks is provided. Two types of deep neural networks, convolutional neural networks (CNNs) and long short-term memory (LSTM) networks, that are used in this thesis are described in further details.

2.1 Problem formulation and Notations

In this section we formalize the problem of imitation learning and introduce some preliminary notations and definitions. Moreover, we define the process of learning from experience which can be used on its own or in conjunction with imitation learning to train an autonomous agent.

The aim of imitation learning is to map observations of the current state to a behaviour based on a set of demonstrations. The behaviour consists of a set of actions dictated by the learned policy. In order for the learning process to be general, it is preferable that agents interact with the environment using *atomic* actions. That is, the lowest level of decisions available in this application. A high level action policy determines the immediate plan of the agent such as ('open the door' or 'kick the ball'). This approach requires sub-policies to break down these decisions into a sequence of atomic actions.

In robotics, a popular way of representing actions is using *Motor primitives.*

Definition 5 *Motor primitives are simple building blocks that are executed in sequence to perform complex motions. An action is broken down into basic unit actions (often* *concerning one degree of freedom or actuator) that can be used to make up any action that needs to be performed in the given problem.*

These primitives are then learned by the policy. In addition to being useful in building complex actions from a discrete set of primitives, motor primitives can represent a desired move in state space, since they can be used to reach any possible state. As in MDPs described above, the transition from one state to another state based on which action is taken is easily tracked when using motor primitives. In this case the output of the policy can represent the change in the current state [16] as follows:

$$\dot{x}(t) = \pi(x(t), t, \alpha) \tag{2.1}$$

Where $\dot{x}(t)$ is the change in the environment's state that is affected by the policy. x is the feature vector, t is the time and α is the set of policy parameters that are changed through learning. While the time parameter t is used to specify an instance of input and output, it is also input to the policy π as a separate parameter. In the context of imitation learning, a policy is trained using a set of collected demonstrations. The demonstrations may come from a designated teacher or another agent. The demonstrations provide the optimal action to a given state, and so the agent learns to reproduce this behaviour in similar situations. This makes demonstrations suited for direct policy learning such as supervised learning methods. More formally:

Definition 6 *A demonstration is presented as a pair of input and output (x, y). Where x is a vector of features describing the state at that instant and y is the action performed by the demonstrator.*

It is clear from this formulation that learning from demonstration does not require the learner to know the cost function optimized by the teacher. It can simply optimize the error of deviating from the demonstrated output such as the least square error in supervised learning. More formally, from a set of demonstrations $D = (x_i, y_i)$ an agent learns a policy π such that:

$$u(t) = \pi(x(t), t, \alpha) \tag{2.2}$$

Where u is the predicted action. So the change in state is represented by an action with which the agent affects the environment. This representation allows for direct policy learning by using observation-label pairs for training. Moreover, it is difficult to model state change in dynamic environments. It is important to note that in this context, the teacher's behaviour is referred to as the optimal policy as it is the goal to reach human level behaviour even if this policy does not achieve the best possible performance. For the rest of this thesis, the expert's behaviour and optimal behaviour are used interchangeably. Such direct policy learning however, is prone to poor generalization. Since the agent's interaction with the environment can change its state, the assumption that predictions are independent and identically distributed is not valid. Indeed most applications require the agent to perform a sequence of dependent actions to achieve a goal. Therefore even a small prediction error can lead the agent to an unfamiliar state. Since demonstrations only provide the optimal action trajectories, a suboptimal policy will be presented with states that are not represented in the demonstrations. More formally the agent is presented with states drawn from a distribution different to the distribution from which the training set $D = (x_i, y_i)$ was sampled.

A policy for autonomous agents can alternatively be learned from experience. The experiences uses for learning may be the agent's own or may come from a different agent. While learning from experience does not require examples of good behaviour to imitate, a reward function is required to evaluate the performance of the current policy in order to optimise it. Reinforcement learning uses the actions performed by the agent, which may not be optimal, along with the reward (or cost) of performing that action given the current state, to learn a policy that maximizes its overall reward. More formally:

Definition 7 *An experience is presented as a tuple (s, a, r, s') where s is the state, a is the action taken at state s, r is the reward received for performing action a and s' is the new state resulting from that action.*

Learning from experience is commonly formulated as a Markov decision process (MDP). MDPs lend themselves naturally to motor actions, as they represent a state-action network and are therefore suitable for reinforcement learning. In addition the Markov property dictates that the next state is only dependent on the previous state and action, regardless of earlier states. This timeless property promotes stationary policies. There are different methods to learn from experience through reinforcement learning that are out of the scope of this thesis. For a survey and formal definitions of reinforcement learning methods for intelligent agents, the reader is referred to [17]. Note that both learning paradigms are similarly formulated with the exception of the cost function; the feature vector $x(t)$ corresponds to s, $u(t)$ corresponds to a and $x(t+1)$ corresponds to the resulting state s'. It is therefore not uncommon (especially in more recent research) to combine learning from demonstrations and experience to perform a task.

Definition 8 *A policy that uses t in learning the parameters of the policy is called a non-stationary policy (also known as non-autonomous [16]) i.e. the policy takes into* *consideration at what stage of the task the agent is currently acting.*

Definition 9 *A stationary policy (autonomous) neglects the time parameter and learns one policy for all steps in an action sequence.*

One advantage of stationary policies is the ability to learn tasks where the horizon (the time limit for actions) is large or unknown [18]. While non stationary policies are more naturally situated to learn motor trajectories i.e actions that occur over a period of time and are comprised of multiple motor primitive executed sequentially. However, these policies are difficult to adapt to unseen scenarios and changes in the parameters of the task [16]. Moreover, this failure to adapt to new scenarios, at one point in the trajectory, can result in compounded errors as the agent continues to perform the remainder of action. In light of these drawbacks, methods for learning trajectories using stationary policies are motivated. An example is the use of structured predictions [19] where the training demonstrations are aggregated with labelled instances at different time steps in the trajectory – so time is encoded in the state. Alternatively, [20] learns attractor landscapes from the demonstrations, creating a stationary policy that is attracted to the demonstrated trajectory. This avoids compounded errors as the current state is considered by the policy before executing each state of the trajectory.

2.2 Deep Learning

Deep learning is a sub-field of machine learning that is concerned with training large neural networks termed: deep neural networks. Deep neural networks are distinct from traditional neural network in that they consist of more layers which allows them to learn more complex models. Deep learning can be thought of as hierarchical feature learning. Each layer learns a higher level representation of the data from the previous layer thus uncovering deeper insights into the data. Recent advances in deep learning algorithms and computational technology have made it possible to effectively train very large and complex neural networks. This section provides a background on deep neural networks. We start by presenting multi-layer perceptrons, describing how they are constructed and how they learn. Following two types of deep neural networks that are used in this thesis are described, namely, convolutional neural networks (CNN) and recurrent neural networks (RNN).

Definition 10 *Artificial neural networks are computing systems inspired by biological neural networks in the brain. Neural networks act as a function approximator that learns a function that maps a given input x to a corresponding output y. Thus providing a function $f : X \to Y$ for any input $x \in X$*

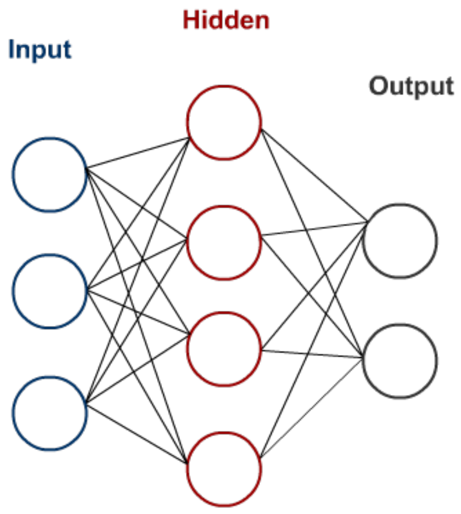


Figure 2.1: Example of an artificial neural network showing connections between the nodes in different layers.

Neural networks are composed of layers each with a given number of nodes. Figure 2.1 provides an illustration of a neural network with an input layer, a hidden layer and an output layer. Connections between the nodes provide parameters that are optimised to learn the required function. This is achieved using two steps: A *forward pass* and a *back propagation* step. The forward pass goes through the network from the input layer to the output layer performing the following computation at each node:

$$y_j = a(\sum_{i=1}^{I} w_{ij} x_i + b_j) \tag{2.3}$$

Where y is the output of the node, x_i is an input to the node from node i, w_i is the weight assigned to the connection with node i, b is a scalar bias and a is an activation function which transforms the weighted sum of the inputs linearly or non-linearly. Activation functions define the structure of the function being estimated.

Common activation functions used in neural networks are the hyperbolic tangent (Tanh) function:

$$a(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.4}$$

rectified linear unit (ReLU):

$$a(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.5}$$

and the logistic sigmoid function:

$$a(x) = \frac{1}{1 + e^{-2}} \tag{2.6}$$

It is clear from the feed forward that neural networks are inherently regressors as the output is a continuous value. However, neural networks are commonly used for classification by using activation functions that transfer the node outputs towards discrete decisions. The neural network learns the function estimation by tuning the values of the weights w and bias b to minimize a cost function of the final output from the forward pass and a given ground truth. The choice of cost function is also an important distinction between classification and regression using neural networks. Mean square error is commonly used for regression problems, to calculate the cost between continuous targets and the output of the network.

$$C = \frac{1}{I} \sum_{i=1}^{I} (u_i - y_i)^2 \tag{2.7}$$

Where C is the cost, u_i and y_i are the networks output and the target for instance i respectively and I is the number of instances. Cross entropy is a common cost function for classification problems, usually with softmax activations. They calculate the likelihood of the neural network choosing the correct class.

$$C = -\frac{1}{I} \sum_{i=1}^{I} \sum_{j=1}^{J} y_{ij} log(u_{ij}) \tag{2.8}$$

Where u_{ij} and y_{ij} are the networks output and the target respectively for instance i and class j, I is the number of instances and J is the number of classes.

Back propagation works in reverse order from the output layer towards the input layer, calculating the changes that need to be made to the parameters in each layer. These changes are based on the gradient of the cost function with respect to the parameters. Intuitively, back propagation calculates how much of the cost can be attributed to each parameter and uses gradient descent to iteratively move towards minimizing the gradient of the cost. More formally Δw_{ij}, the change made to weight of the connection between nodes i, j is calculated as:

$$\Delta w_{ij} = \gamma C a'(\sum_{i=1}^{I} w_{ij} x_i)) x_i \tag{2.9}$$

Where γ is a constant learning rate, C is the cost, $a'(\sum_{i=1}^{I} w_{ij} x_i)$ is the derivative of the activation function and x_i is the input from connection i. The chain rule 2.10 is used to find the gradient of the error starting from the output of the final layer of the network and propagating back through the layer. To calculate the gradients all through the network, the activation functions and cost function need to be differentiable.

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} \tag{2.10}$$

2.2.1 Convolutional Neural Networks

A Convolutional neural network is a type of deep neural network that takes advantage of spatial relations between features to learn high level representation. Inspired by the visual cortex in animals, Convolutional networks apply convolution filters to restrict the connections between nodes. By limiting the number of connections, convolutiuonal layers allow for creating deep networks with fewer parameters. This makes CNNs very effective in learning from images, but they have also shown great potential in learning from other signals where spatial relations between features can be exploited, such as text and audio data. Automatically learning feature representation alleviates the need to hand engineer features which is a major challenge in many applications. Therefore CNNs achieve state of the art performance in a number of domains and are considered one of the most popular neural network classes for modern application.

Definition 11 *The convolution operation applies a filter (or kernel) to a section of the input and is defined as the sum of the element wise product between the filter and the section of the input.*

A convolution layer applies a number of filters to the image using a sliding window. The result is a 3D structure (assuming a 2 dimensional input) where the width and height are a reduction of the original dimensions of the input signal, and the depth is the number of filters used. This can be considered as producing a feature vector for each sub section of the input. Figure 2.2 illustrates applying a convolution filter to an input signal. By segmenting the input signal using the convolutional filters, the number of connections between the nodes is greatly reduced.

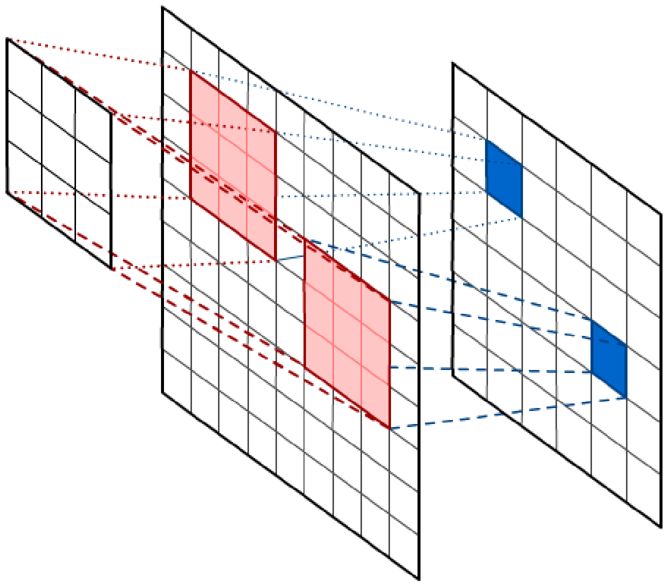


Figure 2.2: Convolution operation on a 2D input using a 3x3 filter

We now present the convolution process. Applying kernel K to input I at point i, k is given as:

$$conv(I, K)_{i,j} = \sum_{m}^{M} \sum_{n}^{N} I(i - m, j - n) K(m, n) \tag{2.11}$$

Where M and N are the dimensions of the kernel. Deep convolutional networks usually consist of a number of stacked convolutional layers followed by a fully connected hidden layer, to map the extracted features to the output layer. The output of the last convolutional layer is flattened into a vector and connected to the hidden layer. The forward pass of the fully connected layer is the same as eq. 2.3. Figure 2.3 illustrates a full convolutional neural network.

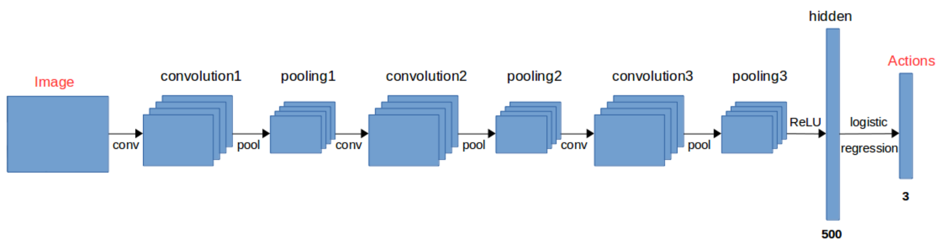


Figure 2.3: Example of an artificial neural network showing connections between the nodes in different layers.

The backward pass in convolutional networks starts with output layer and propagates through the network calculating the gradient for the fully connected layers and then the convolutional layers in sequence using the chain rule 2.10. The convolution operation is differentiable, therefore, the back propagation process is unaffected.

2.2.2 Recurrent Neural Networks

Recurrent Neural Networks are a group of neural networks that take advantage of temporal relations in the data by keeping track of previous samples during learning. Recurrent units retain an internal memory which allows them to process sequences of data such as speech and text. Recurrent units unlike typical feed-forward nodes, recurrent units can have looping connections, thus providing feedback that contributes to the processing of the next time step. Figure 2.4 illustrates how the recurrent connection can be viewed as a sequence of nodes where each node provides an additional input to the next node. In other words, a recurrent node receives the current input in addition to its decision for the previous time step as shown in the following equation.

$$h_{jt} = a(\sum_{i=1}^{I} w_{ij}x_{it} + u_j h_{jt-1}) \tag{2.12}$$

Where h_{jt} is the hidden state of node j at time step t, a is the activation function, w_{ij} is the weight of the connection between nodes i and j and similarly, u_j is a weight on the recurrent connection between node j and its hidden state at the previous time step h_{jt-1}.

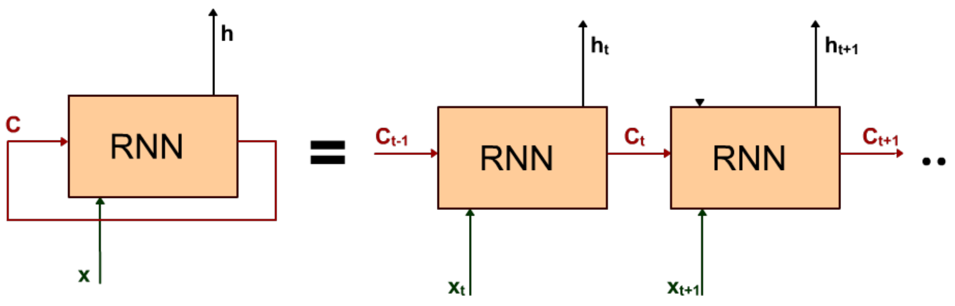


Figure 2.4: Example of a recurrent neural network unit.

Back propagation in RNNs seeks to optimise the values of the weight matrices W and

U. Learning values for U allows the network to learn temporal relationships within sequences of learning. The backward pass in RNNs is dubbed as back propagation through time as the gradient is calculated for the last members of the sequence first and then back propagate to the earlier examples. Since RNNs can have long term temporal dependencies, they are prone to suffer from vanishing and exploding gradients. This occurs because the calculation of the gradient passes through multiplication processes at each step in the chain rule. Therefore the gradient for parameters that are far removed from the cost can become exponentially small or large rendering them useless for learning. Long-short-term-memory (LSTM) networks [21] deal with this issue by including gates in the node that dictate when to hold information in the networks memory and when to forget it. Figure 2.5 shows an LSTM unit containing: a forget gate that controls retaining or forgetting the current cell state; and input gate that controls the current input signal; and an output gate that controls what is output from the node.

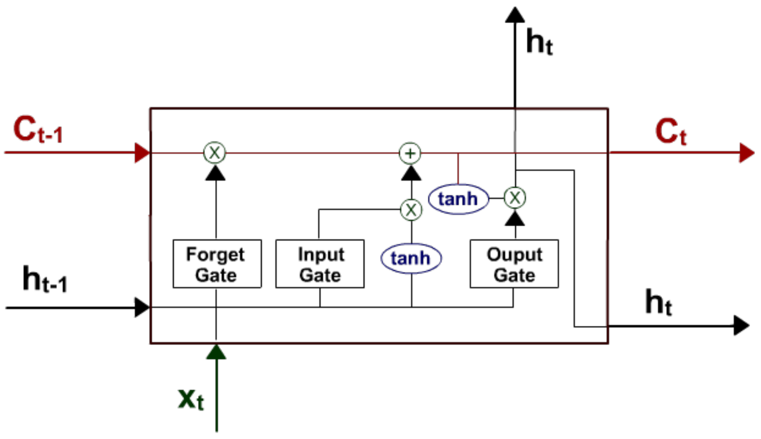


Figure 2.5: Example of an LSTM unit.

2.3 Summary

This chapter provides a background to the topics discussed in this thesis. An introduction to training autonomous agents is provided and the processes of training such agents from demonstrations and experience are formally defined. Following a background to deep learning is presented. Two types of deep neural networks which are used in this work (Convolutional neural networks and recurrent neural networks) are described, as well as a general overview of neural networks.

Chapter 3

Related Work

This chapter presents the main works in the literature related to this thesis. It provides an overview on different imitation learning methods but focuses on methods that employ deep learning and that integrate learning from experience. We also highlight notable works in applications similar to the ones used in this thesis, namely navigation and multi-agent applications. We categorize imitation learning into policies that directly copy behaviour from demonstrations and policies that self-improve based on its current performance. This literature review is published in the following survey paper [11].

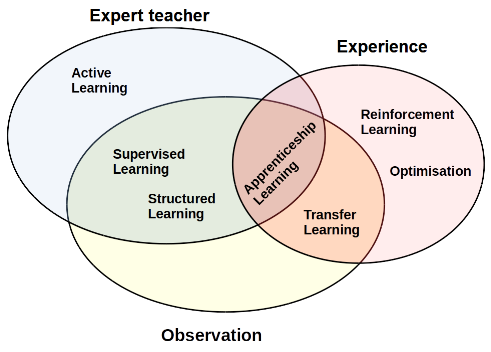


Figure 3.1: learning methods from different sources

Intelligent agents can learn behavioral policies from different sources. Each case needs appropriate learning methods that can best utilize the data source. Figure 3.1 shows a Venn diagram outlining the sources of data employed by different learning methods. An agent can learn from dedicated teacher demonstrations, observing other agent's actions or from experience through trial and error. Active learning needs a dedicated oracle that can be queried for demonstrations. While other methods that utilize demonstrations can acquire them from a dedicated expert or by observing the required behavior from other agents. RL and optimization methods learn through trial and error and do not make use of demonstrations. Transfer learning uses experience from old tasks, or knowledge from other agents to learn a new policy. Apprenticeship learning uses demonstrations from an expert or observation to learn a reward function. A policy that optimizes the reward function can then be learned through experience. The methods proposed in this thesis focus on learning from dedicated experts and the agent's own experiences. We now review the learning approaches for direct and self-improved imitation.

3.1 Direct Imitation

Direct imitation, also known as behaviour cloning [22], is using demonstrations to learn mapping between observations and actions. It is the most straight forward way to learn a policy from demonstrations and usually learns via supervised learning, where the action provided by the expert acts as the label for a given instance. The model is then capable of predicting the appropriate action when presented with a situation. Supervised learning methods are categorized into classification and regression.

3.1.1 Classification

Classification is a popular task in machine learning where observations are automatically categorized into a finite set of classes. A classifier $h(x)$ is used to predict the class y to which an independent observation x belongs; where $y \in Y$, $Y = \{y_1, y_2 \dots y_p\}$ is a finite set of classes, and $x = \{x_1, x_2 \dots x_m\}$ is a vector of m features. In supervised classification, $h(x)$ is trained using a dataset of n labelled samples $(x^{(i)}, y^{(i)})$, where $x^{(i)} \in X$, $y^{(i)} \in Y$ and $i = 1, 2 \dots n$.

Classification approaches are used when the learner's actions can be categorized into discrete classes [9]. This is suitable for applications where the action can be viewed as a decision such as navigation [23] and flight simulators [24]. In [23], a Gaussian mixture model (GMM) is trained to predict navigational decisions. Meta-classifiers are used in [18] to learn a policy to play computer games. The base classifier used in this paper is a neural network. "Super Tux Kart" is a kart racing game where the analogue joystick commands are discretized into 15 buckets, reducing the problem to a 15 class classification problem. So the neural network used had 15 output nodes. "Mario Bros" is a platform game that uses a discrete controller. Actions are selected by pressing one or more of 4 buttons. So in the neural network, the action for a frame is represented by 4 output nodes. This enables the classifier to choose multiple classes for the same instance. Although the results are promising, it is argued that using an Inverse Optimal Control (IOC) technique [25] as the base classifier might be beneficial. In [19] the experiments are repeated this time using regression (see regression section 3.1.2) to learn the analogue input in “Super Tux Kart”. For Mario Bros, 4 Support Vector Machine (SVM) classifiers replace the neural network to predict the value of each of the 4 binary classes. Classification can also be used to make decisions that entail lower level actions. In [26] high level decision are predicted by the classifier in a multi-agent soccer simulation. Decisions such as ‘Approach ball’ and ‘Dribble towards goal’ can then be deterministically executed using lower level actions. An empirical study is conducted to evaluate which classifiers are best suited for the imitation learning task. Four different classifiers are compared with respect to accuracy and learning time. The results show that a number of classifiers can perform predictions with comparable accuracy, however, the learning time relative to the number of demonstrations can vary greatly [26]. Recurrent neural networks (RNN) are used in [27] to learn trajectories for object manipulation from demonstrations. RNNs incorporate memory of past actions when considering the next action. Storing memory enables the network to learn corrective behaviour such as recovery from failure given that the teacher demonstrates such a scenario.

3.1.2 Regression

Regression methods are used to learn actions in a continuous space. Unlike classification, regression methods map the input state to a numeric output that represents an action. Thus they are suitable for low level motor actions rather than higher level decisions. Especially when actions are represented continuous values, such as input from a joystick [19]. The regressor $I(x)$ maps an independent sample x to a continuous value y rather than a set of classes. Where $y \in \mathbb{R}$, the set of real numbers. Similarly the regressor is trained using a set of labelled samples $(x^{(i)}, y^{(i)})$, where $y^{(i)} \in \mathbb{R}$ and $i = 1, 2 \ldots n$.

A commonly used technique is locally weighted regression (LWR). LWR is suitable for learning trajectories, as these motions are made up of sequences of continuous values. Examples of such motions are batting tasks [28] [29] (where the agent is required to execute a motion trajectory in order to pass by a point and hit a target); and walking [30] where the agent needs to produce a trajectory that results in smooth stable motion. A more comprehensive application is table tennis. [31] use Linear Bayesian Regression to teach a robot arm to play a continuous game of table tennis. The agent is required to move with precision in a continuous 3D space in different situations, such as when hitting the ball, recovering position after a hit and preparing for the opponent's next move. Another paradigm commonly used for regression is artificial neural networks (ANN). Neural networks differ from other regression techniques in that they are demanding in training time and training samples. Neural network approaches are often inspired by biology and neuroscience studies, and attempt to emulate the learning and imitation process in humans and animals [1]. The use of regression with a dynamic system of motor primitives has produced a number of applications for learning discrete and rhythmic motions [20] [32] [33], though most approaches focus on direct imitation without further optimization [34]. In such applications, a dynamic system represents a single degree of freedom (DOF) as each DOF has a different goal and constraints [32].

Dynamic systems can be combined with probabilistic machine learning methods to reap the benefits of both approaches. This allows the extraction of patterns that are important to a given task and generalization to different scenarios while maintaining the ability to adapt and correct movement trajectories in real time [35]. In [35] the estimation of dynamical systems' parameters is represented as a Gaussian mixture regression (GMR) problem. This approach has an advantage over LWR based approaches as it allows learning of the activation functions along with the motor actions. The proposed method is used to learn time-based and time-invariant movement. In [36] a similar GMM based method is used in a task-parametrized framework which allows shaping the robot's motion as a function of the task parameters. Human demonstrations are encoded to reflect parameters that are relevant to the task at hand and identify the position, velocity and force constraints of the task. This encoding allows the framework to derive the state in which the robot should be, and optimize the movement of the robot accordingly. This approach is used in a Human Robot Collaboration (HRC) context and aims to optimize human intervention as well as robot effort. Deep learning is combined with dynamical systems in [37]. Dynamic movement primitives (DMP) are embedded into autoencoders that learn representations of movement from demonstrated data. Autoencoders non-linearly map features to a lower dimensional latent space in the hidden layer. However, in this approach, the hidden units are constrained to DMPs to limit the hidden layer into representing the dynamics of the system.

In both classification and regression methods, a design decision can be made regarding the learning models resources. Lazy learners such as kNN and LWR do not need training but need to retain all training samples when performing predictions. On the other hand, trained models such as ANN and SVM require training time, but once a model is created the training samples are no longer needed and only the model is stored, which saves memory. These models can also result in very short prediction times.

3.1.3 Hierarchical Models

Rather than using a single model to reproduce human behaviour, a hierarchical model can be employed that breaks down the learned actions into a number of phases. A classification model can be used to decide which action or sub-action, from a set of possible actions, should be performed at a given time. A different model is then used to define the details of the selected action, where each possible low-level action has a designated model. Figure 3.2 illustrates an example of hierarchical actions. [38] use a hierarchical approach for imitation learning on two different problems. The first is Air Hockey which is played against an opponent, and the objective is to shoot a puck into the opponent's goal while protecting your own. The second game is marble maze; the maze can be tilted around different axis to move the ball towards the end of the maze. Each task has a set of low level actions called motor primitives that make up the playing possibilities for the agent (e.g., Straight shot, Defend Goal, and Roll ball to corner). In the first stage, a nearest neighbour classifier is used to select the action to be performed. By observing the state of the game the classifier searches for the most similar instances in the demonstrations provided by the human expert, and retrieves the primitive selected by the human at that point. The next step is to define the goal of the selected action, for example the velocity of the ball or the position of the puck when the primitive is completed. The goal is then used in a regression model to find the parameters of the action that would optimize the desired goal. The goal is derived from the k nearest neighbour demonstrations found in the previous step. The goals in those demonstrations are input in a local weighted regression model to perform the primitive. In a similar fashion, [39] use a classifier to make decisions in a sorting task consisting of the following macro actions (wait, sort left, sort right and pass). Each macro action entails temporal motor actions such as picking up a ball, moving and placing the ball.

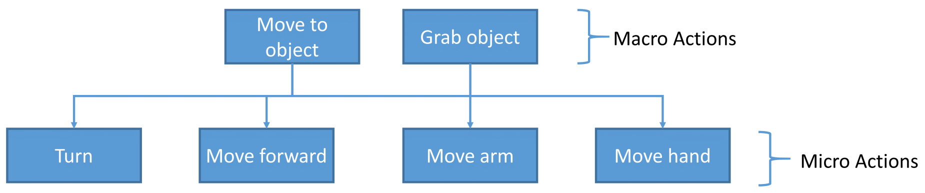


Figure 3.2: Example of hierarchical learning of actions

3.2 Self Improvement

It is often the case that direct imitation on its own is not adequate to reproduce robust, human-like behaviour in intelligent agents. This limitation can be attributed to two main factors: (1) errors in demonstration, and (2) poor generalization. Due to limitations of data acquisition techniques, such as correspondence problem, sensor error and physical influences in kinesthetic demonstrations [9], direct imitation can lead to inaccurate or unstable performance, especially in tasks that require precise motion in continuous space such as reaching or batting. For example, in [40] a robot attempting to walk by directly mimicking the demonstrations would fall because the demonstrations do not accurately take into consideration the physical properties involved in the task such as the robot's weight and centre of mass. However, refinement of the policy through trial and error would take these factors into account and produce a stable motion.

While generalization is an important issue in all machine learning practices, a special case of generalization is highlighted in imitation learning applications. It is common that human demonstrations are provided as sequence of actions. The dependence of each action on the previous part of the sequence violates the i.i.d. assumption of training samples that is integral to generalization in supervised learning [18]. Moreover, since human experts provide only correct examples, the learner is unequipped to handle errors in the trajectory. If the learner deviates from the optimal performance at any point in the trajectory (which is expected in any machine learning task), it would be presented with an unseen situation that the model is not trained to react to. A clear example is provided in [7] where supervised learning was used to learn a policy to drive a car. Given that human demonstrations contained only 'good driving' with no crashes or close calls, when error occurs and the car deviates from demonstrated trajectories, the learner does not know how to recover.

This section discusses indirect ways to learn policies that can complement or replace direct imitation. The policy can be refined from demonstrations, experience or observation to be more accurate or to be more general and robust against unseen circumstances. We categorize the methods for refining the policy into the following groups: reinforcement learning, transfer learning, data aggregation, active learning, apprenticeship learning and optimization.

3.2.1 Reinforcement Learning

Reinforcement learning (RL) learns a policy to solve a problem via trial and error.

Definition 12 *In RL, an agent is modelled as a Markov Decision Process (MDP) that learns to navigate in a state space. A finite MDP consists of a tuple (S, A, T, R), where S is a finite set of states, A is the set of possible actions, T is the set of state transition probabilities and R is a reward function. $TP_s a$ contain a set of probabilities where $P_s a$ is the probability of arriving at state s given action a and where $a \in A$ and $s \in S$. The reward function $R(s_k, a_k, s_{k+1})$ returns an immediate reward for taking an action in a given state and ending up in a new state $a_k \to s_{k+1}$ where k is the time step. This reward is discounted over time by a discount factor $\gamma \in [0, 1)$ and the goal of the agent is to maximize the expected discounted reward at each time step.*

RL starts off with a random policy and modifies its parameters based on rewards gained from executing this policy. Reinforcement learning can be used on its own to learn a policy for a variety of robotic applications. However, if a policy is learned from demonstration, reinforcement learning can be applied to fine tune the parameters. Providing positive or negative examples to train a policy helps reinforcement learning by reducing the search space available [1]. Enhancing the policy using RL is sometimes necessary if there are physical discrepancies between the teacher and the learner or to alleviate errors in acquiring the demonstrations. RL can also be useful to train the policy for unseen situations that are not covered in the demonstrations. Applying reinforcement learning to the learned policy instead of a random one can significantly speed up the RL process and avoids the risk of the policy from converging into a local minimum. Moreover RL can find a policy to perform a task that does not look normal to the human observer. In applications where the learner interacts with a human, it is important for the user to intuitively recognize the agent's actions. This is common in cases where robots are introduced into established environments (such as homes and offices) to interact with untrained human users [41]. By applying Reinforcement learning to a policy learned from human demonstrations the problem of unfamiliar behaviour can be avoided. In imitation learning methods, reinforcement learning is often combined with learning from demonstrations to improve a learned policy when the fitness of the performed task can be evaluated.

In early research, teaching using demonstrations of successful actions was used to improve and speed up reinforcement learning. In [42], reinforcement learning is used to learn a policy to play a game in a 2D dynamic environment. Different methods for enhancing the RL policy are examined. The results demonstrate that teaching the learner with demonstrations improves its score and helps prevent the learner from falling in local minima. It is also noted that the improvement from teaching increases with the difficulty of the task. Solutions to simple problems can be easily inferred without requiring demonstrations from an expert. But as the complexity of the task increases the advantage of learning from demonstrations becomes more significant, and even necessary for successful learning in more difficult tasks [43].

In [44] Gaussian mixed regression (GMR) is used to train a robot on an object grasping task. Since unseen scenarios such as obstacles and the variable location of the object are expected in this application, reinforcement learning is used to explore new ways to perform the task. The trained system is a dynamic system that performs damping on the imitated trajectories. This allows the robot to smoothly achieve its target and prevents reinforcement learning from resulting in oscillations. Using damping in dynamic systems is a common approach when combining imitation learning and reinforcement learning [45][17].

An impressive application of imitation and reinforcement learning is training an agent to play the board game 'Go' that rivals human experts [46]. This work levers learning from demonstrations to train an actor-critic reinforcement learning agent. Firstly a dataset of previous games is used to train a supervised convolutional neural network to play the game. The weights of the network are used to initialize the actor network used for reinforcement learning, so the agent starts exploring from a good starting policy. Secondly a set of recorded games is used to train a network to predict whether the game will end in a win or a loss given the current state. This evaluation function provides feedback to the critic in the reinforcement learning algorithm so it can learn from the estimated consequences of each action. This method significantly outperforms direct imitation [47] and has shown the ability to beat human experts.

A different approach to combine learning from demonstrations with reinforcement learning is employed in [48]. Rather than using the demonstrations to train an initial policy, they are used to derive prior knowledge for reward shaping [49]. A reward function is used to encourage sub-achievements in the task, such as milestones reached in the demonstrations. This reward function is combined with the primary reward function to supply the agent with the cost of its actions. This paradigm of using expert demonstrations to derive a reward function is similar to inverse reinforcement learning approaches [50].

Policy search methods are a subset of reinforcement learning that lend themselves naturally to robotic applications as they scale to high dimensional MDPs [17]. Therefore policy search methods are a good fit to integrate with imitation learning methods. A policy gradient method is used in [51] to improve an existing policy that can be created through supervised learning or explicit programming. A similar approach [52] is used within a dynamic system that was previously used for supervised learning from demonstrations [29]. This led to a series of work that utilizes the dynamic system in

[29] to learn from demonstrations and subsequently use reinforcement learning for self-improvement [45] [53] [54] [55]. This framework is used to teach robotic arms a number of applications such as ball in cup, ball paddling [45] [28] and playing table tennis [31]. In [56] demonstration are used to initialize reinforcement policies. Because RL agents require a large number of trials before it achieves acceptable performance, using RL in many real world applications may not be practical. Therefore demonstrations are used to train an initial policy using supervised loss as well as temporal difference (TD) loss. DQN is then used to retrain the policy by continuing to optimize the TD loss. This method shows significantly faster learning than using DQN from scratch, and outperforms using RL only on a number of Atari games. However, the benchmarks used in this evaluation can already be solved solely using trial and error.

Rather than using reinforcement learning to refine a policy trained from demonstrations, demonstrations can be used to guide the policy search. In Guided policy search [57], a model based approach generates guiding samples from demonstrations using differential dynamic programming (DDP). A model-free policy search algorithm then uses these sample trajectories to explore areas in which it is likely to be rewarded. By following the guidance of demonstrations the agent has faster access to rewards, which expedites learning through reinforcement learning. Recurrent neural networks are incorporated into guided policy search in [58] to facilitate dealing with partially observed problems. Past memories are augmented to the state space and are considered when predicting the next action. A supervised approach uses demonstrated trajectories to decide which memories to store while reinforcement learning is used to optimize the policy including the memory state values.

A different way to utilize reinforcement learning in imitation learning is to use RL to provide demonstrations for direct imitation. This approach does not need a human teacher as the policy is learned from scratch using trial and error and then used to generate demonstrations for training. One reason for generating demonstrations and training a supervised model rather than using the RL policy directly is that the RL method does not act in real-time [59]. Another situation is when the RL policy is learned in a controlled environment. In [60] reinforcement learning is used to learn a variety of robotic tasks in a controlled environment. Information such as the position of target objects is available during this phase. A deep convolutional neural network is then trained using demonstrations from the RL policy. The neural network learns to map visual input to actions and thus learns to perform the tasks without the information needed in the RL phase. This mimics human demonstrations as humans utilize expert knowledge – that is not incorporated in the training process – to provide demonstrations.

For a comprehensive survey of reinforcement learning in robotics, the reader is referred to [17]

3.2.2 Optimization

Optimization approaches can also be used to find a solution to a given problem.

Definition 13 *Given a cost function $f : A \to \mathbb{R}$ that reflects the performance of an agent, where A is a set of input parameters and $\mathbb{R}$ is the set of real numbers, optimization methods aim to find the input parameters x_0 that minimize the cost function. Such that $f(x_0) \leq f(x)\ \forall x \in A$*

Similar to reinforcement learning, optimization techniques can be used to find solutions to problems by starting with a random solution and iteratively improving to optimize the fitness function. Evolutionary algorithms (EA) are popular optimization methods that have extensively been used to find motor trajectories for robotic tasks [61]. EAs are used to generate motion trajectories for high and low DOF robots [62] [63]. Popular swarm intelligence methods such as Particle Swarm Optimization (PSO) [64] and Ant Colony Optimization (ACO) [65] are used to generate trajectories for unmanned vehicle navigation. These techniques simulate the behavior of living creatures to find and optimal global solution in the search space. As is the case with reinforcement learning, evolutionary algorithms can be integrated with imitation learning to improve trajectories learned by demonstration or to speed up the optimization process.

In [40] a genetic algorithm (GA) is used to optimize demonstrated motion trajectories. The trajectories are used as a starting population for the genetic algorithm. The recorded trajectories are encoded as chromosomes constituted of genes representing the motor primitives. The GA searches for the chromosome that optimizes a fitness function that evaluates the success of the task. Projecting the motor trajectories to lower dimension illustrates the significant change between the optimized motion and the one learned directly from kinesthetic manipulation [40].

Similarly in [66] evolutionary algorithms are used after training agents in a soccer simulation. A possible solution (chromosome) is represented as a set of *if-then* rules. The rules are finite due to the finite permutations of observations and actions. A weighted function of the number of goals and other performance measures is used to evaluate the fitness of a solution. Although the evolutionary algorithm had a small population size and did not employ crossover, it showed promising results over the rules learned from demonstrations.

[7] also used evolutionary algorithms to optimize multiple objectives in a racing game. The algorithms evolve an optimized solution (controller) from an initial population of driving trajectories. Evaluation of the evolved controllers found that they stay faithful to the driving style of the players they are modelled after. This is true for quantitative measures such as speed and progress, and for qualitative observations such as driving in the centre of the road.

[67] treat the weights of a neural network as the genome to be optimized. The initial population is provided by training the network with demonstrated samples to initialize the weights. The demonstrations are also used to create a fitness value corresponding to the mean squared error distance from the desired outputs (human actions).

In [68] Particle Swarm Optimization (PSO) is used to find the optimal path for an Unmanned Aerial Vehicle (UAV) by finding the best control points on a B-spline curve. The initial points that serve as the initial PSO particles are provided by skeletonization. A social variation of PSO is introduced in [69], inspired by animals learning in nature from observing their peers. Each particle starts with a random solution and a fitness function is used to evaluate each solution. Then imitator particles (all except the one with the best fitness) modify their behaviour by observing demonstrator particles (better performing particles). As in nature an imitator can learn from multiple demonstrators and a demonstrator can be used to teach more than one imitator. Interactive Evolutionary algorithms (IEA) [70] employ a different paradigm. Rather than use human input to start an initial population of solutions and the optimize them, IEA uses human input to judge the fitness of the solutions. To avoid the user evaluating too many potential solutions, a model is trained on supervised examples to estimate the human user's evaluation. In [71] fitness based search is combined with Preference-based Policy Learning (PPL) to learn robot navigation. The user evaluations from PPL guide the search away from local minima while the fitness based search searches for a solution. In similar spirit [72] train a robot to imitate human arm movement. The difference in degrees of freedom (DOF) between the human demonstrator and the robot obstructs using the demonstrations as an initial population. However, rather than use human input to subjectively evaluate a solution, the similarity of the robot movement to human demonstrations is quantitatively evaluated. A sequence-independent joint representation for the demonstrator and the learner is used to form a fitness function. PSO is used to find the joint angles to optimize this similarity measure. A different method of integrating demonstrations is proposed in [73]. Inspired by Inverse Reinforcement Learning (see section on inverse reinforcement learning 3.2.4), an Inverse Linear Quadratic Regulator(ILQR) framework is used to learn cost function optimized by the human demonstrator. PSO is then employed to find a solution for the learned function instead of gradient methods.

3.2.3 Transfer Learning

Transfer learning is a machine learning paradigm where knowledge of a task or a domain is used to enhance learning of another task.

Definition 14 *Given a source Domain D_s and task T_s, transfer learning is defined as improving the learning of a target task T_t in domain D_t using knowledge of D_s and T_s; where $D_s \neq D_t$ or $T_s \neq T_t$. A domain $D = \{\chi, P(X)\}$ is defined as a feature space χ and a marginal probability distribution $P(X)$, Where $X = \{x_1, ...x_n\} \in \chi$. The condition $D_s \neq D_t$ holds if $\chi_s \neq \chi_t$ or $P_s(X) \neq P_t(X)$ [74].*

A learner can acquire various forms of knowledge about a task from another agent such as useful feature representations or parameters for the learning model. Transfer learning is relevant to imitation learning and robotic applications because acquiring samples is difficult and costly. Utilizing knowledge of a task that we already invested to learn can be efficient and effective.

A policy learned in one task can be used to advise (train) a learner on another task that carries some similarities. In [75] this approach is implemented on two robocup soccer simulator tasks, the first is to keep the ball from the other team, and second to score a goal. It is obvious that skills learned to perform the first task can be of use in the later. In this case advice is formulated as a rule concerning the state and one or more action. To create advice the policy for the first task is learned using reinforcement learning. The learned policy is then mapped by a user (to avoid discrepancies in state or action spaces) into the form of advice that is used to initiate the policy for the second task. After receiving advice the learner continues to refine the policy through reinforcement learning and can modify or ignore the given advice if it proves through experience to be inaccurate or irrelevant.

Often in transfer learning, human input is needed to map the knowledge from one domain to another, however, in some cases the mapping procedure can be automated [76]. For example, in [77] a mapping function for general game playing is presented. The function automatically maps between different domains to learn from previous experience. The agent is able to identify previously played games relevant to the current task. The agent may have played the same game before or a similar one and is able to select an appropriate source task to learn from without it being explicitly designated. Experiments show that the transfer learning approach speeds up the process of learning the game via reinforcement learning (compared to learning from scratch) and achieves a better performance after the learning iterations are complete. The results also suggest that the advantage of using transfer learning is correlated with the number of training instances transferred from the source tasks. Even if the agent encounter negative transfer [74] for example from over-fitting to the source task, it can recover by learning through experience and rectifying its model in the current task to converge in appropriate time [77].

Brys et al [78] combine reward shaping and transfer learning to learn a variety of benchmark tasks. Since reward shaping relies on prior knowledge to influence the reward function, transfer learning can take advantage of a policy learned for one task and perform reward shaping for a similar task. In [78] transfer learning is applied from a simple version of the problem to a more complex one (e.g 2D to 3D mountain car and a Mario game without enemies to a game with enemies).

3.2.4 Inverse reinforcement learning

In many artificial intelligence applications such as games or complex robotic tasks, the success of an action is hard to quantify. In that case the demonstrated samples can be used as a template for the desired performance. In [50], apprenticeship learning (or inverse reinforcement learning) is proposed to improve a learned policy when no clear reward function is available; such as the task of driving. In such applications the aim is to mimic the behaviour of the human teachers under the assumption that the teacher is optimizing an unknown function.

Definition 15 *Inverse reinforcement learning (IRL) uses the training samples to learn the reward function being optimized by the expert and use it to improve the trained model.*

Thus, IRL obtains performance similar to that of the expert. With no reward function the agent is modelled as an MDP/R (S,A,T). Instead the policy is modelled after feature expectations μ_E derived from expert's demonstrations. Given m trajectories $\{s_0^{(i)}, s_1^{(i)}, \dots\}_{i=1}^m$ the empirical estimate for the feature expectation of the expert's policy $\mu_E = \mu(\Pi_E)$ is denoted as:

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}). \tag{3.1}$$

Where γ is a discount factor and $\phi(s_t^{(i)})$ is the feature vector at time t of demonstration i. A such the outer loop represents the number of demonstrations and the inner loop the length of each demonstration. The goal of the RL algorithm is to find a policy $\bar{\pi}$ such that $||\mu(\bar{\pi}) - \mu_E||_2 \leq \epsilon$ where $\mu(\bar{\pi})$ is the expectation of the policy [50].

[79] employ a maximum entropy approach to IRL to alleviate ambiguity. Ambiguity arises in IRL tasks since many reward functions can be optimized by the same policy. This poses a problem when learning the reward function, especially when presented with imperfect demonstrations. The proposed method is demonstrated on a task of learning driver route choices where the demonstrations may be suboptimal and non-deterministic. This approach is extended to a deep-learning framework in [80]. Maximum entropy objective functions enable straightforward learning of the network weights, and thus the use of deep networks trained with stochastic gradient descent [80]. The deep architecture is further extended to learn the features via Convolution layers instead of using pre-extracted features. This is an important step in the route to automate the learning process. One of the main challenges in reinforcement learning through trial and error is the requirement of human knowledge in designing the feature representations and reward functions [17]. By using deep learning to automatically learn feature representations and using IRL to infer reward functions from demonstrations, the need for human input and design is minimized. The inverse reinforcement learning paradigm provides an advantage over other forms of learning from demonstrations in that the cost function of the task is decoupled from the environment. Since the objective of the demonstrations is learned rather than demonstrations themselves, the demonstrator and learner do not need to have the exact skeleton or surroundings, thus alleviating challenges such as the correspondence problem. Therefore, it is easier to provide demonstrations that are generic and not tailor-made for a specific robot or environment.

In addition, IRL can be employed rather than traditional RL even if a reward function exists (given that demonstrations are available). For example, in [81] apprenticeship learning is used to derive a reward function from expert demonstrations in a Mario game. While the goals in a game such as Mario can be pre-defined (such as score from killing enemies and collecting coins or the time to complete the level), it is not known how an expert user prioritizes these goals. So in an effort to mimic human behaviour, a reward function extracted from demonstrations is favoured to a manually designed reward function. The draw back for inverse reinforcement methods is that they can be inefficient for complex problems as the reinforcement learning step is embedded in the optimization loop [82].

3.2.5 Data Aggregation

Definition 16 *Data aggregation allows the agent to perform an initially trained policy and provide corrections from the optimal policy. These samples provide an extra training dataset that contains situations that are not present in the original training set, but are likely to face the agent. This process of data aggregation can be performed iteratively to refine the policy and improve generalization.*

DAGGER [19] formulates the imitation learning problem as a structured prediction problem inspired by [83], an action is regarded as a sequence of dependent predictions. Since each action is dependent on the previous state, an error leads to unseen state from which the learner cannot recover, leading to compounded errors. DAGGER shows that it is both necessary and sufficient to aggregate samples that cover initial learning errors. Therefore, an iterative approach is proposed that uses an optimal policy to correct each step of the actions predicted using the current policy, thus creating new modified samples that are used to update the policy. As the algorithm iterates, the utilization of the optimal policy diminishes until only the learned policy is used as the final model.

[84] propose an algorithm called SIMILE that mitigates the limitations of [19] and [83] by producing a stationary policy that does not require data aggregation. SIMILE alleviates the need for an expert to provide the action at every stage of the trajectory by providing "virtual expert feedback" that controls the smoothness of the corrected trajectory and converges to the expert's actions.

Considering past actions in the learning process is an important point in imitation learning as many applications rely on performing trajectories of dependent motion primitives. A generic method of incorporating memory in learning is using recurrent neural networks (RNN) [85]. RNNs create a feedback loop among the hidden layers in order to consider the network's previous outputs and are therefore well suited for tasks with structured trajectories [86].

3.2.6 Active Learning

Active learning similarly adds new training data provided by the optimal policy based on the agent's initial performance. However, in active learning, the agent is able to query an expert for the optimal response to a given state, thus efficiently sampling the additional training samples.

Definition 17 *A classifier $h(x)$ is trained on a labeled dataset $D_K(x^{(i)}, y^{(i)})$ and used* *to predict the labels of an unlabelled dataset $D_U(x^{(i)})$. A subset $D_C(x^{(i)}) \subset D_U$ is chosen by the learner to query the expert for the correct labels $y^{*(i)}$. The active samples $D_C(x^{(i)}, y^{*(i)})$ are used to train $h(x)$ with the goal of minimizing n : the number of samples in D_C.*

Active learning is a useful method to adapt the model to situations that were not covered in the original training samples. Since imitation learning involves mimicking the full trajectory of a motion, an error may occur at any step of the execution. Creating passive training sets that can avoid this problem is very difficult.

One approach to decide when to query the expert is using confidence estimations to identify parts of the learned model that need improvement. When performing learned actions, the confidence in this prediction is estimated and the learner can decide to request new demonstrations to improve this area of the application or to use the current policy if the confidence is sufficient. Alternating between executing the policy and updating it with new samples, the learner gradually gains confidence and obtains a generalized policy that after some time does not need to request more updates. Confidence based policy improvement is used in [23] to learn navigation and in [39] for a macro sorting task.

In [87] active learning is introduced to enable the agent to query expert at any step in the trajectory, given all the past steps. This problem is reduced to i.i.d. active learning and is argued to significantly decrease the number of required demonstrations.

In [88] active learning is employed in human-robot cooperative tasks. The human and robot physically interact to achieve a common goal in an asymmetric task (i.e. the human and the robot have different roles). Active learning occurs between rounds of interaction and the human provides feedback to the robot via a graphical user interface (GUI). The feedback is recorded and is added to a database of training samples that is used to train the Gaussian mixture model that controls the actions of the robot. The physical interaction between the human and robot results in mutually dependent behaviour. So with each iteration of interaction, the coupled actions of the two parties converge into a smoother motion trajectory. Qualitative analysis of the experiments show that if the human adapts to the robots actions, the interaction between them can be improved; and that the interaction is more significantly improved if the robot in turn adapts to the human's action with every round of interaction.

In [89] the teacher initiates the corrections rather than the learner sending a query. The teacher observes the learner's behavior and kinesthetically corrects the position of the robot's joints while it performs the task. The learner tracks its assisted motion through its sensors and uses these trajectories to refine the model which is learned incrementally to allow for additional demonstrations at any point.

Tables 3.1-3.5 list previous work that utilize imitation learning in various domains. The tables show the year of the study, the data and algorithms used for learning and whether or not self-improvement was employed. The data column describes the sensory information and representations used for training. It is common to use raw or low-level sensory data which we refer to in games and simulated environments as engine data. These are unprocessed features obtained form the engine such as the positions of objects and the angles of joints. It is also common to process the sensory data whether from real or simulated sensors to obtain higher level features that are more useful for training. Engineered features refer to tailor made features that are extracted using functions specifically designed for a specific application. For example, in a driving task, a function that returns which lane the vehicle is currently in from can be calculated from positional data. In Table 3.1 shows imitation learning methods used in the navigation domain. This can include complicated driving systems as well as simulated cardinal movement. In navigation applications, it is important to be aware of the agents surroundings. Therefore, sensory information such as laser scanners and cameras are common.

Table 3.1: Imitation learning methods for navigation

Paper	Data	Learning Method	Self-improvement
Lin 1992[42]	2D positions	Artificial Neural Networks (ANN)	✓
Dixon et al. 2004[90]	Laser scanner	HMM	✗
Saunders et al. 2006[91]	engineered features	KNN	✗
Chernova et al. 2007[23]	engineered features	Gaussian Mixture Model (GMM)	✓
Coates et al. 2008[92]	3D position	Expectation Maximization (EM)	✗

Table 3.2 shows related work in object manipulation tasks. Object manipulation is a popular application in robotics and deals with grabbing and moving objects using the robot's actuators. In this domain it is common to use on-robot sensors to keep track of the state of the robot's pose as well as visual or laser sensors to observe the manipulated objects.

Table 3.2: Imitation learning methods for object manipulation

Paper	Data	Learning Method	Self-improvement
Pook et al. 1993[93]	on-robot sensors	Hidden Markov Model (HMM), K-Nearest Neighbor (KNN)	✗
Oztop et al. 2002[94]	engineered features	ANN	✗
Nicolescu et al. 2003[95]	sonar,laser,visual	graph based method	✓
Asfour et al. 2008[96]	extracted visual features	HMM	✗
Saunders et al. 2006[91]	engineered features	KNN	✗
Guenter et al. 2007[44]	on-robot sensors	GMR	✓
Geng et al. 2011[97]	engineered features	ANN	✗
Levine et al. 2015[60]	raw visual data	ANN	✓

Table 3.3 lists a variety of generic robotic tasks. Robotics is an attractive domain for AI research due to the huge potential in applications that can take advantage of sensing and motor control in the physical world. As in the more specific object manipulation tasks, visual and on-robot sensors are common in this table. Moreover, a number of studies employ body-worn sensors on the human teacher to obtain data about pose and joint movement.

Table 3.3: Imitation learning methods for robotics

Paper	Data	Learning Method	Self-improvement
Mataric 2000[98]	visual features	ANN	✗
Billard et al. 2001[99]	visual/marker data	ANN	✗
Ijspreet et al. 2002[29]	worn-sensors	Local Weighted Regression (LWR)	✗
Ude et al. 2004[100]	marker data	optimization	✗
Nakanishi et al. 2004[30]	on-robot sensors	LWR	✗
Bentivegna et al. 2004[38]	visual/on-robot sensors	KNN, LWR	✓
Schaal et al. 2007[32]	worn-sensors	LWR	✗
Berger et al. 2008[40]	on-robot sensors	direct recording	✓
Mayer et al. 2008[86]	visual/on-robot sensors	ANN	✓
Kober et al. 2009 [28]	on-robot sensors	LWR	✓
Ikemoto et al. 2012[88]	on-robot sensors	GMM	✓
Niekum et al. 2013[101]	visual/on-robot sensors	HMM	✓
Rozo et al. 2013[102]	on-robot sensors	HMM	✓
Mulling et al. 2013[31]	visual/engineered features	Linear Bayesian Regression	✓
Vogt et al. 2014[103]	visual/engineered features	ANN	✗
Droniou et al. 2014[85]	on-robot sensors	ANN	✗

Table 3.4 surveys imitation learning methods used in games. Games have become a popular domain for AI research as they have built-in control mechanisms and scoring measures that can facilitate the design and evaluation of learning methods. Most studies employ data derived from the games engine, whether using the low-level features or extracting high level engineered features. However, there has been recent interest in learning to play games from visual features using the frames displayed on the screen.

Table 3.4: Imitation learning methods for games

Paper	Data	Learning Method	Self-improvement
Giesler et al. 2002[104]	engineered engine features	Naive Bayes (NB), Decision Tree (DT), ANN	✗
Thurau et al. 2004[105]	engine data	bayesian methods	✗
Togelius et al. 2007[7]	engineered engine features	ANN	✗
Munoz et al. 2009[106]	engine data	ANN	✗
Cardamone et al. 2009[107]	engineered engine features	KNN, ANN	✗
Ross et al. 2010[19]	visual features	Support Vector Machine (SVM)	✓
Munoz et al. 2010[108]	engine data	ANN	✗
Ross et al. 2010[18]	visual features	ANN	✓
Ortega et al. 2013[67]	engineered features	ANN	✓
Silver et al. 2016[46]	2D positions	ANN	✓

Table 3.5 lists imitation learning efforts in simulated environments. Simulators can be used to learn real world tasks in a more controlled environment or as prosperously built benchmarks to facilitate AI research. Simulators alleviate many of the imitation learning challenges such as data capturing and sensor error; and thus allow development of new complex learning methods in a controlled and easily reproducible environment. Moreover, in a simulation, it is possible to execute policies faster than real-time and without physical limitations; therefore it is common to use self-improvement through reinforcement learning in conjunction with learning from demonstrations. Similar to games, simulation tasks often employ low-level or engineered engine features.

Table 3.5: Imitation learning methods for simulations

Paper	Data	Learning Method	Self-improvement
Aler et al. 2005[66]	engine data	PART	✓
Torrey et al. 2005[75]	engineered engine features	Rule based learning	✓
Judah et al. 2012[87]	engine data	linear logistic regression	✓
Vlachos 2012[109]	structured text data	online passive-aggressive algorithm	✓
Raza et al. 2012[26]	engineered engine features	ANN, NB, DT, PART	✓
Brys et al. 2015[78]	engine data/engineered features	Rule based learning	✓

3.3 Applications

Advances in imitation learning open the door for a variety of potential applications. Though commercialization and deployable products of such applications may not yet be realized, we present some of the directions taken in the literature and their potential applications. We pay special attention to applications related to the tasks presented in this work, namely navigation and multi-agent tasks.

Autonomous vehicles have been a popular concept in AI from its early days. And recently self-driving cars have been gaining a lot of attention from car manufacturers and tech companies alike. The advancement in sensors and on-board computers in modern cars have rekindled the interest in producing driverless vehicles commercially. Early research in imitation learning focused on this problem, proposing a method for learning to fly an aircraft from demonstrations provided via remote control [24] and self-driving road vehicles [110]. Since then, many researchers have directed imitation learning research to navigational or driving tasks [50] [23] [90] [91] [18] [106] [107]. Imitation in autonomous vehicles is not only concerned with low-level control, but demonstrations can also be used to learn car route selection and planning [79].

Assistive robotics aims to provide intelligent robots that can help elderly or recovering individuals in their day to day activities. Generalization is necessary in most applications as the human partner is usually untrained, so the robot must be able to behave robustly in unseen situations. The Human-robot interaction is not limited to physical assistance. Socially assistive robots can offer help with sociological and mental problems [111] [112] [113]. For robots to be effective in such a social context, their behavior must be human-like to be intuitively recognized by the human partner. Therefore, imitation learning has the potential to be an integral part in assistive robots. The same argument can be made for teaching infants using interactive robots. [88] incorporates interaction in the training process to account for the changing behavior of the human partner. The robot, therefore, can adapt to the human's reactions, as the human naturally makes similar adaptations.

Electronic games is a multi-billion dollar industry, and one in which immersion is an important factor. AI characters that act in a plausible manner within the environment can add greatly to the sense of immersion. Therefore, there is an ever growing demand for believable artificial intelligence to enhance gaming experiences [114]. Similar to Human-robot interaction, the possibilities are too great to consider with explicit programming [104], especially in modern games where the player has more freedom and control and the environments are becoming increasingly complex. A number of studies investigate the effectiveness of imitation learning in video games such as First Person Shooters [104] [115] [116], platformers [67] [18] [19] and racing[18] [19] [106] . These examples are applied to relatively simple problems, with limited action options and few features to consider from the environment; however, they show promising performances from imitation agents that could be extended to more complex games in the future. For instance, the study in [115] showed that not only are imitating agents subjectively believable, but that they also outperform other hand crafted agents. Electronic games provide a friendly platform for advancing imitation learning as they do not require additional systems such as sensors and hardware; and the cost of faults is low compared to applications where faults might endanger people or property. They can also be used as a testbed for artificial general intelligence [117].

Humanoid robots is one of the domains most associated with artificial intelligence. It is one of the most relatable domains, because many of its potential applications are quite obvious. The premise of humanoid robots is to replace some of the workload that humans do. These problems range from specific tasks such as physical chores and housework, to generic problem solving. Since most of the required tasks are already performed by humans, humanoid robot applications are inherently suitable for the concept of learning by imitation. Humanoid robots can learn to perform actions that only utilize part of their skeleton [99] [29] [103] or the entire body [40] [89] [100]. Since humanoid robots are commonly used to interact with humans in a social context, it is important not only to learn how to move but also how to direct its attention to important occurrences. In [118] a robot learns where to look by imitating a teacher's gaze and learns to estimate saliency based on the teacher's preferences.

Another field of robotic applications is **automation**.It is different from the aforementioned domains in that automation is more relevant to industrial tasks while the other domains aim to create AI for personal products and services. However, automation can still be useful in domestic applications [91]. While automation is not a new concept, learning by imitation introduces generalization and adaptability to automated tasks [95]. This means that the robot can act robustly in unseen situations such as the introduction of obstacles or changes in the shape or position of relevant objects. Generalization diminishes the need for supervision and human intervention between small tasks. Imitation learning research done in this direction focus on object manipulation, such as sorting and assembly tasks [91] [93] [94] [89] [95]. Another example of automation is medical procedures. In [86] a robot is trained to automate part of a surgical procedure from surgeons' demonstrations.

3.3.1 Multi-agent Imitation

In many real world application, autonomous agents are required to function in an environment along with other agents or humans, which can result in very dynamic environments. While most research has focused on single agent tasks, imitation learning and multi-agent applications can be a good fit. Learning from demonstrations can be improved in multi-agent environments as knowledge can be transferred between agents of similar objectives. On the other hand, imitation learning can be beneficial in tasks where agents need to interact in a manner that is realistic from a person's perspective. Following we present methods that incorporate imitation learning in multiple agents.

In [119] implicit imitation is used to improve a learner's RL model. A multi-agent setting enables an agent to learn by observing other agents performing a similar task using similar actions to those possessed by the agent. The mentor agent provides demonstrations by performing the task to optimize its objectives, so there is no need for a designated teacher; and the actions of an agent are unknown to other agents. A learner can observe the state changes resulting from the actions of a mentor agent and accordingly refine its model. This premise is useful for real applications where multiple agents act in the same environment. However, this work assumes that the agents are non-interacting, i.e., the consequences of one agent's actions are independent of other agents. Implicit imitation is closely related to transfer learning as a learner acquires the knowledge learned previously by a different learner. This application corresponds to a transductive transfer learning setting where the task is the same but the domains of the mentor and learner are different [74]. An interesting aspect of this approach is that an agent can learn different skills from different mentors. In one experiment two mentors are acting to achieve different goals; the learner uses observations from both agents to learn a more difficult task. Note however, that the tasks were designed so that a combination of the policies used by the mentors form an optimal policy for the learner's problem. [119]. This can be considered as an example of hierarchical transfer learning, where learning solutions to multiple problems can help achieve more complex tasks [76]. However, in this case the knowledge of the tasks is learned through observations and imitation.

Multi-agent settings add complexity to the imitation problem in a number of ways. The learner's state space can be significantly expanded to include the status of other agents, as observing the actions of other agents affects its decision. The reward function is also affected if multiple agents compete or collaborate towards the same goal. The complexity of reward functions can increase even if the agents do not share the same goal. In a competitive setting, in addition to maximizing its own reward, an agent aims to minimize its opponents reward. In a cooperative setting, the total reward of the team might be taken into consideration. These new complexities can be incorporated in the learning process at different levels. For example, [119] exploit the presence of multiple agents to learn from observation, however, robots are non-interacting and act independently of each other. So, the state space and reward function are not affected. In [120], multiple agents collaborate as a team to keep ball from other team in a soccer simulation. The whole team learns through reinforcement learning, but each agent learn a separate policy as different actions executed by the different agents are rewarded with the same reward. While in [26], the soccer agents learn different roles that complement each other to maximize the common goal. In a defensive task, one agent tries to recover the ball from the attacking opponents while the other falls back to act as a goal keeper. The roles are interchangeable, so each agent learns both skills as well as when to assume one of the two roles. In [121] a team of agents is treated as a pattern of policies rather than individual agents. That is a pattern that connects agents in a team that perform complementary roles. This method enables a team of agents to be scaled significantly after training without requiring retraining of different agents for similar roles.

Multi-agent learning from demonstration can however introduce new challenges in terms of acquiring demonstrations. In [39] where active learning is employed in a multi-robot environment, the human expert is required to interact simultaneously with multiple robots. A system is then developed to divide the expert's time among the learners based on their need, by attracting the expert's attention through audio visual cues in order to query information.

3.3.2 Model-based learning and Planning

The methods proposed in this thesis focus on model-free learning approaches that teach the agent to react to the observed state. However, imitation learning can be applied in model-based approaches and used for planning.

Definition 18 *Model based learning is a paradigm where a model M of the environment exists that allows us to know the transition probability $T(s'|s,a)$ of future states. Intuitively, the model provides the result of performing actions before they are executed.*

Definition 19 *Planning is determining a sequence of actions $A = \{a_1, a_2..a_n\}$ to transition from an initial state s_1 to a target state s_t. The entire trajectory of actions A is planned before any actions are executed.*

Model based learning is closely related to planning as it allows the policy to plan long term strategies while knowing the state and the transition probabilities at each step in the trajectory [122, 42].

In [31] a robot arm is trained to play table tennis against an opponent. A model of the physics controlling the ball is use to predict the position of the ball in future time steps. This allows the policy to plan the motions of the paddle to meet the ball. This task illustrates the leverage that a model provides. A model of the ball's movement allows the policy to learn how to hit the ball and prepare for the next hit. A model of the opponent could additionally allow the policy to plan long-term strategies to trick the opponent.

However, preliminary planning can be performed without a model. In [40], the entire trajectory of actions is defined in one shot. This trajectory, while not optimal due to the dynamic nature of the task, serves as a rudimentary sequence that is optimized iteratively using genetic algorithms. In each iteration, the genetic algorithm evaluates the sequence as a whole rather than per action.

A popular approach in autonomous agents is model-based reinforcement learning [123, 124] which provides the next state as well as a reward for a given state-action pair. This approach can be integrated with learning from demonstrations through model-based apprenticeship learning [125]. It is especially useful if the model is differentiable as this allows the policy to be trained directly on the targets using gradient descent with minimal interaction with the environment [126].

The challenge in model-based methods is acquiring the model, which can require substantial expert knowledge and effort and is not feasible in many cases. It is important not to confuse simulators and models as simulators do not predict the state transition before it happens. For example in [87] an active imitation learning approach is used to replace a model-based IRL approach by utilizing a simulator in the absence of a model. To alleviate this challenge, there are recent attempts to learn dynamic models through interaction with the environment [127]. This approach is very attractive as it provides a general method of creating differentiable models.

3.4 Conclusion

This chapter surveys the relevant literature in the area of imitation learning, with a focus on methods that employ deep learning and reinforcement learning and that address relevant applications. Imitation learning methods are categorized into direct imitation methods that learn a supervised policy from demonstrations and self-improvement methods that improve generalization by refining the policy according to the agent's initial performance. There are several different approaches to refine a policy, we categorize them into reinforcement learning, transfer learning, data aggregation, active learning, apprenticeship learning and optimization. Reinforcement learning learns a policy through trial and error based on feed-back from the environment; and thus can learn a policy from scratch or refine an existing one. Reinforcement learning has shown very impressive results in various applications, but can still struggle with sparse rewards. Moreover, trial and error may not be appropriate in some applications due to safety concerns or physical limitations. Apprenticeship learning creates a reward function from the given demonstrations and learns a policy using reinforcement learning. This approach is especially suitable when it is difficult to define a reward function, however IRL approaches can be computationally challenging. Optimization methods such as genetic algorithms and particle swarm methods can be also used to improve a policy based on a fitness function, if the parameters of the policy can be represented as an optimization solution. Data aggregation approaches are some of the most common to enable imitation agents to generalize. By performing the initial policy in a dynamic environment, the agent is able to collect a new dataset from the expert that is relevant to the state space visited by the agent. However, like RL approaches, data aggregation may not be appropriate for application where executing mediocre policies in is not feasible. Similarly, active learning acquires extra training samples by performing the initial policy in the environment. However, in active learning the agent queries the expert for the new samples in order to reduce redundancy and focus on situations where the current policy is ambiguous. While this alleviates the need for extensive human intervention, in some tasks it is difficult for the teacher to provide an isolated action mid-trajectory. Transfer learning allows knowledge from one task to be used to leverage learning in a different task, such as providing context or an initial policy.

Starting with a better than random policy can also alleviate some of the challenges of RL in physical applications. Model-based learning and planning allow learning complex strategies over a sequence of actions. These methods are very attractive if a model is available or is feasible to develop. However, model-free reactive approaches can also learn long-term strategies for example through delayed rewards or policies that retain memory of past experiences. Common domains in imitation learning are also surveyed. Potential real world applications are highlighted as well as common research tasks that facilitate developing new methods.

Chapter 4

Learning Representations from raw visual data

This chapter described the proposed method for learning representations from raw visual data. The method details the processes of acquiring the demonstrations, formulating the data for training, and the training model used to learn from the data. The proposed method is generic, and the network is trained without knowledge of the task, targets or environment in which it is acting. The approach and network architecture described in this chapter is used in conjunction with other contributions in chapters 5, 6 and 7 to tackle different tasks in autonomous navigation applications. Experiments conducted and results to evaluate these tasks are presented in the respective chapters. The proposed feature representation method is published in [10].

4.1 Introduction

In autonomous applications, representing the state in a way that is adequate to train the agent is an important challenge. A special case is imitation learning where demonstrations are provided by an expert, so it is required to represent the demonstrations as well as the agent's observations in a manner that is suitable for learning. Part of the imitation learning processes is to design how the demonstrations are captured and how to present them to the agent. When capturing data, the important question is: what to imitate? In most real applications, the environment is often too complicated to be represented in its totality, because it usually has an abundance of irrelevant or redundant information. It is therefore necessary to consider which aspects of the demonstrations we want to present to the learner. This will help decide which sensory data to acquire and from what perspective. Once the demonstrations are captured the next question is how to represent the captured sensory data as training samples. This includes deciding the feature vector that describes a single sample.

Raw data captured from the source can be used directly for training. However, these feature can be very high dimensional and contain a large number of redundant and irrelevant features. Thus, using raw features for training may be inefficient or infeasible in many problems. For example, when dealing with high DOF robots, describing the posture of the robot using the raw values of the joints can be ineffective due to the high number of dimensions. This is more pronounced if the robot only uses a limited number of joints to perform the action; rendering most of the features irrelevant. This issue also applies to visual information. If the agent observes its surroundings using visual sensors, it is provided with high dimensional data in the form of pixels per frame. However, at any given point, most of the pixels in the captured frame would probably be irrelevant to the agent or contain redundant information. It is therefore common to extract features from raw data for machine learning applications. We make a distinction between two types of extracted features: manually tailored features which are extracted according to expert knowledge of the task and automatically extracted features which are dependant on the data rather than human knowledge. Manually tailored features are extracted from the sensory data using specially designed functions. These methods incorporate expert knowledge about the data and application to determine what useful information can be derived by processing the raw data. Manually designed feature functions are popular with learning from visual data and play an important part in computer vision methods that are used to teach machines by demonstration. However, these features are tailor made for a specific task and sensory input and thus hinder a general learning from demonstration process. This is because for any new task (or even a change in the task), an expert is required to design new feature extractors that are effective for this case. On the other hand, automatic feature extraction can process raw data to create adequate and efficient feature representations based on the provided data. The most relevant information is extracted and mapped to a different domain usually of a lower dimensionality without requiring expert knowledge. While feature extraction methods such as Principal Component Analysis (PCA) have been used to create feature representation in robotics [128, 88, 40, 103], deep learning [129] has shown great success in learning from raw visual data. Learning representations through deep networks can be particularly effective because the final supervised loss is taken into consideration . Convolutional neural networks have been shown to surpass even manually designed features for many computer vision tasks and have the advantage of learning end-to-end without requiring intervention. Figure 4.1 shows the relations between different feature representations.

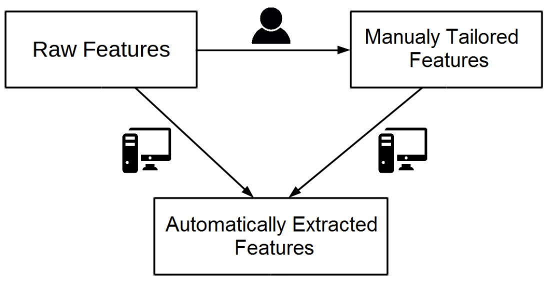


Figure 4.1: Features representations and their relationships. Raw features can be manually manipulated to extract tailored features. While manually tailored or raw features can be automatically processed to extract feature representation.

Autonomous navigation is an important problem that has been recognized in AI research from an early stage. Navigation is an important skill in various robotic applications such as household robotics as well as the main problem in applications such as autonomous vehicles. Learning from demonstrations lends itself to navigation problems as it is difficult, even for experts, to identify an optimal strategy for agents to follow in complex environments. Prioritizing different aspects of navigation such as speed, safety and avoiding obstacles can be better inferred from demonstrations [50]. However, navigation from visual data is challenging as only part of the environment is visible at a time. Moreover, the view of the agent changes constantly as it moves around the environment making it more difficult to observe relations between subsequent states and extract relevant features. This is in contrast for example to object manipulation tasks where a static view contains all the information needed by the agent, and changes from one frame to the next can be more easily tracked.

We now review related and relevant work in autonomous navigation and using deep learning for feature extraction in intelligent agents.

4.1.1 Autonomous Navigation

Navigation is an important skill for intelligent agents due to its relevancy to a variety of applications. Navigation can be a main task as in autonomous vehicle applications [130, 131, 132, 133, 134, 135, 136] or as a base skill for other tasks such as humanoid robots which need to move before performing other tasks [23, 91] An early work [130] proposed a method for learning autonomous control of an aerial vehicle from demonstrations. Since then several papers have proposed learning autonomous aerial navigation using demonstrations [137] and reinforcement learning [131][132][58]. In [135] a robot learns how to navigate through a maze based on its sensory readings. The information available to the robot is a stream from an infra-red (IR) sensor and input from a controller operated by a teacher. The agent learns to map its sensory data directly to the motor primitives provided by the controller. The IR data provides information about the proximity of objects. This sensory information does not allow the agent to differentiate between different objects. In [90] a laser sensor is utilized to enable the agent to detect and identify relevant objects. Instead of mapping the sensory data directly to motor primitives, the agent learns to identify sub-goals from its observations. A more detailed representation of the environment can be provided by visual data. High dimensional visual data can be efficiently provided to intelligent agents thanks to advances in computational resources and communication technology. An agent learns to play a racing game from visual data in [18]. A teacher plays the game using a controller, and the controller's input is captured along with the game's video stream to create a training dataset. The video stream is stored as raw pixels and down sampled versions of the frames are input into a neural network. In [138] a deep reinforcement learning algorithm is used to teach an agent in a racing simulator from raw visual features. The learned policy maps the high dimensional visual input to multiple continuous outputs such as steering and pressing the acceleration pedal. Another racing application is demonstrated in [106] where the training algorithm uses features extracted from the simulator (such as the position and speed of the car). It is shown that learning from demonstration can be used to handle high degree of freedom low level actions, however, features such as those extracted from the simulator are difficult to produce in real world applications. Learning from visual information is not limited to the point of view of the agent. In [133] an imitation learning method is proposed to train a vehicle to navigate over long distances by learning from overhead data captured from satellite and aerial footage. Recently, state of the art deep reinforcement learning methods have been evaluated on 3D navigation tasks [139][140]. However, these benchmark tools are not publicly released.

4.1.2 Deep Learning

Deep learning approaches can be used to extract features without expert knowledge of the data. These approaches find success in automatically learning features from high dimensional data; especially when no established sets of features are available. In a recent study [141], Deep Q-Learning (DQN), a version of Q-learning that employs deep neural networks, is used to learn features from high dimensional images. The aim of this technique is to enable a generic model to learn a variety of complex problems automatically. The method is tested on 49 Atari games, each with different environments, goals and actions. Therefore, it is beneficial to be able to extract features automatically from the captured signals (in this case screen-shots of the Atari games at each frame) rather than manually design specific features for each problem. A low resolution (84×84) version of the coloured frames is used as input to a deep convolutional neural network (CNN) that is coupled with Q based reinforcement learning to automatically learn a variety of different problems through trial and error. The results in many cases surpass other AI agents and in some cases are comparable to human performance. Similarly, [142] use deep neural networks to learn from video streams in a car racing game. Note that these examples utilize deep neural networks with reinforcement learning, without employing a teacher or optimal demonstrations. However, the feature extraction techniques can be used to learn from demonstrations or experience alike. Since the success of DQN, several variations of deep reinforcement learning have emerged that utilize actor-critic methods [140] [138] which allow for potential combinations with learning from demonstrations. In [59] learning from demonstrations is applied on the same Atari benchmark [143]. A supervised network is used to train a policy using samples from a high performing but non real-time agent. This approach is reported to outperform agents that learn from scratch through reinforcement learning. In [60] deep learning is used to train a robot to perform a number of object manipulation tasks using guided policy search

4.2 Proposed Method

In this section we detail our proposed method for learning feature representations for navigation tasks from demonstrations. We begin by describing the process of collecting demonstrations and following the deep neural network model used for training. The full process is illustrated in figure 4.2. Implementation of this method is available at `https://github.com/ahmedsalaheldin/ImitationMASH.git`

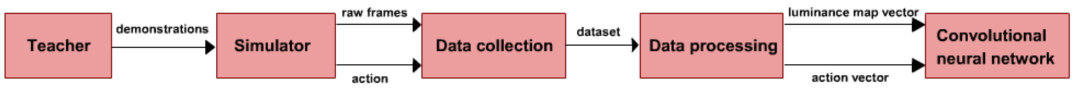
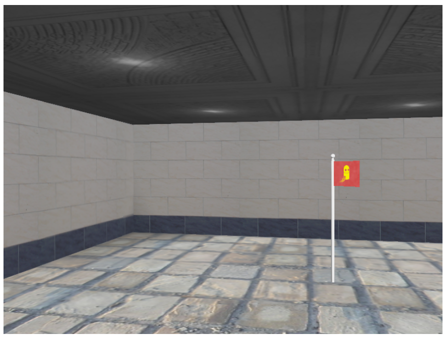


Figure 4.2: Workflow of the learning representation process. The process starts by a teacher providing demonstrations on a given simulator. The demonstrations are collected and processed into a dataset that is fed into a deep convolutional neural network. The network learns high level representations of the raw input features as well as mapping to the action space.

4.2.1 Representing Demonstrations

A teacher providing demonstrations may be assumed to be optimizing an unknown optimal function. Therefore as the teaching policy we use a deterministic optimal policy π^* to control the agent. This policy has access to information from the simulator that is hidden from a human or intelligent agent player such as the position of the agent and the target in 3D space. This information is used to deterministically calculate the optimal action at a given time. The expert provides demonstrations by controlling the agent and performing the task in an optimal manner. In this case the observations are captured from the first person perspective of the agent. Therefore, the agent will relate to the captured demonstrations when it attempts to perform the task and is presented with observations from the same perspective. Figure 4.3 shows examples of observations captured from the demonstrations.


(a) Screenshot from "reach the flag" task

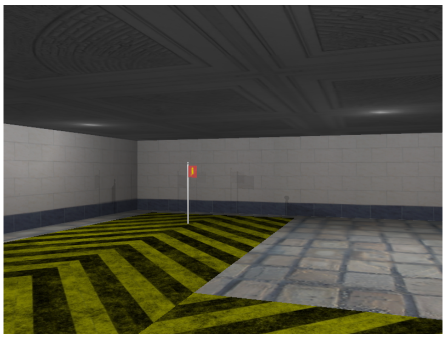

(b) Screenshot from "follow the line" task

Figure 4.3: Example of observations captured during demonstration

For each frame t the view of the agent x_t is captured as well as the action y_t chosen by the optimal policy. This pair (x_t, y_t) is added to the dataset of demonstrations $D = (x, y)$ where x_t is a 120×90 image captured from the first person view of the agent and $y_t = \pi^*(x_t)$ is the action chosen by the expert from a set of possible actions. In the navigation applications at hand the set of actions is $MoveForward, MoveBackwards, TurnLeft, TurnRight$. For each state observation, only the current frame is used unlike deep reinforcement learning methods [144][141] that commonly represent the state by a sequence of rendered frames. The assumption is made that the Markov property holds for the navigation tasks at hand that are formulated as MDPs. That is, at any instance, the current state is sufficient to make a decision and any previous actions and states need not be included in the representation of the current state. This assumption is made based on the expert policy, as the teacher only considers the current state when making a decision. In that case training an imitation learning policy is reduced to a supervised image classification problem;

where the current view of the agent is the image and the action chosen by the teacher is the label. Subsequently the trained agent will be able to predict a decision (as it would be taken by the teacher) given its current view. The captured dataset D is used to train the policy π such that $u = \pi(x, \alpha)$. Where x is a 120×90 image and u is the action predicted by policy π for input x and α is the set of policy parameters that are changed through learning. The policy is trained using a neural network that extracts a high level representation of the observation x and maps it to actions.

4.2.2 Deep convolutional Neural network

To learn the policy we employ a deep convolutional neural network. The proposed network uses several convolution layers to automatically extract features from the raw visual footage. Each convolutional layer learns a number of kernels that act as feature extractors to feed a higher level representation to the next layer. Then a fully connected layer is used to map the learned features to actions. Each convolution layer is followed by a pooling layer that down-samples the output of the convolution layer. The convolution layers take advantage of spacial connection between visual features to reduce connections in the network. The pooling layers reduce the dimensionality to further alleviate the computations needed.

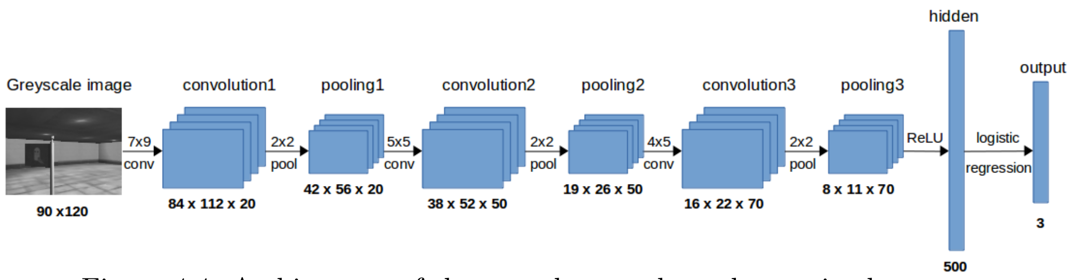


Figure 4.4: Architecture of the neural network used to train the agent

Our network follows the pattern in [141]. It consists of 3 convolution layers each followed by a pooling layer. This is a popular architecture that has proven to be effective and efficient in autonomous agents learning from pixels. The input to the first layer is a frame of 120×90 pixels. We apply a luminance map to the coloured images to obtain one value for each pixel instead of 3 channels, resulting in a feature vector of size 10800. This transformation allows us to use one channel for grey-scale instead of three channels for the RGB colours. Using a single channel in place of three significantly reduces the number of connections in the network and allow training more efficiently. Each convolutional layer is followed by a pooling layer to further reduce the dimensionality. Following is a fully connected layer with a rectifier unit activation function and finally an output layer which directly represents the action available to the agent. Figure 4.4

and Table 4.1 show the architecture of the network. The filter sizes for the three layers are 7×9, 5×5 and 4×5 respectively; and the number of filters are 20, 50 and 70 respectively. The filter sizes and number of filters are set by experimentation that is restricted according to the size of the input image and the number of pooling layers. The pooling layers all use maxpool of shape (2,2). Following the last convolution layer is a fully connected hidden layer with rectifier activation function and fully connected output layer with three output nodes representing the 3 possible actions. Although the application allows for 4 actions, the expert policy never used $MoveBackwards$ in the demonstrations. The training parameters are set through experimentation and rmsprop optimizer is used to help convergence and avoid over-sensitivity to the training parameters [145]. Table 4.1 summarizes the architecture of the network.

Table 4.1: Neural network architecture

Layer	Size of activation volume
Input	120 * 90
Conv1	7 * 9 * 20
Conv2	5 * 5 * 50
Conv3	4 * 5 * 70
FC	500
Output(FC)	3

.

4.3 Conclusion

In this chapter a novel method for representing demonstrations is proposed. The method for capturing the demonstrations and representing observations is detailed. Following, a neural network is used to extract feature representations from the captured observations and map them to atomic actions. The representation process and the architecture of the deep convolutional neural network presented in this chapter are used for multiple navigation tasks in the following chapters. This shows the effectiveness of the feature representations and generality of the learning process across several tasks in chapters 5, 6 and 7. The experiments and results for each problem are presented in the respective chapters.

Chapter 5

Active Data Aggregation

This chapter presents the proposed method for deep active data aggregation. To improve the generalization ability of a direct imitation agent, additional training demonstrations are provided based on the agent's initial performance in a given task. Active learning is employed to efficiently and effectively sample the additional training data. Experiments are conducted on 4 different navigation tasks in a 3D simulated environment. The results show that active learning can signicantly improve the learned policy using a relatively small number of training samples. The method proposed in this chapter is published in [10].

5.1 Introduction

Training a neural network to map raw pixels to actions as presented in chapter 4 can create an autonomous policy capable of effectively learning navigation tasks. However, in many imitation learning applications direct imitation is not sufficient for robust behaviour. One of the common challenges facing direct imitation is that the training set does not fully represent the desired task. The collected demonstrations only include optimal actions performed by the teacher. If the agent deviates even slightly from the optimal trajectory, it arrives at a state that was not represented in its learned policy [7]. This poses a great generalization challenge, especially in dynamic environments. Since an agent's actions dictate the next state of the environment, the assumption that samples are independent and identically distributed (i.i.d) is no longer valid. Therefore the problem is prone to propagation of error that could result in the failure of the task as the agent is essentially presented with samples from a different distribution to the training set. It is therefore necessary to provide further training to the agent based on its own performance of the task. A popular method to improve generalization is data aggregation [19], where the agent is allowed to perform the task and the expert provides the optimal actions for the states visited by the agent to use as additional training data. The original training set is iteratively augmented with the additional training samples and the policy is re-trained in each iteration. This method is shown to improve the generalization of agents in a number of applications, however, providing optimal actions for each state visited by the agent is inefficient. The augmented data sets are likely to contain a lot of redundancy and may add to the imbalance in the dataset. Moreover, using the augmented data set to re-train the model during each iteration can be extremely slow when dealing with large complex models such as deep neural networks. Therefore we propose an active data aggregation approach that samples a small but effective set of samples to augment the training set.

5.2 Proposed Method

Active learning is employed to improve the initial policy learned from demonstrations. This is achieved by acquiring a new data set to train the agent that emphasizes the weaknesses of the initial policy. The agent is allowed to perform the task for a number of rounds. For each prediction the network's confidence is calculated, and if the confidence is low the optimal policy is queried for the correct action. The action provided by the teacher is performed by the agent and is recorded along with the frame image. The confidence is measured as the entropy of the output of the final layer in the network. The entropy $H(X)$ is calculated as:

$$H(X) = -\sum_{i} P(x_i) \log_2 P(x_i) \tag{5.1}$$

Where X is the prediction of the network, $P(x_i)$ is the probability distribution produced by the network for action i.

The active samples are added to the training set and used to update the initial policy. We find that updating a trained network using only the active samples results in forgetting the initial policy in favour of an inadequate one rather than complementing it. Therefore the training set is augmented with the active samples collected from the playing agent. The augmented dataset is used to update the network that was previously trained. We find that it is easier and faster for the network to converge if it is pre-trained with the initial dataset than training from scratch. Algorithm 1 shows the steps followed to perform active learning.

Low confidence predictions are mainly caused by situations that were not covered by the training data. Therefore, for active learning to be effective, it is important that it is performed in the simulation rather than on a collected dataset. Because by performing its current policy in the simulation, the agent arrives at unfamiliar situations where it is not confident in its behaviour and thus utilizes active learning.

Algorithm 1 Active Learning Algorithm

```
 1: **Given:** A policy $\pi$ trained on a Data set $D = (x_i, y_i)$
            Confidence threshold $\beta$
 2: **while** Active_Learning **do**
 3:     $x$ = current_frame
 4:     $u = \pi(x, \alpha)$
 5:     $H(X) = -\sum_i P(u_i) \log_2 P(u_i)$
 6:     **if** $H(X) < \beta$ **then**
 7:         $y = Query(x)$
 8:         perform action $y$
 9:         add $(x, y)$ to $D$
10:     **else**
11:         perform $max(u)$
12: Update $\pi$ using $D$
```

Table 5.1 summarizes key differences between the proposed method, Deep Active Imitation (DAI), and other approaches that use deep learning that learn from raw pixels, Deep-Q-Networks (DQN) [144] and Deep Guided Policy (DGP) [60]. The table shows differences in the approaches such as the methods used to generalize the policy to unseen scenarios, the methods used to gather demonstrations and how the states are constituted from the captured frames. Moreover, it shows differences in the tasks and environments in which the different approaches are utilized. The viewpoint is the perspective from which the state of the environment is captured. Having a fixed point of view may help keep track of changes in the state while having a dynamic viewpoint can be more challenging as the scene changes completely with small movements in the viewpoint. The trajectory refers to the sequence of steps typically needed to successfully complete the task. A longer trajectory can be harder to learn as small errors mid trajectory can propagate and cause failure to reach the target. The environments refers to the settings in which the experiments are conducted. The environment can be randomized at every run, so the agent is faced with unfamiliar states. The more random the environment, the more the agent's policy needs to generalize to the changing circumstances.

Table 5.1: A comparison of deep learning agent approaches

Method	**DAI**	**DQN**	**DGP**
Input	Pixels	Pixels	Pixels
Generalization	Active learning	Q-learning	Policy Gradient
State Representation	Greyscale frame	4 Greyscale frames	RGB frame
Demonstration Source	Teacher	Reinforcement learning	N/A
Viewpoint	Dynamic	Static	Static
Trajectory	Long	Various	Shorter
Environment	3D simulator	2D simulator	Real world
Randomization	Extensive	Extensive	Limited

5.3 Experiments

We conduct our experiments in the framework of mash-simulator [14]. Mash-simulator is a tool for benchmarking computer vision techniques for navigation tasks. The simulator includes a number of different tasks and environments. As learning aids, for each task the simulator provides a reward system as well as optimal policies for a number of tasks. Each task also has a scoring system for evaluation. The tasks differ in the goals that are set, the score system and the failure criteria. The environments differ in both visual and geometric design. Moreover, for any environment, several features are procedurally generated such as the dimensions of the environment, lighting conditions and the positions of relevant objects. All the navigation is viewed from the first person perspective. The player has 4 possible actions: ‘Go forward’, ‘Turn left’, ‘Turn right’ and ‘Go back’. Although there are 4 possible actions, the action ’Go back’ was never used in the demonstrations by the optimal policy, as its function can be achieved by the remaining actions. Therefore the network is only presented with 3 classes in the training set and thus has 3 output nodes. Implementation of the proposed methods in this chapter is available at `https://github.com/ahmedsalaheldin/ImitationMASH.git`

5.3.1 Tasks

The experiments are conducted on the following 4 navigation tasks:

Reach the flag

This task is set in a single rectangular room with a flag placed randomly in the room. The goal is to reach the flag. The task fails if the flag is not reached within a time limit.

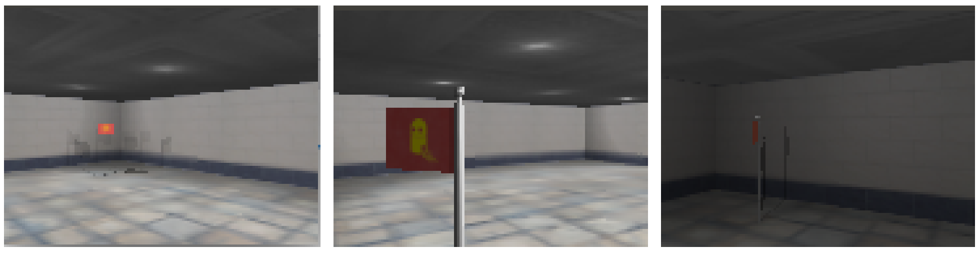


Figure 5.1: sample images from "Reach the flag"

Follow the line

This task is set in a room with directed lines drawn on the floor. The lines show the direction to follow in order to reach the flag. The target is to follow the line to the flag, and the agent fails if it deviates from the line on the floor.

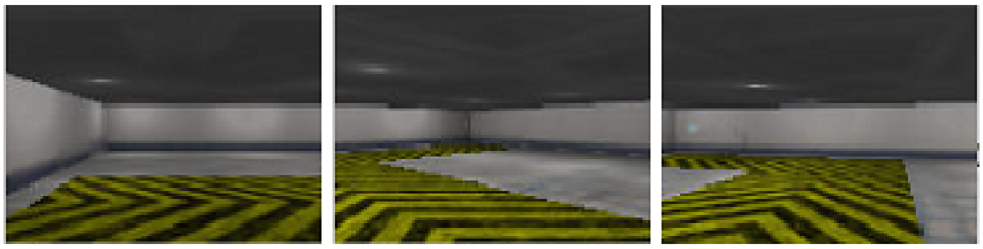


Figure 5.2: sample images from "Follow the line"

Reach the correct object

In this task two objects are placed on pedestals in random positions in the room. The objective is to reach the pedestal with the trophy on it. The task fails if a time limit is reached or if the player reaches the wrong object. The wrong object has the same material of the trophy and can take different shapes.

Eat all disks

This task is set in a large room containing several black disks on the floor. The target is to keep reaching the disks. A disk is 'eaten' once the agent reaches it and disappears.

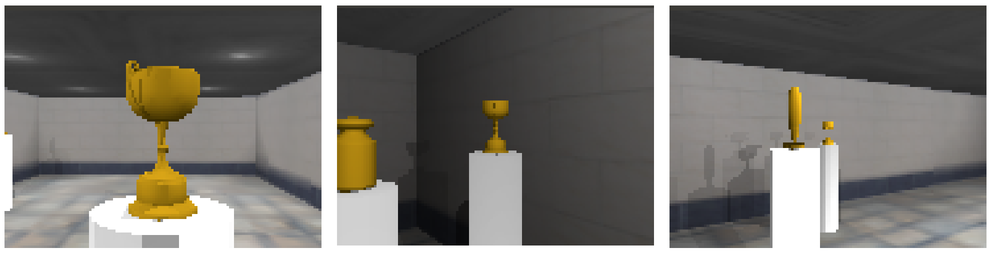


Figure 5.3: sample images from “Reach the correct object”

New disks appear when one is eaten. The goal of this task is to eat as many disks as possible within a time limit.

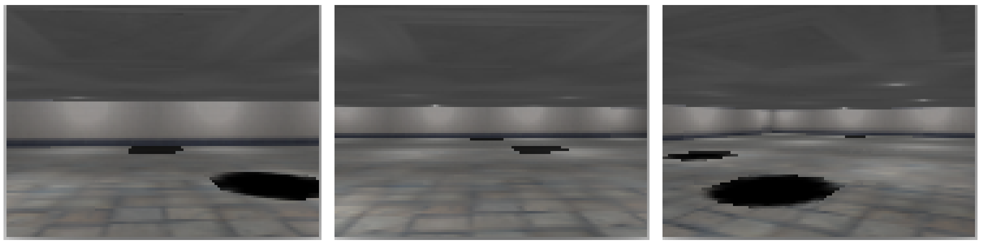


Figure 5.4: sample images from “Eat all disks”

Figures 5.1 - 5.4 show sample images of the 4 tasks in the 120 × 90 size used in the experiments.

5.3.2 Setup

To evaluate the proposed methods, the performance of the agent is measured over 1,000 rounds. A round starts when the task is initialized and ends when the agent reaches the target or a time limit is reached. The number of frames in a round might vary depending on how fast the agent can reach the target. For all tasks, in each round the environment is randomized including room size and shape, lighting and the location of the target and the agent. A time limit is set for each round and the round fails if the limit is reached before the agent reaches the target. The time limit is measured in frames to avoid any issues with different frame rates. The time limit is set as the maximum time needed for the optimal policy to finish the task; which is 500 frames for ”Reach the flag” and ”Reach the correct object” and 5000 frames for ”Follow the line”. In “Eat all disks” the task is continuous, so a time limit was set to match the total number of frames in the other tasks.

5.3.3 Implementation details

Inter-process communication is used to communicate data across the different components of the testbed. The agent acts as a client and communicates with the simulator via a TCP connection as follows: The agent requests a task from the server, the server initiates a round and sends an image to the client. The client sends an action to the server. The server calculates the simulations and responds with a new image. Figure 5.5 shows a flowchart of the data collection process.

The network used for prediction is also decoupled from the agent. The network acts as a predicting server where an agent sends frames that it receives from the simulator and in return receives a decision from the network. The entire process of communication with both servers occurs in real time. This implementation facilitates experimentation, as making changes to the network does not affect the client or the simulator server. Moreover, it is easier to extend this system to physical robots. A predicting server can be located on the robot or on another machine if the robot's computational capabilities are not sufficient. A predicting server can also serve multiple agents simultaneously. The agent client is implemented in C++ to facilitate interfacing with the mash-simulator. The predicting server and the training process are implemented in python using the Theano deep learning library [146]. Figure 5.6 shows a flowchart of the agent performing a task.

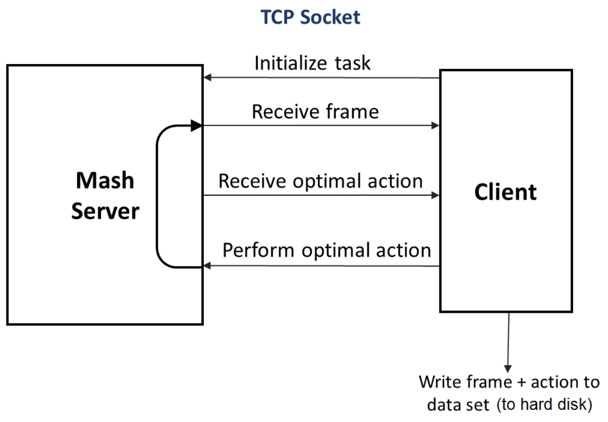


Figure 5.5: Dataset Collection Flowchart

5.3.4 Results

In this section we present the results of the proposed method. The same network and parameters are used to learn all tasks. For each task 20,000 images are used for training. Testing is conducted by allowing an agent to attempt the tasks in the mash-simulator

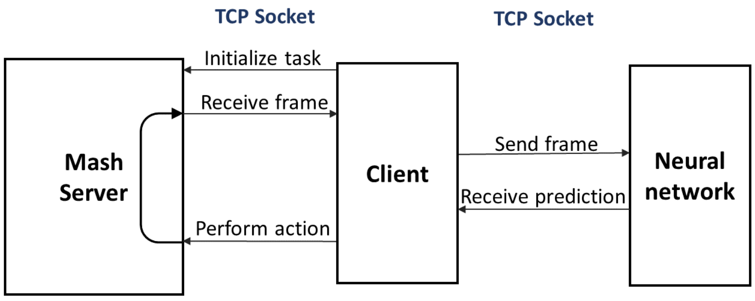


Figure 5.6: Imitation Agent Playing Flowchart

and recording the number of successful attempts. An agent's performance for the first 3 tasks is evaluated as the percentage of times it reaches the target in 1,000 rounds. For "Eat all disks", the performance is measured as the number of disks eaten in 1,000 rounds. We also report the classification error on an unseen test set of 20,000 images collected from the teacher's demonstrations.

Table 5.2 shows the results for the first 3 tasks. The success measure is the percentage of rounds (out of 1000) in which the agent reached the target. While error is the classification error on the test set collected from the teacher's demonstrations. The agent performs well on "Reach the flag" and is significantly less successful in the other two tasks. "Follow the line" is considerably less fault tolerant than "Reach the flag". As a small error can result in the agent deviating from the line and subsequently failing the round. Whereas in "Reach the flag" the agent can continue to search for the target after a wrong prediction. In "Reach the correct object" the agent is not able to effectively distinguish between the two objects. This could be attributed to insufficient visual details in the training set, as the teacher avoids the wrong object from a distance. Qualitative analysis of "Reach the flag" shows that the agent aims towards corners as they resemble the erect flag from a distance. Upon approaching the corner, as the details of the image become clearer, the agent stops recognizing it as the target and continues its search. While this did not pose a big problem in the agent's ability to execute the task it is interesting to examine the ability of CNNs to distinguish small details in such environments. It is also worth noting that the teacher's policy for "Reach correct object" does not avoid the wrong object if it is in the way of the target and achieves 80.2% success rate

Table 5.2: Direct Imitation results

Task	reach the flag	reach object	follow the line
success	96.20 %	53.10%	40.70%
error	2.48%	4.06%	0.86%

Table 5.3 shows results for the 4th task "Eat all disks". The table shows the score of the agent compared to the score achieved using the optimal policy. The agent is shown to achieve 97.9% of the score performed by the optimal policy.

Table 5.3: "Eat all disks" results

Task	Agent	Optimal policy
score	1051	1073
error	1.70%	-

To improve the agent's ability to adapt to wrong predictions and unseen situations, active learning is used to train the agent on "Follow the line". In the other tasks where the agent searches for the target, the optimal policy remembers the location of the target even if it goes out of view due to agent error. Therefore active learning samples include information that is not represented in the visual data available to the agent and thus degrade the performance. This can be rectified by devising a teaching policy that does not use historical information, or by incorporating past experience in the learned model.

Figure 5.7 shows the results of active learning on the "Follow the line" task. Active learning is demonstrated to significantly improve the performance of the agent using a relatively small number of samples. Comparing the classification error with success rate emphasizes the point that the errors come from situations that are not represented in the teacher's demonstrations.

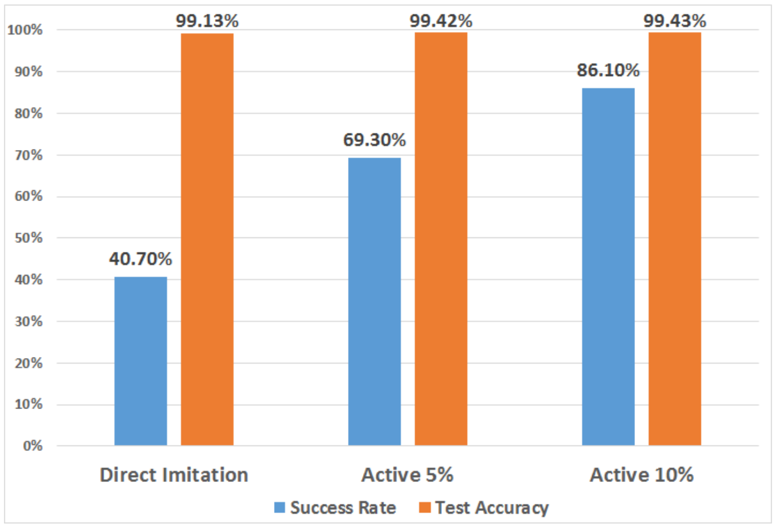


Figure 5.7: Results for active learning on "follow the line" task

The task in which the time limit affected the performance was "Reach the flag". As the agent continues to follow its policy in search of the flag even after performing wrong predictions. The effect of the time limit is evaluated in Figure 5.8 which presents the success rate of “reach the flag” task with different time limits. The horizontal axis represents the time limit as a percentage of the maximum time needed by the teacher. The graph shows that the longer the agent is allowed to look for the target the higher the success rate.

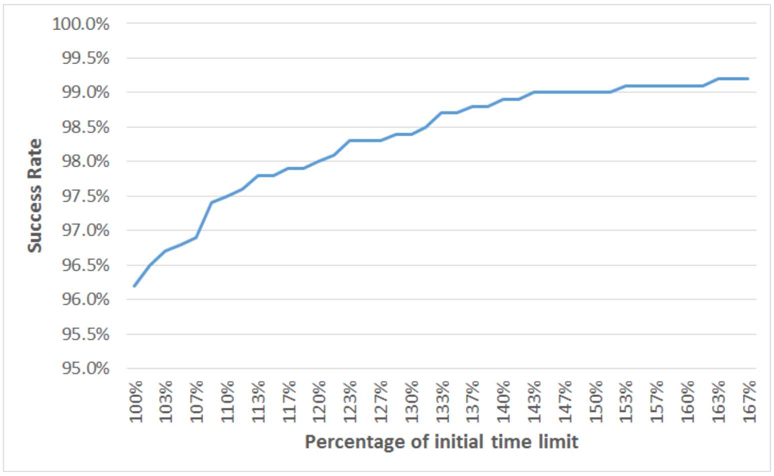


Figure 5.8: Results for “reach the flag” task with increasing time limits

Overall the results show good performance on 3 out of the 4 tasks. They demonstrate the effectiveness of active learning to significantly improve a weak policy with a limited number of samples. Even without active learning the agent can learn a robust policy for simple navigation tasks.

5.4 Conclusion and Future Directions

This chapter describes a novel method for learning autonomous policies for navigation tasks from demonstrations using deep active learning. A general learning process is employed to learn from raw visual data without integrating any knowledge of the task and active learning is employed to address the generalization issue. The experiments are conducted on a test bed that facilitates reproduction, comparison and extension of this work. The results show that CNNs can learn meaningful features from raw images of 3D environments and learn a policy from demonstrations. They also show that active learning can significantly improve a learned policy with a limited number of samples. However, the results also show that data aggregation does not help if the teacher is sub-optimal. Since the approach completely relies on learning from demonstrations even for generalization, it is expected that the performance of the agent depends on the quality of the demonstrations. To alleviate this limitation we propose to address the generalization challenge by allowing the agent to improve its policy using trial and error. This is explored in the following chapter by combining learning from demonstrations with deep reinforcement learning.

Chapter 6

Combining Learning from Demonstrations and Experience

This chapter investigates proposed methods for combining learning from demonstrations and experience by using reinforcement learning to refine imitation learning policies trained using the method described in chapter 4. We investigate using reinforcement learning to improve an imitation policy as well as using demonstrations to help a reinforcement learning policy. The proposed methods are compared to state of the art reinforcement learning methods as well as direct imitation and the active data aggregation method proposed in chapter 5. The work in this chapter is published in [12]

6.1 Introduction

Relying on demonstrations for learning autonomous polices can lead to generalization issues as the demonstrations only present a subset of the possible state and action spaces. Even if data aggregation based on the agent's performance is employed, this additional data is still completely dependent on the teacher's policy. If the teacher is suboptimal or does not follow the learning policy's assumptions such as the Markov property, then the learner's ability to generalize will suffer. This is highlighted in the analysis of the results in chapter 5. Moreover, in many applications the teacher is not able to provide corrective actions mid-trajectory based on the learner's performance. For example it is not feasible for a teacher to correct a tennis swing mid-trajectory, as this is an action that can only be performed in its entirety. Moreover, it is difficult for teachers to place themselves in the agent's situation to perform the optimal action. These limitation for teacher-based generalization motivates employing trial and error to refine and generalize the agent's policy. Learning from experience alleviates the restrictions surrounding teacher corrections. Deep reinforcement learning is rapidly gaining attention due to recent successes in a variety of problems [141, 46, 60, 59, 138, 147]. The combination of deep learning and reinforcement learning allows for a generic learning process that does not consider specific knowledge of the task and learns from raw data. Reinforcement learning (RL) is a popular choice for learning motor actions because most tasks can be modelled as a Markov decision process. Moreover, optimizing a reward function arguably provides a better description of a task than optimizing a policy [50]. However, learning through trial and error from scratch can be difficult when the rewards are sparse and only awarded by the environment after performing long sequences of actions. Moreover, learning through trial and error can lead the agent to perform the task in a way that is different from how a human would behave.

Utilizing both taught behaviour and experience in learning aims to mitigate the limitations of each approach. By allowing the agent to explore using trial and error, it is exposed to new scenarios and is able to generalize without requiring a teacher's involvement. While demonstrations can provide a starting point to learn more efficiently than learning from scratch using trial and error. The combined imitation and reinforcement learning methods are compared to the imitation learning methods presented in chapters 4-5 and two state of the art reinforcement learning methods.

6.2 Proposed Method

In this section we propose methods for combining learning from demonstrations and experience. The policy is learned using DQN [144] while using teacher demonstrations to expedite reinforcement learning. While a demonstrated instance is represented as a pair (x, y), in reinforcement learning additional attributes are added to represent an instance as a tuple (s, a, r, s'). s describes the current state of the agent in its environment and corresponds to x in demonstrations. a is the action taken by the agent and belongs to the same set of possible actions as y. r is a reward provided by the environment for performing action a in state s and s' is the resulting new state. Reinforcement learning assumes the task takes place in an environment $E$where the agent can interact using actions, observe states and receive rewards. To combine learning from demonstrations and experience, the agent is trained using deep reinforcement learning while demonstrations are used to facilitate the training process. The reinforcement learning algorithm follows [141] and uses a convolutional neural network to learn discounted rewards for performed actions. The network optimizes a Q-function $Q(s, a)$ that predicts an estimated reward for the input state-action pair. The Q-function is learned recursively using the Bellman equation.

$$Q(s,a) = \mathbb{E}_{s' \ E}[r + \gamma max_{a'} Q(s',a')|s,a] \tag{6.1}$$

Where γ is a discount parameter and $max_{a'}Q(s',a')$ is the largest estimated reward available to the agent at the next state s'. In the case where s is a terminal state which ends the task, $Q(s,a) = r$ as there is no future state. This ends the recursive learning of Q.

The learning method is model free and does not require a working model of the environment but rather just the experience tuples (s,a,r,s'). The method also learns off-policy; that is the learned policy is different from the performed policy. Therefore an optimal policy π^* which provides the optimal action choice $a^* = \pi^*(s)$ can be used to provide demonstrations through off-policy exploration to guide the agent to reward dense areas in the search space. We investigate two methods for utilizing demonstrations in deep reinforcement learning. The first is to simply initialize the Q-network with weights learned from supervised learning with a data set of demonstrations. Supervised learning is conducted as in chapter 4 on a network with the same architecture as the Q-network. The last layer uses a linear activation function instead of the softmax function used for classification in chapter 4 as the Q-network predicts continuous rewards for each available action. The agent uses random actions and its current policy to explore the environment, so initializing the network helps the agent explore behaviours similar to the teacher's. The second approach is to use demonstrations from the optimal policy π^* to guide the agent's exploration. This way the agent reaches reward dense parts of the state space earlier and more frequently compared to a typical exploration policy. This method ensures that the agent reaches the rewards as a policy initialized using supervised learning may still not perform as well as the teacher. The performance policy alternates between $a_t = \pi^*(s_t)$ and random actions, to encourage exploration beyond the teacher's demonstrations. Note that the choice between using demonstrations and random actions is performed once before each episode not before each action. It is easier in most applications for the teacher to provide demonstrations by performing the whole trajectory. This way the teacher is not required to produce an optimal action in the middle of the trajectory (such as in data aggregation techniques).The performance policy gradually shifts towards using the learned policy π where $a_t = max_a Q(s_t, a; \pi)$ i.e. choose the action with the greatest predicted reward according to the trained neural network. In this approach the information from demonstrations is independent of the agent's learning process, while in the first approach the initialized policy changes with training.

Algorithm 2 Learning from demonstration and experience

```
1:  Given: Teacher policy $\pi^*$
          Exploration factor $\alpha$
          Performance policy $\hat{\pi}$ alternates between $\pi^*$ and random choice according to $\alpha$
          Network $Q(s, a)$ with random weights
2:  for episodes do
3:      for timestep $t = 1 : T$ do
4:          $a_t^* = \hat{\pi}(s_t)$
5:          With probability $\epsilon$, $a_t = a_t^*$
6:          Otherwise $a_t = max_a Q(s_t, a; \pi)$
7:          Perform $a_t$ and get $r_t$,$s_{t+1}$
8:          Given the tuple $(s_t, a_t, r_t, s')$ train $Q(s, a)$:
9:          if $s_{i+1}$ is terminal:
10:             $y_i = r_i$
11:         else
12:             $y_i = r_i + \gamma max_{a'} Q'(s_{t+1}, a'; \theta) + F(s_i, a_i, s_{t+1})$
13:         Optimize $\pi$ using gradient descent for $loss = y_t - Q(s_t, a_t; \pi)$
```

Algorithm 2 summarizes learning from experience using guiding demonstrations. The demonstrations are provided as in traditional learning by demonstration problems, by simply performing the task in an optimal manner. The guided actions of the agent are recorded as an experience tuple (s, a, r, s') and used just like traditional unguided experiences. Unlike [46], no specially designed labelled dataset is needed to pre-train the value function $Q(s', a')$, which makes the training process more generic and streamlined. The task is assumed to be an MDP where the current state represents all past information (no extra context is needed to make a decision). Therefore, a single frame is used as the agent's observation and the resulting policy is stationary (i.e. the policy does not require information about the current position in the trajectory).

6.3 Experiments

The proposed methods are evaluated on the 4 navigation tasks introduced in chapter 5. The experiments compare the proposed methods against using direct imitation and reinforcement learning on their own. Firstly the two reinforcement learning algorithms used for comparison are validated on a simplified version of the navigation task.

6.3.1 Grid Navigation Task

This task is a simplified representation of navigation tasks which facilitates testing and analysis of learning algorithms in controlled manner. The environment is constructed of a grid where each cell is a state in the MDP and the agent is allowed to move between cells using 4 actions (Go Left, Go Right, Go forward, Go Back). Each state is represented by an 84×84 image of the number which reflect the number of this cell in the grid. These states are automatically generated given the dimension of the grid in terms of cells. The goal of the agent is to reach a target cell on the grid. Grids of dimensions 5×5, 15×15 and 30×30 are used in this experiment. This task is simple in that the environment is static i.e. performing the same trajectory results in the same outcome. Therefore, the task does not pose the challenges of generalization. Another simplified aspect is having finite well defined states. However, the task presents other features which are relevant to real navigation tasks. Namely that it requires learning from raw visual data and requires long trajectories of dependent actions to achieve the target. The environment offers no intermediate positive feedback while the agent is performing the task and only supplies a positive terminal reward when the target is reached. This is challenging as in a 30×30 grid, the shortest path to reach a reward consists of 57 steps. To give perspective, in a photo-realistic 3D environment which is used to train deep reinforcement learning agents [139], the shortest path to reach the reward is typically less than 20 actions. Figure 6.1 illustrates this task on a grid of size 5×5. The agent's starting position is shown by the blue marker while the target state is highlighted in green. This task is open source, and an implementation is available at https://github.com/ahmedsalaheldin/MashRL.git

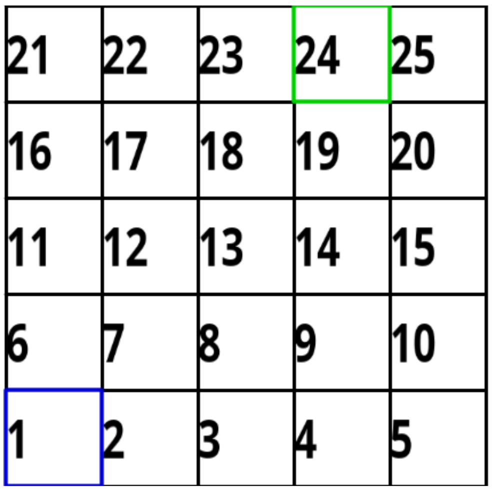


Figure 6.1: Illustration of the Grid Navigation Task

To evaluate the deep reinforcement learning algorithms DQN and A3C, the agent explores the environment using trial and error and receives a positive reward (+1) if it reaches the target and a negative reward (-1) if it selects an action that would take it out of the grid. In this case the agent's position is not changed. The algorithms are run for 1000 epochs, each epoch consisting of 2500 steps. A testing step is conducted after each epoch where the result is 1 if the agent reached the target within a step limit and 0 otherwise.

The proposed methods are then evaluated on navigation tasks in a 3D simulator [14]. For details on the tasks and simulated environments refer to chapter 5. Reinforcement learning algorithms are trained for 100 epochs of 250000 steps each. The same parameters are used for pure reinforcement learning and the proposed methods that utilize demonstrations. A3C utilizes 8 parallel processes. And frame skipping of 5. Frame skipping can greatly help reinforcement learning by shortening the trajectory and enhancing exploration through taking bigger steps. However, delicate navigation can limit the number of frames to skip. For instance, in the "Follow the line" task, navigating the narrow corners of the patterned corridor fails when using high frame skipping values even while following the optimal policy. For supervised learning, each task is trained on 20000 samples.

6.3.2 Inter-process Communication

For both Simulators, the agent is decoupled from the simulator and the learning algorithm. This allows for generic independent modules and facilitates interchanging tasks and learning algorithms. A TCP connection is used to communicate between the different components. The process for collecting demonstrations and supervised learning is shown in chapter 5. Similarly the reinforcement learning process uses a modular structure to facilitate using the same agent with different tasks and allows to change the learning method without affecting how the agent communicates with the simulator.

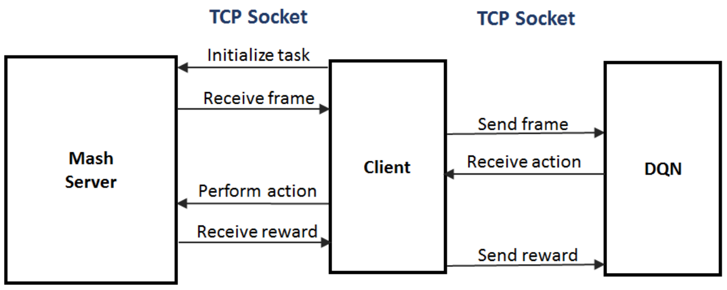


Figure 6.2: Reinforcement learning Flowchart

Figure 6.2 presents the process of learning from experience used in combining reinforcement learning and imitation. The agent communicates with the simulator to receive the state of the environment and the reward and sends them to the learning network. The learning network uses this information to decide the next action and update the policy. The prediction action is sent to the agent which in turn communicates it to the simulator.

6.3.3 Results

Firstly we present the results for learning from experience on the Grid navigation problem. Figure 6.3 shows results comparing DQN and A3C on the three grid sizes. Since success in this task is binary, the score counts how many epochs up to the current epoch have resulted in successful test sessions. This evaluation method produces a graph that shows the improvement and stability of the learned policy over training epochs.

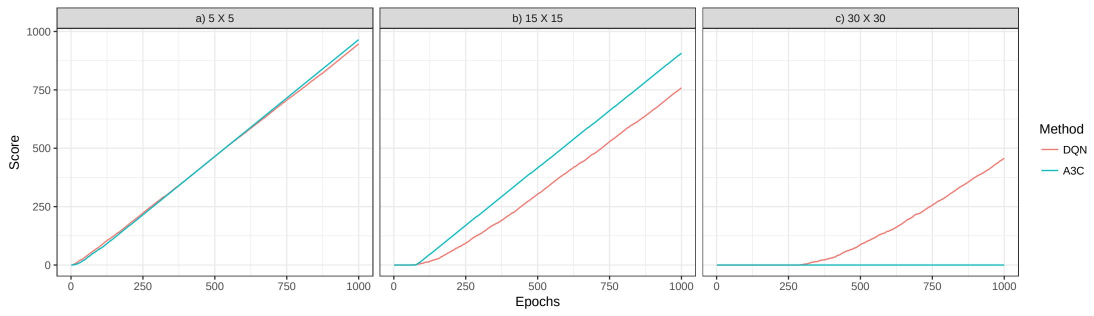


Figure 6.3: DQN and A3C results on the Grid Navigation Task

The results on the Grid tasks show that learning from experience becomes exponentially more difficult as the size of the grid increases. This is evident in the failure of A3C to learn on the 30 × 30 grid. This failure stems from the delayed rewards which makes obtaining feedback less frequent. The agent learns from the more readily available negative rewards to avoid the edges of the grid but is not able to reach the target.

Following, the results for experiments on the Mash simulator are presented. Table 6.1 shows a comparison between direct imitation as presented in chapter 4 and reinforcement learning approaches. The table shows the success rate out of 1000 rounds for each method on 3 navigation tasks in the mash simulator. The results for deep reinforcement learning methods are reported after 100 epochs of training. The table shows that both reinforcement methods are outperformed by direct imitation and ultimately fail to learn a robust policy to solve any of the 3 tasks. Qualitative analysis shows that all successful attempts during testing were achieved by chance without any clear pattern in the learned policy. Since ”Follow the line” requires a longer trajectory and is not as fault tolerant as the other tasks, it is less suitable for random exploration. Thus reaching the target by chance is more difficult and the success rate is 0%.

Table 6.1: Direct Imitation vs RL

Task	reach the flag	reach object	follow the line
Direct imitation	96.20 %	53.10%	40.70%
DQN	6.40 %	6.00%	0.00%
A3C	7.60 %	8.9%	0.00%

Table 6.2 shows results for ”Eat all disks”. For this task, the evaluation measure used is the number of disks eaten in 1000 rounds. The table compares the scores achieved by direct imitation, DQN, A3C and the optimal policy. The results show that direct imitation achieves 97.9% of the score achieved by the optimal policy while again learning from experience failed to produce an effective policy.

Table 6.2: “Eat all disks” results

Task	Direct Imitation	Optimal policy	DQN	A3C
score	1051	1073	51	45
error	1.70%	-	-	-

We further analyse the reinforcement learning results by observing the trend of success over the training period. Figures 6.4 and 6.5 show results for the 4 tasks in terms of rewards received for DQN and A3C respectively over 100 epochs. The test results are reported every 10 epochs and show rewards averaged over the test rounds. The graphs show no pattern of improving the performance with the increasing number of epochs.

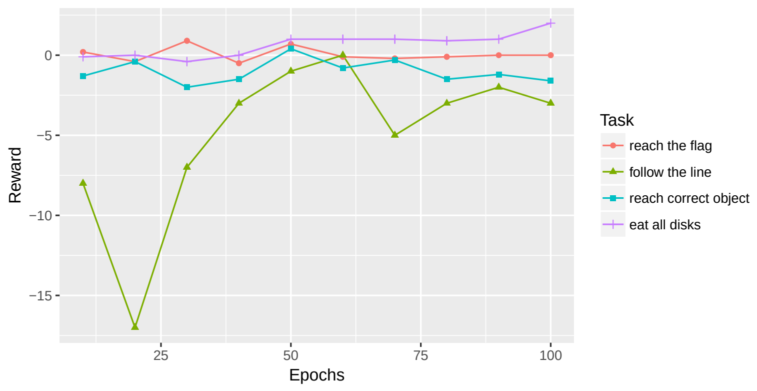


Figure 6.4: Results for DQN on Navigation tasks in MASH simulator

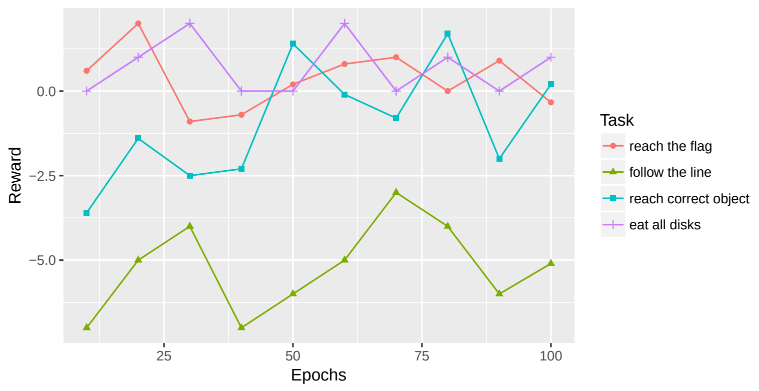


Figure 6.5: Results for A3C on Navigation tasks in MASH simulator

The two proposed methods for combining learning from experience and demonstrations are compared to traditional DQN on the “Reach the flag” task. ’Initialized DQN’ initializes the policy network of DQN with the parameters learned from supervised learning while ’DQN demonstrations’ refers to using demonstrations from the optimal policy to perform off-policy rollouts. Figure 6.6 shows the average rewards every 10 epochs for 100 epoch. The graph shows that utilizing demonstrations using the two proposed methods did not enhance the performance of DQN. The initial policy learned from demonstrations is quickly overwritten and thus provides no benefit to the learning policy or the rollout policy. This happens as there are no constraints to preserve the initial policy once DQN training starts. Guiding the agent by utilizing demonstrations in exploration also did not show any improvement. By looking at the probability distribution of the output layer of the network, we attribute this failure to the fact that the cost function used in DQN training does not consider output nodes other than the performed action. Therefore, when applying a rollout policy of optimal actions, the probabilities of non-used actions change arbitrarily. A cost function that includes all actions could be considered, but since DQN uses a periodically updated target network, the learned parameters for the performed actions will be overwritten with every update.

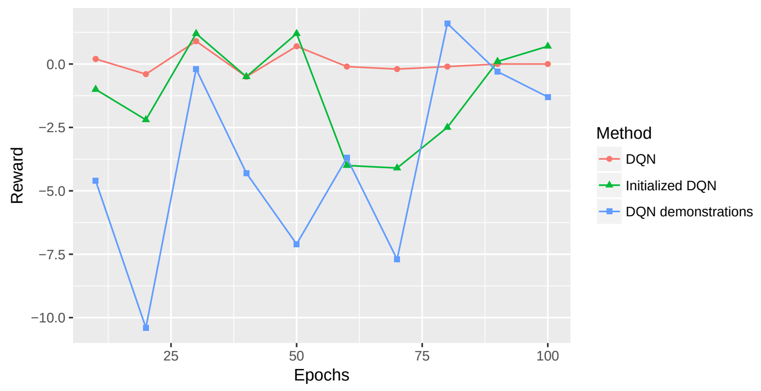


Figure 6.6: Results for combining learning from demonstrations and experience on "Reach the flag"

6.4 Conclusion and Future Work

This chapter investigates novel methods for combining learning from experience and demonstrations. Demonstrations are used to assist reinforcement learning by initializing the exploration policy and guiding the agent to reward dense sections of the state space. The results on 2D and 3D navigation tasks confirm the assertion that learning from trial and error becomes increasingly difficult as the length of the trajectory to the target increases. The results also show that the proposed utilization of demonstration failed to enhance reinforcement learning on the evaluated tasks. Analysis shows that the initialized policy is forgotten quickly when acquiring new experiences and that guiding the exploration policy causes an unbalanced reward estimation function. Therefore, the demonstrations need to be incorporated in a more stable way. Which motivates the use of reward shaping in the next chapter. Using the demonstrations to learn a reward function alleviate these issues as the reward signal is persistent and doesn't affect experience sampling.

Combining learning from demonstrations and experience is a very promising approach and warrants further exploration in the following future steps: Firstly, utilizing demonstrations with policy-based deep reinforcement learning algorithms such as DDPG and A3C to avoid the restrictions of value based estimations. Moreover, we aim to use supervised learning to calculate TD error as well as supervised cost following the approach in [56]. Furthermore adapting the online learning methods in [148] can speed up retraining while overcoming the catastrophic forgetting phenomenon. This can also potentially allow one network to learn multiple tasks.

Chapter 7

Reward Shaping from Demonstrations

This chapter proposes a method for reward shaping from demonstrations to expedite and improve learning through deep reinforcement learning, especially with sparse rewards. Demonstrations from a teacher are used to shape a potential reward function by training a deep supervised convolutional neural network. The shaped function is added to the reward function used in deep-Q-learning (DQN) to perform off-policy training through exploration. The proposed method is demonstrated on navigation tasks that are learned from raw pixels without utilizing any knowledge of the problem. The results show that using the proposed shaped rewards signicantly improves the performance of the agent over standard DQN. The approach proposed in this chapter is published in [13].

7.1 Introduction

Learning from experience can produce robust policies that generalize to dynamic scenarios by balancing exploration and exploitation of rewards. However, finding a solution through trial and error may take too long. Especially in problems that require performing long trajectories of actions with delayed rewards. In such cases it may be extremely difficult to stumble upon rewards by chance and the time to learn a policy to maximize the rewards exponentially increases. This fact is highlighted in the results in chapter 7 where reinforcement learning methods failed to learn a policy in navigation tasks where a large number of steps is required to reach the target. Another drawback is that learning through trial and error may result in a policy that solves the problem differently to how a human would. Performing a task in a manner that is intuitive to a human observer may be crucial in applications where humans and intelligent agents interact together in an environment [5]. Nass et al [6] suggest that humans view computers interacting with them as social agents and that humans interact with them in a manner derived from their experiences interacting with other humans. Therefore, even with the conscious knowledge that an agent is not a human, interaction is improved when the agent behaves in a way that is familiar to its human counterpart.

We propose a reward shaping method for integrating learning from demonstrations with deep reinforcement learning to alleviate the limitations of each technique. Unlike most reward shaping methods, the reward is shaped directly from demonstrations and thus does not need measures that are tailored specifically for a certain task. Moreover, deep learning is used to learn a mapping between raw observations and rewards from the demonstrations. The proposed method uses a deep convolutional neural network to learn a reward shaping function from demonstrations performed by a teacher. This function provides additional rewards based on the teacher's behaviour that are added to the rewards from the environment. The augmented reward function is used to train an agent through Deep-Q-Networks (DQN) [141], a variation of Q-learning that employs deep learning. Both the supervised reward shaping network and the reinforcement learning network utilize stacked convolutional layers to learn reward estimates directly from raw pixels. This approach takes advantage of the extra information provided by demonstrations to expedite and improve reinforcement learning, while being able to generalize by learning through exploration and trial and error. Moreover, an adaptive updating method for the Q-network in DQN is proposed to improve the efficiency and robustness of the learning process.

7.2 Method

This section presents the proposed method for deep reward shaping from demonstrations. The method uses a deep supervised network to learn a shaping function from demonstration. The shaped reward is added to the environment reward used by DQN [144] to speed up and improve policy learning through reinforcement learning. First we formalize the reinforcement learning and learning from demonstrations approaches.

Reinforcement learning assumes the task takes place in an environment E and is formulated as a Markov Decision Process (MDP). An experience is represented as a tuple (s, a, r, s') where s represents the state as observed by the agent, a is the action taken by the agent at state s, r is the reward received for performing action a and s' is the new state resulting from that action. While demonstrations are presented as pairs of input and output (x, y). Where x is a vector of features describing the state at that instant and y is the action performed by the demonstrator. The pair of observation and action (x, y) in demonstrations corresponds to (s, a) in the Markov Decision Process. So the demonstrator can be considered as an optimal policy π^* which provides the optimal action choice $a^* = \pi^*(s)$

The reinforcement learning algorithm works by training a deep convolutional neural network to predict the discounted reward of performing an action. Figure 4.4 illustrates the architecture of the network. More formally, the agent learns by optimizing $Q(s, a)$ where Q is an estimation of the return of performing a at state s which uses the recursive Bellman equation.

$$Q(s, a) = \mathbb{E}_{s'\ E}[r + \gamma max_{a'} Q(s', a')|s, a] \tag{7.1}$$

Where r is the actual reward for performing a at state s, γ is a discount factor for potential future rewards and $max_{a'} Q(s', a')$ is the maximum estimated reward possible at the next state s'. If s is a terminal state (one which ends the task, regardless of result), then $Q(s, a) = r$. The function $Q(s, a)$ is learned via a deep convolutional neural network and is used to provide the agent with actions when presented with a new state. In practice a second network is used to predict the target rewards $Q'(s', a')$ used in training $Q(s, a)$. The reason for that is to provide a constant target for training while updating $Q(s, a)$ to stabilize learning. The target network is updated periodically to be equivalent to $Q(s, a)$. This raises the issue of how long to freeze the target network for before updating it. A freezing period that is too short will not allow $Q(s, a)$ to converge to the target rewards and results in unstable learning. A freezing period that is too long is inefficient since $Q(s, a)$ continues to learn outdated targets. To improve the learning efficiency we propose an adaptive method to update the target network. Convergence will occur at different rates for different tasks or even for different batches within the same task. Therefore rather than a constant freezing period, we set a condition -based on training loss- for updating the target network. Equation 7.2 shows the updating condition.

$$Loss = \frac{(Q(s, a) - [r + \gamma max_{a'} Q'(s', a')|s, a])^2}{2} \leq \varepsilon \tag{7.2}$$

Where ε is a constant indicating how small the loss needs to be before updating the targets.

The main contribution of this approach is to incorporate reward shaping from demonstrations with reinforcement learning in a deep learning context. A shaped reward is an extra reward that is derived from extra information and is added to the reward from the environment. A shaping function $F(s, a, s')$ is used to generate the shaped reward. The shaping function is added to the target reward in equation 7.1 yielding:

$$Q(s,a) = \mathbb{E}_{s'\ E}[r + \gamma max_{a'} Q'(s', a')|s, a + F(s, a, s')] \tag{7.3}$$

Ng et al [49] proved that forming F as function of the transition between states (i.e. the difference in potential between the states) rather than a function of the current state-action pair (s, a) maintains the convergence guarantees of reinforcement learning and preserves the optimal policy. Therefore we express F as the difference between potential functions for states s and s'.

$$F(s, a, s') = \gamma max_{a'} P(s', a') - max_a P(s, a) \tag{7.4}$$

Where $P(s, a)$ is a function estimating the potential of the pair (s, a). We use as the potential function a convolutional neural network trained in a supervised manner on a set of collected demonstrations $D = (x, y)$. The network has the same architecture as the network used to learn $Q(s, a)$ and therefore produces an estimated potential for each action given s. The target outputs y are encoded as one-hot labels, i.e the output is a vector of possible actions with value one for the performed action and zero otherwise. The output layer of the network uses a linear activation function instead of the softmax activation function commonly used in supervised classification problems to avoid sharp potential estimates for unseen states. So P is used as a multivariate regression network rather than a classification network and the predicted potential for each action is a real number. Using a deep network for the potential function has the advantage of being able to learn from raw data and does not require designing representations of the demonstrations. Moreover, unlike [48] the demonstrations do not need to be stored or traversed to calculate the potential for a new state-action pair.

Utilizing this potential based function in equation 7.3 provides extra information from demonstrations to the reinforcement learning algorithm. This alleviates the challenges of sparse environment rewards and allows the agent to get more frequent feedback. Without this extra knowledge, the only policy available for the agent is to explore randomly until it has sampled enough experiences, which is not efficient when the rewards are sparse. Reward shaping speeds up reinforcement learning by limiting the need for extensive random exploration.

Algorithm 3 presents the pseudo code for reinforcement learning with deep reward shaping from demonstrations.

Algorithm 3 DQN with Deep Reward Shaping from Demonstrations

```
1:  given: Teacher demonstrations $D = (x, y)$
        Network $Q(s, a)$ with random weights
        Network $Q'(s', a')$ with random weights
        Network $P(s, a)$ with random weights
        Empty replay buffer $B$
        Loss threshold $\varepsilon$ for adaptive updates
2:  Train $P(s, a)$ on $D$
3:  for episodes do
4:      for timestep $t = 1 : T$ do
5:          With probability $\epsilon$, $a_t$ = random action
6:          Otherwise $a_t = max_a Q(s_t, a; \theta)$
7:          Perform $a_t$ and get $r_t$,$s_{t+1}$
8:          Store the tuple $(s_t, a_t, r_t, s_{t+1})$ in $B$
9:          Randomly select minibatch of
            experiences $(s_i, a_i, r_i, s_{i+1})$ from $B$:
10:         $F(s_i, a_i, s_{i+1}) =$
            $\gamma max_{a'} P(s_{i+1}, a') - max_{a_i} P(s_i, a_i)$
11:         if $s_{i+1}$ is terminal:
12:             $y_i = r_i$
13:         else
14:             $y_i = r_i + \gamma max_{a'} Q'(s_{t+1}, a'; \theta) + F(s_i, a_i, s_{t+1})$
15:         Optimize $\theta$ using gradient descent for:
            $loss = \frac{(y_t - Q(s_i, a_i; \pi))^2}{2}$
16:         if $loss \leq \varepsilon$ :
17:             $Q'(s', a') \leftarrow Q(s, a)$
```

The teacher provides demonstration as in traditional learning by demonstration problems. Unlike [46], no specially designed labeled dataset (that includes extra information other than state and action, such as evaluation measures of the performance) is needed to pre-train $Q(s', a')$ or $F(s, a, s')$, which makes the training process more generic and streamlined. The task is assumed to be an MDP where the current state represents all past information (no extra context is needed to make a decision). Therefore a single image frame is used as the agent's observation and the resulting policy is stationary (i.e does not require information about the current position in the trajectory).

The neural network architecture used to optimize Q and P is a deep architecture with three convolutional layers that follows the network architecture in [141] with the exception of using a single frame as input. The convolutional layers are followed by a fully connected (FC) hidden layer and finally an output layer. A rectified linear activation function (ReLU) is used for all layers apart from the output layer in which a linear activation function is used. Table 7.1 summarizes the network architecture.

Table 7.1: Neural network architecture

Layer	Size of activation volume
Input	84×84
Conv1	$8 \times 8 \times 32$
Conv2	$4 \times 4 \times 64$
Conv3	$3 \times 3 \times 64$
FC	512
Output(FC)	4

7.3 Experiments

In this section we describe the experiments conducted to evaluate the proposed approach and present the results. Implementation of the proposed method including the task used for evaluation is available at `https://github.com/ahmedsalaheldin/MashRL.git`

7.3.1 Experimental Setup

The proposed method is evaluated on the 2D grid navigation task introduced in chapter 6. The experiments are conducted on grids of size 5×5, 15×15 and 30×30. Figure 7.1 illustrates the navigation task on a 5×5 grid. The task involves the agent navigating from the starting state (highlighted in blue) to the target state (highlighted in green) by moving in one of the four cardinal directions. Each state is represented by an image showing the number of the state as visualized in the figure.

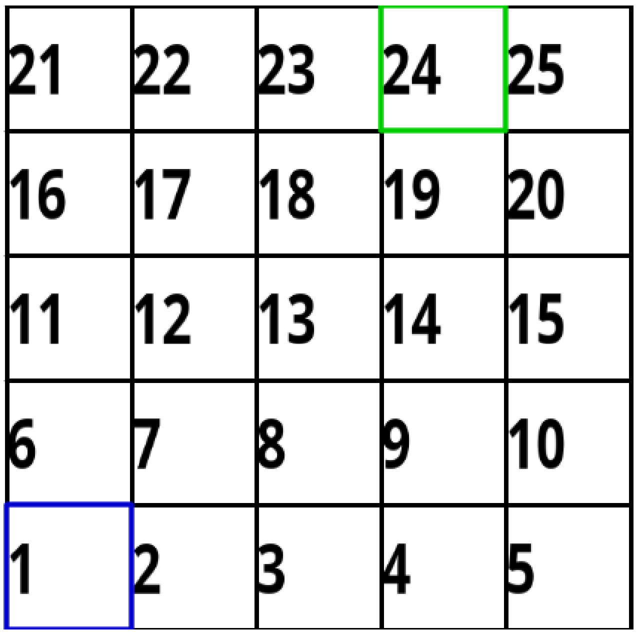


Figure 7.1: Illustration of the Grid Navigation Taks

To evaluate the proposed method we conduct several experiments for each grid size. Firstly, the adaptive method for updating the target network is evaluated against static freeze parameters. Typically a large freeze parameter (10000) is used to ensure convergence but smaller values may result in faster training. Adaptive updating is compared against freezing the target network for 10000, 2500, 500 and 100 steps. The loss threshold ε is set to 0.02. This comparison is done using the DQN algorithm without reward shaping. The second experiment compares the proposed reward shaping approach with DQN using adaptive updating for both approaches. The second experiment is repeated using the best performing static freeze parameter (500) to show that RL can benefit from demonstrations regardless of the updating method. Finally we compare the proposed approach with DQN while using limited exploration. As mentioned in Section 3, the prior knowledge incorporated through reward shaping from demonstrations provide a base for the policy to start learning. As such, the need for extensive random exploration is limited. Exploration is controlled by the parameter ϵ which decides if the agent gathers samples randomly or according to the current learned policy. Following [141], the learning rate used is 0.00025 and ϵ decays to 0.1 over training time. The supervised training of the potential function $P(s, a)$ is executed on demonstrations performed by a deterministic optimal policy. The policy performs 5 complete trajectories of the task to gather enough samples for supervised training. For all experiments, the agent is allowed to train for 1000 epochs. After each epoch, the agent performs the current policy in a test session and the score is reported. Since success in this task is binary, the score is defined as the number of successful test sessions up to the current epoch. Using such an expanding window produces a monotonically increasing graph that reflects the rate of learning and the stability of the learned policy.

7.3.2 Results

This section presents the results of the proposed method. Figures 7.2, 7.3, 7.4 and 7.5 show the results of the 4 experiments conducted on grids of sizes 5×5, 15×15 and 30×30. The X axis represents the number of epochs used for training. The Y axis represents the score achieved by the agent at that test session. The score shows how many training epochs resulted in a policy that successfully solves the problem up to the current epoch. Figure 7.2 shows a comparison of different static freezing parameters against the proposed adaptive measure for updating the target network. The graphs show that adaptive updating achieves better results than all static parameters. For grid sizes 5×5 and 15×15, setting the freezing parameter to 100 produced the second best results with a very slightly lower curve than adaptive updating. Freezing parameter 500 closely follows, and increasing the freezing parameter results in slower learning. Similar results are shown on grid size 30×30, however, the best performing static parameter is 500 as using 100 fails to learn completely. This is due to requiring more time to converge on more complex tasks and highlights the difficulty of choosing static freezing parameters for different tasks and the advantage of using adaptive updating.

Figure 7.3 evaluates the proposed reward shaping method against DQN. Both approaches use adaptive updates for this comparison. The graphs show that using reward shaping results in a more stable policy faster than traditional DQN. The same observations are made in figure 7.4 which uses a static freezing parameter to show that the reward shaping approach does not depend adaptive updates. These results demonstrate the benefits of the proposed reward shaping approach over standard deep reinforcement learning.

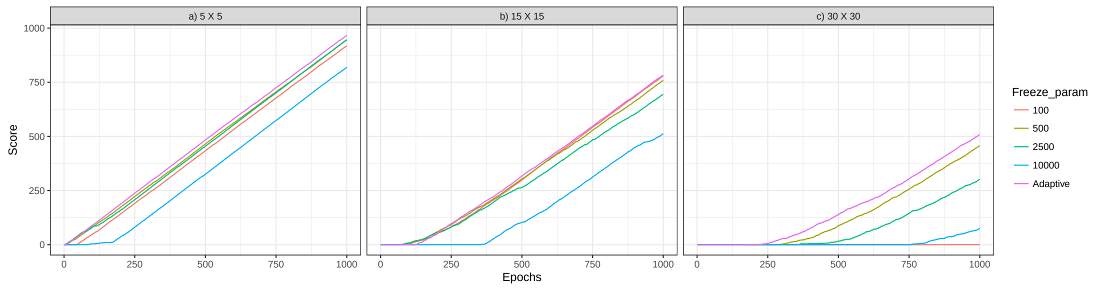


Figure 7.2: Adaptive vs static network updating

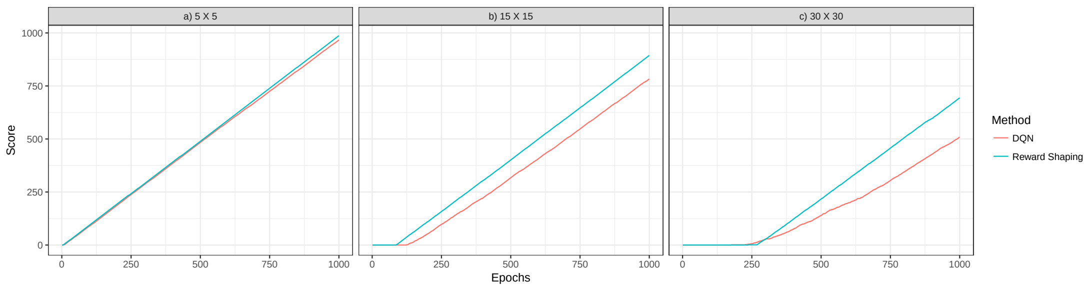


Figure 7.3: Reward shaping vs DQN, adaptive updating

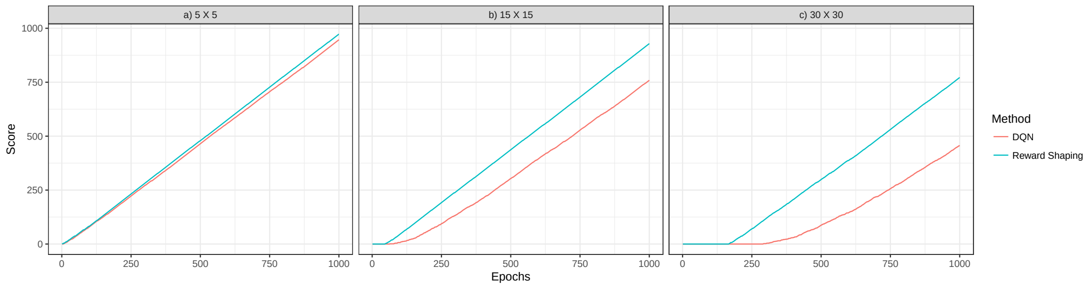


Figure 7.4: Reward shaping vs DQN, freeze parameter = 500

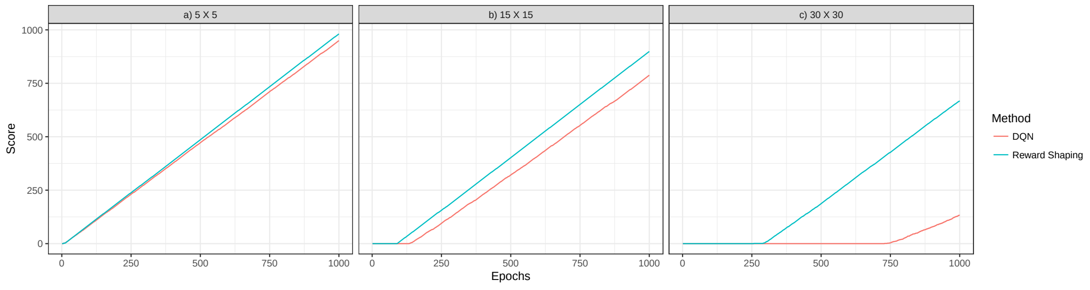


Figure 7.5: Reward shaping vs DQN, $\epsilon = 0.4$

Figure 7.5 compares the reward shaping approach to DQN using an initial ϵ of 0.4 instead of 1. This decreases the initial exploration performed by the agent to collect samples and their corresponding rewards. The graphs show that reward shaping continues to outperform DQN. On the 30×30 grid, the performance of DQN drops significantly due to the increased search space, while reward shaping results in faster and more stable learning. This indicates that the prior knowledge extracted from demonstrations provides guidance to the sampling policy and results in a learning process that is more robust to changes in exploration parameters. A noteworthy observation is that all experiments show that the benefits of the proposed method are more pronounced the larger the grid size. This is because the challenges addressed in this method increase as the grid becomes bigger. The number of states increases and the number of steps needed to complete the task also increase which makes the rewards available to the agent even sparser.

7.4 Conclusion and Future Work

This chapter proposes a novel method for deep reward shaping from demonstrations to improve deep reinforcement learning. Learning from demonstrations allows a generic approach to reward shaping and learns from raw visual data without requiring specific information about the task. Moreover, an adaptive approach to updating the target network is proposed that is shown to benefit deep reinforcement learning whether with or without the use of reward shaping and alleviates the need to manually select parameters suitable for the task. The results are conducted on a 2D navigation task and show that the proposed reward shaping approach speeds up and improves deep reinforcement learning and provides increased stability against exploration policies. While the task is simple, it poses realistic challenges such as sparse rewards and raw high dimensional input. However, in more realistic applications the environment would be more dynamic. Therefore, in the next chapter we address the generalization problem in a multi-dimensional dynamic environment by utilizing deep recurrent neural networks to keep a memory of performed trajectories. Other future steps include testing the proposed approach on learning various navigation tasks in a more realistic simulator [14]. We also aim to incorporate reward shaping from demonstration with A3C [140] which is considered the current state of the art in deep reinforcement learning.

Chapter 8

Deep Imitation Learning from Sequences

This chapter presents the proposed method to learn sequences of actions by utilizing memory in deep neural networks. Long short-term memory networks are utilized to capture the temporal dependencies in a teacher's demonstrations. This way past states and actions provide context for performing following actions. The proposed methods are evaluated on a benchmark soccer simulator and compared to supervised learning and data-aggregation approaches. The results show that utilizing memory while learning significantly improves the performance and generalization of the agent and can provide a stationary policy. The methods and experiments presented in this chapter are submitted for review to the international joint conference on neural networks (IJCNN 2018).

8.1 Introduction

The methods described in this thesis so far deal with states and actions as discrete instances. Although most autonomous applications involve performing sequences of actions to achieve a goal, most learning methods process instances separately as independent and identically distributed (i.i.d.) samples. These methods rely on the hypothesis that the observed state contains enough information to make an accurate decision; and that performing a series of accurate independent decisions will accumulate to effective behaviour. This hypothesis overlooks the dependencies between actions which can be key in planning long trajectories of actions. This is especially sensitive in imitation learning the teacher might inherently be relying on memory, without this information being presented to the learning agent. Even if an accurate decision can be made from the current state alone, the teacher might choose a different course of action based on previous experience. If this additional information is not presented to the agent, it will not be able to learn from the demonstrated behaviour as highlighted in Chapter 5. Moreover, even if sampling training data is dependent on previous actions such as data aggregation methods, the learning algorithm does not take temporal relationships between these observations into account. Using memory of past events as context, allows the policy to learn different reactions to similar observations in different point along the trajectory [86]. It is therefore necessary to represent training demonstrations as sequences and learn to reproduce dependant action trajectories.

Recurrent neural networks have shown great success in learning from sequences [149, 150]. They capture temporal dependencies by having looping connections so the nodes consider previously processed samples along with new input to produce a decision. However, most RNN applications involve processing the sequence in its entirety before producing a decision or generating an output sequence [151]; which is not suitable for real time autonomous agents. Some applications such as handwritten text generation utilize RNNs to generate a sequence one step at a time [152]. However, these sequences are generated in isolation from other factors while autonomous agents are required to react mid trajectory to dynamic environments. For that, imitation learning requires new RNN based methods that can learn from long sequences of dependent actions and react based on real time observations of the environment.

This chapter proposes representing demonstrations as sequences of dependent state-action pairs and using a long-short-term-memory network (LSTM) to learn a policy. The LSTM network learns a mapping between states and actions while taking into consideration memory of previous events and actions and the temporal dependencies between these instances. This approach is demonstrated on the "robocup soccer simulator" [15] ; a multi-agent soccer simulator. The multi-agent setting provides a dynamic environment for which generating static sequences is not suitable as the policy is required to react to the other agents' actions. This makes the simulator a popular benchmark for intelligent agents. Unlike most machine learning methods, the proposed LSTM network learns from raw low level sensory data, without the need for engineered feature extraction. Similarly, the policy performs low-level parametrized actions that making a decision as well as predicting continuous values for the actuators simultaneously. Performing sequences of these low level actions makes up the desired behaviour without manually engineering high level strategies. To evaluate the proposed LSTM approach its performance is compared to the hand-crafted teacher policy, and policies learned via neural networks without memory (MLP). To evaluate the generalization ability of RNNs in imitation learning, the proposed approach is further compared to a data aggregation method [19] conducted on the MLP agents.

8.2 Related Work

Recurrent neural networks can be used to generate sequences by considering the past generated samples. Such an approach is used in [153] to generate continuous handwriting. An extension to this approach is also proposed in [153] that allows the generated sequence to be conditioned on a sequence of input text characters. Clearly, such approaches can be very relevant to imitation learning if the actions can be formulated as a generated sequence conditioned on a sequence of observed states. Similarly sequence to sequence learning [151] has been gaining a lot of attention recently. However, for most applications, the entire input sequence is analysed before generating the output sequence, while autonomous agents are required to act in real-time to every sensory input.

An LSTM based system is proposed in [86] to learn how to perform surgical procedures by controlling a robotic arm. The network is trained on demonstrations by a human expert. Although the static setting of the surgery allows for policies that replicate manually designed trajectories, this supervised learning approach provides better generalization.

The robocup simulator is a popular benchmark for intelligent learning methods as it shares many characteristics with real world applications. A cooperative defensive task is learned in [26] using demonstrations provided by two human players simultaneously. Several classifiers are used to learn from the demonstrations and the results are favourably compared to human performance and simple hand-coded agents. However, this approach employs high level strategies as the decision to be learned by the agent, such as “approach the ball” or “block attacker’s path” which in turn need to be translated into low level actions through manual programming. High level actions also enable learning the task through evolutionary algorithms [154] as the solution space becomes smaller. Similarly, in [155, 120, 156] reinforcement learning is used to learn high level actions. It is noteworthy that each paper employs a different set of macro-actions; so each new macro-action has to be manually designed. Deep reinforcement learning is used in [157] to learn an offensive task from raw sensory data. The reinforcement learning policy is used to predict low-level parametrized actions and thus does not require manual policy design. However, the organic reward in this task (scoring a goal) is very sparse and requires performing long trajectories of low-level actions to reach this state. As reinforcement learning exploration fails to reach the environment’s reward, this approach employs a manually engineered reward function that guides the agent to perform desired behaviours. This engineering requires substantial task knowledge and limits the general application of this approach.

8.3 Method

The method proposed in this chapter minimizes the need for expert knowledge by utilizing the low level sensory features and learning a mapping to atomic parametrized actions. The method learns solely from demonstrations and does not require any explicit tailored engineering. We start by describing the process of data collection and representation. Demonstrations are provided by a teacher that performs the task for a number of rounds. For each round the teacher attempts to score a goal; the round ends with successful or unsuccessful attempt. A plethora of hand-crafted agents exist for robocup soccer simulator and can serve as the teacher to provide examples of effective behaviour. Existing agents are also available to control the opponent to provide a realistic setting for the demonstrations.

Each round is represented as a sequence of state-action pairs. For each frame t the state of the environment x_t is captured along with the action taken by the teacher y_t and are added to the sequence $S_i = x, y$. The state x_t represents low level information about the agent's surroundings and is captured from the agent's point of view using it's simulated sensors. So all the information about the field and the objects and players in it are captured relative to the agent's position and status. The action y_t is chosen from a set of the low level parametric actions available to the agent. That is, the agent decides what move to perform from its list of actuators as well as one or more continuous values that serve as parameters for the selected actuator. Such atomic actions performed in a sequence construct a higher level behaviour that is usually identified and modelled manually in other studies. The captured sequences are used to construct the training dataset $D = S_1, S_2..S_n$ is used to train recurrent neural network. learn from raw sensory data.

The training set is used to train a deep recurrent neural network. The network consists of 3 stacked LSTM layers containing 100, 50 and 6 nodes respectively, followed by a reshaping layer to present the 6 output values for all samples in the batch to the loss function. The loss function calculates the error for the predictions of the entire batch rather than the final prediction only. This is because unlike most RNN applications we are interested in producing accurate predictions at each frame rather than optimizing one prediction after reading the entire sequence. The LSTM layers are used to extract high level temporal features from the raw input and the context provided by the networks memory. The LSTM layers utilize hyperbolic tangent(tanh) activation functions. The output nodes in the final layer correspond to parametrized actions and are used to predict continuous values for the 6 possible parameters for the agent's actuators. The output layer utilizes linear activations and a mean square error loss function is used, therefore the network behaves as a multivariate regressor. The actuator with the highest predicted parameter value is selected for execution by the agent. This method allows for prediction values for multiple parameters simultaneously. Figure 8.1 shows a visualization of LSTM units and illustrates how the proposed network learns from sequences. At each time step, the node receives the new input x_t and the output of the previous step h_{t-1} along with its internal state. The node updates its state and produces an output that is fed back in the next time step.

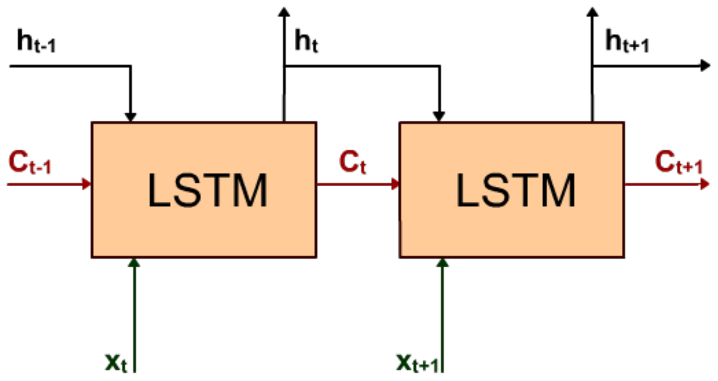


Figure 8.1: Illustration of LSTM units

In many applications the output of the nodes in the final layer is not produced until the end of the sequence and is only fed into the next time step without being output as the network's prediction. In contrast, the proposed network does not read the entire sequence before producing a decision or generating an output sequence. Instead, at each instance of the input sequence the network predicts an output. Thus generating the output sequence step by step with the input, at each step utilizing all the information available up to this instance. By representing the demonstrations as sequences, this approach provides context for most of the samples facing the agent.

However, this makes the prediction dependent on the position of the sample in the sequence. For example if the agent starts performing the trained policy mid episode, the current frame will be treated as if it is at the beginning of the sequence even though it is not. To ensure the stationarity of the agent's policy, we train another network on a modified version of the training set D in which all the sequences $S_1, S_2..S_n$ are augmented into one list of samples. This list is subsequently segmented into segments of uniform length that serve as the new training sequences to be fed into the LSTM network. Figure 8.2 illustrates the segmentation of the artificial sequences. This arbitrary creation of sequences presents different states in different parts of the training sequences while maintaining a temporal dependency between the consecutive instances in a sequence. This approach is not expected to outperform training on fully structured sequences given that complete sequences are always presented to the agent during testing. However, it demonstrates that the proposed approach does not depend on reproducing entire training sequences and that utilizing memory in imitation learning is beneficial even if the beginning and end of the sequence are unknown.

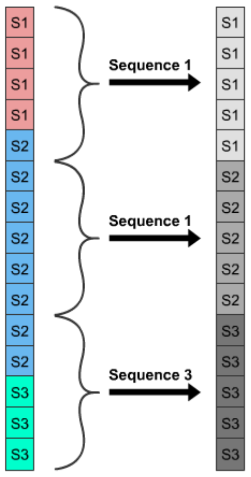


Figure 8.2: Re-segmenting the demonstrated sequences into arbitrary sequences

Table 8.1 highlights the differences between the proposed method and other intelligent methods used for "Robocup". Most methods rely on manually engineered features and high-level actions which require significant task specific knowledge and engineering which does not allow for a general learning process. [157] uses deep learning to alleviate the need for engineering features and directly map raw features to low-level actions. However, designing dense reward functions to guide the agent require similar effort and produce similar results to manually engineering the policy. A change in the setting such as the number of players on the field requires designing new reward functions. This is in contrast with organic reward fucntions which are directly provided by the rules of the game. The approach proposed in this chapter is general and only receives knowledge about the task from the demonstrations. Providing new demonstrations for changes in a task is considerably easier than designing reward functions or low-level policies to execute the high-level decisions made by the policy.

Table 8.1: A comparison of machine learing approaches for rocbocup

Method	Learning	Features	Actions	Rewards
Jain et al.[155]	Reinforcement learning	Selected	High-level	Engineered
Raza et al.[26]	Supervised (various)	Engineered	High-level	N/A
Stone et al.[120]	Reinforcement learning	Engineered	High-level	Organic
Masson et al.[156]	Reinforcement learning	Selected	High-level parametrized	Engineered
Hausknecht et al.[157]	Reinforcement learning	Raw	Low-level parametrized	Heavily-engineered
Ours	Supervised (LSTM)	Raw	Low-level parametrized	N/A

The proposed LSTM network is compared to a multi-layer perceptron that does not have a memory and treats all frames as independent and identically distributed samples. In this case the sequences are augmented to create one training set, from which batches of samples are drawn. Keeping the sequence of samples without utilizing memory can be detrimental to training as the samples in training batches will be too similar and lack diversity. Therefore, when training the MLP, the entire dataset is shuffled before sampling the training batches to ensure that they contain diverse samples from a variety of situations. This is similar to the replay buffer approach used in [144] which is a key factor in the success of deep reinforcement learning. The architecture of the MLP consists of 3 fully connected layers containing 100,50 and 6 nodes respectively. The first 2 layers utilize rectifier activation functions (ReLU) and the output layer uses a linear activation function.

Moreover, to evaluate the generalization of the proposed LSTM approach, it is compared to a data aggregation approach that is applied during MLP training. DAGGER [19] is a seminal data aggregation method that aims to enhance generalization in imitation learning by providing additional training samples based on the agent's initially trained policy. This approach is similar to the method proposed in Chapter 4 but does not utilize active learning as the problem is not as computationally intensive as 3D navigation from pixels. The agent is allowed to perform the task using the policy trained using the MLP. For each frame the teacher provides the optimal action for the state observed by the agent and a new training dataset is collected. The agent stochastically chooses to perform the teacher's instruction or the action predicted based on its current policy. The new samples are added to the original training set and used to train a new agent. The new samples show states that are likely to be visited by the agent according to its trained policy and thus improving generalization. This process is repeated iteratively and the new set of training instances are aggregated into the final training set.

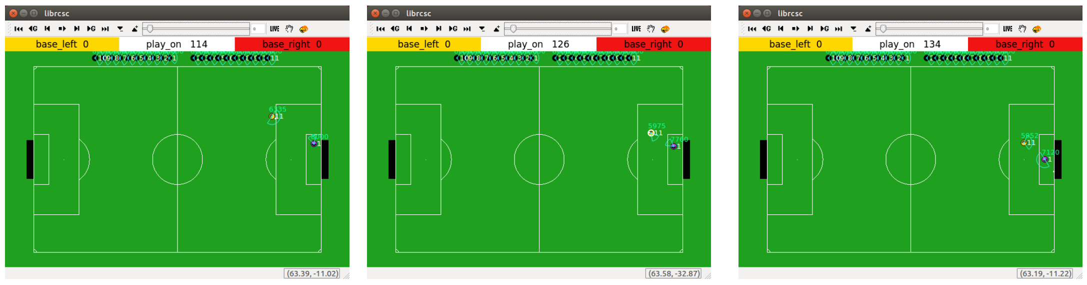
8.4 Experiments

8.4.1 robocup

The robocup soccer simulator [15] is a 2D simulator that allows for full soccer matches between 11 player teams. The simulator is used for a worldwide competition since 1997 to create artificial intelligence for soccer agents. This competition is a popular benchmark for artificial intelligence as it contains a number of real characteristics and challenges found in real applications. This task provides a dynamic environment where multiple agents act simultaneously and in real time to accomplish given targets. The team based setting allows for cooperative and competitive strategies between multiple agents. Moreover, the decision making is decentralized, meaning that each agent makes it's own decisions in order to reach the team's shared goal. This provides an extra challenge especially as the environment is sensed from each agent's perspective and not relative to the world; as well as the availability of sensor and communication error. More closely related to this study is the fact that playing soccer requires performing long sequences of actions that depend on previous actions as well as actions from other agents. Because soccer is a familiar activity, this application provides extensive evaluation of the agents' performance; not only according to the well established rules of soccer but also qualitatively analysing the agents' behaviour through 2D visualization. Implementation of the proposed methods in this chapter is available at `https://github.com/ahmedsalaheldin/RoboCupLSTM`

8.4.2 Half-Field-Offence

This study is conducted on a simplified sub problem of soccer simulator called Half-field-offence (HFO) [158]. Over the years, researchers have used simplified versions of the game of soccer to create intelligent autonomous agents in a more controlled setting [159]. As the name suggests HFO takes place in half the soccer field and is only



(a) (b) (c)

Figure 8.3: Illustration of the Half-field-offence environment

concerned with the task of offence. The round is initiated with the offensive player and the ball randomly placed in the half-field. The objective of the offence is to score in the opponents goal while the defence tries to intercept the ball. The round ends if a goal is scored or the defence captures the ball or the ball goes out of the half-field bounds or if a time-limit is reached. Multiple agents can be added to both teams but for the sake of simplicity we follow the original HFO study [157] and keep the agents to one per team (an offensive player and a defensive goalie). In our experiments we use demonstrations provided by the offensive agent to teach an intelligent agent to play the offensive role. A simple hand crafted agent is used as the teacher to facilitate communication with the server. More complex higher performing hand crafted agents can be used to provide demonstrations in the future. Figure 8.3 shows the a sequence within the HFO environment with the learned policy controlling the offensive agent. Fig. a) shows the agent approaching the goal with the ball. Fig. b) shows the agent kicking the ball towards the goal. Fig c) shows the ball crossing the line to score a goal.

8.4.3 Experimental Setup

The experimental evaluation compares between 4 learning methods. Firstly the proposed LSTM trained on the captured sequences. This method referred to as "LSTM episodic" as each episode or round of HFO makes up a training sequence. Secondly, "LSTM segmented", where the LSTM model is trained on the uniformly segmented sequences. The third method "MLP" trains a supervised multi-layer perception on the training set. And finally "MLP shuffled" is similar to "MLP" but shuffles the dataset before training the model. Moreover, data aggregation of 3 iterations is applied to the MLP based approaches. All models are trained on the same collected demonstrations consisting of 20000 samples. Data aggregation adds a further 5000 samples for each iteration. The sequence length used for training "LSTM segmented" is 80 samples. The models are trained offline for 1500 epochs and the trained networks are saved to be later used by the agent in real time.

We use a client that is decoupled from the learned models to connect to the simulator server so that the same client can be used to execute any learned policy. The client communicates with the simulator to receive the raw sensory data observed by the agent at each frame and send the decisions of the neural network to control the agent's actions. The models are evaluated on 1000 rounds of HFO. Each round can end in one of 4 outcomes. Firstly, a goal is scored, which is the best possible outcome, followed by the defence capturing the ball, then the ball going out of bounds and finally running out of time represents the poorest behaviour by the agent. During training, this ranking of outcome is not taken into consideration as the learning is not reward based, but rather depends on imitating the provided demonstration. Table 8.2 shows the percentage of each outcome achieved by the teacher used to provide the demonstrations; in 1000 rounds of playing.

Table 8.2: HFO results for the hand crafted teacher

Method	**Goal**	**Defence**	**Bounds**	**Time**
Teacher	44.37%	51.43%	4.19%	0%

8.4.4 Results

Firstly, the results comparing the proposed LSTM approach to imitation learning without memory are presented. Figure 8.4 shows the results for the 4 trained models "LSTM episodic", "LSTM segmented", "MLP" and "MLP shuffled". The results are shown for 1000 rounds of testing. The graph shows the percentage of rounds that resulted in the 4 possible outcomes: goal scored, captured by the defence, ball out of bounds and out of time. The percentage of goals scored is the most important measure as scoring is the primary objective of the task, however the other measures show the rest of the picture. The proposed method "LSTM episodic" has resulted in the highest percentage of goals, similar to the teacher's performance and outperforms networks without memory with statistical significance. "LSTM segmented" comes in second place also outperforming the MLP methods with statistical significance, demonstrating that utilizing memory is the contributing factor in the effectiveness of the learned policy, even if the beginning and end of the sequence are unknown.

The results also show that shuffling the training set resulted in significantly more goals. This corroborates the hypothesis that using dependant sequences of samples to train models without memory can be detrimental as the training batches lack diversity. The remaining measures show the robustness of utilized imitation learning methods with small percentages of unwanted outcomes ("out of bounds" and "out of time") especially the LSTM based methods. Qualitative analysis of the performance shows that running out of time is usually the result of the agent getting stuck and constantly performing the same action. As can be expected the teacher never produced this outcome and it is considered the poorest behaviour displayed by the imitating agents. Being stuck is an indication of ambiguity in the agent, and the extremely low percentage of this behaviour in the LSTM agents demonstrates that utilizing memory significantly improves the generalization of the learned policy in addition to its effectiveness.

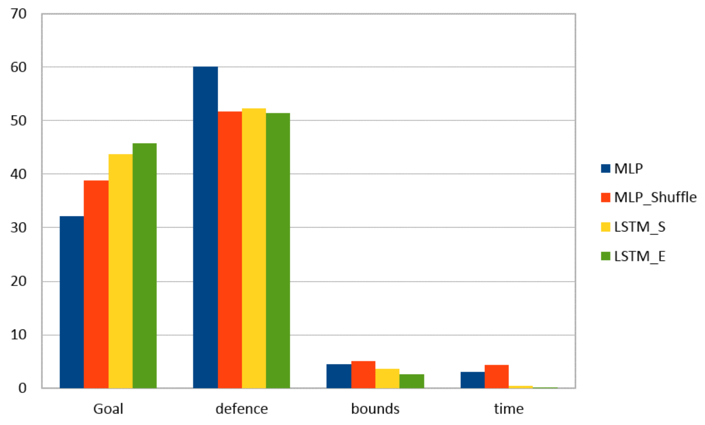


Figure 8.4: Results for robocup half field offence. The outcomes presented in the graph are: Goal: The offensive agent scored a goal, Defence: The ball was captured by the defence, Bounds: The ball went out of bounds, Time: A time limit was reached before any of the other outcomes

Following, the results for data aggregation are presented. Figure 8.5 shows the results for using data aggregation on the MLP network, with and without shuffling the data. With "MLP shuffle", the entire data set is shuffled each iteration, so the training batches can contain samples from all the aggregated datasets. The graph shows the percentage of goals scored in 1000 rounds for supervised learning (using the original training set), and three iterations of data aggregation. The graph shows that for "MLP shuffle", there is no significant improvement in the percentage of scored goals. Without shuffling, we can see an improvement in scoring for the first two DAGGER iterations but this pattern does not hold for the third iteration. For both methods, the graph shows no consistent improvement in scoring with increasing the DAGGER iterations and in all cases does not reach the performance of the LSTM approaches.

Tables 8.3 and 8.4 shows the complete results for data aggregation with multi-layer perceptrons, with and without shuffling respectively. The results show that aggregating new samples does not necessarily decrease the percentage of undesirable outcomes. For both approaches, there does not appear a pattern for decreasing the "out of time"

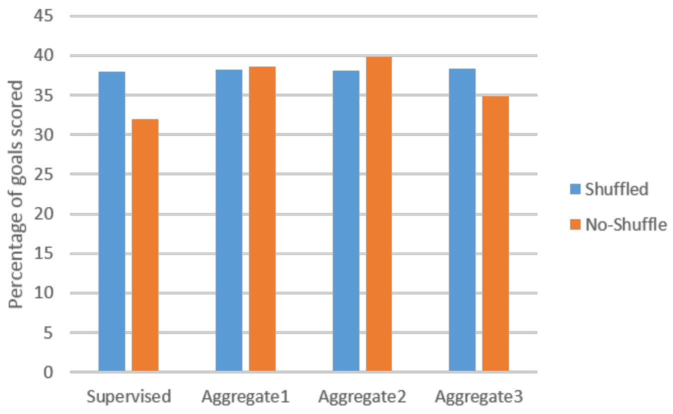


Figure 8.5: Scoring percentage for MLP with data aggregation

percentages with increasing iterations of data aggregation. In "MLP no-shuffle" where the scoring rate was substantially improved with the first iteration of data aggregation, we observe that this improvement is accompanied by a huge rise in the percentage of "out of time" rounds. This emphasises that data aggregation in this study does not provide a consistent improvement in the agent's performance. Although data aggregation utilizes more information, by sampling demonstrations from likely states, it fails to improve the generalization of the agent compared to the proposed LSTM approach.

Table 8.3: MLP data aggregation results with shuffling

Method	**Goal**	**Defence**	**Bounds**	**Time**
Supervised	38.83%	51.74%	5.02%	4.40%
Aggregate 1	38.17%	55.12%	4.46%	2.23%
Aggregate 2	38.01%	53.09%	5.68%	3.20%
Aggregate 3	38.31%	53.14%	2.98%	5.56%

Table 8.4: MLP data aggregation results without shuffling

Method	**Goal**	**Defence**	**Bounds**	**Time**
Supervised	32.19%	60.12%	4.56%	3.11%
Aggregate 1	38.59%	49.08%	4.27%	7.94%
Aggregate 2	39.81%	54.86%	3.78%	1.53%
Aggregate 3	34.89%	59.89%	3.67%	1.53%

8.5 Conclusion and Future Work

This chapter proposed an imitation learning approach for learning from sequences in a dynamic environment. A demonstration is represented as an ordered sequence of state-action pairs. The states are represented by a feature vector of low level sensory information from the agent's perspective. The actions available to the agent are low level parametrized actions. A deep Long-short-term-memory network is used to learn a policy that retains a memory of past experiences and learns from the entire demonstrated trajectory of actions. The trained model uses memory to provide context to improve generalization and predicts an action at every frame in real-time. Results on a multi-agent soccer simulator show that learning from sequences using memory networks can significantly outperform learning from i.i.d samples and reach comparable performance to the teacher. Using the memory to provide context when learning from sequences outperforms data aggregation methods for improving generalization and is much faster to train. Moreover, it is also shown that the proposed LSTM method can be stationary by training on sequences that are arbitrarily segmented from the demonstrations without a significant drop in performance. Experiments using multi-layer perceptrons show that if the model has no memory when learning from sequences, shuffling the training data can result in a significant improvement in performance as the samples in the training batches become more diverse. In the next step, we aim to use a number of high performing agents from the robocup competitions to provide the demonstrations and include more agents in the game. Memory networks are proposed to learn from partially observed states by keeping track of observations in past frames when considering the current frame.

Chapter 9

Conclusion and Future work

This chapter summarises the main outcomes and conclusions resulting from this body of work.

This thesis contributes to the advancement of the state of the art in imitation learning by proposing novel deep learning based approaches. In particular, the methods proposed in this thesis address two key challenges: 1- Representation of sensory information in a manner that is adequate for learning and general enough to be applicable to various problems. 2- Generalization of the learned policies to new situations.

9.1 Summary

This section summarizes the conclusions made from the studies and experiments in this thesis.

Firstly, the representation problem is addressed by a novel method that reduces learning from demonstrations to image classification and utilizes convolutional neural networks to automatically extract relevant features from raw visual data. This approach is shown to learn effective policies from pixels in a number of different tasks using the same network architecture. This conclusion is corroborated by a number of studies that use similar CNN based methods for learning a variety of problems using reinforcement. However, treating imitation learning as an image classification problem greatly reduces the number of training samples required compared to other deep autonomous agent approaches. Directly imitating the demonstrations however, is shown to be prone to generalization problems as well as being dependent on the quality of the provided demonstrations.

To address the generalization problem, we propose an active data aggregation method that is incorporated with the deep imitation learning approach. Deep active imitation allows the agent to query the teacher for a decision in ambiguous situations. Using active learning provides efficient sampling of the added information and can significantly improve generalization using relatively few aggregated training samples. However, depending on the teacher to improve the agent's generalization remains restricted by the teacher's performance. The teacher can provide suboptimal demonstrations or utilize information not available to the agent such as memory of past events. Moreover, it may not be applicable to provide demonstrations mid trajectory for data aggregation.

Therefore, to alleviate the dependency on the teacher, we propose to address the generalization problem by combining imitation learning with reinforcement learning. These two approaches complement each other as reinforcement learning explores new situations without supervision while demonstrations provide valuable information about how to perform the task. Combining learning from demonstrations and experience is proposed by initializing reinforcement learning policies using supervised learning and using demonstrations to guide the exploration policy. Although these methods did not result in good behaviour, guiding exploration through demonstrations is an interesting direction to follow in future work. We attribute the poor performance to unbalanced value updates, as the terminal states only provide updates for certain actions. Updating all action values did not work either as the values are overwritten due to the extended update freezing period. In the future we propose to alleviate this imbalance by using policy-based or actor-critic reinforcement learning. A number of very recent studies are starting to investigate this direction. In [160, 161], the policy based Deep Deterministic Policy Gradients (DDPG) [138] is used. In [162] an actor-critic method is used where all action values for the critic are updated, this could perhaps be attributed to quick updates which were not applicable in our tasks.

A more stable method of levering demonstrations in deep reinforcement learning is proposed by incorporating knowledge from the demonstrations in the reward signal. A deep imitation learning policy is used for reward shaping, to create artificial rewards in non-terminal states based on a teacher's demonstrations. The benefit of this approach over the previously proposed combination is two fold. Firstly, it is stable as the reward signal doesn't get overwritten and doesn't affect experience sampling. Secondly, it ensures frequent rewards based on the similarity of the performed action to the demonstrated trajectories. The results show that the deep reward shaping approach significantly expatiates and improves DQN without requiring any task specific knowledge as the reward shaping is automatically learned from raw demonstrations. Moreover, a method for dynamically updating the value network is shown to provide more efficient and robust learning than fixing the updating parameter. This method is evaluated on a simple task that non the less reflects a number of realistic challenges. Future work will include applying reward shaping from demonstrations to more dynamic environments.

To tackle a dynamic multi-agent environment, we propose a method that addresses demonstration representation as well as generalization using recurrent neural networks. The raw demonstrations are represented as sequences of dependent state-action pairs rather than i.i.d. samples and used to learn a policy using long-short-term-memory networks. Thus taking advantage of the temporal dependencies between the samples in demonstration and execution. This method is shown to be stationary and does not depend on knowing the position of the current instance in the sequence. A stacked LSTM network is used to learn from raw demonstrations and predict low-level parametrized actions without requiring any task-based knowledge. This approach is shown to significantly outperform memoryless networks that treat demonstration samples as i.i.d instances. The LSTM approach is also shown to generalize better than data aggregation which did not consistently improve the generalization of direct imitation policies.

9.2 Discussion

This section provides a closing statement that discusses key conclusions from this thesis and provides recommendations for future directions.

One of the themes of this thesis is utilizing deep learning methods to deal with the high dimensional raw data that is commonly associated with imitation learning problems. Our methods illustrate that deep neural networks whether convolutional or recurrent greatly facilitate general learning processes.This is demonstrated by the networks' ability to learn different tasks without requiring task specific information. Another example is using the same CNN architecture for direct imitation, reward shaping and reinforcement learning without requiring task specific information. In addition to learning hierarchical representations, we tap into deep neural networks' ability to learn multi-variate regression problems. Thus being able to learn problems that entail high degrees of freedom in an end-to end fashion. However, the issue that faces deep neural networks in imitation learning and other applications alike is lack of interpretability. While the networks are designed to learn higher level representations with each layer, it is difficult to extract what features they learned. This could be especially useful in autonomous agent applications as it can facilitate transfer learning and allow mixing learning paradigms that can level from expert knowledge.

This work also highlights the issue of generalization as one of the most important challenges in imitation learning. Even minute supervised error in the machine learning models can lead to accumulation of error and complete failure of the task. This leads us to conclude that direct imitation is an unreliable strategy. Addressing generalization in autonomous agents mainly relies on allowing the agent to perform its current policy to gather extra information in order to refine the policy. While we show that data aggregation methods can significantly improve upon direct imitation, the limitations of data aggregation methods are highlighted in the "Mash-simulator" and "robocup" experiments. Similar challenges are also evident in other applications [163]. This leads us to the conclusion that generalization is better achieved by utilizing reinforcement learning and RNNs. LSTMs are also shown to benefit reinforcement learning [140], so this motivates using LSTMs with a combination of reinforcement and imitation learning going forward.

Another theme that is touched upon in this thesis is benchmarking. We argue that reproducibility is an important factor in autonomous agent research as it facilitates comparing and improving existing methods. It is common to use robots or simulators that are not available to other researchers or to design experimental environments that are difficult to reproduce. Recently a number of games and simulators [143, 164, 15] are being extensively used as benchmarks to facilitate the reproduction of results and comparison of different approaches. However, highlighting the challenges and characteristics presented by each individual task can help produce more impactful research as this provides a basis for choosing the tasks with which the proposed methods are demonstrated. In addition some state of the art research still employs custom made simulators that are not available to other researchers. In this thesis we demonstrate our proposed methods on a number of simulated applications. The simulators used are open-source and have clear evaluation criteria for benchmarking. Each application possesses a unique set of challenges and features that make it appropriate for evaluating a given method. To further facilitate reproducibility, implementations of the proposed methods are available on Github.

9.3 Future Directions

Intelligent autonomous learning is a relatively young field and a number of promising future directions are identified to address the challenges of representations and generalization. So far most research assumes the demonstrations are provided from suitable perspective to the learner, and are only concerned with creating adequate feature representations. However, humans learn by watching other people perform tasks from the third person view and then replicates the task from the first person view. [165] propose to learn a mapping between the two perspectives using generative adversarial networks (GAN) [166] to create representations seamlessly.

GANs can also be used to address the generalization problem by generating trajectories that resemble demonstrations by learning a discriminator [82, 167]. This follows the same motivation as inverse reinforcement learning which aims to alleviate the need to engineer reward functions. GAN based methods provide a less complex alternative to inverse reinforcement learning which does not scale well to realistic applications. However, GAN methods for imitation have also been proven difficult to train [163], so further research is needed to produce general GAN-based methods that do not require fine-tuning.

Learning a world model is a trending topic goes beyond generalization through experience [127], as the policy can predict the outcome without going through the experience. Learning such a model is a challenging task but it can be learned from examples or interaction which opens a lot of possibilities for research. This approach would not only enable agents to predict the outcomes of unseen situations, but can directly learn a policy form the model if it is differentiable; instead of generating samples from the model for training.

Bibliography

[1] Billard A, Calinon S, Dillmann R, Schaal S. Robot programming by demonstration. In: Springer handbook of robotics. Springer; 2008. p. 1371–1394.

[2] Schaal S. Is imitation learning the route to humanoid robots? Trends in cognitive sciences. 1999;3(6):233–242.

[3] Schaal S. Learning from Demonstration. In: Mozer MC, Jordan MI, Petsche T, editors. Advances in Neural Information Processing Systems 9. MIT Press; 1997. p. 1040–1046. Available from: http://papers.nips.cc/paper/1224-learning-from-demonstration.pdf.

[4] Bakker P, Kuniyoshi Y. Robot see, robot do: An overview of robot imitation. In: AISB96 Workshop on Learning in Robots and Animals; 1996. p. 3–11.

[5] Coyle D, Moore J, Kristensson PO, Fletcher P, Blackwell A. I did that! Measuring users' experience of agency in their own actions. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM; 2012. p. 2025–2034.

[6] Nass C, Steuer J, Tauber ER. Computers are social actors. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM; 1994. p. 72–78.

[7] Togelius J, De Nardi R, Lucas SM. Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on. IEEE; 2007. p. 252–259.

[8] Russell SJ, Norvig P. Artificial Intelligence: A Modern Approach. 2nd ed. Pearson Education; 2003.

[9] Argall BD, Chernova S, Veloso M, Browning B. A survey of robot learning from demonstration. Robotics and autonomous systems. 2009;57(5):469–483.

[10] Hussein A, Gaber MM, Elyan E. Deep active learning for autonomous navigation. In: International Conference on Engineering Applications of Neural Networks. Springer; 2016. p. 3–17.

[11] Hussein A, Gaber MM, Elyan E, Jayne C. Imitation Learning: A Survey of Learning Methods. ACM Comput Surv. 2017 Apr;50(2):21:1–21:35. Available from: http://doi.acm.org/10.1145/3054912.

[12] Hussein A, Elyan E, Gaber MM, Jayne C. Deep imitation learning for 3D navigation tasks. Neural computing and applications. 2017;p. 1–16.

[13] Hussein A, Elyan E, Gaber MM, Jayne C. Deep reward shaping from demonstrations. In: Neural Networks (IJCNN), 2017 International Joint Conference on. IEEE; 2017. p. 510–517.

[14] Mash-Simulator; 2014. https://github.com/idiap/mash-simulator.

[15] Kitano H, Asada M, Kuniyoshi Y, Noda I, Osawa E. RoboCup: The Robot World Cup Initiative. In: Proceedings of the First International Conference on Autonomous Agents. AGENTS '97. New York, NY, USA: ACM; 1997. p. 340–347. Available from: http://doi.acm.org/10.1145/267658.267738.

[16] Schaal S, Ijspeert A, Billard A. Computational approaches to motor learning by imitation. Philosophical Transactions of the Royal Society B: Biological Sciences. 2003;358(1431):537–547.

[17] Kober J, Bagnell JA, Peters J. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research. 2013;32(11):1238–1274.

[18] Ross S, Bagnell D. Efficient reductions for imitation learning. In: International Conference on Artificial Intelligence and Statistics; 2010. p. 661–668.

[19] Ross S, Gordon GJ, Bagnell JA. A reduction of imitation learning and structured prediction to no-regret online learning. arXiv preprint arXiv:10110686. 2010;.

[20] Ijspeert AJ, Nakanishi J, Schaal S. Learning attractor landscapes for learning motor primitives; 2002.

[21] Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997;9(8):1735–1780.

[22] Cocora A, Kersting K, Plagemann C, Burgard W, De Raedt L. Learning relational navigation policies. In: Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on. IEEE; 2006. p. 2792–2797.

[23] Chernova S, Veloso M. Confidence-based policy learning from demonstration using gaussian mixture models. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM; 2007. p. 233.

[24] Sammut C, Hurst S, Kedzier D, Michie D, et al. Learning to fly. In: Proceedings of the ninth international workshop on Machine learning; 2014. p. 385–393.

[25] Ratliff N, Bradley D, Bagnell JA, Chestnutt J. Boosting Structured Prediction for Imitation Learning. In: Proceedings of the 19th International Conference on Neural Information Processing Systems. NIPS'06. Cambridge, MA, USA: MIT Press; 2006. p. 1153–1160. Available from: http://dl.acm.org/citation.cfm?id=2976456.2976601.

[26] Raza S, Haider S, Williams MA. Teaching coordinated strategies to soccer robots via imitation. In: Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on. IEEE; 2012. p. 1434–1439.

[27] Rahmatizadeh R, Abolghasemi P, Bölöni L. Learning Manipulation Trajectories Using Recurrent Neural Networks. arXiv preprint arXiv:160303833. 2016;.

[28] Kober J, Peters JR. Policy search for motor primitives in robotics. In: Advances in neural information processing systems; 2009. p. 849–856.

[29] Ijspeert AJ, Nakanishi J, Schaal S. Learning rhythmic movements by demonstration using non-linear oscillators. In: Proceedings of the ieee/rsj int. conference on intelligent robots and systems (iros2002). BIOROB-CONF-2002-003; 2002. p. 958–963.

[30] Nakanishi J, Morimoto J, Endo G, Cheng G, Schaal S, Kawato M. Learning from demonstration and adaptation of biped locomotion. Robotics and Autonomous Systems. 2004;47(2):79–91.

[31] Mülling K, Kober J, Kroemer O, Peters J. Learning to select and generalize striking movements in robot table tennis. The International Journal of Robotics Research. 2013;32(3):263–279.

[32] Schaal S, Mohajerian P, Ijspeert A. Dynamics systems vs. optimal controla unifying view. Progress in brain research. 2007;165:425–445.

[33] Kober J, Mohler B, Peters J. Learning perceptual coupling for motor primitives. In: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE; 2008. p. 834–839.

[34] Kober J, Peters J. Learning motor primitives for robotics. In: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE; 2009. p. 2112–2118.

[35] Calinon S, Li Z, Alizadeh T, Tsagarakis NG, Caldwell DG. Statistical dynamical systems for skills acquisition in humanoids. In: 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012). IEEE; 2012. p. 323–329.

[36] Rozo L, Bruno D, Calinon S, Caldwell DG. Learning optimal controllers in human-robot cooperative transportation tasks with position and force constraints. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE; 2015. p. 1024–1030.

[37] Chen N, Bayer J, Urban S, Van Der Smagt P. Efficient movement representation by embedding Dynamic Movement Primitives in deep autoencoders. In: Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on. IEEE; 2015. p. 434–440.

[38] Bentivegna DC, Atkeson CG, Cheng G. Learning tasks from observation and practice. Robotics and Autonomous Systems. 2004;47(2):163–169.

[39] Chernova S, Veloso M. Teaching collaborative multi-robot tasks through demonstration. In: Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on. IEEE; 2008. p. 385–390.

[40] Berger E, Amor HB, Vogt D, Jung B. Towards a simulator for imitation learning with kinesthetic bootstrapping. In: Workshop Proceedings of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR); 2008. p. 167–173.

[41] Calinon S, Billard A. A framework integrating statistical and social cues to teach a humanoid robot new skills. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Workshop on Social Interaction with Intelligent Indoor Robots. LASA-CONF-2008-020; 2008. .

[42] Lin LJ. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning. 1992;8(3-4):293–321.

[43] Lin LJ. Programming Robots Using Reinforcement Learning and Teaching. In: Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2. AAAI'91. AAAI Press; 1991. p. 781–786. Available from: http://dl.acm.org/citation.cfm?id=1865756.1865798.

[44] Guenter F, Hersch M, Calinon S, Billard A. Reinforcement learning for imitating constrained reaching movements. Advanced Robotics. 2007;21(13):1521–1544.

[45] Kober J, Peters J. Imitation and reinforcement learning. Robotics & Automation Magazine, IEEE. 2010;17(2):55–62.

[46] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529(7587):484–489.

[47] Clark C, Storkey A. Training deep convolutional neural networks to play go. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15); 2015. p. 1766–1774.

[48] Brys T, Harutyunyan A, Suay HB, Chernova S, Taylor ME, Nowé A. Reinforcement learning from demonstration through shaping. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI); 2015. .

[49] Ng AY, Harada D, Russell S. Policy invariance under reward transformations: Theory and application to reward shaping. In: In Proceedings of the Sixteenth International Conference on Machine Learning. vol. 99; 1999. p. 278–287.

[50] Abbeel P, Ng AY. Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning. ACM; 2004. p. 1.

[51] Kohl N, Stone P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on. vol. 3. IEEE; 2004. p. 2619–2624.

[52] Peters J, Schaal S. Reinforcement learning of motor skills with policy gradients. Neural networks. 2008;21(4):682–697.

[53] Kober J, Peters J. Movement templates for learning of hitting and batting. In: Learning Motor Skills. Springer; 2014. p. 69–82.

[54] Buchli J, Stulp F, Theodorou E, Schaal S. Learning variable impedance control. The International Journal of Robotics Research. 2011;30(7):820–833.

[55] Pastor P, Kalakrishnan M, Chitta S, Theodorou E, Schaal S. Skill learning and task outcome prediction for manipulation. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE; 2011. p. 3828–3834.

[56] Hester T, Vecerik M, Pietquin O, Lanctot M, Schaul T, Piot B, et al. Learning from Demonstrations for Real World Reinforcement Learning. arXiv preprint arXiv:170403732. 2017;.

[57] Levine S, Koltun V. Guided policy search. In: Proceedings of The 30th International Conference on Machine Learning; 2013. p. 1–9.

[58] Zhang M, McCarthy Z, Finn C, Levine S, Abbeel P. Learning deep neural network policies with continuous memory states. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE; 2016. p. 520–527.

[59] Guo X, Singh S, Lee H, Lewis RL, Wang X. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In: Advances in Neural Information Processing Systems; 2014. p. 3338–3346.

[60] Levine S, Finn C, Darrell T, Abbeel P. End-to-End Training of Deep Visuomotor Policies. arXiv preprint arXiv:150400702. 2015;.

[61] Nolfi S, Floreano D. Evolutionary Robotics: The Biology,Intelligence,and Technology. Cambridge, MA, USA: MIT Press; 2000.

[62] Rokbani N, Zaidi A, Alimi AM. Prototyping a biped robot using an educational robotics kit. In: Education and e-Learning Innovations (ICEELI), 2012 International Conference on. IEEE; 2012. p. 1–4.

[63] Min HQ, Zhu JH, Zheng XJ. Obstacle avoidance with multi-objective optimization by PSO in dynamic environment. In: 2005 International Conference on Machine Learning and Cybernetics. vol. 5. IEEE; 2005. p. 2950–2956.

[64] Zhang Y, Wang S, Ji G. A comprehensive survey on particle swarm optimization algorithm and its applications. Mathematical Problems in Engineering. 2015;2015.

[65] Zhang C, Zhen Z, Wang D, Li M. UAV path planning method based on ant colony optimization. In: 2010 Chinese Control and Decision Conference. IEEE; 2010. p. 3790–3792.

[66] Aler R, Garcia O, Valls JM. Correcting and improving imitation models of humans for robosoccer agents. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on. vol. 3. IEEE; 2005. p. 2402–2409.

[67] Ortega J, Shaker N, Togelius J, Yannakakis GN. Imitating human playing styles in super mario bros. Entertainment Computing. 2013;4(2):93–104.

[68] Sun TY, Huo CL, Tsai SJ, Liu CC. Optimal UAV flight path planning using skeletonization and particle swarm optimizer. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence). IEEE; 2008. p. 1183–1188.

[69] Cheng R, Jin Y. A social learning particle swarm optimization algorithm for scalable optimization. Information Sciences. 2015;291:43–60.

[70] Gruau F, Quatramaran K. Cellular encoding for interactive evolutionary robotics. In: Fourth European Conference on Artificial Life. MIT Press; 1997. p. 368–377.

[71] Bongard JC, Hornby GS. Combining fitness-based search and user modeling in evolutionary robotics. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM; 2013. p. 159–166.

[72] Lin HI, Liu YC, Chen CL. Evaluation of human-robot arm movement imitation. In: Control Conference (ASCC), 2011 8th Asian. IEEE; 2011. p. 287–292.

[73] El-Hussieny H, Assal SF, Abouelsoud A, Megahed SM, Ogasawara T. Incremental learning of reach-to-grasp behavior: A PSO-based Inverse optimal control approach. In: 2015 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR). IEEE; 2015. p. 129–135.

[74] Pan SJ, Yang Q. A survey on transfer learning. Knowledge and Data Engineering, IEEE Transactions on. 2010;22(10):1345–1359.

[75] Torrey L, Walker T, Shavlik J, Maclin R. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In: Machine Learning: ECML 2005. Springer; 2005. p. 412–424.

[76] Torrey L, Shavlik J. Transfer learning. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. 2009;1:242.

[77] Kuhlmann G, Stone P. Graph-based domain mapping for transfer learning in general games. In: Machine Learning: ECML 2007. Springer; 2007. p. 188–200.

[78] Brys T, Harutyunyan A, Taylor ME, Nowé A. Policy Transfer using Reward Shaping. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems; 2015. p. 181–188.

[79] Ziebart BD, Maas AL, Bagnell JA, Dey AK. Maximum Entropy Inverse Reinforcement Learning. In: 23rd AAAI Conference on Artificial Intelligence and the 20th Innovative Applications of Artificial Intelligence Conference, AAAI-08/IAAI-08; 2008. p. 1433–1438.

[80] Wulfmeier M, Ondruska P, Posner I. Maximum Entropy Deep Inverse Reinforcement Learning. arXiv preprint arXiv:150704888. 2015;.

[81] Lee G, Luo M, Zambetta F, Li X. Learning a Super Mario controller from examples of human play. In: Evolutionary Computation (CEC), 2014 IEEE Congress on. IEEE; 2014. p. 1–8.

[82] Ho J, Ermon S. Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems; 2016. p. 4565–4573.

[83] Daumé Iii H, Langford J, Marcu D. Search-based structured prediction. Machine learning. 2009;75(3):297–325.

[84] Le HM, Kang A, Yue Y, Carr P. Smooth Imitation Learning for Online Sequence Prediction. Proceedings of the 33rd International Conference on Machine Learning. 2016;.

[85] Droniou A, Ivaldi S, Sigaud O. Learning a repertoire of actions with deep neural networks. In: Development and Learning and Epigenetic Robotics (ICDL-Epirob), 2014 Joint IEEE International Conferences on. IEEE; 2014. p. 229–234.

[86] Mayer H, Gomez F, Wierstra D, Nagy I, Knoll A, Schmidhuber J. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. Advanced Robotics. 2008;22(13-14):1521–1537.

[87] Judah K, Fern A, Dietterich TG. Active imitation learning via reduction to iid active learning. arXiv preprint arXiv:12104876. 2012;.

[88] Ikemoto S, Amor HB, Minato T, Jung B, Ishiguro H. Physical human-robot interaction: Mutual learning and adaptation. Robotics & Automation Magazine, IEEE. 2012;19(4):24–35.

[89] Calinon S, Billard AG. What is the teachers role in robot programming by demonstration?: Toward benchmarks for improved learning. Interaction Studies. 2007;8(3):441–464.

[90] Dixon KR, Khosla PK. Learning by observation with mobile robots: A computational approach. In: Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on. vol. 1. IEEE; 2004. p. 102–107.

[91] Saunders J, Nehaniv CL, Dautenhahn K. Teaching robots by moulding behavior and scaffolding the environment. In: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction. ACM; 2006. p. 118–125.

[92] Coates A, Abbeel P, Ng AY. Learning for control from multiple demonstrations. In: Proceedings of the 25th international conference on Machine learning. ACM; 2008. p. 144–151.

[93] Pook PK, Ballard DH. Recognizing teleoperated manipulations. In: Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on. IEEE; 1993. p. 578–585.

[94] Oztop E, Arbib MA. Schema design and implementation of the grasp-related mirror neuron system. Biological cybernetics. 2002;87(2):116–140.

[95] Nicolescu MN, Mataric MJ. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. ACM; 2003. p. 241–248.

[96] Asfour T, Azad P, Gyarfas F, Dillmann R. Imitation learning of dual-arm manipulation tasks in humanoid robots. International Journal of Humanoid Robotics. 2008;5(02):183–202.

[97] Geng T, Lee M, Hülse M. Transferring human grasping synergies to a robot. Mechatronics. 2011;21(1):272–284.

[98] Mataric MJ. Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics. In: Imitation in animals and artifacts. Citeseer; 2000. .

[99] Billard A, Matarić MJ. Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture. Robotics and Autonomous Systems. 2001;37(2):145–160.

[100] Ude A, Atkeson CG, Riley M. Programming full-body movements for humanoid robots by observation. Robotics and autonomous systems. 2004;47(2):93–108.

[101] Niekum S, Chitta S, Barto AG, Marthi B, Osentoski S. Incremental Semantically Grounded Learning from Demonstration. In: Robotics: Science and Systems. vol. 9; 2013. .

[102] Rozo L, Jiménez P, Torras C. A robot learning from demonstration framework to perform force-based manipulation tasks. Intelligent service robotics. 2013;6(1):33–51.

[103] Vogt D, Amor HB, Berger E, Jung B. Learning Two-Person Interaction Models for Responsive Synthetic Humanoids. Journal of Virtual Reality and Broadcasting. 2014;11(1).

[104] Geisler B. An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game. Citeseer; 2002.

[105] Thurau C, Bauckhage C, Sagerer G. Learning human-like movement behavior for computer games. In: Proc. Int. Conf. on the Simulation of Adaptive Behavior; 2004. p. 315–323.

[106] Munoz J, Gutierrez G, Sanchis A. Controller for torcs created by imitation. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. IEEE; 2009. p. 271–278.

[107] Cardamone L, Loiacono D, Lanzi PL. Learning drivers for TORCS through imitation using supervised methods. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. IEEE; 2009. p. 148–155.

[108] Muñoz J, Gutierrez G, Sanchis A. A human-like TORCS controller for the Simulated Car Racing Championship. In: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on. IEEE; 2010. p. 473–480.

[109] Vlachos A. An investigation of imitation learning algorithms for structured prediction. In: In Proceedings of the European Workshop on Reinforcement Learning (EWRL). Citeseer; 2012. p. 143–154.

[110] Pomerleau D. Neural Network Vision for Robot Driving. In: Arbib M, editor. The Handbook of Brain Theory and Neural Networks; 1995. .

[111] Feil-Seifer D, Mataric MJ. Defining socially assistive robotics. In: Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on. IEEE; 2005. p. 465–468.

[112] Bemelmans R, Gelderblom GJ, Jonker P, De Witte L. Socially assistive robots in elderly care: A systematic review into effects and effectiveness. Journal of the American Medical Directors Association. 2012;13(2):114–120.

[113] Tapus A, Tapus C, Mataric MJ. The use of socially assistive robots in the design of intelligent cognitive therapies for people with dementia. In: Rehabilitation Robotics, 2009. ICORR 2009. IEEE International Conference on. IEEE; 2009. p. 924–929.

[114] Hingston P. Believable bots. Can Computers Play Like People. 2012;.

[115] Gorman B. Imitation learning through games: theory, implementation and evaluation. Dublin City University; 2009.

[116] Thurau C, Bauckhage C, Sagerer G. Imitation learning at all levels of game-AI. In: Proceedings of the international conference on computer games, artificial intelligence, design and education. vol. 5; 2004. .

[117] Schaul T, Togelius J, Schmidhuber J. Measuring intelligence through games. arXiv preprint arXiv:11091314. 2011;.

[118] Shon AP, Grimes DB, Baker CL, Hoffman MW, Zhou S, Rao RP. Probabilistic gaze imitation and saliency learning in a robotic head. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE; 2005. p. 2865–2870.

[119] Price B, Boutilier C. Implicit Imitation in Multiagent Reinforcement Learning. In: Proceedings of the Sixteenth International Conference on Machine Learning. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1999. p. 325–334. Available from: http://dl.acm.org/citation.cfm?id=645528.657625.

[120] Stone P, Sutton RS, Kuhlmann G. Reinforcement learning for robocup soccer keepaway. Adaptive Behavior. 2005;13(3):165–188.

[121] DAmbrosio DB, Stanley KO. Scalable multiagent learning through indirect encoding of policy geometry. Evolutionary Intelligence. 2013;6(1):1–26.

[122] De Weerdt M, Ter Mors A, Witteveen C. Multi-agent planning: An introduction to planning and coordination. In: In: Handouts of the European Agent Summer. Citeseer; 2005. .

[123] Doya K, Samejima K, Katagiri Ki, Kawato M. Multiple model-based reinforcement learning. Neural computation. 2002;14(6):1347–1369.

[124] Kuvayev D, Sutton RS. Model-based reinforcement learning. Citeseer; 1997.

[125] Liang Y. Model-based Apprenticeship Learning for Robotics in High-dimensional Spaces. 2014;.

[126] Baram N, Anschel O, Mannor S. Model-based adversarial imitation learning. arXiv preprint arXiv:161202179. 2016;.

[127] Nair A, Chen D, Agrawal P, Isola P, Abbeel P, Malik J, et al. Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation. arXiv preprint arXiv:170302018. 2017;.

[128] Curran W, Brys T, Taylor M, Smart W. Using PCA to Efficiently Represent State Spaces. arXiv preprint arXiv:150500322. 2015;.

[129] Bengio Y. Learning deep architectures for AI. Foundations and trends® in Machine Learning. 2009;2(1):1–127.

[130] Sammut C, Hurst S, Kedzier D, Michie D, et al. Learning to fly. In: Proceedings of the ninth international workshop on Machine learning; 1992. p. 385–393.

[131] Abbeel P, Coates A, Quigley M, Ng AY. An application of reinforcement learning to aerobatic helicopter flight. Advances in neural information processing systems. 2007;19:1.

[132] Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, et al. Autonomous inverted helicopter flight via reinforcement learning. In: Experimental Robotics IX. Springer; 2006. p. 363–372.

[133] Silver D, Bagnell J, Stentz A. High performance outdoor navigation from overhead data using imitation learning. Robotics: Science and Systems IV, Zurich, Switzerland. 2008;.

[134] Ratliff N, Bradley D, Bagnell JA, Chestnutt J. Boosting structured prediction for imitation learning. Robotics Institute. 2007;p. 54.

[135] Chernova S, Veloso M. Confidence-based policy learning from demonstration using gaussian mixture models. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM; 2007. p. 233.

[136] Ollis M, Huang WH, Happold M. A bayesian approach to imitation learning for robot navigation. In: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. IEEE; 2007. p. 709–714.

[137] Ross S, Melik-Barkhudarov N, Shankar KS, Wendel A, Dey D, Bagnell JA, et al. Learning monocular reactive uav control in cluttered natural environments. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on. IEEE; 2013. p. 1765–1772.

[138] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971. 2015;.

[139] Zhu Y, Mottaghi R, Kolve E, Lim JJ, Gupta A, Fei-Fei L, et al. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. arXiv preprint arXiv:160905143. 2016;.

[140] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, et al. Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:160201783. 2016;.

[141] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–533.

[142] Koutník J, Cuccu G, Schmidhuber J, Gomez F. Evolving large-scale neural networks for vision-based reinforcement learning. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM; 2013. p. 1061–1068.

[143] Bellemare MG, Naddaf Y, Veness J, Bowling M. The arcade learning environment: An evaluation platform for general agents. arXiv preprint arXiv:12074708. 2012;.

[144] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602. 2013;.

[145] Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:160904747. 2016;.

[146] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints. 2016 May;abs/1605.02688. Available from: http://arxiv.org/abs/1605.02688.

[147] Heinrich J, Silver D. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. arXiv preprint arXiv:160301121. 2016;.

[148] Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu AA, et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences. 2017;p. 201611835.

[149] Graves A, Mohamed Ar, Hinton G. Speech recognition with deep recurrent neural networks. In: Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. IEEE; 2013. p. 6645–6649.

[150] Mikolov T, Karafiát M, Burget L, Cernockỳ J, Khudanpur S. Recurrent neural network based language model. In: Interspeech. vol. 2; 2010. p. 3.

[151] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Advances in neural information processing systems; 2014. p. 3104–3112.

[152] Graves A, et al. Supervised sequence labelling with recurrent neural networks. vol. 385. Springer; 2012.

[153] Graves A. Generating sequences with recurrent neural networks. arXiv preprint arXiv:13080850. 2013;.

[154] Pietro AD, While L, Barone L. Learning in RoboCup keepaway using evolutionary algorithms. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc.; 2002. p. 1065–1072.

[155] Jain D, Shah M, Garg BK. Watchdogs 2D Soccer Simulation;.

[156] Masson W, Ranchod P, Konidaris G. Reinforcement learning with parameterized actions. arXiv preprint arXiv:150901644. 2015;.

[157] Hausknecht M, Stone P. Deep reinforcement learning in parameterized action space. arXiv preprint arXiv:151104143. 2015;.

[158] Hausknecht M, Mupparaju P, Subramanian S, Kalyanakrishnan S, Stone P. Half field offense: An environment for multiagent learning and ad hoc teamwork. In: AAMAS Adaptive Learning Agents (ALA) Workshop; 2016. .

[159] Stone P, Kuhlmann G, Taylor ME, Liu Y. In: Bredenfeld A, Jacoff A, Noda I, Takahashi Y, editors. Keepaway Soccer: From Machine Learning Testbed to Benchmark. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 93–105. Available from: https://doi.org/10.1007/11780519_9.

[160] Večerík M, Hester T, Scholz J, Wang F, Pietquin O, Piot B, et al. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv:170708817. 2017;.

[161] Nair A, McGrew B, Andrychowicz M, Zaremba W, Abbeel P. Overcoming exploration in reinforcement learning with demonstrations. arXiv preprint arXiv:170910089. 2017;.

[162] Asadi K, Allen C, Roderick M, Mohamed Ar, Konidaris G, Littman M. Mean Actor Critic. arXiv preprint arXiv:170900503. 2017;.

[163] Liu A. Imitation Learning with THOR;.

[164] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. OpenAI Gym. CoRR. 2016;abs/1606.01540. Available from: http://arxiv.org/abs/1606.01540.

[165] Stadie BC, Abbeel P, Sutskever I. Third-Person Imitation Learning. arXiv preprint arXiv:170301703. 2017;.

[166] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. In: Advances in neural information processing systems; 2014. p. 2672–2680.

[167] Fu J, Luo K, Levine S. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. arXiv preprint arXiv:171011248. 2017;.