FFI-RAPPORT

17/01510

Data-driven behavior modeling for computer generated forces

a literature survey

—
Rikke Amilde Løvlid
Linus J. Luotsinen (FOI)
Farzad Kamrani (FOI)
Babak Toghiani-Rizi (FOI)

Data-driven behavior modeling for computer generated forces

a literature survey

Rikke Amilde Løvlid
Linus J. Luotsinen (FOI)
Farzad Kamrani (FOI)
Babak Toghiani-Rizi (FOI)

Norwegian Defence Research Establishment (FFI) 1 December 2017

Keywords
maskinlæring
datagenererte styrker

FFI-rapport
FFI-RAPPORT 17/01510

Prosjektnummer
1380

ISBN
P: 978-82-464-3010-2
E: 978-82-464-3011-9

Approved by
Ole Martin Mevassvik, *Research Manager*
Tor-Odd Høydal, *Acting Director*

Summary

Computer Generated Forces (CGFs) have been used within military simulation-based, training and decision support applications for decades. CGFs are autonomous or semi-autonomous entities that typically represent military units such as tanks, soldiers, and combat aircrafts. Their main purpose, at least within training applications, is to reduce human role playing, which allows for more efficient use of military training facilities. Although CGFs are undoubtedly useful, their behavioral capabilities are often limited to follow doctrines, rules of engagement or heuristics identified by human domain experts that not necessarily represent the behavior observed in the CGF's real-world counterpart.

In this report we introduce and provide a literature review of works related to the Data-Driven Behavior Modeling (DDBM) approach, which is an alternative to the traditional domain expert based behavior modeling approach. In DDBM computers are employed to analyze and identify behavioral patterns in data using machine learning techniques. DDBM is believed to more efficiently produce CGFs that are more objective and realistic compared to CGFs where the behavioral patterns mainly have been identified using subjective human experts.

This report is the result of a collaborative effort between the Norwegian Defence Research Establishment (FFI) and the Swedish Defence Research Agency (FOI), "Technical arrangement number 4 FFI-FOI - Terrain Analysis and Synthetic Agents".

Sammendrag

Datagenererte styrker har lenge blitt brukt i militære simuleringsbaserte verktøy for trening og beslutningsstøtte. Datagenererte styker er autonome eller semiautonome entiteter som representerer militære enheter, for eksempel stridsvogner, soldater og fly. Hovedformålet med bruk av datagenererte styrker er å kunne gjennomføre øvelser med mindre personell ved at en person kan styre flere militære enheter. Selv om datagenererte styrker er nyttige i dag, er den autonome oppførselen ofte begrensa til å følge ideelle regler identifisert av domeneeksperter. De reflekterer ikke nødvendigvis den menneskelige oppførselen til de virkelige systemene som de datagenererte styrkene representerer.

Denne rapport introduserer og presenterer en litteraturoversikt over datadrevet oppførselsmodellering (DDBM), som er et alternativ til den tradisjonelle domeneekspertbaserte måten å modellere oppførselen til datagenererte styrker på. I DDBM brukes maskinlæringsalgoritmer for å analysere og identifisere oppførselsmønstre og for å generere oppførselsmodeller fra eksempler på ønska oppførsel. Målet er å kunne generere oppførsel mer effektivt og lage oppførselsmodeller som er mer objektive og realistiske.

Denne rapporten er resultatet av et samarbeid mellom Forsvarets forskningsinstitutt (FFI) og Totalförsvarets forskningsinstitut (FOI) i Sverige, "Technical arrangement nummer 4 FFI-FOI - Terrain Analysis and Synthetic Agents".

Contents

1 **Introduction** 7
- 1.1 Purpose of this report 7
- 1.2 Terminology 7

2 **Literature survey** 9
- 2.1 Observational learning 9
  - 2.1.1 Data acquisition 10
  - 2.1.2 Feature extraction 12
  - 2.1.3 Machine learning techniques 14
- 2.2 Experiential learning 14
  - 2.2.1 Machine learning techniques 15
- 2.3 Hybrid learning 17

3 **Discussion and conclusion** 18
- 3.1 Challenges 18

Appendix

A **Summaries of selected articles and papers** 25
- A.1 Identifying physical team behaviors from spatial relationships 25
- A.2 Automated robot behavior recognition applied to robotic soccer 25
- A.3 Automatic recognition of human team behaviors 26
- A.4 Learning collaborative team behavior from observation 28
- A.5 A case-based reasoning framework for developing agents using learning by observation 28
- A.6 Learning from observation using primitives 29
- A.7 Building high-performing human-like tactical agents through observation and experience 29
- A.8 A two-stage genetic programming approach for non-player characters 30
- A.9 Modeling physical capabilities of humanoid agents using motion capture data 30
- A.10 Evolving models from observed human performance 31
- A.11 Learning human like behavior for computer games 31
- A.12 The behavior modeling of computer generated warship forces system based on neural network 32
- A.13 Adaptive computer-generated forces for simulator-based training 32
- A.14 Demonstration-based behavior programming for embodied virtual agents 32
- A.15 Goal-directed learning to fly 33
- A.16 Learning goal hierarchies from structured observations and expert annotations 33
- A.17 Human-level control through deep reinforcement learning 34
- A.18 Dueling network architectures for deep reinforcement learning 35
- A.19 Asynchronous methods for deep reinforcement learning 35
- A.20 Mastering the game of Go with deep neural networks and tree search 36

A.21 Continuous control with deep reinforcement learning 38
A.22 Reinforcement learning with unsupervised auxiliary tasks 38

1 Introduction

Computer Generated Forces (CGFs) are autonomous or semi-autonomous actors within military, simulation based, training and decision support applications. CGFs are often used to replace human role-players in military exercises to, ultimately, improve training efficiency. The modeling and development of CGFs is a complex, time-consuming and expensive endeavor where military domain expertise and doctrinal knowledge are interpreted and programmed into the CGF by hand. Furthermore, CGFs often represent human actors and behaviors (pilots, soldiers, manned systems, etc.) making it an even more challenging task.

In recent years the Artificial Intelligence (AI) research community has achieved some remarkable results where Intelligent Agents (IA) successfully defeated human champions in games such as chess [10], Jeopardy [17] and Go [48]. AI researchers have demonstrated that Machine Learning (ML) algorithms [38] can be used to learn IA behaviors from recorded observations such as log-files, GPS coordinate traces and, more recently, pixels from images and video [40, 41].

Modeling the behavioral rules of CGFs using machine learning, which we refer to as Data-Driven Behavior Modeling (DDBM), has many potential advantages compared to the traditional modeling approach where the behavioral rules are manually hand-crafted using subject matter experts and doctrines. Using DDBM, the modeling efficiency with respect to cost and time may improve, in particular, when modeling complex CGFs designed to mimic human actors and behaviors within complex environments. The DDBM approach may also improve behavior realism and objectiveness resulting in better and more realistic training and decision support tools.

This report is organized as follows. Chapter 2 provides a survey of work related to DDBM. We have not tried to make and exhaustive search, but have included a selection of papers that provide an overview of the state of the art in the field. More details about some of the papers are included in Appendix A. Potential benefits and challenges of using DDBM are addressed in Chapter 3.

1.1 Purpose of this report

The purpose of this report is to introduce the reader to the concepts of DDBM and to review related works where DDBM, or a similar approach, has been evaluated and applied in the past.

The intended audience of this report are researchers in the military modeling and simulation community and personnel/operators at military training and decision support facilities.

1.2 Terminology

Let us list a few terms that may be useful when reading this report:

- *Artificial intelligence (AI)*: Intelligence shown by machines.

- *Agent*: In the AI-community an agent is defined as anything capable of perceiving and acting upon its environment [45].
- *Intelligent agent (IA)*: An IA is an agent that acts humanly or rationally [45].
- *Computer generated forces (CGF)*: The CGF is a specialization of an IA, primarily employed in simulation based military decision support and training tools.
- *Supervised learning*: Machine learning algorithms that learn from labeled datasets.
- *Unsupervised learning*: Machine learning algorithms that learn from unlabeled datasets.
- *Reinforcement learning*: Machine learning algorithms that learn by optimizing reinforcement, reward or fitness functions.
- *Observational learning*: Modeling behaviors from observations of the desired behavior. Typically implemented using supervised machine learning techniques.
- *Experiential learning*: Modeling behaviors using a trial-and-error approach where a target simulator is used to improve the behavior model based on an evaluation, fitness or reward function. Typically implemented using reinforcement machine learning techniques.
- *Hybrid learning*: Combining observational- and experiential learning.
- *Data-driven behavior modeling (DDBM)*: Using observational, experiential or hybrid learning to generate behavior models.

2 Literature survey

DDBM is the use of machine learning techniques to automatically generate behavior rules[1] from examples of correct behavior or some measurement of what is correct, good, or representative. Using machine learning it is possible to learn behaviors online, i.e. during operations, or offline, i.e. prior to being applied. In this report we focus mainly on the offline learning techniques.

In this chapter we present the main findings of our literature review. The review has been organized to emphasize the work-flow within the three main learning strategies, *observational*, *experiential* and *hybrid*. Extended summaries of selected papers are listed in Appendix A.

2.1 Observational learning

In observational learning, the goal is to develop a behavior model for a CGF entity by *observing* the behavior of the agent whose behavior should be learned (so called original agent). The data collected from the original agent performing an activity, in a simulation or in the real world, is used to train a CGF entity to act similarly when attempting to perform the same activity under similar conditions [52].

Other terms used in the literature that are largely synonymous to observational learning or learning from observation are *learning from demonstration* and *learning by imitation*. In learning from demonstration a human purposely demonstrates how to perform a given task in order to make the agent perform the same task, whereas in learning from observation, the one being observed does not need to be a willing participant [44]. Learning from imitation or mimicking refers to learning the exact same actions, whereas observational learning is primary about learning to induce some effects in the environments.

In the machine learning community, the term learning by observation is often used in a broad sense and refers to the fact that the training data is a set of observations. However, although the observed data is used to learn (e.g. in handwritten character recognition), in general, the data does not demonstrate how to perform a task, or how to teach any behavioral skills [15]. We use the term observational learning in a more specific context, by learning from observation we mean how to learn the behavior of an observed entity performing some activity.

Observational learning is essentially similar to supervised learning in the sense that it learns from the observed data. However, there are some principal differences between these two. In observational learning, the input data is a trace of human performance with different length, and it is not necessarily clear where one example starts and where it ends. Furthermore, the labels are implicit rather than explicit. In traditional supervised learning the input data is explicitly defined with features and labels [52].

Several authors have investigated observational learning in different domains, using a variety of techniques [44, 15, 26, 27]. For instance, Johnson and Gonzalez [26, 27] presented a prototype

[1]With behavior rules we do not necessarily mean if-then-rules. It can be any kind of behavior representation, such as a neural network or a decision tree.

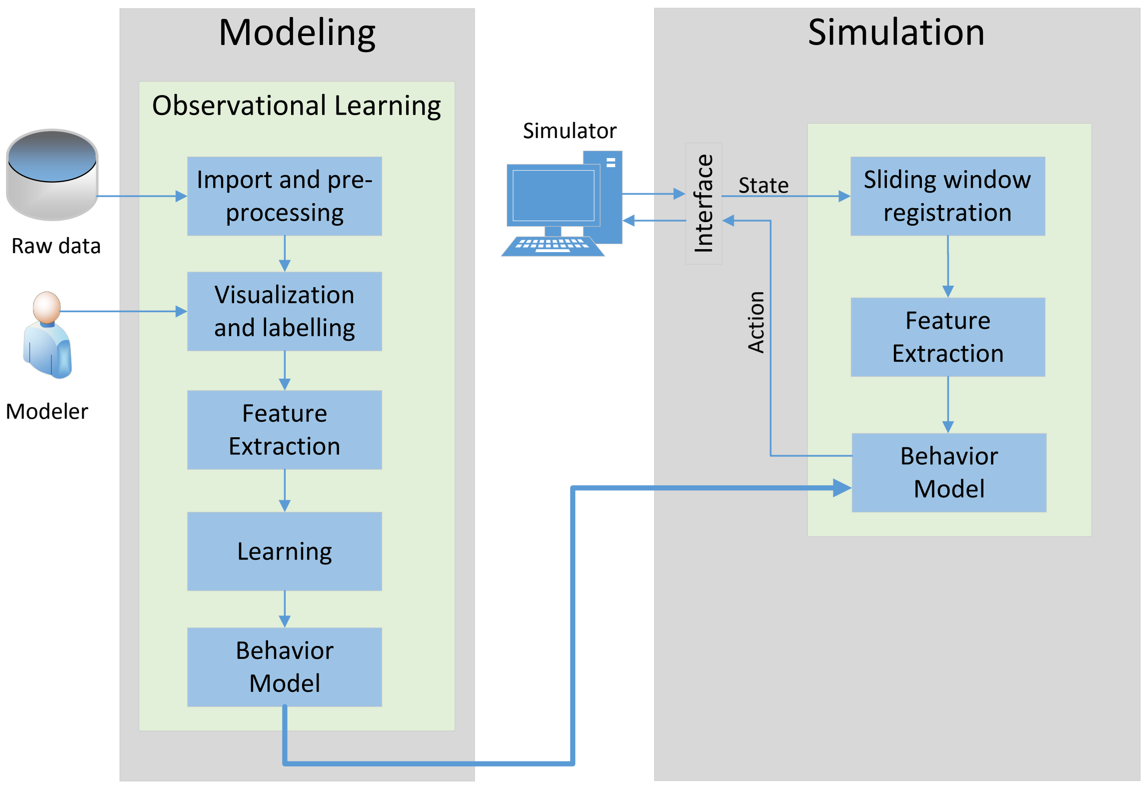


Figure 2.1 *Observational learning.*

with the focus on learning team behavior from observations. The approach is semi-autonomous and observations are manually processed to identify domain specific contexts representing different states of the observed behavior. In robotic research it is common to use learning from demonstration, where a human demonstrates and teaches a robot on how to perform a given task [4, 8].

The observational learning strategy can be divided into three steps as illustrated in Figure 2.1:

1. *Data acquisition*: In this phase raw data is acquired from sensors, exercises, simulations, etc. The data is visualized and pre-processed to filter out any invalid data points and if needed supplement the dataset with synthetic data points.
2. *Feature extraction*: In this phase the dataset is further processed using feature extraction functions capable of identifying key features in the data that ultimately reduces the complexity of the learning task.
3. *Learning*: In the learning phase machine learning techniques and algorithms are used to identify rules, patterns and structures that define the CGF's behavior.

2.1.1 Data acquisition

In observational learning the majority of the modeling work consists of collecting, visualizing and pre-processing datasets that, at a later stage, can be fed into the machine learning algorithms. The quality and quantity of the data is perhaps the most important contributing factor to successfully applying observational learning. Unfortunately, to the best of our knowledge, none of the modeling and simulation frameworks or platforms available today embeds machine learning or DDBM capabilities [1]. Hence, there is a lack of software tools tailored for, and specifically targeting, the requirements of the DDBM approach with respect to acquisition, visualization and pre-processing. As a result researchers and modelers typically rely on a mix of general purpose and ad hoc tools when modeling agents and CGFs using DDBM. In the remainder of this section we will briefly highlight a few of these tools used for observational learning.

In recent years several companies turned to big data analysis and data mining techniques to gain an advantage over their competitors, to better understand consumer behaviors etc. This trend resulted in the development of several commercially available software tools, such as RapidMiner[2] and PredictionIO[3]. Besides managing large amounts of data these tools are capable of analyzing and identifying patterns in large datasets.

In the military domain a lot of work has gone into creating tools such as SIMDIS[4], that relatively quickly can be used to reconstruct military incidents using Geographic Information Systems (GIS) and data originating from sensors (GPS, sonar, radar, etc.). These tools typically include playback functionality to improve the end-users ability to analyze incidents as they evolve over time.

Data acquisition and visualization tools are also used in military exercises for planning and evaluation purposes. The Exonaut TEM (Training Exercise Manager)[5], which is used by the Swedish Armed Forces, is an example of a relatively advanced tool capable of planning, coordinating and evaluating large military exercises involving thousands of trainees.

A feasible way of collecting data is to employ a simulator and record the actions of a human performer carrying out the task of interest in a simulated milieu. Using a simulator instead of the real world to collect data offers several benefits. Simulated environment provides a much more flexible settings to design specific scenarios (including difficult or dangerous situations), which support the objective of the studied research. Furthermore, using simulation removes the requirements of complex sensors and image recognition algorithms [16].

Argall *et al.* provide an overview of data acquisition methods for robots that are learning from demonstration [3]. Two potential correspondence issues are identified: 1) Do the recorded data contain the exact states and actions that are experienced by the teacher? 2) Are the recorded states and actions the same as those that the learner would observe?

Teleoperation is the simplest method for data acquisition in robotics. Here the teacher steers the robot using a remote control and the states and actions are recorded from the robot point of view. A corresponding method for CGFs would be to record the actions and states while manually controlling the CGFs. A more complex setting would be to use records of human actions while performing some task in the real world or in another virtual environment.

Active learning is a machine learning method in which the learning algorithm may interactively pose queries to the teacher. The main idea behind active learning is that by choosing the data

[2]https://rapidminer.com/

[3]https://prediction.io/

[4]https://simdis.nrl.navy.mil/

[5]https://www.4cstrategies.com/exonaut-products/training-and-exercise-manager

by which the algorithm learns, it can achieve performance of greater accuracy. Active learning is especially well-motivated in scenarios where copious unlabeled data is available, but labeling data manually is time-consuming and expensive [47]. Since learning by demonstration permits intervention with the learning process, it should be possible to apply methods and findings in active learning to perform learning by demonstration.

Generative adversarial nets (GAN) [21] is an interesting approach that can be used to generate synthetic data using machine learning. The main idea of GAN is to learn two models using only a limited amount of unlabeled training data. The first model is called the generator and its main goal is to generate realistic data given some random input. The second model is called the discriminator and its purpose is to classify and separate real (i.e. training data) from generated data. During the learning phase these models are competing against each other and as a result they iteratively improve their capability to generate and discriminate data respectively.

2.1.2 Feature extraction

Although computers are efficient in data processing, they generally do not perform well if the data is not presented in an appropriate format. Hence, gathered raw data cannot typically be used directly, and different pre-processing measures (filtering, partitioning, etc.) and feature extraction are required to prepare the data and represent it in a way that is understandable and meaningful for the ML algorithms. Pre-processing sometimes include transforming spatial data to a reference system in order to ensure that the observation can be generalized to seemingly different but in fact equivalent situations.

The simple act of observing human performers can be quite difficult, given that computers are not as efficient as humans in processing visual data and recognizing patterns. Therefore, the first step in learning by observation is to observe the behavior and represent it as data that is easily processed by a learning algorithm [26].

Representing observed behavior can be performed manually, automatically, or by a combination of these two approaches. For instance, Sukthankar *et al.* manually annotated the data to label individual domain-specific behaviors of human motion captures, such as walking, probing, inspecting, and covering [53]. In later work they used an automatic approach to identify physical team behaviors from spatial relationships [55, 54] in the military operations in urban terrain (MOUT) domain.

Modeling team behaviors is a challenging task since agents in the team often execute different actions simultaneously. Teams operating in physical domains have spatial characteristics that can be used to recognize the team behavior. Example of these characteristics are spatial formations including the relative position of the agents and their positions in relation to buildings, doorways and other static objects.

Sukthankar and Sycara [55] exploited the distinctive spatial structure to identify team behaviors in MOUT operations. The main idea was to compare the spatial relationships between physical entities in a static scenario with a model library to identify the team behavior based on a similarity measure [55]. A collection of unique spatial models that had no temporal order represented each team behavior. Features, such as positions of relevant entities, entity types, pairwise constraints between entities (e.g., line of visibility or lack of visibility), and scaling constraints (e.g., limitation of distance between agents) constituted each spatial model. Although the scenario contained both static (buildings, obstacles, etc.) and dynamic (foot soldiers) entities, the analyzed models are static, i.e., they represent snapshots of spatial formations. To improve generalization capabilities, a set of legal transforms (rotation, translation and scaling) must be defined. These transforms ensure that a model developed for one scenario can match behaviors observed in other spatial layouts. The ability of an identified model to generalize to similar circumstances is an essential part of data pre-processing. Without generalization it is unfeasible to exhaustively enumerate all possible occurrence of relationships in different layouts.

A more complicated scenario arises when the goal is to identify team behaviors considering the temporal aspect and execution order of events. Although, certain team behaviors that include highly discriminating spatial relationships can be identified from snapshots of states, many behaviors are recognizable only by examining the succession of states as they unfold.

The overall structure of these problems is similar to problems in temporal pattern recognition (e.g., speech and gesture recognition) in that unobserved Markovian states are inferred from a sequence of observed features. Hidden Markov Models (HMM) have been widely applied in speech recognition, handwritten character recognition, and natural language processing. Thus, it is appealing to use HMM for the problem of behavior recognition. Several researchers have successfully applied HMM to behavior recognition [22, 54, 36].

Han and Veloso [22] use HMMs to represent and recognize strategic behavior of agents in simulated robotic soccer, where it is advantageous for an agent to identify and predict the other agents' behaviors. Although the robotic soccer domain is multi-agent, the work in [22] is limited to one agent and the ball. In this application, the discrete set of states represent the agent's mental state, and the set of observations represent its physical state. Separate HMMs are trained and used to represent the behaviors. Given an observation, the HMM with the highest probability determines the current behavior and strategy of the robot.

Unlike domains such as speech recognition, which can naturally be segmented in words, the segmentation of robot behaviors is not well-defined. The robot behavior changes continuously and there is no gap (segmentation point) between different behaviors. To work around this problem, new HMMs are instantiated at regular intervals. If a HMM is instantiated at a point of time close to a behavior start time, then the behavior is recognized.

It is generally time consuming to create HMM of team behaviors through knowledge engineering (manual creation of the model). Moreover, the hidden nodes of the HMM, which correspond to the mental state of the agents are not always intuitively identifiable. Luotsinen *et al.* [36] have developed an interactive application for the editing and manipulation of recorded observations to simplify the identification and extraction of representative examples from relatively large observation databases. The resulted set of representative examples is then used as training and validation data to automatically learn teamwork behavior HMMs.

In [54], movement data (position and orientation) of members in a MOUT-team is recorded over time. In this work, behavior recognition is performed using a set of HMMs (one for each recognizable team behavior).

Johnson and Gonzalez [26, 27] and Fernlund et al. [15, 16] segment the behavior by dividing it into different predefined contexts manually. The behavior in each context is learned separately. Stein and Gonzalez show that dividing the behavior into contexts greatly reduces the complexity of the learning process [51].

Behavior can also be divided into primitives, representing small parts of the task execution. The use of primitives is particularly popular in robot motor control [46], were the primitives represent small parts of a movement. Recorded movements used for learning can be segmented into these primitives and new motions can be realized by new combinations of primitives. Bentivegna and Atkeson use pre-defined primitives to teach a robot to play air hockey [8]. When segmenting the training data, a nearest neighbor algorithm is used to identify the most similar primitive.

2.1.3 Machine learning techniques

A variety of machine learning techniques that fall into the supervised learning category are used for learning by observation. Supervised learning consists of a learning algorithm with training data which include the inputs and expected outputs. The learning algorithm classifies the data and creates a behavior model for the agent from the data. Examples of supervised learning algorithms that are used for observational learning are back propagation for artificial neural networks (ANN), decision tree learning, Bayesian networks, and support vector machines (SVMs).

In many cases the inputs and outputs are not explicitly defined and must be extracted from a sequence of data demonstrating the correct behavior over time. The example behaviors and the corresponding results are input and the behavior model is the output of the learning algorithm [26].

An alternative technique targeting observational learning is inverse reinforcement learning (IRL) [43]. The general idea of IRL is to learn a reward function from observations and then apply this function within a reinforcement learning algorithm to model or learn the observed behavior.

In recent years deep learning (DL) [33, 20] has emerged as a promising technique capable of learning from raw data (e.g., images, video, audio) without the need for explicit feature engineering. That is, the feature extraction phase, as discussed in Section 2.1.2, is embedded within the learning algorithm. There are several advantages with this approach: 1) it does not require the modeler to develop complex feature extraction functions; 2) the hand-crafted features are selected based on human intuition that not necessarily reflects the machine's actual preferences for best performance; and 3) computational performance is improved as there is no need to compute the hand-crafted feature functions. Compared to traditional ANNs, often referred to as shallow learning, the DL approach employs ANNs that are deeper (using more layers, neurons and connections). As a result, DL models typically have higher capacity, i.e. they can model or approximate more complex functions, at the cost of requiring larger datasets and more computational resources for training.

DL in supervised learning has significantly improved the performance of a wide range of applications such as face recognition and identification [57], lip-reading [12], autonomous driving [11], image description synthesis [29] and Q&A systems [17].

2.2 Experiential learning

In experiential learning, the main idea is that the CGF can learn and optimize its behavior using a trial-and-error approach within the target simulator. The approach resembles human learning through practice when carrying out an activity, where the agent executes a sequence of actions in different settings and learns from the outcome of actions taken [50]. As illustrated in Figure 2.2, experiential learning is an iterative process. Usually the agent starts with a random behavior model. In each iteration the agent use the behavior model to decide how to act. An evaluation, fitness, or reward function is used to measure the performance of the agent, and some learning algorithm updates the behavior model based on this measure.

While in observational learning, the objective is to create agents whose behavior are as similar to observed behavior as possible, in experiential learning, the focus is to improve the performance of the agents as much as possible [50]. In experiential learning the agent explores the solution space and learns by maximizing its performance measure within a simulator.

Contrary to observational learning, in experiential learning the performance measure cannot determine the correctness of an action at any given time. Instead it grades the result of the behavior as it unfolds over time [50].

One issue with experiential learning is that the agents may learn interesting but inappropriate behavior. That is, although the performance criteria is met, the agent may not behave as one would expect a human to carry out the activity.

There is a substantial body of research using the experiential learning approach, although they do not explicitly use the term experiential learning (e.g. see [2, 37, 58]). Aihe and Gonzalez [2], propose using reinforcement learning to compensate for situations where the domain expert has a limited knowledge on the subject being modeled. Merrick and Maher [37] present motivated reinforcement learning agents to create non-player game agents that explore their environment and learn new behaviors, in response to interesting experiences. Teng *et al.* [58] use a self-organizing neural network that learns incrementally through real-time interactions with the environment and improves air combat maneuvering strategies of CGFs.

Reinforcement learning has also been applied using deep learning techniques to learn action selection models capable of maximizing future reward. Mnih *et al.* [40, 41] demonstrated this in the context of playing Atari-games. Refer to Appendix A.17 for a detailed summary of this work.

As illustrated in Figure 2.2, experiential learning consists of two alternating steps, evaluation and learning. The latter will be addressed in further detail in the next section.

2.2.1 Machine learning techniques

There are several well known machine learning techniques that can be used to implement experiential learning. Reinforcement learning [56], genetic algorithms (GA) and genetic programming (GP) [32] appear to be appropriate methods for experiential learning [52].

Reinforcement learning (RL) is based on incremental rewards and punishments to teach the agent how to accomplish a task successfully. It is a trial-and-error method, where the output cannot be classified as correct or faulty at any point, but the outcome of taken actions are evaluated by the learning algorithm. Actions that result in successful outcomes are reinforced, while those that result in failures are weakened.

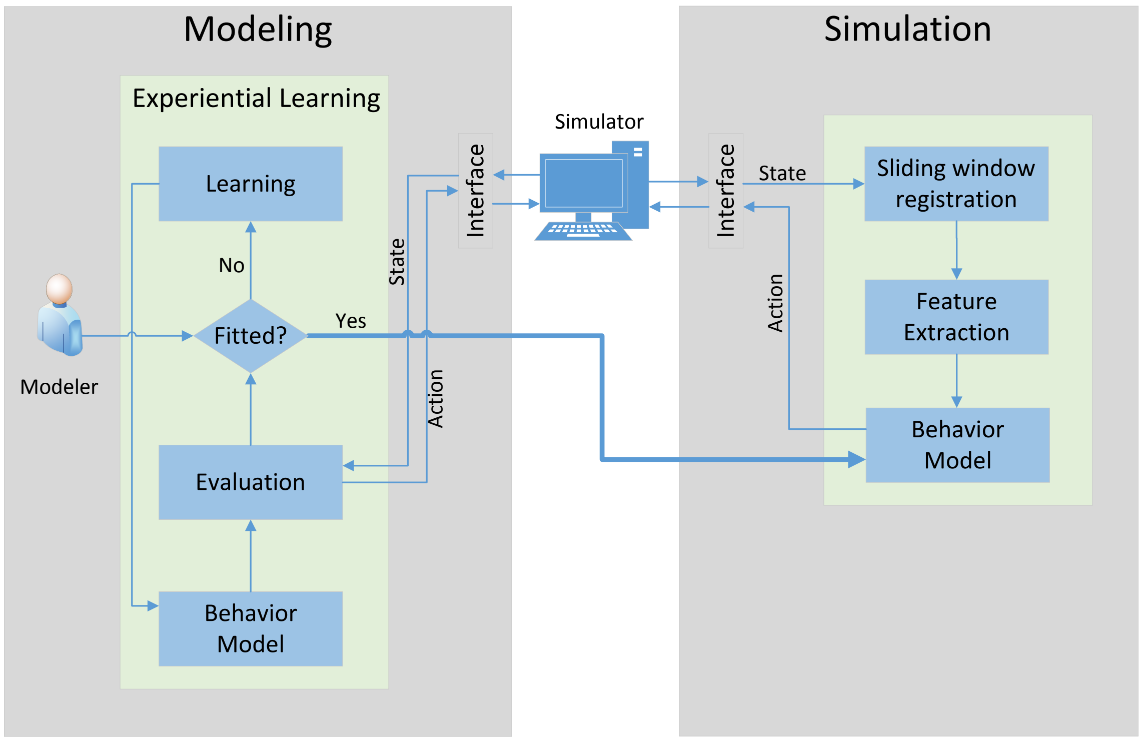


Figure 2.2 *Experiential learning.*

Genetic algorithm [23] is a stochastic search heuristic inspired by Darwin's theories of evolution and natural selection. In this method, a population of candidate solutions (individuals) are randomly generated and evolved in an iterative process toward better solutions using genetic operators such as selection, crossover and mutation. The iteration is stopped after a defined stop criterion is fulfilled, and the solution with the best performance in all generations is presented as the solution of the problem. GA usually uses linear binary array of bits to encode the individuals as a genetic representation (chromosomes) of the solution space. This representation provides a convenient way for simple implementation of genetic operations.

The size of initial population as well as the termination criteria, choice of genetic operators and parameters, which steer to what degree these operators affect the evolution of the population, all depend on the nature of the problem and require a lot of experiments. Another limitation of GA is that like all other heuristic methods, there is no guarantee that it will find the global optimum and may converge towards a local optimum.

Genetic programming [32] is a method that is closely related to GA. In GP, each individual in the population is represented by a computer program. The computer program is modeled using a tree structure where internal nodes represent functions (add, multiply, etc.) and statements (conditional, loop, etc.). Leaves in the tree represent input data, constants and variables.

A difference between GP and GA is that the individuals in a GP population have a variable length whereas individuals in GA are fixed length. In GP, the crossover operator typically swap subtrees of individuals to generate new individuals. As a result, the crossover operator may end up increasing the depth of the trees.

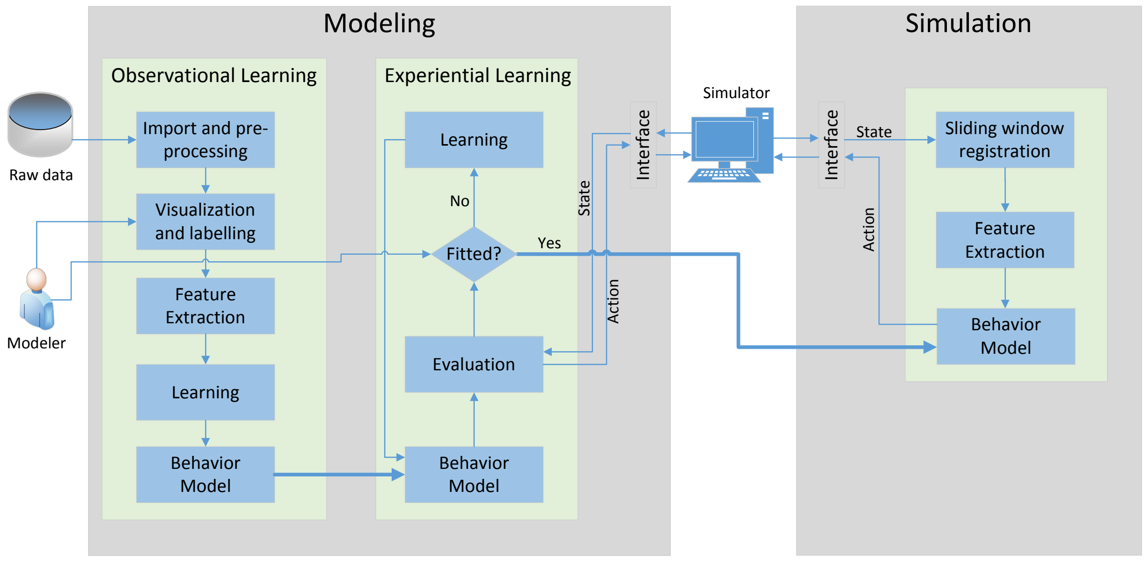


Figure 2.3 *Hybrid learning.*

2.3 Hybrid learning

Several authors have used a *hybrid approach*, combining observational learning and experiential learning methods [8, 50]. The hybrid approach is similar to experiential learning, with the main difference that the agent, instead of random initial solutions, improves solutions that are obtained by observational learning. This is illustrated in Figure 2.3.

In the work of Bentivegna and Atkeson [8], a robot playing air-hockey first observes and learns the behavior of an expert. Second, a reinforcement learning process is used to improve the learned behavior.

Stein and Gonzalez [50] use a hybrid method, in which agents learn tactical skills by observation as well as by experiments (in different domains). The authors suggest that the agents using the hybrid approach are both human-like and perform better than the original human. Learning from observation makes the agents behave human-like, and during the experiential learning phase the performance of the agent is optimized.

AlphaGo [48] is another example where hybrid learning is used to learn how to play the game of Go. A thorough summary of this work is presented in Appendix A.20.

3 Discussion and conclusion

In this report we have introduced and reviewed works related to the DDBM concept where machine learning techniques are applied to create the ”behavioral rules” of a CGF using observations and recorded data. The review covers two different strategies, observational learning and experimental learning, and how they can be combined.

The main advantages of DDBM, if it can be successfully applied to real-world problems, are:

- The military end-users might be able to add new behavior themselves by showing a CGF how to behave.
- It might be faster to add new behavior. This will depend on the cost of generating and preparing training data.
- Observational learning might lead to behavior that is more human like. Human like behavior is not always optimal or even rational, which makes it challenging to model with traditional methods.

3.1 Challenges

Although the DDBM approach appears promising, it also presents a new set of challenges that could limit its applicability in military simulation based tools:

- *Not enough data:* As the dimensions of the feature vectors increase, more data are required to capture the observed behavior using the DDBM approach. However, more data also imply more work for the modeler, which in turn may reduces the efficiency of the approach. There are several dimension reduction algorithms, for instance Principal Component Analysis (PCA) [14] that can be applied to address this challenge. This problem is also related to *the curse of dimensionality* [7].
- *Incomplete and noisy data:* Data originating from military exercises or operations are often noisy and incomplete, which ultimately affects the performance of the DDBM approach. To address this challenge particle or Kalman filters can be used [28].
- *Terrain problems:* The behavior of military entities depends on the surrounding environment. Hence, it is important that these aspects are also encoded within the feature vector. This challenge can be addressed by utilizing advanced geographical analysis tools to calculate line-of-sight, accessibility of various terrain types, etc.
- *Black-box problems:* Machine learning algorithms typically generate models that are too complex for any human to interpret. As a result, the models are considered black-boxes. This is perhaps the most severe drawback of using DDBM or machine learning techniques to model CGF behaviors. Although this challenge is addressed in initiatives such as the explainable AI (XAI) program[6] there is, to the best of our knowledge, no solution available yet.

[6]https://www.darpa.mil/program/explainable-artificial-intelligence

- *Real-time requirements and processing bottle-necks:* CGFs developed using DDBM rely on feature extraction functions to process and adapt the raw data into a format compatible with the selected machine learning algorithms. In military applications these feature will most likely include advanced terrain analysis, which may impact the overall real-time performance of the CGF. This problem can be solved by carefully weighing model fidelity vs. processing performance.

Bibliography

[1] N. Abdellaoui, A. Taylor, and G. Parkinson. Comparative analysis of computer generated forces' artificial intelligence. *NATO Modelling and Simulation Group (NMSG) Symposium (MSG-069): Use of M&S in Support to Operations, Irregular Warfare, Defence against Terrorism, and Coalition Tactical Force Integration*, October 2009.

[2] D. Aihe and A. Gonzalez. Context-driven reinforcement learning. In *Proceedings of the Second Swedish-American Workshop on Modeling and Simulation*, Cocoa Beach, FL, 2004.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[4] C. G. Atkeson and S. Schaal. Learning tasks from a single demonstration. In *In Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 1706–1712, 1997.

[5] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[7] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[8] D. Bentivegna and C. Atkeson. Learning from observation using primitives. In *IEEE International Conference on Robotics and Automation (ICRA).*, volume 2, pages 1988–1993, 2001.

[9] L. Bölöni, L. J. Luotsinen, J. N. Ekblad, T. R. Fitz-Gibbon, C. Houchin, J. Key, M. A. Khan, J. Lyu, J. Nguyen, R. Oleson, G. Stein, S. V. Weide, and V. Trinh. A comparison study of 12 paradigms for developing embodied agents. *Software: Practice and Experience*, 38(3):259–305, 2008.

[10] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu. Deep blue. *Artif. Intell.*, 134(1-2):57–83, jan 2002.

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2722–2730, Washington, DC, USA, 2015. IEEE Computer Society.

[12] J. S. Chung and A. Zisserman. Lip reading in the wild. In *Asian Conference on Computer Vision*, 2016.

[13] J. Dinerstein, P. K. Egbert, D. Ventura, and M. Goodrich. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence*, 24(4):235–256, 2008.

[14] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[15] H. K. G. Fernlund. *Evolving Models from Observed Human Performance*. PhD thesis, University of Central Florida, 2004.

[16] H. K. G. Fernlund, A. J. Gonzalez, M. Georgiopoulos, and R. F. DeMara. Learning tactical human behavior through observation of human performance. *IEEE Transactions on Systems Man and Cybernetics*, 36:128–140, February 2006.

[17] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.

[18] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[19] M. W. Floyd and B. Esfandiari. A case-based reasoning framework for developing agents using learning by observation. In *ICTAI*, pages 531–538. IEEE, 2011.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[22] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In J. Hollerbach and D. Koditschek, editors, *Robotics Research: the Ninth International Symposium*, pages 199–204. Springer-Verlag, London, 2000. Also in the Proceedings of IJCAI-99 Workshop on Team Behaviors and Plan Recognition.

[23] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[24] A. Isaac and C. Sammut. Goal-directed learning to fly. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 258–265. AAAI Press, 2003.

[25] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[26] C. L. Johnson and A. J. Gonzalez. Learning collaborative behavior by observation. In S. Draghici, T. M. Khoshgoftaar, V. Palade, W. Pedrycz, M. A. Wani, and X. Zhu, editors, *ICMLA*, pages 99–104. IEEE Computer Society, 2010.

[27] C. L. Johnson and A. J. Gonzalez. Learning collaborative team behavior from observation. *Expert Syst. Appl.*, 41(5):2316–2328, 2014.

[28] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME - Journal of Basic Engineering*, pages 35–45, March 1960.

[29] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3128–3137, June 2015.

[30] T. Könik and J. E. Laird. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64(1-3):263–287, 2006.

[31] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[32] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

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

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016.

[35] L. Luotsinen, J. Ekblad, A. Wu, A. Gonzalez, and L. Bölöni. A two-stage genetic programming approach for non-player characters. In *FuturePlay 2005 online proceedings (http://www.futureplay.org/papers/paper-181_luotsinen.pdf)*, October 2005.

[36] L. J. Luotsinen, H. Fernlund, and L. Bölöni. Automatic annotation of team actions in observations of embodied agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 9. ACM, 2007.

[37] K. Merrick and M. L. Maher. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '06, New York, NY, USA, 2006. ACM.

[38] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. Technical report, DeepMind Technologies, 2013.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[42] Y. Nan, W. Jiangyun, and S. Yaoxing. The behavior modeling of computer generated warship forces system based on neural network. In *Fluid Power and Mechatronics (FPM), 2011 International Conference on*, pages 920–925, 2011.

[43] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML ’00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[44] S. Ontanon, J. L. Montana, and A. Gonzalez. Towards a unified framework for learning from observation. In *IJCAI Workshop on Agent Learning Interactively from Human Teachers*, 2011.

[45] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[46] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In P. Dario and R. Chatila, editors, *Robotics Research. The Eleventh International Symposium*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 561–572. Springer Berlin Heidelberg, 2005.

[47] B. Settles. Active learning literature survey. Technical Report Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 1210 W. Dayton St. Madison, WI, 2010.

[48] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[49] D. Silver, G. Lever, N. Heess, T. Degris, D. Wiestra, and M. Riedmiller. Deterministic policy gradient algorithms. *ICML*, 2014.

[50] G. Stein and A. Gonzalez. Building high-performing human-like tactical agents through observation and experience. In *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, pages 792–804, 2011.

[51] G. Stein and A. J. Gonzalez. Learning in context: enhancing machine learning with context-based reasoning. *Applied Intelligence*, 41(3):709–724, 2014.

[52] G. Stein, A. J. Gonzalez, and C. Barham. Combining NEAT and PSO for learning tactical human behavior. *Neural Computing and Applications*, pages 1–18, 2014.

[53] G. Sukthankar, M. Mandel, K. Sycara, and J. Hodgins. Modeling physical capabilities of humanoid agents using motion capture data. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, pages 344–351, NYC, NY, jul 2004.

[54] G. Sukthankar and K. Sycara. Automatic recognition of human team behaviors. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO)*, Edinburgh, Scotland, jul 2005.

[55] G. Sukthankar and K. Sycara. Identifying physical team behaviors from spatial relationships. In *Proceedings of Conference on Behavior Representation in Modeling and Simulation*, pages 142–149, Universal City, CA, may 2005. Winner of recommended reading award.

[56] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998.

[57] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1701–1708, Washington, DC, USA, 2014. IEEE Computer Society.

[58] T.-H. Teng, A.-H. Tan, and L.-N. Teow. Adaptive computer-generated forces for simulator-based training. *Expert Systems with Applications*, 40(18):7341–7353, 2013.

[59] C. Thurau, C. Bauckhage, and G. Sagerer. Learning human like behavior for computer games. In *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior*, 2004.

[60] E. Todorov. Mujoco: Modeling, simulation and visualization of multi-joint dynamics with contact, 2016.

[61] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[62] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. *ICML*, 2015.

[63] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 2000.

A Summaries of selected articles and papers

A.1 Identifying physical team behaviors from spatial relationships

Assuming that team activities in physical domains have a distinctive spatial structure (e.g., the relative position of the team members with each other and to static entities), this structure can be used to identify the team behavior of Military Operations in Urban Terrain (MOUT). This paper [55] presents a Random Sampling and Consensus (RANSAC) based algorithm [18] for recognizing the behavior of a team by comparing the physical features of a team behavior with models from a library of created models.

A similarity function scores possible configurations by comparing static entities (e.g., doors and buildings), dynamic entities (e.g., opponents and teammates), transform validity, spatial proximity, and preservation of visibility constraints. The feasibility of the method is demonstrated by recognizing the physical behaviors in a simulated MOUT scenario of a firing team. The recognition is performed by analyzing snapshots of an annotated 2D overhead map. The strength of the algorithm, according to the authors, is that it is robust to spatial variations, generalizes across scenarios, and can be executed in real-time.

In an initial phase, using an authoring tool, one or several spatial models corresponding to each physical behavior are constructed. The library of spatial models then can be used to classify formations of MOUT entities on a 2D map.

It is important to note that no particular temporal structure, or execution order is associated to the models. Thus, the behaviors of MOUT team are represented only by spatial relationships of the entities that are common during the execution of that behavior.

Models are generalized using the set of legal transforms (rotation, translation and scaling). Given a set of spatial models and valid transforms, a statically robust technique (RANSAC) is used to determine which spatial models are applicable to the current annotated map. This method can be summarized as the follows: for each model under consideration, two entities are randomly selected. Two candidate entities on the annotated map having compatible types are randomly selected. These two pairs of points are sufficient to fully specify a transform matrix T (transform hypothesis). T is used to transform the location of every entity in the model to the map. The obtained transform is tested using the distance between the transformed entities in the model with corresponding entities in the map, and models which have a score less than a threshold are rejected.

A.2 Automated robot behavior recognition applied to robotic soccer

This paper [22], formulates robot behaviors (in robotic soccer domain) as Hidden Markov Models (HMMs) to address the problem of autonomous high-level behavior recognition from observed low-level state features. Robot's behaviors is decomposed as the states in HMMs (called Behavior

HMMs). A Behavior HMM is defined to have four different states: (i) initial state (the start of the execution of the behavior), (ii) accept state (successful completion of a behavior), (iii) intermediate state (between initial and accept states), and (iv) reject state (states not reachable or not relevant to the behavior).

The transition probability $Pr(S_{t+1} = s_j | S_t = s_i)$ (for the state transition $x_i \leftarrow s_j$), models the noise in the system and any discrepancies between the model and the actual behavior. The observations denoted by o_i are generated from the location of the objects in the field. There are different methods that generate the observations, depending on what behavior should be recognized. For example, in *absolute position*, the location and orientation of the robot, (x, y, θ) are used to directly compute o_i (using a segmentation of the continuous space (x, y, θ) into regions), while in *object relative*, o_i is computed using the relative location of an object to another (e.g., in the ball-centric behavior, the value of $(x_{robot} - x_{ball}, y_{robot} - y_{ball}, \theta)$ after segmentation is used).

Instead of modeling all behaviors together as one HMM, in order to avoid complexity, the authors choose to use one HMM for each behavior, executing concurrently during recognition. Since behaviors are not mutually exclusive (e.g intercept a ball and go to the ball behaviors), it is possible that two HMMs shows high accepting probabilities simultaneously.

Each behavior is a sequence of state traversals, starting from the initial state, and completing at the accept state. The robot will either idle or start executing another one, after it has executed one behavior. It may also terminate the execution of an behavior and start executing a new one.

The Behavior HMM can recognize execution of a behavior, only if it is instantiated roughly at the time when the real behavior starts executing. In non-continuous domains such as speech recognition, this problem is addressed by word segmentation. However, the segmentation of the robot soccer behaviors is not as well defined. The execution of a sequence of robot behaviors is continuous and it is very hard to find segmentation points between behaviors. To address this problem, the authors choose to instantiate a recognizer at regular intervals, expecting that the start time of the behavior is close enough to one of these recognizers. In order to avoid the number of HMMs to explode, a mechanism for removing HMMs from the system is required. The HMMs are removed if one of the two conditions is fulfilled: 1) a behavior specific timeout threshold is elapsed, 2) a HMM reaches a high probability of rejecting state, indicating that it is unlikely that robot is doing what the Behavior HMM is trying to recognize.

A.3 Automatic recognition of human team behaviors

In this paper [54], the authors present a method for recording, representing, and recognizing human team behaviors in a simulated MOUT scenario. For this reason, they have developed a customized version of the computer game *Unreal Tournament*, that logs all players data (position and orientation), while team members participate in a scenario of a firing team moving through an urban area. Team behavior recognition is achieved using a set of HMMs after the data are translated into a canonical reference frame. Each behavior is modeled by a HMM and the behavior corresponding to the model with the highest log-likelihood is identified as the behavior of the team for a given sequence.

The following are the characteristics of the problem examined in the paper: (i) the team acts in a (simulated) physical domain, (ii) all agents are always performing tasks relevant to accomplishing team goals (tightly coupled teamwork), (iii) the team is never concurrently executing multiple group behaviors (to be compared with [22], where concurrent behaviors were possible), (iv) the majority of the agent's actions involve physical movement, and (v) the focus is on recognizing the complex spatial relationships that exist between the human team members rather than simple location descriptors (e.g., 'in room').

Both spatial and temporal characteristics are used to perform behavior recognition (i.e. invariant spatial relationships in state representation and temporal relationships in the transition matrix of HMM). This can be compared with [55], where the same authors use merely spatial relationships (in MOUT scenarios) that are highly discriminative and rarely appear in other behaviors for recognizing team behavior. However, the method in this paper is more powerful, since it can recognize behaviors that are only distinguishable by examining sequences of spatial relationships as they evolve through time.

The focus of this paper is on behaviors used to approach and enter a building, that is, *stacked movement*, *bounding overwatch*, and *buttonhook entry*. Identifying these behaviors is difficult based on their rather similar static snapshots, motivating using both spatial and temporal relationships. However, in the performed experiments many spatial and temporal cues that exist in a real MOUT scenario are deliberately omitted (to examine the raw accuracy of the classifier). For example, static spatial features (e.g., doors and walls) are not incorporated into the model.

Moreover, the recognitions are performed at a very low-level and do not use inter-window dependencies, which in reality are important cues to the behavior. For example, in a typical building clearing operation, there are often long periods of bounding overwatch followed by a single buttonhook entry through a doorway. Incorporating this higher-level domain knowledge into the model, which provides more accuracy over a full-length MOUT scenario requires using a hierarchical HMM and is postponed to future work.

To achieve translation invariance, at any given time, the configuration of the agent team is described using the centroid of positions of the agents calculated as:

$$c_{j,t} = \frac{1}{A} \sum_{\forall a} x_{a,j,t},$$

where a is an index over A agents, j is an index over W overlapping windows, t is and index over the T frames in a given window, and $x_{a,j,t}$ is the vector containing the position, (x, y) of the agent a at frame t in window j. However, instead of rotating each frame independently, the authors define a shared canonical orientation for all the frames in a window. For this reason, the displacement of the team centroid over the window: $d_j = C_{j,T} - C_{j,1}$ is used as the principal axis of the data points for that window. Thus, the canonical coordinates, x', is calculated by

$$x'_{a,j,t} \equiv R_j x_{a,j,t} - c_{j,t},$$

where R_j is the rotation matrix that rotates all of the data in each window to align its canonical orientation with the x-axis. Moreover, agents' velocity is defined locally as

$$v_{a,j,t} \equiv ||x'_{a,j,t+1} - x'_{a,j,t}||.$$

The goal of the classification is then to select the most appropriate behavior model, for each transformed window. This classification is performed using a set of HMMs, one for each behavior *b*. The model with the highest log-likelihood of generating the observed data is selected. The observation space is continuous and approximated by a multivariate Gaussian distribution. The structure for each behavior HMM is determined based on domain knowledge, and the number of states differs among behaviors. One can say that, each hidden state is the idealized snapshot of the team formation (at some point in time), where the observation tuple (in canonical coordinates) is modeled by a Gaussian distribution.

A.4 Learning collaborative team behavior from observation

Johnson and Gonzalez [26, 27] present a prototype, called Contextually-based Observational Learning of Teamwork System (COLTS), which is developed focusing on the learning of teamwork behavior using observations. This work presents a semi-autonomous approach where observations are manually processed to identify and create domain specific contexts representing the different states of the observed behavior. The prototype is separated into three modules:

- The observer module which is responsible for acquiring observations and identifying contexts.
- The learning module used to automatically create behavior maps, one for each context, using the processed observations. A behavior map relates situational information to an action and the goal of the learning module is to identify actions for all situational combinations.
- The run-time module responsible for executing the behavior models created with the aforementioned modules. The run-time module also contains a generalization technique to ensure that the agents always will chose an action even if the situation was never seen in the training data.

The use of contexts limits the observational training data to embed situations only relevant within the context. This approach reduces the complexity of the learning process to ensure the scalability and extensibility of the cloned behaviors.

The COLTS approach is tested using two discrete event simulations. In the first experiment a bucket brigade team is created capable of efficiently moving water from a source (lake) to a sink (fire). Observations are collected from the simulated team and fed into the COLTS prototype. Results show that the COLTS prototype is able to imitate the team with perfect results. The COLTS prototype is also tested using a more complex pursuit-evasion game with good results.

In future work the authors aim to automate the observer module, i.e. identification of contexts, by extending on previous works where contexts have been automatically created for single, non-cooperative agents. Furthermore, the authors plan on extending the prototype to include, in addition to the discrete event simulations, continuous simulation environments.

A.5 A case-based reasoning framework for developing agents using learning by observation

Floyd and Esfandiari [19] present the Java Learning by Observation Framework (jLOAF), which is capable of learning behaviors by observation in complex, real-world and partially observable environments using a case based reasoning (CBR) approach. The main contribution of this work is the abstract nature of the framework, which enables it to be used independently of the application domain. The authors illustrate this by providing four different case studies:

- an obstacle avoidance robot with a touch and sonar sensor,
- a grabbing robotic arm with a color sensor, touch sensor and sound sensor,
- a soccer playing agent in simulated soccer environment having more complex sensory capabilities that make it capable to sense objects and their locations, and
- a software agent playing Tetris.

The paper shows how inputs and actions from several different domains (both physical and simulated environments) can be modeled in the JLOAF framework, and the same agent, without changing the reasoning module, can learn a variety of behaviors and replicate them.

A.6 Learning from observation using primitives

Bentivegna and Atkeson [8] introduce a framework where an agent or robot first learns the behavior of an expert or teacher by observing its behavior followed by a refinement process where the behavior is improved, beyond the observed behavior, using a learning by practice approach. The framework uses manually hand-crafted primitives, which represent abstract actions to reduce the search space of the learning process to a manageable dimensionality. Primitives are solutions to small parts of a task that can be combined to complete the task. A solution to a task may be made up of many primitives, each of which may consist of smaller primitives. However, the number of primitives are much less than the frequency at which the robot generates commands to all its actuators.

In this work the framework is evaluated using a robot that learns how to play air-hockey. In this domain typical primitives represent hitting the puck in different directions (right, left, straight), blocking the puck, etc. The robot learns from observation by segmenting the available observations into different types of primitives. During game-play a nearest neighbor algorithm is used to identify among all observed primitives, the most similar primitive, given the current state or situation.

A.7 Building high-performing human-like tactical agents through observation and experience

In the work of Stein, Gonzalez and Barham [50, 52] agents learn tactical skills by observation as well as by experimentation (practice). Agent skills are acquired from three domains with increasing complexity: 1) Chaser - The goal is to chase a single, pre-programmed, evading agent; 2) Sheep - The goal is to force multiple, pre-programmed, sheep into a pen; and 3) Car - The goal is to control a car, to avoid hitting other autonomous cars, and to complete as many laps as possible.

To acquire observational data two human players are asked to play the above-mentioned games multiple times using a force-feedback joystick. The joystick activities are recorded in log-files and used to compare similarity between the observationally trained agents and the human performers. For each domain a performance metric is developed that measures the agents as well as the human players success while playing the games. The performance metrics are used to improve agent behavior in the learning by experimentation phase.

The authors suggest that the hybrid approach is capable of creating agents that are human-like and at the same time perform well or better than the original human. Learning by observation creates the initial agent mimicking the human performer. This is followed by the experimental learning phase that allows the agent to learn from random situations never experienced before. In other words, in the first phase the agent is optimized with respect to human similarity and in the second phase the agent is optimized with respect to performance.

Results are promising indicating that it is possible to use this approach when creating agents that are human-like. Humanness is evaluated both qualitatively, using joy-stick logs, as well as quantitatively using questionnaires. Furthermore, human players are allowed to evaluate humanness using haptics by sensing the movement of the agent through the joystick's force feed-back functionality.

In [51] the authors extend on their work to show that a divide-and-conquer approach, using context based reasoning, towards machine learning problems is beneficial.

A.8 A two-stage genetic programming approach for non-player characters

The work by Luotsinen et.al. [35] presents a two-stage learning by doing approach where first tactical then strategic agent behaviors are learned using genetic programming. The approach is evaluated using a simple game called feed-fight-mate (FFM). The approach is further evaluated and compared against 11 other agent paradigms (neural networks, reinforcement learning, forward reasoning, game theory, etc.) in [9] with respect to agent performance and amount of development effort.

A.9 Modeling physical capabilities of humanoid agents using motion capture data

Sukthankar et al. [53] demonstrate a method for modeling an agent's physical capabilities (e.g., running, jumping and sneaking) using motion capture data from human subjects. From the motion capture data a *motion graph* [31] is extracted and a cost map for the agent actions is created. A planner then uses this cost model to select between equivalent goal-achieving plans.

Motion graph is a method for creating realistic (visually-smooth), controllable motion using a collection of motion capture data. It is a directed graph consisting both of pieces of original motion and automatically generated transition. In order to join pairs of frames in the database, one should define a proper distance metric that secures that there is a match between the position and velocity of corresponding body parts in subsequent segments of motion [31].

Using a motion graph, all generated candidate sequences are visually smooth and human-like behaviors. These different sequences are then evaluated based on their costs to choose the most appropriate one. The cost is calculated using two criteria: 1) time, which is directly proportional to the path length in frames (F), and 2) goal achievement, that is the distance of the character's position ($r(x, y)$) and the desired goal position ($g(x, y)$). The cost metric is:

$$C = \alpha F + \beta ||r(x, y) - g(x, y)||^2,$$

where C is cost, and α and β are empirically-determined scaling factors.

A.10 Evolving models from observed human performance

Fernlund et al. use genetic algorithms to generate agents that simulate the behavior of different drivers in a driving simulator [16, 15]. The behaviors of the drivers are categorized into different contexts a priori, e.g. urban driving, intersection turning and traffic light driving. The agents learn the drivers behavior in each of the contexts as well as when to change context.

Each agent is evaluated based on how good it is at simulating the behavior of the corresponding driver, not on how good it is at driving. With this approach it is possible to generate slightly different behaviors for the same situation. This can, for example, be used to create more unpredictable enemy behavior models.

A.11 Learning human like behavior for computer games

Thurau, Sagerer and Bauckhage investigate learning strategic behavior in the commercial computer game Quake II [59]. In this game the strategic goals consist of collecting various items and securing important areas. Artificial agents learn these goals from humans playing the game.

For a representations of the virtual world, a neural gas algorithm is used to learn a waypoint map. The training data consist of all positions visited by a human player during various executions, and the result of applying the neural gas algorithm is a set of positions that is frequently visited by a player. The waypoint map is completed by connecting nodes when there is an observable player movement from one node to the other. Every position on the 3D map is assigned to one of these nodes called position prototypes using a nearest neighbor criterion.

A player's state is represented by a vector consisting of the player's inventory (which weapons or items it has picked up), its current armor and health value. Similar observed player states are grouped to form state prototypes using a neural gas algorithm.

Each position prototype is assigned to one state prototype, forming a cluster of positions for each state. This clustering leads to a number of movement patterns for each state, consisting of a sequence of positions that start when entering the sate and ends when a state change occurs.

For each state a potential field force distribution is computed that recreate the movement patterns typically observed in that state. Changes in state, typically an item pickup, cause switches among the force fields, leading the agent to pursue a new sub goal.

Indirectly, what is learned is where the important items and areas in that particular game environment are and how to move towards them. For military applications, it would probably be more interesting to learn to identify strategic areas in a new environment based on the map.

A.12 The behavior modeling of computer generated warship forces system based on neural network

Nan, Jiangyun and Yaoxing present a surface warship behavior model in a CGF system [42]. They train one feed forward neural network for target identification and one for threat assessment. The target type and threat level were used together with knowledge of available weapons when deciding whether or not weapons should fire immediately and if so, which kind of weapon should be used.

The target identification neural network is trained to distinguish airplanes, helicopters, ships, missiles, torpedoes and submarines based on the targets vertical and horizontal velocity, height, detection distance and radar cross-section. Training examples are generated from predefined rules describing the different targets with the use of the same characteristics, i.e. vertical and horizontal velocity, height etc. The advantage of using the neural network and not the predefined rules for target identification is that the neural network can provide an answer for all input conditions.

A.13 Adaptive computer-generated forces for simulator-based training

Teng, Tan and Teow present an adaptive CGF for simulation-based training [58]. The CGF is used for training 1-v-1 air combat maneuvers, and they argue that a CGF that is able to adapt to the performance of the trainee will increase the realism of the training experience. The CGFs behavior is realized by using a self-organizing neural network that is trained with reinforcement learning.

A.14 Demonstration-based behavior programming for embodied virtual agents

Dinerstein et al. use a combination of programming by demonstration and data-driven behavior synthesis to generate deliberate behavior for agents in complex virtual environments [13]. Target behavior is demonstrated by the user and recorded as sequences of abstract actions that can be recombined to produce novel behavior.

During training the user demonstrate behaviors by controlling the character. The demonstrated behaviors are recorded as a sequences of actions based on fixed or variable time steps. All possible sub-sequences with a given length are then organized by using the *k-means* clustering algorithm. The result is a tree that is used when searching for a new sub-sequence to perform.

The goal of the agent is represented by a fitness function that can be explicit or learned during demonstration. The agent planes its behavior by simulating the outcome of candidate behavior segments and is able to plan and re-plan in non-deterministic environments.

The findings suggest that a relatively small amount of demonstrated behavior is needed in order to achieve interesting behavior, and the authors argue that limiting the action space to demonstrated action sequences makes planning fast and the behavior natural. The technique is tested on several applications with good results.

A.15 Goal-directed learning to fly

Issac and Sammut use learning by observation, or behavior cloning as they call it, to learn to fly maneuvers in turbulence of an aircraft simulator [24]. The training data are collected from real pilots flying the simulator. The training data consist of several behavior traces for each maneuver. For example, the *climb* maneuver is recorded with four different climb-rates, the *turn* maneuver is recorded with four different turn-rates (positive and negative) etc.

The learned control system is twofold with a goal level and a control level. First, the goal level learns appropriate values for turn-rate, climb-rate and acceleration based on desired heading, altitude and airspeed. Second, the control level learns PID controllers which use the difference between current and desired turn-rate, climb-rate, etc. to controll elevators, ailerons, throttle and flaps.

Both the goal and the control level is represented by model trees, one for each variable. Model trees are similar to regression trees, only the leaf nodes can be functions, not only constants.

A.16 Learning goal hierarchies from structured observations and expert annotations

Könik and Laird describe a framework for learning higher level, goal directed behavior based on expert observations [30]. The system uses a symbolic, rule based knowledge representation and the agent architecture SOAR is used as the execution system of the agent program.

One major challenge when learning from observation is that the expert's intentions are not directly available to the learner. Könik and Laird tackle this by making the experts annotate the behavior traces with the goals he/she was pursuing. The annotated traces that are used for learning consist of the selected actions, a symbolic representation of the observed situation, and a symbolic representation of the goal situation that motivates the actions. Additionally, general background knowledge, like it is possible to go trough a door, is implemented separately and is available to the agent at any time.

The behavior knowledge is represented as a hierarchy with goals and subgoals as internal nodes and actions as leaf nodes. Four decision concepts are learned independently for each node: selection condition (when it should be selected), overriding selection condition (when it should be selected even when another is active), maintenance condition (what must be true for the node to continue being active) and termination condition (when the node is completed and should be terminated).

An *inductive logic programming* (ILP algorithm) is used for learning. This algorithm first generates a very specific rule using a single positive example. In the next step, rules consisting of substructures of this initial rule are generated in a heuristic search, where each one of these rules are tested for coverage of positive and negative examples.

In the experiments, records of hand-coded SOAR agents are used to generate training data. This is a cost-effective way to test the framework. At the end of learning, the learned concepts are compared to the original hand-coded concepts. The results seems promising, but further experiments with more complex data are required. The authors also plan to add a second mode to the learning system where the agent generates behavior while the expert gives feedback.

A.17 Human-level control through deep reinforcement learning

A difficulty in Reinforcement Learning (RL) has been to derive efficient representation of the environment from high-dimensional sensory inputs and to generalize from past experiences in order to improve its performance to new situations. In this paper [41], the authors create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks, which for long as been a central goal of general artificial intelligence.

To do this, they proposed a novel agent that uses a deep Q-network (DQN), a deep convolutional neural network, to solve a RL problem. In order to showcase this result, the agent is trained to play on a number of games in the Arcade Learning Enviroment (ALE), which emulates the Atari 2600 enviroment and its respective games [6].

By interacting with the environment through observations, actions and rewards, the agent in RL selects actions in a fashion that maximizes its cumulative future reward. The DQN approximates the optimal action value function as seen in the equation below where the maximum sum of rewards r_t that are discounted by γ at each time step t. This is achievable by the behavior policy $\pi = p(a|s)$, after making the observation s and taking the action a.

$$Q^*(s,a) = \underset{\pi}{max}\mathbf{E}[r_t + \gamma r_{t+1} + (\gamma)^2 r_{t+2} + ...|s_t = s, a_t = a, \pi]$$

Since RL problems are known to be unstable when a nonlinear function approximator, such as deep neural networks, are used to represent the action-value (also referred to as the Q-value), the authors propose two methods of achieving a more stable learning process and increase the potential convergence. The first one is called *Experience Replay* (ER) and is a biologically inspired mechanism that saves a fixed number of past *experiences* to a memory. These experiences reuse a randomly sampled batch in each training step of the network to smooth the data distribution and reduce the correlations in the observation sequence, which results in better generalization in the performance of the agent. The experiences are saved tuples of transitions from state, action, reward and next state: (s_t, a_t, r_t, s_{t+1}).

The second method was to use two separate policy networks to estimate the action-values, an online network with weights θ_i and a target network with weights θ_i^- (where θ_i^- represent the weights of the online network at an earlier iteration i). The online network was used to estimate the action values $Q(s_t, a_t; \theta_i)$ of each state s_t, and target network was then used to estimate the target values

$Q(s_{t+1}, a_{t+1}; \theta_i^-)$ of the next state s_{t+1}, which would then be saved to the ER memory and later be used to update the online network. The online network would be trained using mini batches (including past experiences from the ER memory $\mathcal{D}$) using the loss function below at iteration i:

$$L_i(\theta_i) = \mathbf{E}_{(s_t,a_t,r_t,s_{t+1})\sim U(D)}\left[\left(r_t + \gamma \underset{a_t}{max} Q(s_{t+1}, a_{t+1}; \theta_i^-) - Q(s_t, a_t; \theta_i)\right)^2\right]$$

The authors of the paper present results from where DQN agent was trained and evaluated by comparing its performance on 49 games from the RL literature and professional human game testers. Without incorporating any prior knowledge about the Atari 2600 games, it managed to outperform the best previously existing RL methods on 42 out of the 49 games. Furthermore, the performance of the DWN agent was at a level that was comparable to that of a professional human game tester across all 49 games, achieving more than 75% of the human score on 29 out of them.

A.18 Dueling network architectures for deep reinforcement learning

This paper [62] takes an alternative but complementary approach to the paper described in Section A.17. The authors suggest a new network architecture that is suited for methods like Q-learning. They define an advantage function $A(s_t, a_t)$ that relates the value of the state $V(s_t)$ and action values $Q(s_t, a_t)$ such that:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

The value $V(s_t)$ is a measure of how good it is to be in state s_t (e.g. are you *winning* or *losing*) and $Q(s_t, a_t)$ measures the values of choosing a particular action a_t when in that state. What the advantage function then represents is the relative measure of importance of each action, which leads to better generalization of the agent over similar-valued states.

The proposed network architecture is separated into two steams, one estimating the single value states $V(s_t)$ and another one estimating the action advantages $A(s_t, a_t)$. The action values $Q(s_t, a_t)$ are derived through the combination of the output from these networks, using:

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t)$$

The performance of the dueling network architecture is evaulated through, amongst other, the Atari 2600 enviroment [6] and shows dramatic improvements over previously existing approaches to Deep Reinforcement Learning.

A.19 Asynchronous methods for deep reinforcement learning

The authors of this paper [39] suggest an alternative, asynchronous, method of training agents via Deep Reinforcement Learning (DRL). Techniques like Experience Replay (ER), as described in

Section A.17, has several drawbacks in terms of memory usage and computation per interaction. Also, the stored samples in the ER memory were generated from an older policy, which requires off-policy learning methods like Q-learning.

The asynchronous methods instead use a number of asynchronously executed agents in parallel on multiple instances of the environment to collaboratively train a global model. This helps to reduce the correlations in the agents' data, since the agents are likely to be experiencing a variety of different states at each time step. Also, asynchronous DRL methods do not rely on specialized hardware, such as Graphical Processing Units (GPUs) or massively distributed architectures, and can run on a single machine's standard multi core Central Processing Unit (CPU) instead.

The authors developed four asynchronous DRL algorithms that they evaluate in the paper:

- Asynchronous one-step Q-learning
- Asynchronous one-step SARSA
- Asynchronous n-step Q-learning
- Asynchronous Advantage Actor-Critic (A3C)

where the A3C algorithm was separated into two variants: A3C with a feed forward neural network (A3C-FF) and with an additional Long Short-term Memory (LSTM) layer (A3C-LSTM).

The methods of this paper are evaluated primarily on the same Atari 2600 domain as in Section A.17 and the 3D car racing game TORCS [63]. Further, the A3C algorithm is also evaluated using the Mujoco domain [60] and Labyrinth from the DeepMind Labs enviroment [5].

In the Atari 2600 environment, they manage to surpass previous state-of-the-art results with the A3C algorithms while training for half the time. Further, they show that A3C also manages to succeed at a variety of continuous motor-control tasks as well as navigating 3D mazes using a visual input.

A.20 Mastering the game of Go with deep neural networks and tree search

In this paper [48], it is described how several techniques were used to build a complex model (named *AlphaGo*) consisting of different Deep Neural Networks (DNNs) that could outplay professional players in the Chinese board game Go. It experiments with the approach of combining the following methods of optimization:

- Supervised learning; by training a DNN to imitate expert moves.
- Reinforcement learning; training two DNNs by letting the model play against randomly sampled earlier iterations of itself.
- Performing Monte Carlo Tree Search using the two new DNNs

The general idea behind AlphaGo was that all games of perfect information have an optimal value function $v^*(s)$ that can determine the outcome of the game from every possible board position (or state s) given perfect play. This can be computed recursively using a tree search with approximately b^d possible sequences of moves, where the *breadth* of the game (also referred to as number of legal moves per position) and the *depth* of the game (which is referred to as game length). The paper uses the size of the search tree in chess ($b \approx 35, d \approx 80$) as an example to showcase how the search tree size in the game Go ($b \approx 250, d \approx 150$) makes computing a recursive search tree unfeasible. The authors propose two general principles to reduce the search space of Go. The first one is to reduce the depth of the search space through position evaluation. This allows them to truncate the search tree at state s and replacing the subtree below s by an approximate of the value function $v(s) \approx v^*(s)$. The next principle is to reduce the breadth of the search by sampling actions from a policy $p(a|s)$ that describes a probability distribution of all available actions a in state s.

The first stage of training AlphaGo was to train it on predicting (or imitating) expert moves using a dataset consisting of 30 million different positions (or states) within the game using supervised learning. The trained network is referred to as the policy network p_σ with weights σ. Its architecture consisted of a total of 13 hidden layers, including convolution layers and fully connected layers. The final layer output a softmax probability distribution over all legal actions a. The training was done by randomly sampling state-actions pairs (s, a) and using Stochastic Gradient Descent (SGD) to update the network in the direction to maximize the likelihood of the human move a selected in state s, using the loss function stated in the equation below.

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a_t|s_t)}{\partial\sigma}$$

In the second stage of training AlphaGo, the weights σ were used to initialize an identically structured *policy* network p_ρ such that $\rho = \sigma$. By letting the new policy network p_ρ play against randomly selected previous iterations of itself, the network was further trained and improved using Reinforcement Learning techniques. The reward function $r(s)$ used in the reinforcement learning would return zero for all non terminal states, +1 for winning and -1 for losing. The weights were updated at each time step by SGD in the direction that would maximize the expected outcome using the loss function in the equation below. By now, the p_ρ would win against p_σ in 80% of the head-to-head games.

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\rho} r(s_t)$$

The final stage of training AlphaGo focused on evaluating the position evaluation by estimating the value function $v^p(s)$, that would predict the outcome from position s by both players given policy p. The value function was approximated using a value network $v_\theta(s) \approx v^*(s)$, where $v^*(s)$ would be the assumed hypothetical optimal value function under perfect play. Structurally, the value network looked similar to the previous networks with the exception that it only output a single scalar value. It was trained using regression in state-outcome pairs (s, r) and SGD to minimize the mean squared error between the predicted value $v_\theta(s)$ and the corresponding outcome $r(s_t)$ using the loss function in the equation below.

$$\Delta\theta \propto \frac{\partial \log v_\theta(s_t)}{\partial\theta}(r(s_t) - v_\theta(s))$$

Ultimately, AlphaGo combines the policy and value networks in a Monte Carlo Tree Search-algorithm that selects actions by a lookahead search. This resulted in a state-of-the-art artificial Go-player that manages to beat previously existing artificial Go-players in 494 out of 495 games (99.8%). AlphaGo also managed to beat the winner of 2013, 2014 and 2015 Go Championship Fan Hui in five out of five played games.

A.21 Continuous control with deep reinforcement learning

These authors of this paper [34] adapt the ideas from the underlying success of deep Q-learning (DQL) methods and experiment with applying them to the continuous action domain using an Actor-Critic approach. While methods like DQL (as described in Section A.17) can solve high-dimensional observation spaces, they can only handle discrete and low-dimensional action spaces. Tasks such as physical control of objects have high dimensional action spaces and thus requires an iterative optimization process that continuously optimizes at every step.

The Deterministic Policy Gradient [49] (DPG) maintains a parameterized Actor function $\mu(s|\theta^{\mu})$ which specifies the current policy by deterministically mapping states to that specific action, while the Critic $Q(s_t, a_t)$ is learned like in Q-learning.

The Deep DPG (DDPG) algorithm is similar to DQL [41], but instead uses two separate Actor and Critic networks. Other features of the DQL algorithm, such as Experience Replay memory and a slowly changing targets networks, are also used to improve the stability and the performance of the DDPG agent. Furthermore, the authors solve the challenge of exploration while learning by treating the problem of exploration independently from the learning algorithm. Given the exploration policy μ', noise sampled form a suitable noise process $\mathcal{N}$, to the actors policy:

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$$

The DDPG agents manage to learn how to play numerous MuCoJo [61] control tasks (gripper, puck striking and locomotion) through both low-dimensional state description, such as joint angles and positions, as well as high-dimensional renderings of the environment. The DDQP agent also notably learns in far fewer steps than the DQN algorithm (2.5 million steps, a factor of approximately 20 fewer steps). The authors ultimately conclude that it potentially may solve even more difficult problems than what was experimented with in the scope of this paper.

A.22 Reinforcement learning with unsupervised auxiliary tasks

Most succesful and noted results in deep reinforcement learning come from agents that directly maximise their cumulative extrinsic reward. With the notion that the environments contain a wider variety of possible training signals, the authors of this paper [25] explore an alternative agent that also maximizes many other pseudo-reward functions simultaneously, allowing it to also operate with the absence of extrinsic rewards.

The algorithm, named UNsupervised REinforcement and Auxiliary Learning (UNREAL), uses an network architecture that approximates both the optimal policy and the optimal value function. In addition to that, it also makes auxiliary predictions serve to focus the agent on important aspects of the tasks. These include the long-term as well as short term predictions of cumulative rewards. To further increase efficiency, it uses the Experience Replay mechanism as described in Section A.17, preferentially replaying rewarding sequences.

The auxiliary tasks are incorporated to promote faster training, more robust learning and ultimately better performing agents. By defining them as an additional pseudo-reward function in the environment the agent is interacting with, the authors formally define the auxiliary control task c by the reward function $r^{(c)}$. With the set of possible states $\mathcal{S}$, the space of available actions $\mathcal{A}$, this can be expressed:

$$r^{(c)} : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$$

Given the total set of tasks C where each task $c \in C$, the agent has a separate policy for each task $\pi^{(c)}$ and a base policy π. The overall objective in the training is then to maximize the total performance across the auxiliary tasks. With the discounted return for auxiliary reward $R^{(c)}_{t:t+n} = \sum_{k=1}^{n} \gamma^k r^{(c)}_t$, the set of parameters θ of π and all $\pi^{(c)}$'s, this can be expressed as:

$$\underset{\pi}{\text{argmax}} \mathbf{E}_{\pi}[R_{1:\infty}] + \lambda_c \sum_{c \in C} \mathbf{E}_{\pi_c}[R^{(c)}_{1:\infty}]$$

There are many types of auxiliary tasks, but the authors of this paper focus on two specific types:

- **Pixel changes** - Since changes in the perceptual stream often corresponds with important events in an environment, the authors train the agent to maximize changing of perceived pixels.
- **Network features** - The networks of an agent already learn how to extract task-relevant high-level features of the environment and can hold useful quantities for the agent to learn to control. Thus, the activation of the hidden units within the agent's network is an auxiliary reward in itself. The authors therefore train the agent to maximize the activation of the units within a specific hidden layer of the network.

The UNREAL agent ultimately combines the benefits of two state-of-the-art methods in deep reinforcement learning. Its primary policy is trained using Asynchronous Advantage Actor-Critic (A3C) (as described in Section A.19), while the auxiliary tasks are trained on recent sequences of stored experiences that are randomly sampled.

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

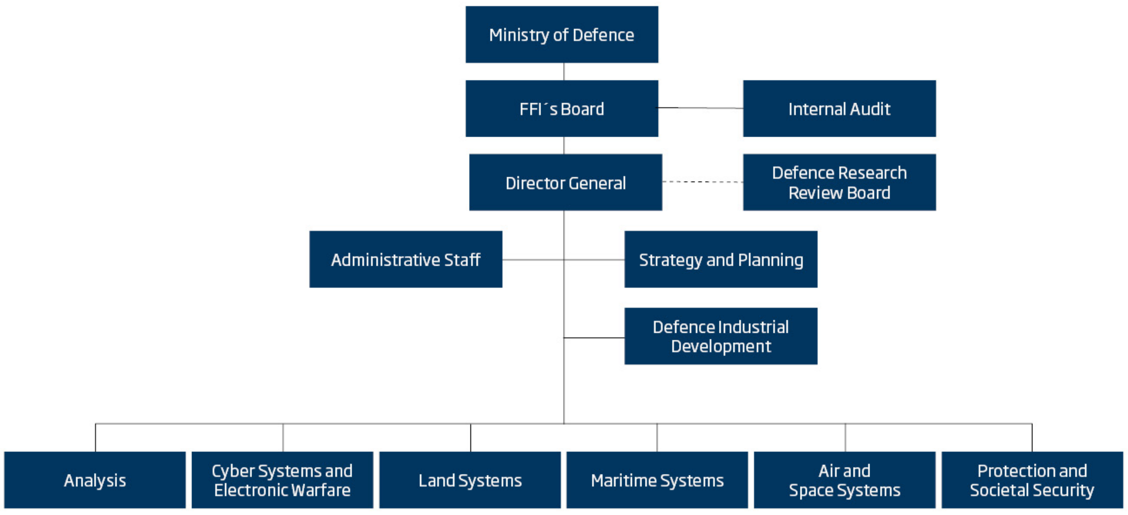
FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFI's organisation



Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment