On Structured Domain Generation for Generalization in Reinforcement Learning

Master Thesis

Serena Bono
Kreiman Lab
Boston Children Hospital
Harvard University
Department of Robotics, Systems and Control
ETH University

Advisor: Spandan Madan
Supervisors: Prof. Dr. Gabriel Kreiman, Prof. Dr. Marco Hutter

December 20, 2022

Abstract

The study of generalization in reinforcement learning (RL) aims to produce algorithms for agents and policies that generalize to novel, unseen environments. There are many real-life applications that could benefit from such a result. For instance, algorithms tend to be trained in simulation to deal with the shortage of data and tested in the real world. The deployment can be tedious especially when the real data distribution is significantly different from the simulation one. In contrast, humans seem to generalize just fine among domains. This work investigates the gap between humans and machines. Furthermore, when testing generalization, most literature suggests modifying the game by alterations to the game elements or by noise addition to the pixel of the image. Though understandable, such an approach introduces indirect, uncontrollable changes at the level of the RL agents. The main contribution of this work is the creation of controlled increasingly different environments to test and compare the generalization ability of classical tabular RL (Q learning), deep RL (DQN), and humans. For this purpose, we choose Pacman, a popular Atari game, and add increasing noise to the transition function of the game. Such an implementation, is controllable, as it is trivial to plot performance as a function of increasing noise, and interpretable, as the noise is directly added to the agent via the Bellman Equation. Preliminary results show that humans perform worse than machines under this setup. This suggests that classical alterations used to test generalization might be unfavorable for the agent and more natural for humans, therefore setting an unfair comparison. Additionally, we compare the generalization and learnability of RL agents on increasingly noisy setups, finding interesting and counterintuitive results. Agents trained on the nonnoisy version of the environment report better performances when tested on different levels of noise, than when learning directly on the noisy environments. This result suggests that learning on an easier version of the environment might be beneficial for better performance on harder versions, opening up new avenues in the study of the robust deployment of the agent in the real world.

Acknowledgements

I would like to sincerely thank my supervisors Prof. Dr. Gabriel Kreiman and Prof. Dr. Marco Hutter. They enabled me to realize this thesis by providing me all the means and assistance necessary for achieving my goals.

Furthermore, I would like to dearly thank my advisor Spandan Madan. He supported me patiently with his knowledge, his experience and his know-how in this field. I really appreciate your support, it was great learning from you.

Contents

List of Figures

Chapter 1

Introduction

1.1 Survey on Generalization in Reinforcement Learning (RL)

Reinforcement Learning agents are used in a myriad of applications ranging from autonomous vehicles [13], algorithm control [4], and robotics [15]. Reality is dynamic, open-ended, and always changing, and to be able to fulfill such demanding applications, RL agents need to have the capability to transfer and adapt to unseen environments during their deployment. State-of-the-art RL algorithms still have issues in generalization to slightly different data distributions, leading to policies that perform badly on even slightly adjusted environment instances. Such an issue demands further study on how to generate agents able to design robust and flexible policies general enough to be applied to different scenarios but also personalized enough to guarantee acceptable performance. In section 2.5 I rigorously define reinforcement learning and highlight the main issues in generalization. Most of the references are taken from [14].

1.2 Biological Correlates of Reinforcement learning

Additionally, I was interested in comparing the generalization abilities of RL agents with human performance. The main motivation for such comparison is that RL, unlike any other branch of artificial intelligence is able to automatically embed priors that resemble human operant learning. In stark contrast with machines, humans are able to learn faster, even just by watching experts, and flexibly generalize to the novel, unseen environments. In this work, I am keen on investigating the reasons for such a generalization gap with the dual purpose of improving SOTA algorithms in RL and generating better models of the human brain. In section 2.6 I explain psychological and neuroscientific correlations to RL and highlight similarities and dissimilarities between human and machine models. Most of the references are taken from [20].

1.3 Challenges and Motivation

The main driver for this work is the lack of rigor in the study of generalization for reinforcement learning. In the literature, agents are usually trained in some setup and tested in some other setup considered to be similar by their human designer. Currently, there is no way of quantifying the difference between the environments. The only two metrics we can reason about are the total final performance and the performance gap between the training and the testing environments. In this author's opinion, this leaves a lot of unanswered questions. Indeed, if we use the final performance to quantify such a difference, it is extremely difficult to find patterns that give insights into how changing the environment impacts the downstream performance. We clearly need a structured way of generating worlds that have quantifiable distance from each other and see how agents perform

when tested onto such environments. Another motivation for this work is to create a unifying framework to test and compare deep RL, classical tabular RL, and humans.

Chapter 2

Related Work

In this chapter, I provide the fundamental theoretical background in order to be able to contextualize and understand the discussion of the subsequent chapters. The chapter begins with a brief introduction to reinforcement learning, starting from the mathematical formulation, the concept of MDPs, and the STAR model. Moreover, the specific RL method used in this thesis, Q-Learning, followed by a brief survey in the field of generalization in RL. Additionally, it provides insights into its biological correlates.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a category of Machine Learning algorithms used for training an intelligent agent to perform tasks or achieve goals in a specific environment by maximizing the expected cumulative reward. As shown in Figure 2.1, the agent gets information about the current state, chooses an action, and receives feedback on the consequences of its action. Such feedback is used to improve future guesses. The process is iterative and guaranteed to converge to what is called an optimal policy. The fundamental mathematical model of reinforcement learning is the *Markov Decision Process* (MDP).

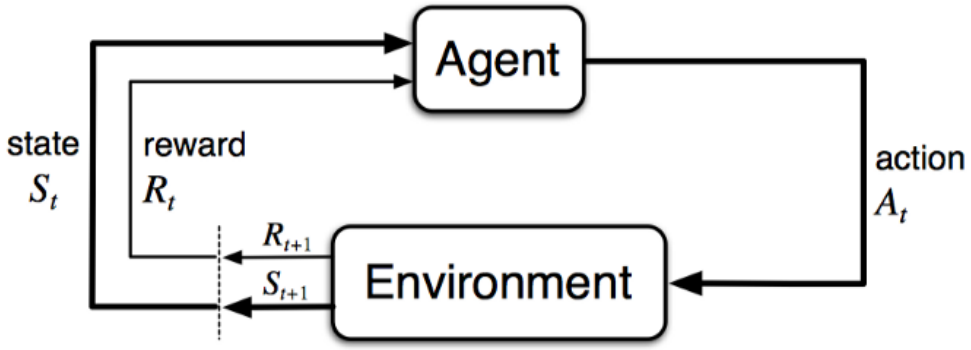


Figure 2.1: Schematic representation of the RL agent and the environment interaction

2.2 Markov Decision Process and the Star Model

Markov Decision Processes (MDPs) are a way to formalize, mathematically, sequential decision-making processes, where actions not only affect immediate rewards, but also future states and rewards. This formalization follows the Markov property, which means that the future must be independent of the past given the present. A discrete MDP would be modeled as the tuple $< S, T, A, R, \lambda >$.

2.3 STAR model (State, Transition Function, Action, Reward)

The $STAR$ model is extremely useful to rigorously define worlds:

1. *The state space* is the set of all attainable states of the environment. States can be completely observable or only partially observable, in either case, they describe the information that the agent gets from the environment. Additionally, state spaces can be continuous or discrete depending on the nature of the world.

2. *The transition function* (TF) is a 3D matrix containing all possible transitions from state s, given action a. Most worlds only admit a limited set of transitions (few attiguous states), therefore TFs are generally very sparse. The dimensionality of such a component is $S \times A \times S$: the complexity of the world increases exponentially with the number of states and actions and this is the fundamental reason why tabular algorithms are only able to deal with small state spaces.

3. *The action space* is the set of all possible actions the agent can take to interact with the environment. The action spaces can be continuous or discrete.

4. *The reward function* (RF) associates every state and action with a given reward. We train the agent to take the "best" actions by giving a positive, neutral, or negative reward based on whether the action has taken the agent closer to achieving a specific goal. The future discounted reward is directly optimized by the Bellman Equation (BE).

Assuming the MDP is correctly defined, we need to design an optimization function embodying the final objective of maximizing cumulative rewards. The *Cumulative Discounted Reward* is defined in Equation 3.1.

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.1}$$

The *Value Function* is defined as the expectation of the cumulative discounted reward, starting in state s, and following policy π. (Equation 3.2)

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{2.2}$$

The *Action-Value Function*, also known as Q-function is defined in Equation 3.4. In this case, this function represents how good it is to take action a while being in state s and following policy π.

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{2.3}$$

The goal is to find the optimal policy for the agent, using the Bellman Expectation Equation defined in equations 3.9 and 2.5, the optimal value function and action-value function can be defined as shown in equations 2.6 and 2.7.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} T(s', r|a, s) \left[r + \gamma v_\pi(s') \right]. \tag{2.4}$$

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s', r|a, s) \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s', a'). \tag{2.5}$$

$$v^*_\pi(s) = \max_a Q^*(s, a) = max_a (r(s, a) + \lambda \sum_{s' \in \mathcal{S}} T(s', r|a, s) v^*(s')) \tag{2.6}$$

$$Q^*(s,a) = (r(s,a) + \lambda \sum_{s' \in \mathcal{S}} T(s', r|a, s) v^*(s')) = (r(s,a) + \lambda \sum_{s' \in \mathcal{S}} T(s', r|a, s) \max_a Q^*(s', a')) \tag{2.7}$$

Oftentimes, in real-world applications, it is unfeasible to collect the entire state space and define the transition function. Algorithms like Q-Learning, automatically learn Q values and implicitly parametrize the state space via linear estimators (tabular Q-Learning) or nonlinear estimators (deep Q-Learning).

2.4 Q-Learning

Q-Learning is one of the most often used RL algorithms. It learns to predict the quality of an action in a specific state (Q value). Q-Learnining is a model-free RL method, meaning that the agent does not have an internal representation of the environment, it learns via interaction in a trial-and-error fashion. In stark contrast with the model-based algorithms, Q-Learning doesn't focus on planning, but only on learning. The main objective of Q-Learning is that of determining an optimal Q-function ($Q(s_t, a_t)$) that associates a value to every action-state pair, it does so by iterating untill convergence with the updating rule defined in Equation 2.8. Even though it might resemble the Reward Function, the Q-function is the result of the interactions of the agent with the environment with the objective of optimizing the cumulative discounted reward. Approximators of such function can be linear, like in the case of tabular Q-Learning, or non-linear, like in the case of Deep Q-Learning (DQN) [17].

$$Q_(s,a) \leftarrow Q_(s,a) + \alpha(r(s,a) + \gamma(\max_{a'} Q(s', a') - Q(s,a)) \tag{2.8}$$

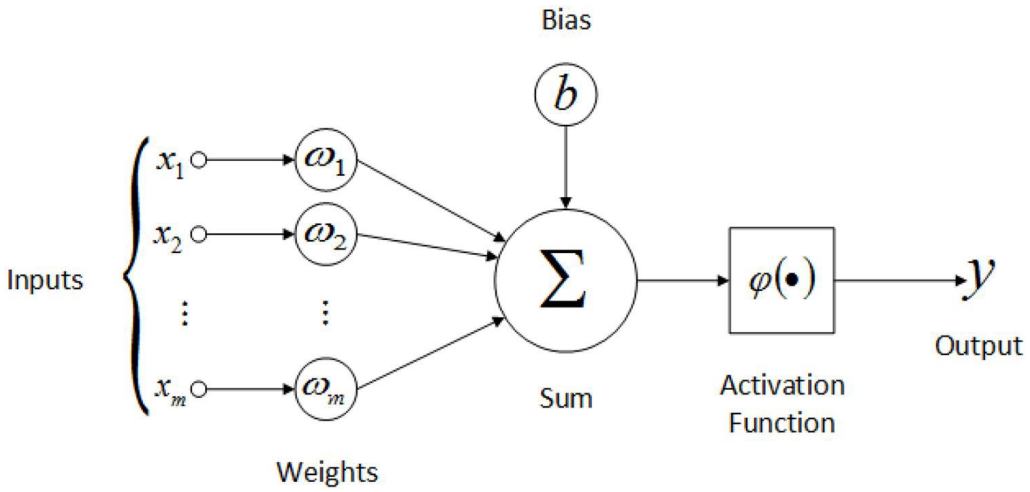


Figure 2.2: Sketch of a non-linear approximator

2.5 The Generalization Benchmark

Generalization in Reinforcement Learning aims to produce algorithms whose policies generalize well in novel, unseen environments at deployment time. Such policies have to show flexible behavior and robustness to distribution shifts. The problem is of fundamental importance because of the vast range of algorithms attaining incredible performance in simulation, but failing when slight changes are introduced in the testing environments. The current state of the field lags behind studies on generalization in other branches, mostly because for reinforcement learning the matter

is significantly more complicated. Generalization is a class of problems, rather than a specific problem. This introduction aims to categorize existing benchmarks as well as current methods, with the hope of providing critical discussion on the current state of the field.

2.5.1 Contextual Markov Decision Processes (CMDPs)

We refer to generalization as the problem sets called zero-shot policy transfer [10]. The policy trained on some environments is evaluated zero-shot on a collection of different environments. It follows that to be able to rigorously talk about generalization we need a way of talking about a collection of tasks, environments, or levels. One possible formalization is that of *Contextual Markov Decision Processes* (CMDP) [9], [8]. The reward function, transition function, initial state distribution, and emission function all take the context as input, the choice of context determines everything about the MDP apart from the action space, which we assume is fixed. The agent is trained on a subset of contexts C_{train} and tested on a disjoint subset C_{test}. There are at least two ways of comparing the generalization performance of models. One is to look at the absolute performance on evaluation tasks, and the other is to look at the generalization gap. A benchmark task is a combination of a choice of the **Context** and a suitable **Evaluation Protocol**.

2.5.2 Context

Within contexts, the literature mainly delineates two ways of generating CMPD: Procedural Content Generation (PCG) and Controllable Environments. Those differ in the controllability of the user. The former is completely defined by a seed value, meaning there is no direct control over the features of the environment, and the last allows more freedom by letting the user decide on individual components independently.

2.5.3 Evaluation Protocol

The evaluating protocols are defined by context efficiency. This is analogous to sample efficiency, where only a certain number of samples are allowed during training, but instead, we place restrictions on the number of contexts. As shown in Figure 2.3, there are three different regimes. The dots represent the training samples, while the colored squares different data distributions. The orange square represents interpolation, the yellow square represents combinatorial interpolation, the green square represents single-factor extrapolation, and the blue square represents extrapolation. Given any training protocol, testing samples can inhabit any other region in the grid. For clarity's sake, I'll illustrate the concept with the example taken from [14].

Consider a policy trained in a CMDP where the context set consists of values for friction and gravity strength. During training, the environments have either a friction coefficient between 0.5 and 1 but gravity fixed at 1, or gravity strength ranging between 0.5 and 1 but friction fixed at 1 (the light blue line in Fig. 2.3 C). Testing contexts that take friction and gravity values within the training distribution are full interpolation (e.g. $(f = 0.5, g = 1)$,$(f = 1, g = 1)$*), contexts that take values for friction and gravity which have been seen independently but not in combination are combinatorial interpolation (e.g.* $(f = 0.5, g = 0.5)$,$(f = 0.5, g = 0.9)$*, the yellow area), and contexts which take values for friction and gravity which are outside the seen ranges during training are full extrapolation (e.g.* $(f = 0.2, g = 0.5)$,$(f = 1.1, g = 1.5)$*, either the dark blue or green areas).*

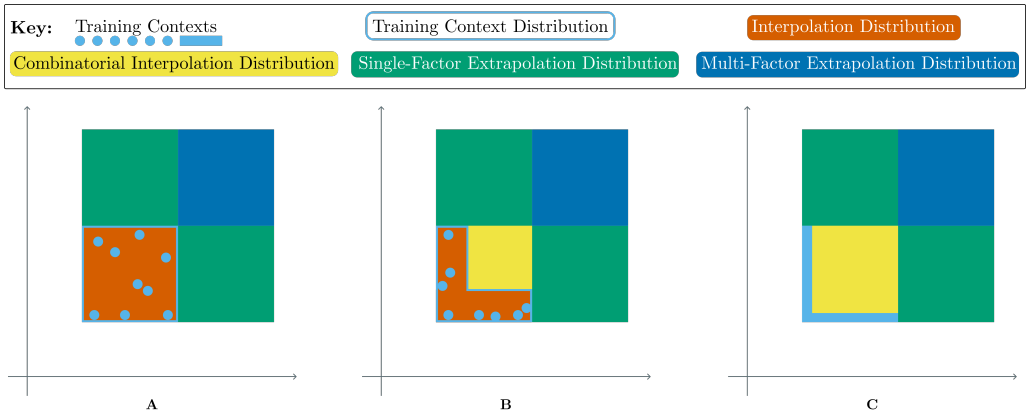


Figure 2.3: Evaluation Protocols

Given a formal definition of the generalization benchmark, we can move to classify methods that tackle generalization in RL.

1. **Increased similarity**
2. **Handle differences between training and testing environments**
3. **Target RL-specific issues or optimization improvements**

2.5.4 Generalization Methods

Those methods are summarized in Figure 2.4

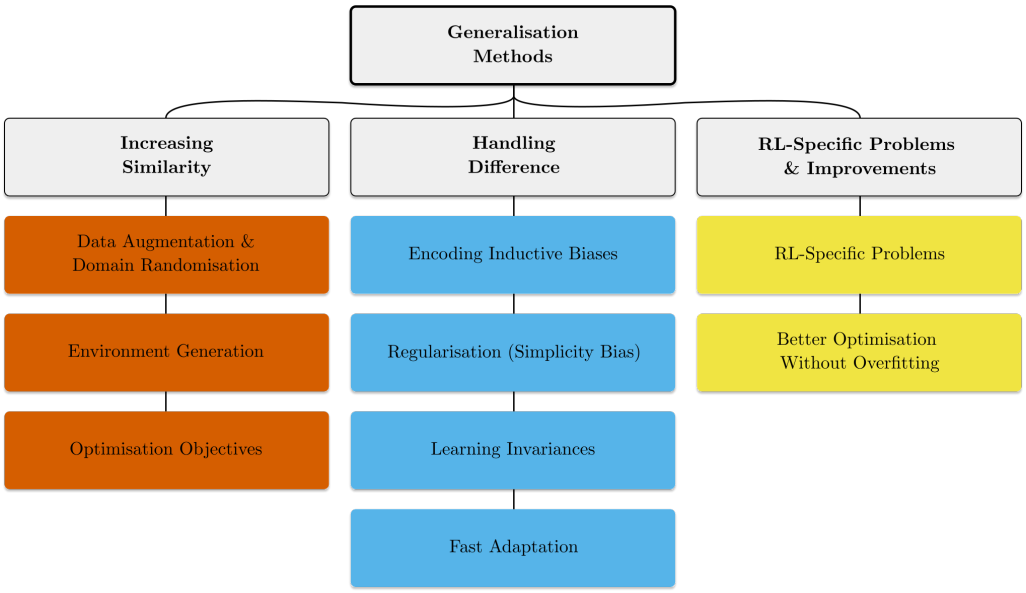


Figure 2.4: Generalization Methods

2.5.5 Increasing Similarity Between Training And Testing

The closer the testing environment is to the training environment, the smaller the generalization gap between the two. This method proposes to reduce the generalization gap, by designing the train setup to be as close as possible to the desired testing setup. In stark similarity with other machine learning paradigms, this objective can be reached by tackling the dataset via data augmentations or randomization, environment generation, the loss function, or the optimization of different objectives.

2.5.5.1 Data Augmentation and Domain Randomization

One interesting approach along these lines is Data Augmentation (DA) [25]. DA tries to increase similarity with the model using a dual approach: on the one end increases the interpolation similarity with the training set and on the other end it enforces the learning of invariances. In the literature, there are many examples of such training strategies, both tackling the data augmentation generation and the time a which the augmentation is applied. Additionally, instead of enriching the training dataset, another approach would be to increase the variability of the training environment itself. Such an ambitious goal can be achieved by Domain Randomization (DR) [22]. DR is an approach that requires full access to the parametrization of the model. It proposes to randomize the variables controlling the environment with the hope that the testing environment would be contained in such a rich training dataset. Both DR and DA increasingly augment or randomize the data, making optimization very difficult, and often this makes the methods much less sample-efficient.

2.5.5.2 Environment generation

Environment generation (EG) is the technique of designing new MDPs that are optimized to aid training [24]. In this context, levels of the game are fed in a lifelong learning fashion. The generated environment should get closer to the testing environment as the training proceeds.

2.5.5.3 The Optimization Objective

Works have tackled the optimization objective (implicitly or explicitly) of the environment. Changing the data distribution could be seen as implicitly changing the optimization objective, works along this line include [12]. Additionally, literature has focused on methods for robust AI, that deal with the generalization performance by assuming a worst-case optimization approach. In this context, the minimum performance is maximized across a vast range of environmental perturbations with the goal of improving generalization to unseen dynamics [1], [7].

2.5.6 Handling Differences Between Training And Testing

Other works approach the matter from a different perspective. Such research focuses on completely disregarding dissimilar features while focusing on common features. Work in this direction tackled the addition of indicative bias, regularization, learning invariances, and fast adaptation.

2.5.6.1 Exploring Inductive Biases

Bias addition is a technique that requires hand-engineering architectures or objectives to not rely on features that are expected to change. Such approaches tackle both the optimization objective, with the addition of regularization terms to the loss function [23], and the architectures that incorporate new modules functional to the specific bias [21].

2.5.6.2 Standard Regularization

Standard regularization is an easy, but powerful strategy when dealing with generalization. This is commonly motivated by a paraphrased Occam's razor: the simplest model will generalize the best. Task-agnostic regularisation has a wide range of applications and usually leverages techniques that discourage learning a more complex or flexible model, so as to avoid the risk of overfitting. Regularization techniques include augmentation, dropout, batch norm, L2 weight decay, policy entropy, and more.[6]

2.5.6.3 Learning Invariances

Another attempt to regularize data is that of encouraging learning invariances. For instance, effective representations should be invariant to differences in lighting, position, angles, and relative dimension. If the training set is rich enough, the variability within it might incorporate that of the testing set. Several approaches use multiple contexts to learn an invariant representation [3], [19]. Others try to learn behaviourally similar (similarity of short future action sequences) representations. [16].

2.5.6.4 Fast Adaptation

Fast adaptation is a method that aims to generalize by adapting online to the testing contexts. Approaches along this line, tackle inferring the type of testing context by representing environments in some higher-order embedding, sampling from such embedding, and then conditioning the policy [26]. Alternatively, approaches use hard-coded adaptation techniques which do not rely on explicitly inferring a context. [18].

2.5.7 RL-Specific Problems And Improvements

The methods above could be applied, with no loss of generalization, to a wide range of algorithms in supervised learning. Differences between reinforcement learning and such algorithms cannot be ignored.

2.5.7.1 RL-Specific Problems

In this section, I will present works that deal with specific problems of generalization for RL agents. Issues proper only to RL agents are the nonstationarity of the data distribution and the need for exploration. There have been some attempts to study how nonstationarity affects generalization, and they lean toward the result that policies learn features that do not generalize well, even if they achieve the same training performance. [11]

2.5.7.2 Better Optimization Without Overfitting

Several works aim at improving generalization performance without overfitting. One strategy is to adjust the training regime and architecture by separating the policy and value function and optimizing only the value function [5], which allows the value head to be optimized for longer while not causing the policy to overfit. Another angle on better optimization is the use of model-based RL (MBRL). Such approaches impose structure on the learning pipeline claiming that learning a model as an intermediate step can narrow down the set of possible value functions more than directly using the Bellman equation [2].

2.6 Biological Correlates of Reinforcement Learning

Reinforcement Learning is a biologically motivated algorithm on multiple levels. On a higher level, psychology and RL agents share the trial-and-error learning strategy. Additionally, model-based and model-free paradigms strongly resemble concepts of habitual and goal-directed behavior as explained in Section 2.6.1.6. On a lower level, reinforcement learning is the computational model of choice in the study of dopamine-production neurons, as explained in Section 2.6.2. Reinforcement learning mechanisms seem to underlie much of human and animal behavior. The motivation to shed light on its biological correlates is dual, on the one end, the similarities with the biological system are entrancing and the hope is that the interplay of machine learning and neuroscience will enable a better understanding of our neurological traits. On the other end, the

objective is that of generating machine learning algorithms able to get closer to the efficiency and flexibility of the human brain.

2.6.1 Psychology

The similarity between reinforcement learning algorithms and human behavior is striking, but not surprising, as its basic algorithms were inspired by psychological theories. Of course, RL does not have the ambition to exhaustively model human behavior, though is worth exploring their similarities to encourage a spirit of collaboration between the two branches.

2.6.1.1 Classical, or Pavlovian, Conditioning and Instrumental Conditioning

The distinction in reinforcement learning between algorithms for prediction and algorithms for control parallels animal learning theory's distinction between classical, or Pavlovian, conditioning and instrumental conditioning. The key difference between instrumental and classical conditioning experiments is that in the former the reinforcing stimulus is contingent upon the animal's behavior, whereas in the latter it is not. Both conditionings are widespread in the animal world, and have been modeled through the years by appropriate RL algorithms.

2.6.1.2 Classical Conditioning: TD algorithm and the influence of Rescorla–Wagner

Under natural conditions, the normal animal must respond not only to stimuli that themselves bring immediate benefit or harm but also to other physical or chemical agencies. Connecting new stimuli to innate reflexes in this way is called classical, or Pavlovian, conditioning. Evolutionarily speaking, natural stimuli "unconditioned stimuli" (USs), trigger inborn responses "unconditioned responses" (URs). The animal can be taught to react similarly through "conditioned stimuli" (CSs), via classical conditioning, in response to predictive stimuli "conditioned responses" (CRs). A common example is that of a dog responding to the command informing that food is ready. The dog receives the command (CS), and responds by running toward the owner (CR), because it was trained to recognize that such a command implies food being served. Models of classical conditioning have historically struggled to represent two phenomena: higher-order conditioning and blocking. Higher-order conditioning is a further level of conditioning that is attained when an intermediate CS is introduced in between the initial CS and the CR. In the dog example, the animal might first learn to associate a bell with food (first-order conditioning), but then learn to associate a light with the bell (second-order conditioning). Blocking is the phenomenon in which a previously-learned thought process prevents or delays the learning and conditioning of a new behavior. If the dog learns that the bell is associated with the food and then is presented with a combined stimulus of the bell and a whistle, it will hardly learn that the whistle by itself is associated with food. TD model of classical conditioning is an example of reinforcement learning principles accounting for animal learning behavior. This model generalizes the influential Rescorla–Wagner model by including the temporal dimension where events within individual trials influence learning, and it provides an account of second-order conditioning and blocking.

2.6.1.3 Instrumental Conditioning: Law of Effect and Learning by Trial and Error

In instrumental conditioning experiments, learning depends on the consequences of behavior: the delivery of a reinforcing stimulus is contingent on what the animal does. Learning by trial and error is at the base of the instrumental conditioning aspect of reinforcement learning. Such a formulation is strictly related to The Law of Effect principle developed by Edward Thorndike. This law postulates that responses closely followed by satisfaction become firmly attached to the situation and therefore more likely to reoccur when the situation is repeated. Essential features of reinforcement learning correspond to those theorized by the Law of Effect. In particular, animal learning

is selective, meaning that they try alternatives and select among them by comparing their consequences resembling reward maximization. Moreover, it is associative as the alternatives found by selection are associated with particular situations, or states, to form the agent's policy, resembling value functions state-action associations. Additionally, there are other aspects of instrumental conditioning resembling reinforcement learning strategies, one example being "Shaping". The shaping principle for animal learning claims that defining curricula of increasing difficulty aids learning. In the case of reward sparsity or inaccessibility, reinforcement learning agents benefit as well from starting with an easier problem and incrementally increasing its complexity.

2.6.1.4 Delayed Reinforcement: Eligibility Traces, Value Functions

One of the major criticism directed toward *The Law of Effect* is that it requires a backward effect on connections, most researchers could not conceive of how the present could affect something that was in the past. Additionally, such concern is amplified by the fact that learning can even occur when there is a considerable delay between an action and the consequent reward or penalty. This is called the problem of Delayed Reinforcement. Reinforcement learning algorithms present two basic mechanisms for addressing the problem of delayed reinforcement: eligibility traces and value functions learned via TD algorithms. Eligibility Traces (ET) in particular show strong correlates in the nervous system. ET are traces left in the nervous system that persists for some time after the stimulus ends. The temporal gap between the CS offset and the US onset present during stimulus traces makes learning possible.

2.6.1.5 Model Based Reinforcement Learning and Cognitive Maps

Experiments conducted in the mid-20th century ratified the advent of model-based learning over the simplest model-free way as the representation of choice for human behavior because RL algorithms using such a paradigm appeared to have elements in common with what psychologists call cognitive maps. Cognitive maps are representations of the environment learned in the absence of rewards or penalties. Conducted experiments purported to demonstrate the ability of animals to learn cognitive maps as alternatives to, or as additions to, state–action associations, and later use them to guide behavior, especially when the environment changes unexpectedly.

2.6.1.6 Model-Free and Model-Based Algorithms vs. Habitual and Goal-Directed Behavior

Reinforcement learning's distinction between model-free and model-based algorithms corresponds to the distinction in psychology between habitual and goal-directed behavior. Model-free algorithms make decisions by accessing information that has been stored, whereas model-based methods plan ahead using a model of the agent's environment. According to most computational neuroscientists, animals use both model-free and model-based processes. Model-based systems' planning process chains together short-term predictions. Such predictions are more accurate in the early epochs when less experience is available compared to the long-term predictions of the model-free process. Instead, when extensive experience is available, the model-free process becomes more accurate because planning requires simplifications and shortcuts to be feasible. According to this idea, one would expect a shift from goal-directed behavior to habitual behavior as more experience accumulates.

2.6.2 Neuroscience

Research in the field of computational neuroscience is revealing striking correspondences between the brain's reward system and the theory of reinforcement learning. Neural pathways of reward involved in the reward systems are complex and relatively poorly understood. One way forward could be that of modeling the brain using architectures and algorithms that resemble its mechanisms. With this goal in mind, many researches have tried to reverse engineer the brain at a biological level, by comparing it with state-of-the-art machine learning. Among AI algorithms, reinforcement learning is the one with more striking similarities, as it is able to approximately explain the dopamine cycle and the dopamine-production neurons.

2.6.2.1 The Reward Prediction Error Hypothesis of Dopamine

According to the theory of reward prediction error (RPE), neurons producing dopamine, a neurotransmitter essential in mammals for reward-related learning and behavior, leverage their phasic activity to deliver an error between an old and a new estimate of expected future reward. Researchers in the late 90s recognized parallels between TD errors and the activity of such neurons. Among the similarities, if an earlier cue precedes the expected one, that has already acquired the predicted value, the dopamine is produced in correspondence with the earlier cue, and if, the predicted rewarding event is omitted, a dopamine neuron's response decreases below the baseline shortly after the expected time of the rewarding event. Therefore, phasic responses of dopamine neurons can be considered reinforcement signals.

2.6.2.2 Actor–Critic Algorithm as a Model of the Brain

A prominent hypothesis is that the brain implements an actor-critic algorithm. In the vanilla actor-critic architecture, the 'actor' is the component that learns policies, and the 'critic' is the component that evaluates whatever policy is learned by the actor and 'criticizes' it. The actor and the critic are implemented by ANNs based on the policy-gradient actor–critic method, as shown in Figure 2.5. The two structures in the brain that play critical roles in reward-based learning, and may function respectively like an actor and a critic are the dorsal and ventral subdivisions of the striatum. The TD error appears to be critical for modulating synaptic plasticity in both structures, but its effect on learning is different for each component. For the actor, the TD error is responsible for updating the action probabilities in order to reach higher-valued states, and for the critic, such an error indicates the direction and magnitude in which to change the parameters of the value function in order to improve its predictive accuracy.

2.6.2.3 Reward-Modulated Spike-Timing-Dependent Plasticity

Lending plausibility to the existence of actor-like synaptic plasticity in the brain, neuroscientists have discovered a form of Hebbian plasticity called spike-timing-dependent plasticity (STDP).In STDP, the relative timing of pre-and postsynaptic activity determines the direction of synaptic change. In particular, the synaptic strength is increased if the spike arrives shortly before the postsynaptic neuron fires. If the timing relation is reversed, with a presynaptic spike arriving shortly after the postsynaptic neuron fires, then the strength of the synapse decreases. These experiments point to the existence of contingent eligibility traces having prolonged time courses.

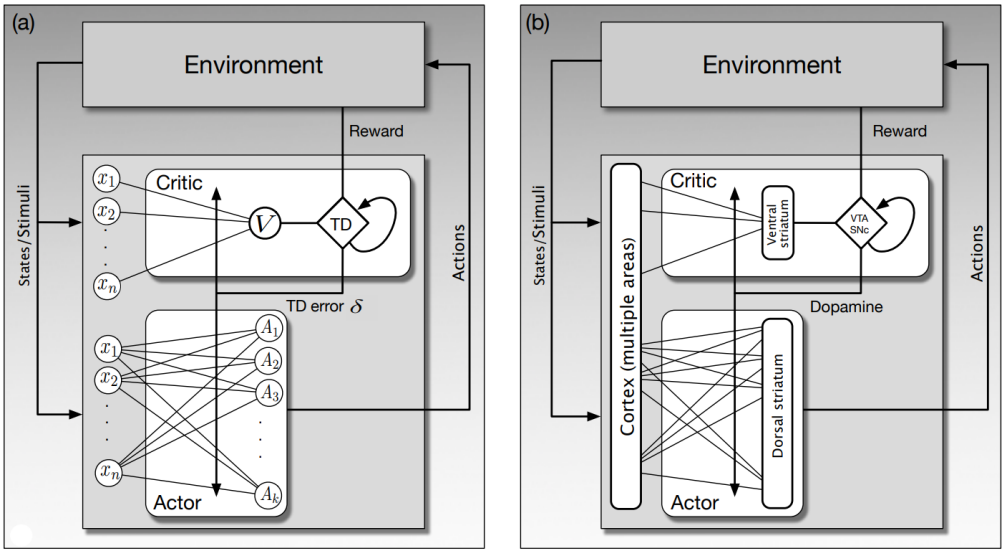


Figure 2.5: Actor-Critic Architecture

2.6.2.4 Hedonistic Neurons

According to Klopf's hypothesis of the "hedonistic neuron", individual neurons seek to maximize the difference between synaptic inputs considered as carrying positive rewards and those considered as carrying negative rewards. Such distinction is based on the consequences of their own action potentials, just like reinforcement learning agents. In other words, under this hypothesis, individual neurons can be trained with response-contingent reinforcement, just like an animal can be trained in an instrumental conditioning task. In order to be modified (trained), synapses have to be temporarily marked as eligible. This is known as the eligibility theory and it states that a synapse's legacy is modifiable only if a reinforcing signal arrives while the synapse is eligible.

2.6.2.5 Collective Reinforcement Learning

Changes in the legacies of the synapses of all of these units depend on the reinforcement signal. Each individual neuron can be considered a reinforcement learning agent that aims to improve the consequences of its own action potentials. Under such formulation, the situation can be modeled as a team problem. The field of multi-agent reinforcement learning deals with optimizing performance when learning by populations of reinforcement learning agents. If each team member uses a sufficiently capable learning algorithm, the neuron population can learn collectively to improve the overall performance as evaluated by the globally-broadcasted reinforcement signal, regardless of the lack of communication among team members. Such an approach provides a neurally plausible alternative to the widely-used error-backpropagation method.

2.6.3 Model-Free or Model-Based?

Reinforcement learning's distinction between model-free and model-based algorithms is proving insights into neural bases of habitual and goal-directed learning and decision-making. Research so far points to the idea that there is no unifying nature proper of the entire brain, likely there are some brain regions more involved in one type of process than the other. Nevertheless, the picture remains unclear because our brain lacks neat separation between model-free and model-based processes. Some examples of regions that appear to follow the model-based paradigm are the dorsomedial striatum, the prefrontal cortex, and the hippocampus, while an example of model-free processes is the dorsolateral striatum.

Chapter 3

Methods

3.1 Mathematical Problem Statement

We aim to add noise directly into the transition function (TF) of the Bellman Equation (3.1).

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} T(s', r|a, s) \left[r + \gamma v_\pi(s')\right]. \tag{3.1}$$

The final goal is to control the noise we add to the transition function. *How do we add the noise?* The transition function could be rewritten such that the noise is directly added to the level of the single agents, like in Equation 3.4.

$$T(X_t|X_{t-1}, a) = \sum_{X^*} T(X_t, X^*|X_{t-1}, a_{pacman}, a_{ghost}) \tag{3.2}$$

$$T(X_t|X_{t-1}, a) = \sum_{X^*} T(X_t|X^*, X_{t-1}, a_{pacman}, a_{ghost}) T(X^*|X_{t-1}, a_{pacman}, a_{ghost}) \tag{3.3}$$

$$P(X_t|X_{t-1}, a) = \sum_{X^*} T(X_t|X^*, a_{ghost}) T(X^*|X_{t-1}, a_{pacman}) \tag{3.4}$$

The Equation 3.4 is more interpretable by humans because we view the agents of the game as distinct. Though, there is no guarantee that the noise added to the agent would reflect the level of noise we want to add to the transition function (Equation 3.2). The parameterization of the final noise is not known even in the case of a simple Gaussian distribution, it becomes more and more untractable as we increase the number of agents. Therefore we directly add the noise to the transition function of Equation 3.2.

3.2 A Unifying Nomenclature for Atari Games

Components of the game can be classified into 4 categories according to their interaction with the environment.

- **Dynamic Controllable**: dynamic elements controlled by the agent's learned policy
- **Dynamic non Controllable**: dynamic elements controlled by a previously defined distribution independent from the agent's.
- **Static interactable**: static elements that interact with dynamic agents and have few discrete states that change according to the interaction.
- **Static non-interactable**: static elements that interact with dynamic agents, but never change according to the interaction.

Such nomenclature is natural and directly reflected in the Transition Function and Reward Function. For instance, static non-interactable elements can be completely controlled by the TF. Adding a wall in the grid is trivial, it just means setting all the probabilities of converging to state s' to 0, namely $\sum_s \sum_a P(s'|s,a) = 0$. Similarly, static intractable are usually only reflected in the Reward Function.

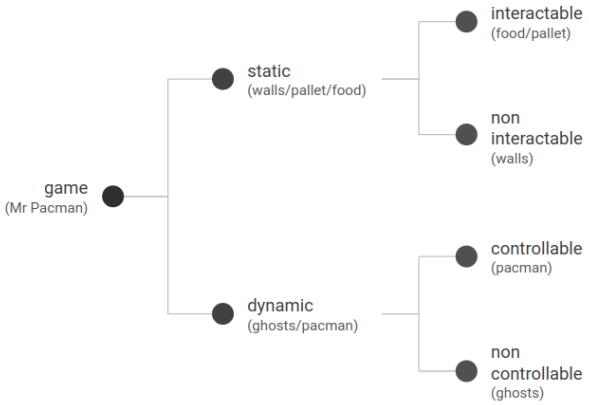


Figure 3.1: Agents' nomenclature

In Figure 3.2, few Atari games are classified according to such nomenclature.

game	dynamic controllable	dynamic uncontrollable	static interactable	static non-interactable
Adventure	adventurer	outmanoeuvre dragons	lives, time, sword, a bridge, c	map
Air Raid	spaceship	flying saucers	lives/flying bullets	map
Alien	human/sword	aliens	lives, time, eggs/pulsar	map
Amidar	gorilla	Warriors-chickens	lives, time, corner-boxes	map
Assault	vehicle	small drones-mother ship	lives	map
Asterix	Asterix	lyres	lives, objects, flying bullets	map
Asteroids	spaceship	asteroids/flying saucer	lives	map
Atlantis	defender	enemies	lives, Atlantis' installations, f	map
Blank Heist	bank robber's getaway car/sti	police	lives, gas tank	map
Battle Zone	tank	enemy vehicles	lives	map
Beam Rider	spaceship	enemy ships/space debris	lives, flying bullets	map
Berzek	human	evil robots	lives, time	map
Bowling	ball		pins	map
Boxing	fighter	adversarial	lives	map
Breakout	paddle/ball		lives, blocks	map
Carnival	gun	targets/chickens	flying bullets	map
Centipede	magic spells	centipede/spiders/fleas	wands/mashrooms	map
Chopper Comr	elicopter	enemy aircraft	trunk convoys/ flying bullets	map
Crazy Climber	climber	obstacles	lives	map
Defender	spaceship	alien ships/humans	lives/ flying bullets	map
Demon Attack	human	waves of demons	reserve bunkers	map
Double Dunk	player/ball	adversarial player	sets	map

26

Figure 3.2: Atari subdivided with nomenclature

3.3 Chosen Environment and Grids

These are the environments of choice, with the respective grids. In this first paragraph, I have listed them and briefly explained the main differences with standard implementations.

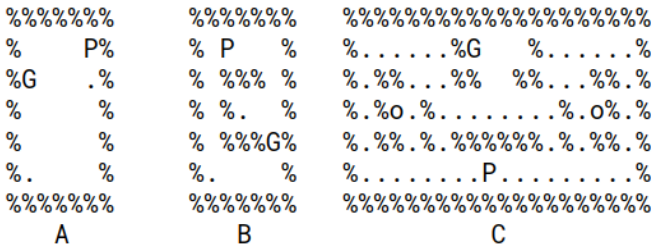


Figure 3.3: Grids layouts for Pacman. A: *v2*, B: *v3*, C: *SmallClassic*

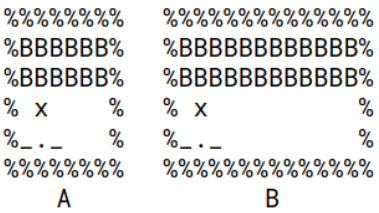


Figure 3.4: Grids layouts for Breakout. A: *small*, B: *medium*

3.3.1 Pacman

3.3.1.1 State Space representation

In *Pacman*, the dynamic controllable agent (namely the Pacman P), moves in an environment made of static non-interactable elements such as walls ($\%$), static interactable elements, such as food (.), pallets (o) and dynamic uncontrolled agents, such as the ghosts (G), as shown in Figure 3.3. The purpose of Pacman is to eat all the food in the minimum possible time. With the purpose of achieving such a goal, the Pacman interacts with the environment, occasionally eating pallets, which make it able to eat ghosts, and consuming the food it encounters on the way.

3.3.1.2 Maintaining the markovian property

To assure a correct factorization of the transition function into $\sum_{s'} P(s'|s,a)$, we need to ensure the *Markovian Property*, meaning that the current state s' conditioned on the previous state s, and previous action a, has to be independent on past state-action pairs. The initial game configuration allows different ghost types. Among others, the directional ghost constitutes a problem. The directionality is not embedded in the state space representation, as it only considers one possible representation for all the ghosts, as shown in Figure 3.3. Namely, there is no way of inferring which is the current directionality of the ghost from the current state representation. We could introduce different elements to encode the current ghost directionality, though it seemed to the authors that it would have unreasonably increased the complexity of the game. Therefore, in our implementation, the ghost loses this constraint, as it is able to stir in any direction.

3.3.2 Breakout - still in implementation

3.3.2.1 State Space representation

In *Breakout*, the dynamic controllable agent (namely the Bar ___) directs the ball toward the static interactable blocks B, as shown in Figure 3.4. The purpose of the game is to destroy all the blocks in the minimum possible time. The target x is just a way of representing the directionality of the ball in the state space. (Discussion in the following paragraph)

3.3.2.2 Maintaining the markovian property

For the correct factorization of the transition function, we again need to ensure the *Markovian Property*. The nature of the game requires preserving the directionality of the ball (the ball can only continue moving in its direction or invert its direction when it encounters a wall or a block). The directionality cannot be inferred by the current definition of the state s. There are several ways to account for the problem, one idea is to consider the motion of the ball until it encounters an impediment as part of the same time segment and represent the directionality using a helper target pointing in the desired direction, like in Figure 3.4.

3.4 A Unifying Nomenclature for States

Each game is composed of agents, objects, and rules. The rules of the game limit attainable positions in the grid. With the term illegal I describe an entity that somehow breaks the game rules. States are completely defined by the agents' positions and the objects' positions, while steps are individual agent's movements. There are some states in the game that are never attainable after any number of legal steps, due to game rules, and such states are going to be defined as illegal. *Illegal States* are composed of agent positions that can be individually attained, but never simultaneously. The states that include illegal positions of any of the agents are defined as *Impossible States* (those states include the ones where agents are on the walls) and not considered. All attainable states are named *Legal States*. States with no legal successor are named terminal states.

3.5 Frameworks

From now on I'll be explaining the framework from Pacman's perspective, keeping in mind that all that will be said also applies to Breakout and any other game coded under the constraints expressed above. The implementation of Pacman is based on the Pacman Berkeley codebase.

3.5.1 Transition function is not exposed in existing implementations

The transition function is not exposed in the existing implementation but implicitly queried through a chain of function calls. Such an implementation makes systematic noise addition impossible. In my framework, the transition function is pre-computed and used to control transitions between states.

3.5.2 Computing the transition function of legal states

The transition function of the game is the function mapping each state and action to the successor legal state. Pacman is deterministic in its normal configuration meaning that the probability of going from any state to its legal successor given any legal action is always 1. Such transitions are computable by considering all the paths generated by all the possible combinations of actions of Pacman and ghosts. We want to compute the transition function for legal states. All the possible combinations of actions that originate from a certain state and terminate in the legal successors of such state need to be stored. Once a state is processed and its legal successor computed such a state should not be processed anymore in the future, even though, with all likelihood, it is going to appear as a legal successor of some other legal state. The solution we implemented is creating a graph where each node is the agent and each edge is the probability of transitioning to an intermediate state, (where only that agent has moved), given the agent's legal action, like in Figure 3.5. The legal states are found at the end of an entire agents' rollout and stored in the transition matrix with the action being the initial Pacman action and the value the product of the probabilities on the edges. Each successor state is enqueued (to be further processed) only if it is not yet stored in the dictionary containing all visited states in a Depth-First-Search (DFS) fashion.

If there are more successors (meaning we have reached a terminal state), nothing is enqueued. The process stops when there are no more states in the queue. This accounts to play any possible playable game. For the purpose of generating a 3D matrix containing all possible combinations of state-next state-Pacman action, states need to be enumerated.

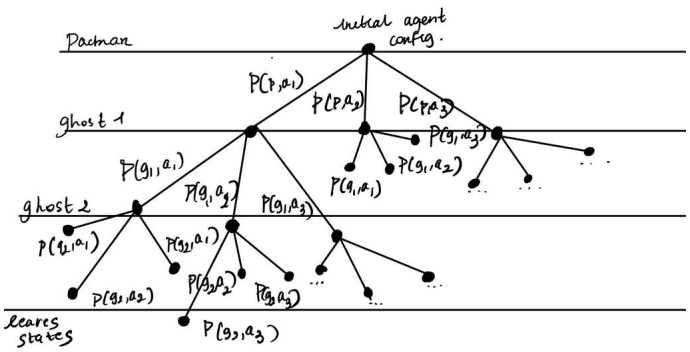


Figure 3.5: Sketch of the tree for transition matrix computation

3.5.3 Hash Function of the state space

To enumerate states, one trivial solution would be to create a dictionary associating the stringified version of the state space to any natural number. The state is defined by the Pacman, the ghosts, the food, and the pallets, the transition function is independent of the food and pallet configuration, as transitioning from state s to state s' retains the same probability in the presence or absence of the food in the transitioning state. Additionally, given the previous state food configuration, and Pacman's action, we are able to completely define the configuration of the successor state, in agreement with the markovian property. Following such a thought process, we created a hash function efficiently associating the position of the ghost and the Pacman to their state representation. The hash function takes as input the grid coordinate position of the Pacman and the ghosts (number from 1 to N squares in the grid) and computes the number in basis 10, as shown in Figure 3.6.

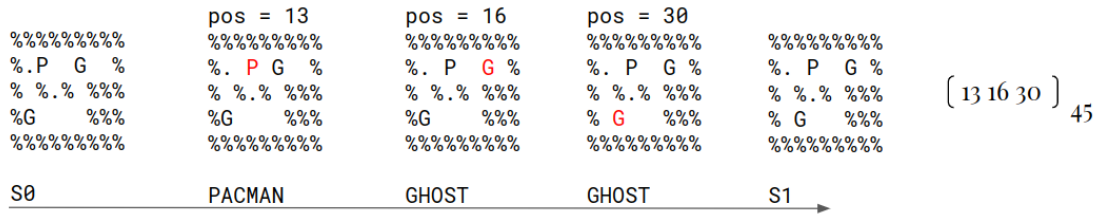


Figure 3.6: Hashmap computation

3.5.4 Compressing the transition function to store it in memory

The transition function has to be pre-computed and stored in memory. Though, the computed legal transition is the only non-zero entry of the transition matrix, making it very sparse. This would require allocating a lot of space and quickly becomes infeasible. The matrix is compressed in a dictionary storing only interesting information having as keys the hash values of states and as values the non-zero probabilities.

3.5.5 Adding distributed noise

Distributed noise is the first step in our project. The noise has to be introduced to every possible transition prior to the start of the game. Precomputing and storing the transition matrix is too

computationally expensive. Instead, we compute slices of the transition matrix on the fly and fix the seed for each computation to ensure reproducibility. The numbers are drawn from various distributions (Gaussian, Poisson) and randomly generated using CUDA parallel computing. The seeds are computed by multiplying a seed mesher (time at the start of the execution) by the hash value of the state and summing the chosen action, as to ensure the uniqueness of the seed for each layer of the matrix. With such an implementation, each state-action pair has a unique associated layer making up the final 3D shape of the matrix, the noise is added to the such layer and then all values are re-normalized.

3.5.6 Additional Problems in adding noise

We need to be careful in the noise addition: the probability of the legal action to be taken when the distributed noise is added goes to zero as the number of states increases, therefore it requires proper normalization. Let's say that I sample n states from a gaussian distribution with mean 0 and standard deviation 1, then I add the legal states probability and re-normalize. Legal states would have a prob of $\frac{1+noise}{n_{states}} \to 0, n_{states} \to \infty$, as shown in Equation 3.9. Therefore, we need to multiply the 1 by the n_{states}. Another important thing to consider is the number of agents. The percentage of illegal states out of the total number of states increases exponentially with the agent's number, therefore the total probability to end up in an illegal state increases exponentially. Let's say that there are i impossible positions,n illegal positions and m legal positions in the grid, the probability of an agent (Pacman or ghost) ending up in an illegal position given a uniform distribution over the positions would be $\frac{n}{n+m}$. Mind though that only one agent needs to be in an illegal position for the entire state to be illegal. Therefore the probability of ending up in an illegal state, given 2 agents is $1 - (\frac{n}{n+m})^2$. From an illegal state the probability. In the single-agent case:

$$P(s = legal) = n_{legal} * (\frac{1 + noise}{n_{states}}) \tag{3.5}$$

$$P(s = illegal) = n_{illegal} * (\frac{noise}{n_{states}}) \tag{3.6}$$

$$n_{legal} = 4 \tag{3.7}$$

$$n_{illegal} = n_{states} - 4 \tag{3.8}$$

$$P(s = legal) = 4 * (\frac{1 + noise}{n_{states}}) \propto \frac{1}{n_{states}} \tag{3.9}$$

$$P(s = illegal) = (n_{states} - 4)(\frac{noise}{n_{states}}) \propto 1 \tag{3.10}$$

To maintain a fear comparison among grid, which is independent of n_{agents} and n_{states}, we need to multiply the probability of the legal state to occur by $(n_{states})^{n_{agents}}$.

3.6 The Agents

We test different reinforcement learning agents with the purpose of comparing their performance in a structured way. The chosen agents are classical tabular Q-Learning with various exploration strategies (the experiment section reports only the performance of Boltzmann and e-greedy exploration strategies), deep Q-Learning (DQN), and humans.

3.7 Pipeline

There are some significant changes in the original pipeline of the Pacman Berkeley repository. The dictionary containing all legal transitions is pre-computed at the beginning of each new game, and the noise is stored either before the game starts (for relatively small state spaces) or computed on the fly as explained above. Then the agent queries the noisy transition function for possible actions. The probability of reaching state s', given current state s through action a is not 1 anymore, the value is slightly lower and there is a nonzero probability of reaching any other state in the grid. Finally, the next state is decided by performing a choice among all possible transitions weighted by their respective probabilities. After the next state has been chosen, the framework computes the state representation to be fed back to the agent for the next query and so on. Such state representation varies according to the nature of the agent itself: tabular Q-Learning needs the stringified version of the environment, while DQN requires a snapshot of the newly reached state configuration.

3.8 Visualizations for Generalization and Learnability

We use the proposed framework to study generalization and learnability for reinforcement learning agents. Generalization is defined as the ability to adapt to new, unseen environments, in this context we use the first-shot paradigm, as explained in the theory overview. While learnability is defined as the ability to learn a new environment starting from scratch.

3.8.1 Reward Graphs

Reward graphs (RG) are probably the most informative visualization type. They plot the reward obtained averaged across different agents as a function of the number of epochs the agent was trained on. RGs is essential to compare the difference in performance with the increase in noise. The first value of a RG is the random agent performance as the agent is tested on an environment it has never seen before. The upper bound is the agent trained on the non-noisy version of the environment and the lower bound is the random agent performance.

3.8.2 GIFs

GIFs are visualizations of the agent's learned policy. Those are useful to see how the agent's performance increases through the epochs, and what the agent learns.

3.8.3 Random Walk Agents

Random walk graphs are useful to visualize the random walk performance of the agent as a function of the noise injected into the environment. They are an important baseline of agent performance.

3.8.4 Heatmaps

Heatmaps visualize the occupancy of each position in the grid. They are computed by counting the times the agent lands in a specific position and normalizing such values.

Chapter 4

Experiments and Results

4.1 Experiments

4.1.1 First Experiment

The first experiment aims to compare generalization under the proposed benchmark for humans and machines. The chosen type of noise is drawn i.i.d. from a Gaussian distribution parametrized by the mean and the standard deviation. Machines and humans are tested under the same noise conditions: 10 levels equally spaced between 0 and 1 for both mean and standard deviation, resulting overall in 20 noise levels. RL agents are trained until convergence and then tested in a first-shot paradigm in a noisy environment. The purpose of the experiment is to fairly compare humans and machines, therefore we need similar performances for the nonnoisy environment. The chosen grids are smallClassic and v3, shown in Figure 4.1.

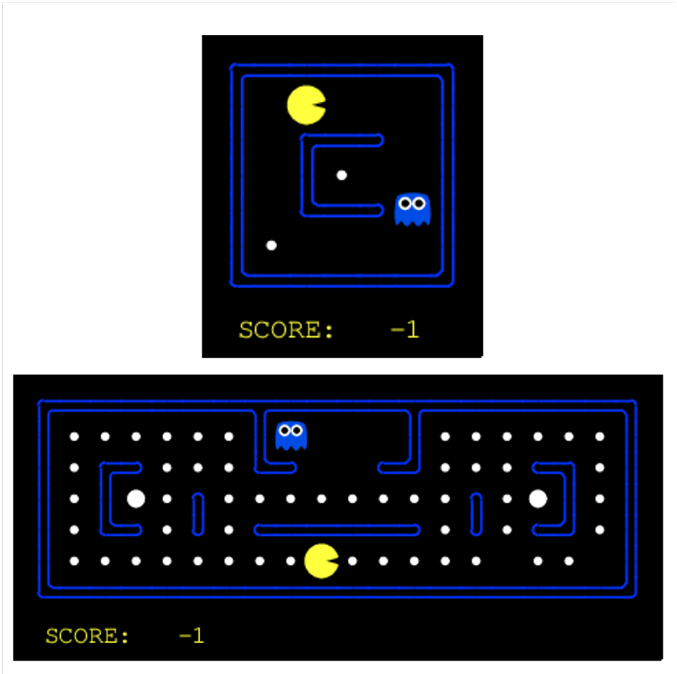


Figure 4.1: *SmallClassic* and *v3*

4.1.1.1 Machines

Boltzmann Q-learning and DQN are trained until convergence on the chosen grids and then tested in noisy environments. The performance of agents is variable, therefore, to attain statistical signif-

icance 200 agents are averaged for all noise levels each playing 1000 games. Tabular Boltzmann Q-Learning performance is knowingly poor on *smallClassic*. Since we want to match the non-noisy performance, we decide to remove *smallClassic*, and only test the agent on *v3*. *SmallClassic* is maintained for DQN.

4.1.1.2 Humans

With the purpose of roughly estimating human abilities in Pacman, before the start of each trial humans are tested on a nonoisy version of both grids. Only humans attaining a certain threshold are considered further in the study. Those represent the "experts", namely they are considered pairs of trained agents. Experts not necessarily needed to pass the test for both grids, if their performance was good enough for only one of the two grids, they were considered trained for that one, while results for the other were discarded. The practice trial consists of 10 games for each grid. After such a trial, the experiment begins: each human was required to play the noisy version of Pacman for about 30-40 minutes, concluding approximately half of the noise levels. Results are averaged among 20 subjects.

The results of the experiment are reported in Figures 4.2, 4.3.

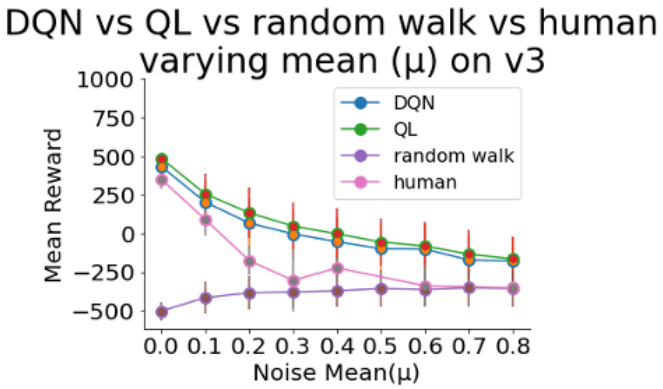


Figure 4.2: First Experiment results: humans vs. machines performance on *smallClassic*

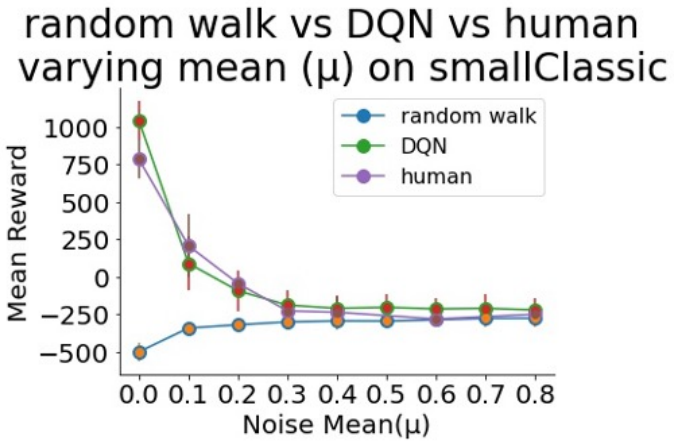


Figure 4.3: First Experiment results: humans vs. machines performance on *v3*

4.1.2 Second Experiment

The purpose of the second experiment is to test learnability and generalization in RL agents for different levels of noise. In this case, we wish to visualize the entire training process to observe the learning curves of agents. This requires different metrics of evaluation. Each agent is trained for a total of N games: every $n_{training}$ games it is tested on $n_{testing}$ games and the performance is averaged and stored. This process is then repeated for 500 agents for statistical significance. For generalization the $n_{testing}$ games are evaluated for every noise level, while for learnability the agent is trained and tested in the noisy environments. The chosen type of noise is drawn i.i.d. from a Gaussian distribution parametrized by the mean and the standard deviation. Machines are tested under the same noise conditions: 10 noise levels equally spaced between 0 and 1 for both mean and standard deviation, resulting overall in 20 noise levels. All tabular Q-Learning algorithms are trained and tested on blocks of 10 games for a total of 1000 training games, while DQN is trained for 40, tested on 10, for a total of 4000 training games.

Results are shown in Figures 4.4, 4.5, 4.6, 4.7.

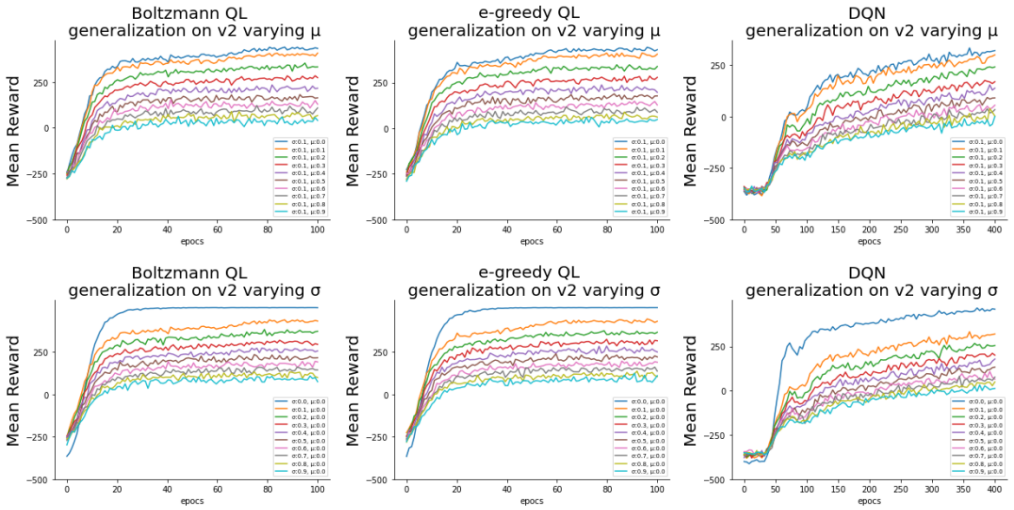


Figure 4.4: Generalization on *v2*

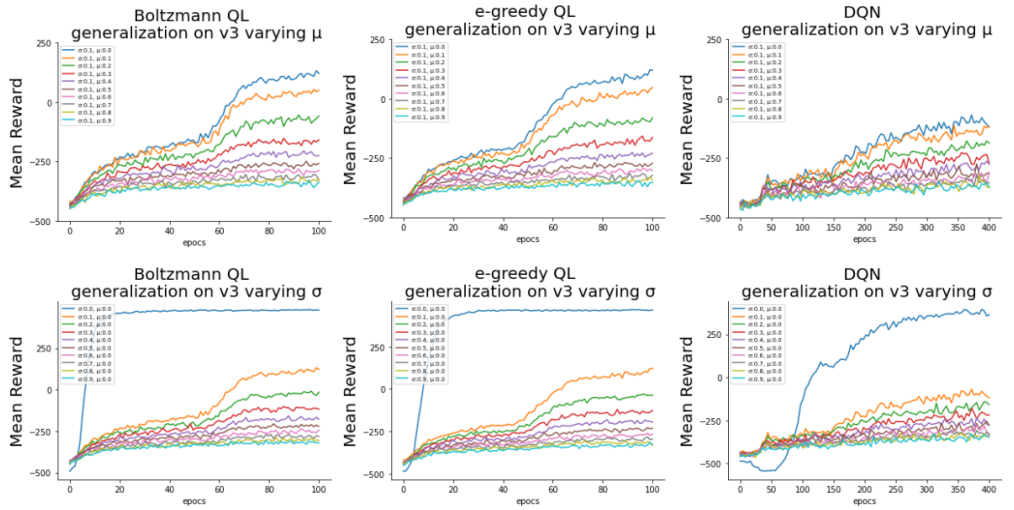


Figure 4.5: Generalization on *v3*

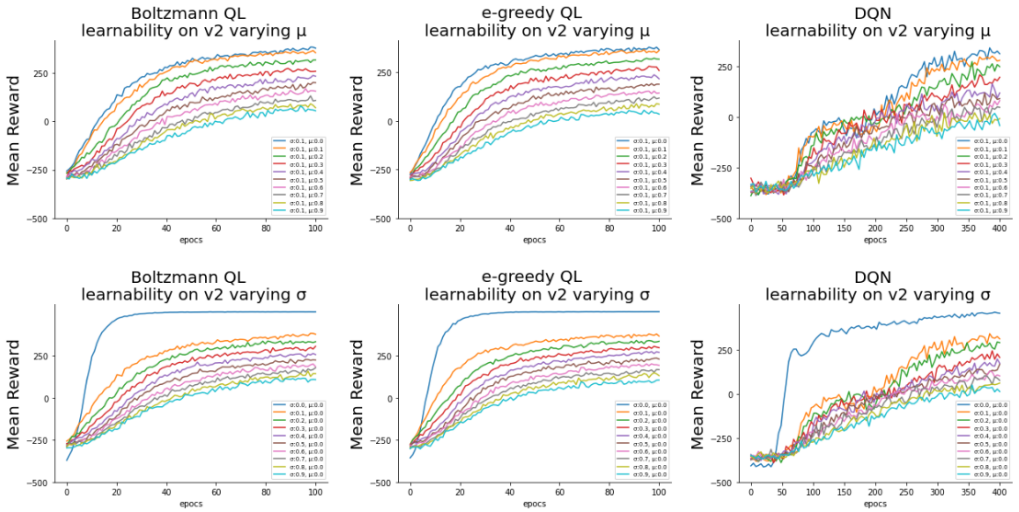


Figure 4.6: Learnability on *v2*

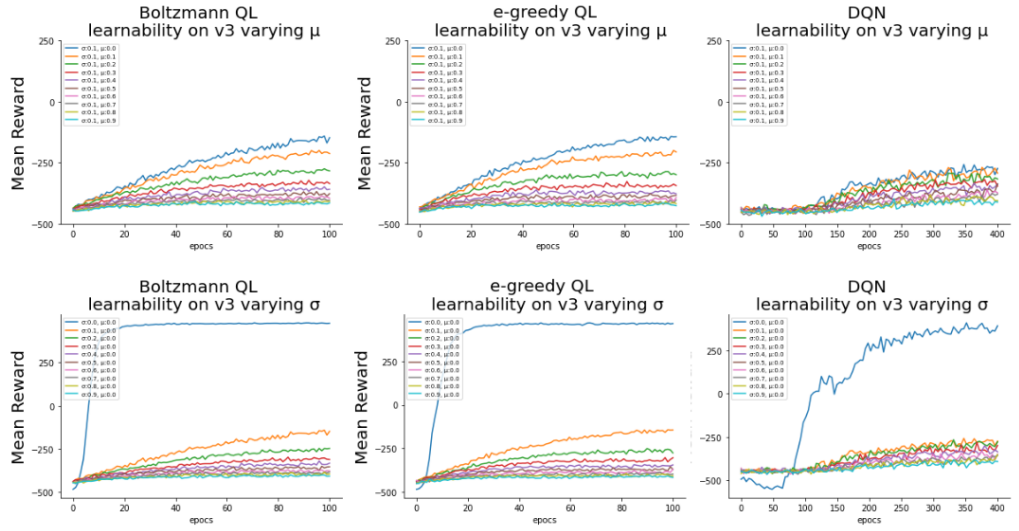


Figure 4.7: Learnability on *v3*

Chapter 5

Discussion

5.1 First Experiment

The first result is extremely interesting. Humans perform worse than RL agents under the defined benchmark. The gap between humans and machines is much more significant when testing on *v3* than *smallClassic* (Figures 4.2, 4.3). Such a result seems to suggest that the performance of humans improves with the size of the grid. This is not surprising since when Pacman teleports human response is delayed by a certain reaction time, and if the position of Pacman is too close to one of the ghosts the action might result in the death of the agent. Indeed, the probability of the ghost and the Pacman attaining contiguous positions increases when the state space dimensionality decreases. In future experiments, it might be interesting to remove the reaction time variable and see if the gap still persists. In such a case, a more interesting hypothesis might come into play. For example, humans might need to re-orientate in space, or additionally, the working memory information might need to be refreshed in bigger grids. Future works might design new experiments that eliminate the reaction time component and focus on new, diverse causes.

5.2 Second Experiment

There are many interesting results that can be observed for Experiment 2 under the given implementation (Figures 4.4, 4.5, 4.6, 4.7).

5.2.1 Worse learning speed for DQN than tabular Q-learning

The generalization and learnability of DQN are poorer with respect to tabular Q-Learning, probably because the considered environments are too small, and the very simple and efficient tabular Q-Learning algorithm trains a lot faster than a neural network.

5.2.2 Logarithmic decrease in performance

The distance in the converged performance for the different levels of noise seems to decrease logarithmically. Looking closer to the zero noise level might give interesting results. Is there a regime shift? Would a really small amount of noise cause a big gap in generalization performance? To answer such questions we train an agent on an environment with Gaussian distributed noise of standard deviation $1e^{-11}$ and mean 0 (2^{nd} red line from the top on the bottom right of Figure 5.2). Interestingly enough, the gap remains quite big for the negligible noise addition, though it seems to reduce when moving closer to the nonnoisy environment. It would be interesting to further explore low noise levels also for learnability, the performance might be boosted because of the nearly unperturbed environment.

5.2.3 Drop in performance for DQN in early epochs

Observing closely the DQN performance in both generalization and learnability, the nonnoisy agent performs worse than the random walk on the first few epochs of training, while the noisy environments outperform it. We were wondering whether training an agent on a noisy version of the environment for the first 500 epochs would increase the convergence speed of the agent. We tested this hypothesis, and the results are reported in Figure 5.2 (purple lines on the bottom left of Figure 5.2). It appears that the overall performance is impeded by such a training strategy. Namely, the agent still converges to the expected performance, but the convergence speed is decreased.

5.2.4 Steep increase for generalization on *v3*

There is a steep increase when looking at the generalization performance on v3 for the noisy environments, which is not present in the nonnoisy version (Figure 4.5). The noisy setup is only used for testing, meaning that the increase in performance must come form the training of the agent in the nonnoisy setup. This is proof of the fact that the noisy environment performance extremely benefits from prolonged training. This suggests that while the performance on the normal environement remains unchanged, probably due to the triviality of the grid, the agent does not stop learning. This learning is beneficial when the agent is faced with a more complicated scenario, as it might need to leverage additional tactics to win.

5.2.5 Generalizaiton beats Learnability!

The most interesting result is found when comparing generalization and learnability for all agents across all grids. The performance for learnability is lower than for generalization, as shown in Figure 5.1 for *v3*. This is an incredible result. It seems to suggest that **learning on an alternative MDP is actually beneficial toward finding the optimal policy for the given MDP**. Reasoning further, the noise introduced to the environemnt might make it too unstructured for any learning to be feasible, though the transferred learning from a more structured environment helps the agent exploit the underlying patterns that it was unable to discover.

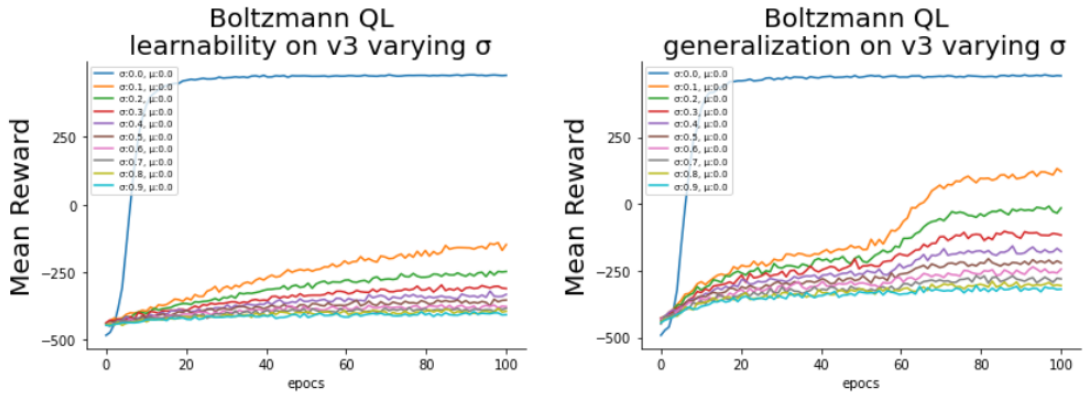


Figure 5.1: Comparison between generalization and learnability for *v3*

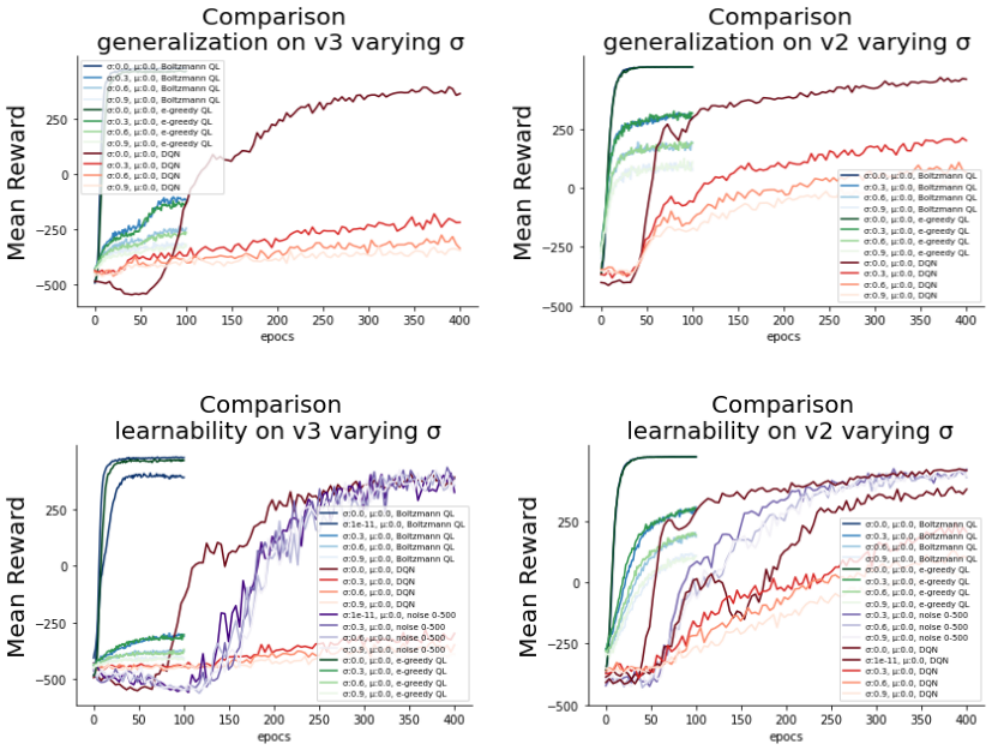


Figure 5.2: Comparison between generalization and learnability

Chapter 6

Conclusion

We presented a new framework for the creation of controlled increasingly different environments to test and compare the generalization ability of classical tabular RL (Q learning), deep RL (DQN), and humans. We conducted a pilot study on Pacman, a popular Atari game, by adding increasing noise to its transition function. In the first experiment, we tested and compared the performance of humans and RL agents on the proposed benchmark. Our results show that humans perform worse than machines, suggesting that previous works might have set an unfair comparison. The performance appears sensitive to the dimension of the grid and the hypothesis we make is that humans' poor results might be due to response delays caused by their reaction time. In the second experiment, we compared the generalization and learnability of RL agents on increasingly noisy setups, finding interesting and counterintuitive results. First, we find that the performance for learnability is lower than for generalization. This suggests that learning on an alternative MDP might actually be beneficial toward finding the optimal policy for the original MDP. Additionally, the noisy environment performance extremely benefits from prolonged training of the agent on the normal version of the environment. Future work for the first experiment might want to investigate more interesting hypothesis accounting for performance decrease, such as humans might need to re-orientate in space, or additionally, the working memory information might need to be refreshed in bigger grids. For the second experiment, further study is needed to understand the incredible result on generalization and learnability. Additionally, it might me interesting to verify the presence of a domain shift in performance between agents trained on the nonnoisy version versus the noisy version of the environment.

Bibliography

[1] Mohammed Amin Abdullah, Hang Ren, Haitham Bou Ammar, Vladimir Milenkovic, Rui Luo, Mingtian Zhang, and Jun Wang. Wasserstein robust reinforcement learning, 2019.

[2] Ankesh Anand, Jacob Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Théophane Weber, and Jessica B. Hamrick. Procedural generalization by planning with self-supervised world models, 2021.

[3] Martin Bertran, Natalia Martinez, Mariano Phielipp, and Guillermo Sapiro. Instance-based generalization in reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11333–11344. Curran Associates, Inc., 2020.

[4] Lucian Buşoniu, Tim de Bruin, Domagoj Tolić, Jens Kober, and Ivana Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28, 2018.

[5] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient, 2020.

[6] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 09–15 Jun 2019.

[7] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning, 2020.

[8] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P. Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability, 2021.

[9] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes, 2015.

[10] James Harrison, Animesh Garg, Boris Ivanovic, Yuke Zhu, Silvio Savarese, Li Fei-Fei, and Marco Pavone. Adapt: Zero-shot adaptive policy transfer for stochastic dynamical systems, 2017.

[11] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning, 2020.

[12] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay, 2020.

[13] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey, 2020.

[14] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning, 2021.

[15] Jens Kober, J. Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013.

[16] Bogdan Mazoure, Ahmed M. Ahmed, Patrick MacAlpine, R Devon Hjelm, and Andrey Kolobov. Cross-trajectory representation learning for zero-shot generalization in rl, 2021.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[18] Younggyo Seo, Kimin Lee, Ignasi Clavera, Thanard Kurutach, Jinwoo Shin, and Pieter Abbeel. Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning, 2020.

[19] Anoopkumar Sonar, Vincent Pacelli, and Anirudha Majumdar. Invariant policy optimization: Towards stronger generalization in reinforcement learning, 2020.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. ACM, jun 2020.

[22] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[23] Marin Vlastelica, Michal Rolínek, and Georg Martius. Neuro-algorithmic policies enable fast combinatorial generalization, 2021.

[24] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019.

[25] Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen. Image data augmentation for deep learning: A survey, 2022.

[26] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification, 2017.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

On structured Domain Generation for generalization in Reinforcement learning

Authored by (in block letters):

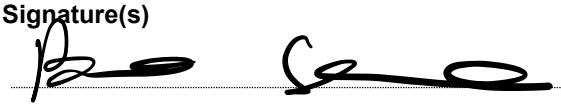
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Serena	Bono

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Turin 19-12-2022	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.