

Universal Artificial Intelligence – Evaluation and Benchmarks

By

Pallavi Mishra

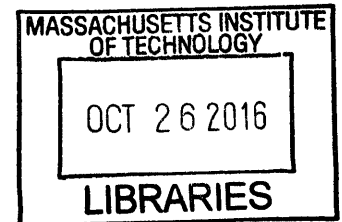
Submitted to the faculty in partial fulfillment of the requirements for the degree of
Master of Science in Engineering And Management

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

©2016 Pallavi Mishra, All rights reserved.



The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature redacted

Signature of Author: _____

Pallavi Mishra
System Design and Management Program
18 May 2016

Signature redacted

Certified by: _____

James M. Utterback
David J. McGrath jr(1959) Professor of Management and Innovation
and Professor of Engineering Systems
Thesis Supervisor

Signature redacted

Accepted by: _____

Patrick Hale,
Director
System Design and Management Program

Reader: Sebastien Helie, Ph.D. Assistant Professor
Department of Psychological Sciences, Purdue University

Universal Artificial Intelligence - Evaluation and Benchmarks

Pallavi Mishra

Contents

1	Preface	5
2	Abstract	7
3	Acknowledgment	8
4	Introduction	9
5	Universal Artificial Intelligence	11
5.1	Theory	11
5.1.1	Universal Artificial Intelligence Agent	11
5.2	Modeling of Universal AI	14
5.2.1	Agents	16
5.2.2	Environments	16
5.2.3	Reward Functions	17
6	Theoretical Features to Practical Implementation	20
6.1	Computational Properties of Agents and Environments	20
6.1.1	Environments	21
6.1.2	Agents	23
6.2	Practical benchmark requirements	24
6.2.1	Finite set of randomly selected environments	24

6.2.2	Environments should be reward-summable	25
6.2.3	Set of selected environments should be diverse enough for general- ization	27
6.2.4	Environments should be reward-sensitive	28
6.2.5	Training and testing phases and environments	30
6.2.6	Interactions with environment should be asynchronous	30
6.2.7	Environments should be representative	31
6.2.8	Adaptive complexity of environments	31
6.2.9	Incorporate physical time and speed of agent	32
6.2.10	Aggregate not average rewards	32
6.2.11	Incorporate multi-agent interaction into complexity measure	36
7	Case Study: ALE Platform for General AI Agents	40
7.1	Game-based Environment Features	40
7.2	Evaluation Metrics	41
7.3	Arcade Learning Environment	43
7.3.1	Atari 2600	43
7.3.2	Scoring metrics	44
7.3.3	Benchmark Results	46
7.3.4	Comparison against proposed practical benchmark requirements	50
7.3.5	Proposed extensions	54
8	Conclusion	58

List of Figures

1	Figure 1 Agent Environment Framework The agent is the decision maker and learns from its interactions with the environment through rewards and observations	15
2	Figure 2 Markov Decision Process The state transitions as shown here are partly random and partly under the control of the agent. State 1 will transition to State 2 if the agent takes action a1 with a probability of 40% whereas there is a 70% chance it will transition to State 3 dependent on action a1 of the agent	28
3	Figure 3 Screenshots of Pitfall! and Space Invaders Pitfall: The agent controls Pitfall Harry, navigating the jungle, collecting treasures, avoiding obstacles in a limited time Space Invaders: One of the earliest successful games in Atari 2600 that involved shooting alien spaceships	44
4	Figure 4 Screenshots of Donkey Kong and Pac-man Donkey Kong: One of the earliest games in the platform game genre Pac-man: One of the legendary games on Atari 2600 that involves the agent to navigate through a maze while avoiding adversarial entities	44
5	Table 2: Initial Benchmark Results published by the ALE Platform	48
6	Figure 6 Deepmind's DQN Results Comparison of Deepmind's DQN algorithm with the other reinforcement learning methods in literature	49
7	Figure 7 Visual Representations for game states in the last layer of the DQN [34] 	56

1 Preface

Since the beginning of modern Artificial Intelligence research in the 1950s, researchers have constantly underestimated the difficulty of producing general cognitive and adaptable intelligent systems with high degree of autonomy for multi-purpose use in tasks including but not limited to space exploration, mining and hazardous environment exploration as well as everyday household tasks. The optimism for a general purpose AI surged in 1970s and was largely given up by the late 1980s after the spectacular failure of the Fifth Generation Computer Systems project in Japan (Crevier, 1993). However, considering that we seem to be far from a general artificial intelligence as we were in the past, the field of AI has been fairly successful since 1990s, by not focusing on a general purpose intelligence but on specific sub-problems that can produce consistent, verifiable results and become commercially deployable. These specialized systems include technologies such as neural networks, computer vision and data mining systems.

Some mainstream AI researchers still believe in general artificial intelligence and are striving to combine these specialized systems that solve sub-problems into an integrated cognitive architecture – a system of systems of sorts from which general artificial intelligence can emerge. Several cognitive architectures have been proposed and analyzed for implementation of various aspects of general intelligence. Some of them have been implemented in autonomous systems as well. However, no standard benchmark tests exists in this research community to evaluate and compare the capabilities of various general artificial agents. Most of the benchmark tests evaluate one parameter of expected intelligence or the other. The various proposed general and mathematically formalized tests are for the most part, impractical. For example, some specific AI tests are designed to evaluate common sense capabilities, whereas others tests these architectures based on autonomy, or learning or meta-learning and so on. The lack of a cohesive and common benchmark alludes to the fact that the community has found it difficult to agree on the definition of general intelligence and its progress till date.

The motivation of this thesis is to analyze the current methods and tests designed to evaluate artificial agents for general intelligence and evaluate if a common framework for

is of practical use or not. The hypothesis we will test in the thesis is: based on the most acceptable understanding of the definition of intelligence and the comparison of current benchmark tests for the same, can one propose a single, common framework to evaluate general artificial intelligence?

2 Abstract

The fields of artificial intelligence has struggled since it's inception about fundamental question of what intelligence means and how to measure it. The underlying issue of defining intelligence and it's formal measure are sensitive issues in human culture, both in respect to humans and more so in respect to machines. Several attempts have been made to generalize the definition of universal intelligence and derive formal benchmark tests from such definitions. In this thesis, we will review the definition of universal intelligence and attempt to aggregate the salient features of mathematically formalized tests proposed for the same. The combined theoretical features for benchmark will then be used to analyze one promising platform - the Arcade Learning Environment (ALE) that integrates Atari 2600 games to test domain independent artificial agents. We will suggest practical ways to incorporate these features into the ALE platform to manage limitations of computing resources used to generate required environments for agents. The limitation of resources is not only a practical constraint but also a factor that should be included in defining any practically useful measure of intelligence. We learn from the exercise that defining intelligence by generalizing it is a self-defeating goal and that, intelligence is best defined with respect to the physical, time and computing resource-related constraint in which the agent operates.

An agent with unlimited resources can adapt to infinite set of environments, but there can be no practical implementation of such an agent. Since physical universe itself has limited although large set of information encoded in the environment with a possibly finite set of non-repeating states, in order to be of practical use, the benchmarks tests should account for physical resources as well as physical time. This constraint related view calls for context-specific measure of intelligence rather than a cumulative total reward based measure across a defined set of environments.

3 Acknowledgment

I would like to thank my thesis advisor Professor James Utterback for giving me his valuable guidance on undertaking the challenges in research and life. He has given me immense freedom to explore the area of my interest.

I would like to thank my thesis reader Assistant Professor Dr Sebastien Hélie, Department of Psychological Sciences, Purdue University for guiding me in my research, reviewing this work and providing many helpful suggestions.

To my husband, Abhishek Ray, who has supported me tremendously in every possibly way imaginable throughout these two years and who continues to inspire me.

I would like to thank our Program Director, Pat Hale, who has been generous, kind and supportive at every single step of my way during the course of the program.

I would also like to give special thanks to Professor Leslie Kaelbling and Professor Gerald Jay Sussman of MIT for some wonderful discussions.

4 Introduction

There have been innumerable theories about what constitutes intelligence. Legg and Hutter in [1] survey various formal and informal definitions collected over a period of years. There are a total of 70 odd definitions in the survey that have been categorized into definitions from psychologists, AI researchers and in the general category. In the end, they devise a concise definition that they finally adopt as practical definition for universal intelligence:

“Intelligence measures an agent’s ability to achieve goals in a wide range of environments.” – Legg and M. Hutter

There is one particular definition that captures the paradox of AI effect - *AI is whatever it hasn’t been done*:

“Our mind contains some processes that enable us to solve problems we consider difficult. ‘Intelligence’ is our name for whichever of processes we don’t understand yet” – Marvin Minsky, Society of Mind

In his book, Society of Mind, Minsky notes that this particular definition is disliked by a number of people as it is bound to change as we learn more about human psychology. According to him, the concept of intelligence is like a stage magician’s trick that disappears as soon as we discover it. He argues that, in the same way as animal and plant life seemed mysterious to us a few centuries ago, until we understood the mechanism of cell-machines and hereditary codes, we are making constant progress in how we understand the underlying theoretical basis that give rise to human thought and intelligence.

Till date, several computational theories including compression, sequence prediction, information encoding, and pattern recognition which are pertinent to certain kinds of difficult problems have been discovered and subsequently used to create applications in the real world that are of substantial economic value. But certainly there is no computational model that appears to be remotely as intelligent (and self-aware) as humans. People argue that computers can only do what they are explicitly programmed to do. This is certainly true *now*. There is no machine that can halt itself when presented with the MU problem as

made famous by Douglas Hofstadter in [2]. The MU puzzle involved transforming an axiomatic string MI to MU using a set of rules or theorems in each step. It is not immediately clear given the elaborate rules whether such a transformation is possible. A human player would most certainly give up after a series of unsuccessful trials and would question the system whereas, there is no way a computer can step out of this formal system and halt voluntarily.

Measuring Intelligence

There have been several attempts to measure various different aspects of intelligence over time. Numerous tests have been designed for human cognitive development and problem solving performance as surveyed in depth by [1] - starting with Alfred Binet in 1905 [3], Stanford-Binet test in 1950 and other versions [4], Stern's IQ test [5] and so on. The Wechsler Adult Intelligence test for example [6] involved verbal components such as knowledge, basic arithmetic, vocabulary and short term memory and non-verbal components such as picture completion, spatial perception, problem solving, symbol search and object assembly. It was later criticized due its strong cultural bias. To circumvent that problem, Raven proposed a culturally neutral cognitive functionality test in 2000 [7]. At the same time in history, psychologists have intensely studied animal behavior. Several studies have been done about cognition and intelligence in animals, for instance by Zentall in 2011 [8], Herman and Pack in 1994 [9]. Others have studied how human bias affects our understanding of animal intelligence [10].

Machine intelligence related research started much later with the first Turing test [11]. Most of the tests that do not involve human subjects have been controversial because of suspected anthropomorphic biases. Turing test itself has been criticized over the years [12] [13] and revised subsequently with several modifications over the years as researchers found it too anthropomorphic [14]. In the AI research community, there is no de-facto standard of measuring machine intelligence. It has been proposed, however, that any reasonable intelligence tests should have certain properties [15] [16] : 1. Repeatability 2. Statistical in-variability 3. Predictability for intelligence related future tasks 4. Measure of learning and adaptation (dynamic test) 5. Measure of prediction ability based on simplest hypothesis explaining observation of (complex) pattern

With the above in mind, in this thesis we explore the theoretical foundations for an exhaustive test for universal intelligence. We begin by defining universal intelligence and then enumerate some of the computational basis of learning and problem solving in machines especially reinforcement learning. We will then look at an emerging new platform to test domain-independent learning agents called the Arcade Learning Environment (ALE). We will compare the current feature set in the platform to the ideal features of an exhaustive benchmark to suggest new extensions and features to the platform.

5 Universal Artificial Intelligence

5.1 Theory

In this section we briefly touch upon the theories behind an ideal intelligent agent, few reasoning principles and computational mechanisms that give rise to learning and problem solving capabilities for intelligent agents.

5.1.1 Universal Artificial Intelligence Agent

Legg and Hutter [17] have defined a universally intelligent agent called AIXI, an agent that can find the optimal policy to maximize future rewards in *any* given environment. This is a theoretically perfectly intelligent agent but mathematically uncomputable since the environments and their complexity measures are essentially infinite. They have mathematically formalized a classical definition of perfect intelligence from Russel and Norvig in their popular modern AI book in 2005 [18] which states that intelligence is defined as “the ability to operate successfully in a wide variety of environments”. Therefore a perfectly intelligent agent should be able to operate in any given environment. As expected, there is severe criticism and skepticism of this idea from the AI community [19]. However, this idea can be taken as an ‘ideality’ measure of intelligence that can possibly never be realized in the real universe and we can work our way backwards to realize physical agents that operate within certain constraints of environments and capabilities. Let us now look at some of the underlying computational basis for universal intelligence (as defined above).

Inductive Inference and Bayes Rule

Inductive inference is the process by which one (an agent) observes the world and infers the cause behind the observations. Much of modern science has been based on the principles of inductive inference. The process of inductive inference involves creating a valid theoretical model that can explain an observed phenomenon in order to understand the underlying mechanism of the environment to predict future events and outcomes. In order to distill one valid hypothesis by avoiding unnecessary explanations for several other consistent hypothesis, Occam’s razor suggests that the simplest hypothesis is the most likely. It is not always easy to distill the simplest hypothesis from a given set, if complexity of all consistent hypothesis are nearly the same. So, once a set of hypothesis $h \in H$ has been developed given the observed data D , Bayesian inference rules can be used to calculate the likelihood of different hypothesis by using Bayes rule as follows:

$$P(h | D) = \frac{P(D | h) P(h)}{P(D)} \quad (1)$$

Here $P(D)$ is the evidence or observed data, $P(h)$ is the distribution of hypothesis h over the space of all hypothesis in the set *before* any data has been observed, also known as the *prior distribution*. A question naturally arises of how to determine this prior probability of hypothesis before seeing any evidence or data. This is a point of intense debate about the principles and validity of Bayesian inference. However, supporters of Bayesian theory argue that inductive inference works within some consistent principles or beliefs that ideally defines the prior distribution space. Another way to avoid defining the prior is to use some well known prior probability distributions that does not allow any biases to creep into the process of induction.

Solomonoff’s induction and universal prior

The problem of universal prior was tackled by Solomonoff in [20] where he was dealing with prediction of binary sequences from an unknown arbitrary computable distribution. He defined a ‘prior’ for a sequence beginning with a given string based on the probability that a universal Turing machine running a randomly generated program would compute a sequence that begins with the given string. Randomly generated algorithm means that every bit has been produced with a uniform probability. For a binary sequence, it can mean that every bit has been produced by a flip of a coin - example head gives 1 and tail gives

us a 0.

The underlying assumption he made was that the observed data was generated from *some* algorithm. Thus, given an observed data or string, if we run every single hypothesis (or program) through the turning machine, one after the other, we can find the valid hypothesis if the output matches the observed data. There can be several (randomly generated) programs that can produce the output beginning with the given string and so, there has to be a way to assign likelihoods to each of the valid hypothesis. Since we are dealing with randomly generated binary sequences, each bit is a coin flip is 50% likely and contributes 1/2 to the final probability of a string. Shorter strings therefore have a higher probability than longer strings. This theory is also compatible with Occam's razor.

Definition 1

Solomonoff's prior for a string $x \in B$ where B is the space of all binary sequences is defined as:

$$M(x) := \sum_{p:U(p)=x^*} 2^{-l(p)} \quad (2)$$

Where U is a universal Turing machine that runs program p to output a binary sequence beginning with x and l is the length of the program p .

This leads to the idea that strings produced by shorter programs have higher likelihood in terms of universal prior and therefore simpler compared to string produced by longer programs. This leads us to the definition of Kolmogorov's complexity of a given binary sequence or a string s :

$$K(s) = \min_{p \in B} \{l(p) : U(p) = s\} \quad (3)$$

If there is no such program p that can generate the given string s , then $K(s) = \infty$. There is why Kolmogorov complexity is uncomputable since it is not possible to determine if there exists such a program that generates the particular output string or not in advance.

5.2 Modeling of Universal AI

The most natural way to formulate a mathematical model of an domain independent artificially intelligent agent is to start with no prior knowledge of the world but still accomplish a task by trial and error. This type of self-learning model is called Reinforcement or Temporal Difference Learning in the artificial intelligence literature. The idea is that an artificially intelligent agent should be able to learn what to do in absence of any training and get better at whatever it is trying to accomplish over time.

Reinforcement Learning

For learning to take place, each play of the game must yield much more information. This is achieved by breaking the problem into components. The unit of success is the goal. If a goal is achieved, its subgoals are reinforced; if not, they are inhibited. – Allen Newell

This technique is a way of rewarding or punishing the agent every step of the way without specifying explicitly how the goal is to be achieved. The agent learns to solve a problem in a dynamic environment by trial and error. This class of problems is significantly different than supervised learning problems.

This learning model scales well in simple as well as complex tasks. A good example of reinforcement learning in context of complex task is the game of chess. There are two ways for an agent to learn to play chess. The first way is to get a set of instructions for rules of the games as a prior knowledge, however since general rules may not cover all the different possible moves, the agent needs to be instructed about valid moves at every single position of the game. This is a tedious way to teach an agent to play the game and such detailed instructions are rarely available in the real world for complex tasks. For an agent to be truly domain independent, it needs to have the ability to explore its environment and learn the optimal behaviour on its own. Imagine the same example of chess but now with no prior rule book. If there is some way the agent can know if it made a good or a bad move, it can automatically learn the rules of the game. One way that the agent learns is by receiving feedback on its actions from its environment. The agent then aim to improve its game based on the response it receives from the environment - in our example, the environment

is the chess board, chess pieces and the opponent.

In many highly complex tasks, reinforcement learning is the only feasible way to train an agent to perform at high levels. Otherwise, it becomes rather hard and tedious for a human to enumerate and evaluate every single possible situation that the agent may face. The space of possible observations and action can become quite large for the agent to effectively search for the optimal next move. One example of such a domain is self-driving cars.

There are several frameworks for reinforcement learning. Some frameworks consider rewards or feedback as part of its input percept while others keep it separate from sensory input. Some frameworks reserve the reward towards the end of the task or activity while in others, rewards are more frequently distributed. We will discuss the most relevant properties of these frameworks with respect to agents, environments and rewards in order to synthesize a general computational model for our purpose of evaluation of a general agent. The most important point to keep in mind with respect to a reinforcement learning framework is that - feedback or reinforcements serve to define optimal policy in environments based on Markov Decision Processes (MDP) that maximizes the total reward for the agent in that environment. But again, not all environments are markov decision processes as we will see shortly.

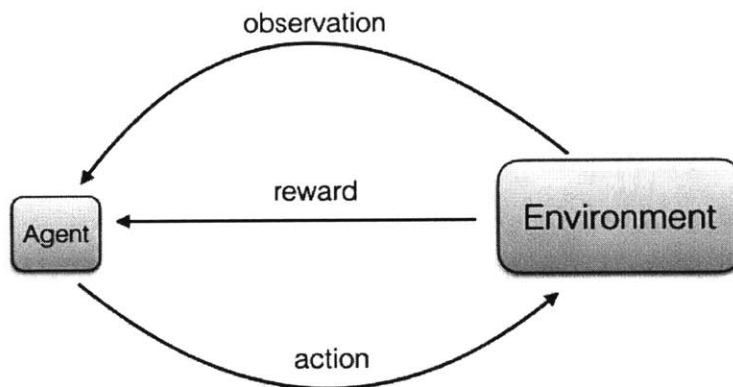


Figure 1 | **Agent Environment Framework** | The agent is the decision maker and learns from its interactions with the environment through rewards and observations

5.2.1 Agents

There are four basic kinds of agent programs that embody the principles underlying almost all intelligent systems [18]:

- Simple reflex agents: These agents choose actions based only on the current perception of their environment, ignoring the percept history.
- Model-based reflex agents: These agents keep track of the world that it has seen before using some kind of a model that can range from simple boolean circuits to a scientific theory and complex hypothesis. The agents use this model to take current actions while updating the internal model of the world based on new observations.
- Goal-based agents: These agents are programmed with a goal that describes desirable situations and outcomes. The agent uses the internal model of the world to navigate its way in the environment to achieve its goals.
- Utility-based agents: These agents use an internal utility function to track their performance. The agent chooses actions to maximize its expected utility over a period of time.

Some more types:

- Random: This is a non-intelligent agent that just chooses actions randomly.
- Learning based: A learning based agent is similar to a utility based agent but is more advanced in the sense that it can update its internal knowledge to adapt to the environment.

5.2.2 Environments

There are several classes of environments studied in artificial intelligence based on Bernoulli scheme, Markov chains and Markov decision processes among others. Interestingly, only some of these classes have properties that makes it possible for any agent to reach an optimum behavior in that environment. In [21] Legg has proposed that his universal intelligent agent would be capable of optimizing its behaviour in all environments that admit self-optimizing agents.

- Passive Environment
 - Bernaulli scheme environments
 - Stochastic processes
 - Markov chains
- Active Environment
 - Bandit problems
 - MDPs and n-th order MDPs
 - Partially observable MDPs
 - Ergodic MDPs.

Some popular classes of active environments in artificial intelligence are Function Maximizing Problems, Repeated Strategic Games, Classification problems.

5.2.3 Reward Functions

Rewards are used to learn optimal or nearly optimal policies for the environment without any prior knowledge of the complete or partial model of the environment. Optimal reward functions are often designed for environments and agents to shape desired behavior of agents [22].

Definition 2

Actions available to an agent is represented by one of symbols in the set of all possible actions. For example: $A = \{left, right, up, down\}$

Rewards are taken from any subset R of rational numbers. For example from a set defined as $\{0...1\}$

Observations are a finite set O . For example, a grid of $m * n$ matrix where each cell can be either 0 or 1.

o_i, a_i and r_i are the observation, action and reward in cycle i

Perception is a set of observation and reward $\langle r_i, o_i \rangle$ in cycle i

In classical AI and cognitive science literature, the sequence of events is generally: observation, action and reward. A possible sequence of events can be represented as $o_1 a_1 r_1 o_2 a_2 r_2 \dots o_n a_n r_n$. The environments are generally probabilistic and agents can be probabilistic as well.

We will use the following notation going forward: For an agent denoted by π , the probability of π taking an action a_k after a sequence of events $o_1 a_1 r_1 o_2 a_2 r_2 \dots o_k$ is represented by the string $\pi(a_k | o_1 a_1 r_1 o_2 a_2 r_2 \dots o_k)$. The environment will be denoted by μ and the probability of the environment giving reward r_k after a sequence of events $o_1 a_1 \dots o_{k-1} a_{k-1} r_{k-1}$ is represented by the string $\mu(r_k | o_1 a_1 r_1 o_2 a_2 r_2 \dots o_{k-1} a_{k-1} r_{k-1})$.

Example 1

Consider a sample test setup where an agent (say, a robot) is given an action space $A = \{K_1, K_2, K_3\}$ where K_1 corresponds to pressing the key marked 1, K_2 corresponds to key marked 2 and so on. The observation space $O = \{L_1, L_2, L_3\}$ is a set of LEDs that are marked as L_1 , L_2 and L_3 respectively. At each turn of the game, exactly one of the three LEDs blinks while the rest are switched off. The robot observes the set of LEDs and chooses to press any of the available keys. After the agent perceives and acts on the observation, it is presented with a reward $R = \{0, 1\}$ where 0 corresponds to no power boost and 1 corresponds to a power recharge of 10 minutes. The goal of the agent is to be able to play the game for as long as possible and so, power recharge is a positive reward.

The observation o_i is randomly generated from a uniform distribution of probabilities over the three LEDs. The environment denoted by μ produces the next reward r_i using the function described below, where 1 and 0 indicate the probability of the transition and the statement inside *if* denotes the conditions for the transition.

$$\mu(r_i|o_1a_1\dots o_{i-1}a_{i-1}r_{i-1}) = \begin{cases} 1 & \text{if } \begin{cases} (a_{i-1} = K_1 \text{ and } o_{i-1} = L_1) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = L_2) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = L_3) \text{ and } (r_i = +1) \end{cases} \\ 1 & \text{if } \begin{cases} \neg\{(a_{i-1} = K_1 \text{ and } o_{i-1} = L_1) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = L_2) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = L_3)\} \text{ and } (r_i = 0) \end{cases} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The optimal policy in this game for the robot is to keep pressing the key with the number that corresponds to the number of blinking LED. Although this environment seems trivial at this point, this representation can be adequately complex to represent several of the environmental attributes that we will discuss about in the following chapters.

Definition 3

The universal intelligence γ as defined by Legg and Hutter [1] in [19], of an agent π is:

$$\gamma(\pi, U) = \sum_{\mu=i}^{\infty} p_U(\mu) \cdot V_{\mu}^{\pi} \quad (5)$$

where environments denoted by μ are generated on a universal turing machine denoted by U and $p_U(\mu)$ represents the probability of each environment getting generated by U . V_{μ}^{π} is the reward gathered by agent in each environment.

6 Theoretical Features to Practical Implementation

In this section, we formulate the key features expected of a practical benchmark evaluation criteria for general agents based on the survey of various theoretical models for test of general, domain-independent artificially intelligent agent. We have discussed so far how theoretically complete notion of universal probability and generalizable universal intelligent agent such as AIXI are mathematically uncomputable. This means that no machine is guaranteed to reproduce the kind of environments and agents we discussed due to halting problem [23].

In this section, we discuss the practical features of the test such as how to sample environments from a limitless set of available or feasible environments, how to structure rewards within the environment that best tests the ability of the agent to learn an optimal policy without cheating, what are the practical constraints to creating such a benchmark in terms of computational speed, memory and time.

6.1 Computational Properties of Agents and Environments

As mentioned in the beginning of the previous chapter, agents, environment and actions are most natural and widely accepted way to formalize training and evaluation for a general purpose self-learning agent in a variety of environments. In this section, we look into the computational complexity of generating environments, computing the next set of observations for the environment depending on the actions of the agent and calculating the rewards for the agent based on its chosen action in response to the full or partial observation it makes in the environment.

Environments can be encoded using different components as required by the host machine such as its visual output, backend algorithms for observation generation based on agent action (for a dynamic environment) or a fixed sequence of observations generated (for a passive environment) and an output interface. No matter which framework, machine or programming language we choose to render an environment in real life, there always exists a simple function to convert the structured information that defines the entire environment into a unique sequence of strings say, x . Similarly, we can also re-generate

the environment from the given string x if we understand the encoding to translate this information back and forth. This means, we can technically talk about an environment as a sequence of bits or a string. This is what we will do in the rest of this section.

6.1.1 Environments

We have discussed in the previous section that the Universal Probability P_U of a string x is such that a ‘prefix-free’ universal machine U , regardless of its physical implementation and programming language, will generate strings of lower complexity with higher probability [24] based on the Kolmogorov complexity (K) of the string x , as defined by:

$$P_U := 2^{-K_U(x)} \quad (6)$$

The problem with this approach however is that, the Kolmogorov complexity K of the output string is an uncomputable function. That is, we cannot determine in advance (within a finite amount of time) the number of bits in a computer program that generates environments with a given amount of information content. We care about the Kolmogorov complexity and universal probability distributions because of the fundamental ways in which we understand the universe and its innate probability distributions. It has been shown how universal probability distribution generates the probability for a string of given complexity by LeCun in [25]. LeCun et al explain the deep connection between complexity of the string and its probability distribution meaning that a randomly tossed computer program on a universal machine $c1$ would generate a string S of Kolmogorov complexity K and then halt is given by:

$$P(c1, S) := \lim_{L \rightarrow \infty} 2^{-L} \sum_P \delta_{eval(c1, P)}^S \delta_{Len(P)}^L \quad (7)$$

This basically boils down to $P(c1, S) \approx 2^{-K(c1, S)}$ as the limit on L (length in bits of program generating string S) converges. The point here is this: as the length of program required to produce the string increases, the probability of randomly generating the program is halved with each additional bit of the program.

We want our string x to be generated by a computer program because, in theory, this

is a fair and scalable mechanism to generate test environments for an agent. We are using a coin toss to determine the bits in our computer program because, in theory again, this mechanism lets us generate every single type of environment possible, recursively. Several proposals for universal intelligence tests [26] [27] are using universal probability because many physicists and computer scientists believe that there is essentially ‘one’ probability distribution in the universe and they believe it makes sense to translate this distribution to computer programs, such as using coin toss for generating random computer programs.

Even if we agree with using Kolmogorov complexity and universal probability to generate infinite variety of environments, we still haven’t solved the problem of Kolmogorov complexity being uncomputable. There are several suggested workarounds to this problem. Some of which are to use a time-bound [28] or weighted version of Kolmogorov complexity. Perhaps the most popular is Levin’s [29] Kt Complexity:

$$Kt_U(x) := \min_{\text{where } U(p)=x} \{l(p) + \log \text{time}(U, p, x)\} \quad (8)$$

Where p denotes the program p that generates string x , $U(p)$ denotes the result of execution of p on machine U , $\text{time}(U, p, x)$ denotes the computation time taken by machine U to execute p in order to produce x . This way, we force a time limit on computation to conditionally make the Kolmogorov complexity measure bounded and therefore computable for practical purposes. The other reason this may be a popular choice is that Levin showed how it is possible to achieve optimal search or learning strategies based on Kt complexity [29]. Hutter’s [30] versions of universal intelligence are based on the results of Levin’s proof that shows for every computable environment, there exists an optimal learning strategy.

In order to train and test an agent then, all we need to do is generate a variety of environments encoding a sequence of observation patterns of varying complexity. The complexity measure of this environment is directly proportional to the ease of predictability of sequence of observations made by the agent. This complexity measure helps evaluate the performance of the agent. Unfortunately, the problem is not that simple. As we discuss below.

6.1.2 Agents

Lets now look at the computational requirement for the agent to understand various environments and achieve its goals in a variety of environments. No matter how complex the string x is, assuming the output to be a computable (enumerable) environment based on the natural universal probability discussed above, a prediction maximizing algorithm exists for each unique s that would eventually be able to identify any regularity in the data - as shown by Solomonoff in his seminal work [24]. To solve the problem of prediction of any arbitrarily long sequence of string, Solomonoff replaced the unknown prior probabilities in his induction theorem with the universal 'natural' probability to prove that a prediction algorithm exists in principle for any computable string. Hutter [31] tried to do the same trick for AI agents that interact with their environments to create his AIXI model of universal intelligence. Based on Hutter's AIXI theory, Legg in [21] tackles the question whether universal predictors like AIXI for computable sequences are themselves computable? The simple answer is no. Legg concludes that highly general, elegant and constructive (or Turing computable) theories of prediction do not exist unlike Solomonoff's induction theory which is elegant and general but non-computable. Moreover, Legg argues that even if we want to have a predictor that can learn to predict all sequences upto a high level of Kolmogorov complexity, the underlying constructive theory must be too complex to analyze mathematically. Remember that, we are assuming unlimited computational resources when we talk about complexity measures of these predictors. In fact, Legg concedes that even very powerful predictors of high Kolmogorov complexity sequences are necessarily very complex because of the underlying problem of existence of computable sequences that are very expensive to compute.

A question that comes up naturally: why are we obsessed with the theory of prediction models and their complexity? And the reason is that, several classes of problems are equivalent to or depend directly on sequence prediction. Reinforcement learning is one such class of problem. In the general framework of reinforcement learning, an agent should make a choice in every iteration cycle about which one of various possible actions to take in order to maximize future (or cumulative) rewards from the environment. At every cycle, it must implicitly or explicitly predict which action now would lead to how much

reward in the present and in the future. Therefore computable predictors have a direct application for reinforcement learning models for agents. In the following discussion about features of the benchmark test, we will strongly prefer environments and agents that are computable with limited resources and find a common ground between mathematically complete versus practically useful features of agents and environments. The other important constraint that we will focus on with respect to the agent especially is - speed. For all practical purposes, agents that learn quickly are preferred in tasks over those that don't. This key aspect will be incorporated as one of the key constraint in creating a useful benchmark test.

6.2 Practical benchmark requirements

6.2.1 Finite set of randomly selected environments

The first obvious requirement for creating a practical framework for general AI agents is to have a finite set of environments for agent interactions so that evaluations can terminate in a finite amount of time. The agent should therefore be presented a set of randomly selected environments generated without bias using some type of a distribution (we will discuss more about choosing distribution later in this chapter). The other obvious requirement would be that for computational feasibility the class of environments chosen should be recursively enumerable. Recursively enumerable type of languages (or rules that produce environment as 'strings' as we have been referring to them in general) are type-0 in Chomsky hierarchy. This property allows a computer program to be able to generate all valid environments of the chosen class. Given that a strong requirement for the benchmark test would be to evaluate virtual (software) agents, we will impose the computability requirement strictly to the class of environments we choose.

In addition to above, there are two major problems that arise as a result of translating the mathematical theory of agents capable of optimizing in an infinite number of environments into a practical test. As Dowe in [27] points out, the first one is with summation of scores in each environment and the other is summation of rewards of actions in each of these environments since the life of agent in each environment is assumed to be infinite in [21]. Dowe et al. suggest limiting the number of interactions between the agent and the en-

environment besides choosing a computable variant of K . The problem of infinite interaction cycles (exploration of the environment) and accumulation of infinite rewards is sufficiently clear. It has also been shown that for active agents, given a MDP based environment, the agent can find an optimal policy in its environment by limiting its exploration to spaces in the environment where a trial optimal policy is the most sensitive to state transitions [32]. In many ways, this kind of exploration policy for maximizing cumulative rewards is intuitive to humans learning a new video game, for example. This shows that limiting the cycles for environment exploration by the agent to a reasonable limit does not interfere in agent finding optimal policies for the environment for several classes of environments as shown in [32]. Therefore, we not only limit the number of environments that the agent samples but also the number of cycles it gets to sample each of these environments itself.

6.2.2 Environments should be reward-summable

Definition from [21]: A reward-bounded or reward-summable environment is an environment μ in a set of environments E housing an agent π with rewards in each environment denoted by V iif $\forall \mu \in E$:

$$V_{\mu}^{\pi} < 1 \tag{9}$$

In the reinforcement learning framework that we are working with, a reward-maximizing agent seeks a sequence of actions or ‘policy’ to do so in the environment that presents rewards and observations to the agent. Given that the control of rewarding and providing non-rewarding percepts in form of observation lies with the environment, the goal is implicitly defined by the environment. Therefore in order to test the agent in any way, it is sufficient for us to fully define the environment itself. In many situations as well as in real life, the desired goal may be more complex than just maximizing current rewards. One simple example would be the temporal relative significance of actions: should the desired goal be to value near term or long term gains? A reward-bounded environment not only normalizes the rewards in a way to make the total possible reward finite but also shapes the behavior of an intelligent agent by weighing rewards at different times in future differently.

If $\gamma_1, \gamma_2, \dots$ are respective discounting factors for rewards in each successive cycle, where $\forall i : \gamma_i \geq 0$ and $\sum_i^\infty \gamma_i < \infty$ in order to avoid infinite weighted sums. The discounting factor can be absorbed in the reward function itself and the following constraint can be imposed on the total returned reward from each environment:

$$V_\mu^\pi := E \left(\sum_i^\infty r_i < 1 \right) \quad (10)$$

Example 2

Lets consider the previously discussed Example 1 wherein a robot is given an action space $A = \{K_1, K_2, K_3\}$ where K_1 corresponds to pressing the key marked 1, K_2 corresponds to key marked 2 and so on. The observation space $O = \{L_1, L_2, L_3\}$ is a set of LEDs that are marked as L_1, L_2 and L_3 respectively.

This time with slightly different reward mechanism where:

$$\mu(r_i | o_1 a_1 r_1 \dots o_{i-1} a_{i-1} r_{i-1}) = \begin{cases} 1 & \text{if } \left\{ \begin{array}{l} (a_{i-1} = K_1 \text{ and } o_{i-1} = L_1) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = L_2) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = L_3) \text{ and} \\ (r_i = 1/2^{i-1}) \end{array} \right. \\ 1 & \text{if } \left\{ \begin{array}{l} \neg \{ (a_{i-1} = K_1 \text{ and } o_{i-1} = L_1) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = L_2) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = L_3) \} \text{ and} \\ (r_i = 0) \end{array} \right. \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

To illustrate the property of reward-bounded environment, let's take into consideration an agent that chooses a fixed action regardless of the action-observation-reward sequence. Now, if the observation o_i is randomly generated between three possible set of option, the cumulative reward collected by the agent over an infinite number of turns of the game is:

$$V_{\mu}^{\pi} = E\left(\sum_{i=1}^{\infty} r_i\right) = \frac{1}{3} \left(\sum_{i=1}^{\infty} \frac{1}{2^i}\right) = \frac{1}{3} \quad (12)$$

It can be easily seen that rewards get smaller as the number of turns increases. This also means that initial few turns have the most impact on the overall reward achieved. The benefit of having a reward-bounded environment is design a benchmark in a way that there is a maximum possible computable reward which makes the tests computable. The way the rewards are structured and the discounting factor are important design parameters to shape the environment and optimal agent policy.

6.2.3 Set of selected environments should be diverse enough for generalization

In theory, any environment that has a finite description but offers infinite number of possible cycles must have a pattern - even if it is just random. The agent in theory can be moved to the next environment after it discovers the pattern and gets optimal cumulative rewards (or fails to discover it and performs sub-optimally) in each one of these environments. If resources on the environment and agent was not limited in real life, we won't have to worry about the selection or exclusion of specific environments that are too trivial for our test. Generating environments algorithmically by enumeration may output several environment that are ineffective for testing along with the ones that are useful. Dowe in [27] suggests several criterion for environment selection to device effective tests. Ineffective environments may be too simplistic in reward generation, substantially overlapped with previously seen environments, non-interactive and non-dynamic. In order to make the best use of resources, it is suggested that the ineffective classes of environments should be identified and excluded explicitly.

Empirically evaluating general competency on a handful of parametrized benchmark problems is, by definition, flawed. Such an evaluation is prone to method over fitting [33]. The high-level requirements of set of environments selected [34] are - 1) Environments should be varied enough to claim generality 2) Each environment should be interesting and representative of a practically useful scenario 3) There should not be an experimenter's

bias (a third party should create test) 4) More complex and difficult environments should be weighted more compared to simple and easy ones.

6.2.4 Environments should be reward-sensitive

Of the several class of possible environments, Markov decision process (MDPs) as depicted in Figure 2 provides a natural and useful framework to architect environments for intelligent self-optimizing agents especially in the domain of reinforcement learning. The basic idea is to create such a universe where the next state is partly dependent on the agent's action and partly random. The transitions from previous to next state is probabilistic and states can be fully or partially observable to the agent. The underlying assumption is that the current reward and the next state are probabilistic functions of the current observation and actions only.

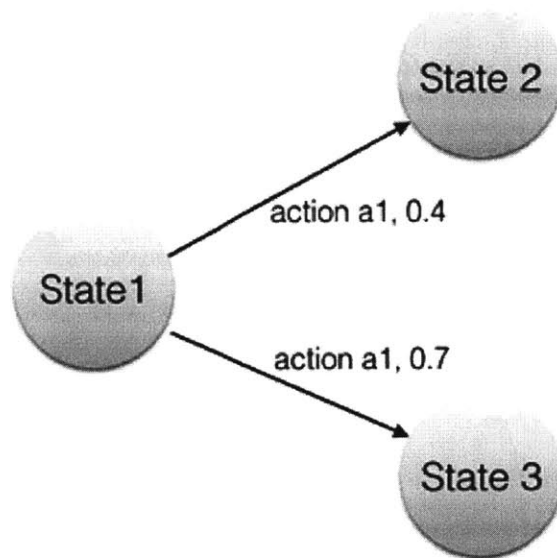


Figure 2 | **Markov Decision Process** | The state transitions as shown here are partly random and partly under the control of the agent. State 1 will transition to State 2 if the agent takes action a1 with a probability of 40% whereas there is a 70% chance it will transition to State 3 dependent on action a1 of the agent

The probabilistic aspect of MDPs allows encoding of real-world uncertainty and randomness like weather related changes to our plans for example. Also, in nature, a strong characteristic of our learning is only being able to fully realize the implications of our actions after those actions have already been taken. Both these features make MDP a natural choice for testing learning algorithms. A special class of MDP are ergodic MDPs wherein any possible state is reachable from in one or more number of cycles from any given state. Several of the classes of environments mentioned in section 5.2 can be proved to be ergodic[21]. It has been proven that ergodic environments admit self-optimizing agents and so, this feature has been used to define universal agents in theory. The problem however with this property is that, since a good state can be recovered in a fixed number of cycles, the agent can exploit this property and give itself multiple chance to learn from its mistakes and improve over time. In real life, not many classes of environments have this property. Therefore, Dowe in [27] proposes that environments should be sensitive to agent's actions in offering a reward in a way that a wrong action can lead to a state that cannot be recovered from. This is a perfectly valid assumption given that many real life tasks are non-ergodic. In order to prepare for such rough environments, living organisms for example, get some level of training or learning from other members of the same species. As we impose this restriction on environments, in order to make the learning possible for an agent with no prior knowledge, we will incorporate a training phase and a test phase similar that in supervised machine learning.

Definition: An environment is n -action reward sensitive if for every sequence of actions $a_1, a_2 \dots a_k$ there is a positive number $m < n$ such that there are two other sequence of actions $b_1, b_2 \dots b_m$ and $c_1, c_2 \dots c_m$ so that the sum of rewards of sequence $a_1, a_2 \dots a_k b_1, b_2 \dots b_m$ and $a_1, a_2 \dots a_k c_1, c_2 \dots c_m$ are different.

Several classes of games for example can be non reward-sensitive as there exist several sequence of actions such that one can ultimately lose in the final round by different margins. In order to effectively determine the agent's performance, this restriction can be overcome by taking into account the margin of win or loss into the reward. This leads to a more practically useful benchmark as no time is wasted in allowing agents to perform actions when there is no scope of effecting final reward.

6.2.5 Training and testing phases and environments

A classic dilemma in reinforcement learning is as the ‘exploration-exploitation’ trade-off. Exploitation refers to the agent choosing an action whose expected reward is known and gets rewards that is close to what the agent expects. Exploration refers to choosing a new actions whose rewards are unknown to the agent for the learning more about the environment and exploring possibly better action-reward functions. It is argued in [35] that averaging the reward between initial learning and later phases does not allow agents to explore their environments optimally in order to perform as best as they could. They propose two separate phases for the agent: learning and evaluation phase. It is agent’s decision to switch from learning to evaluation irreversibly. This structure can help evaluate different types of agents: from ones that do not learn and choose to skip the learning phase altogether, to those that employ evolutionary approach and maximize their time in learning phase by iterating on their algorithms. This structure lends itself well to testing all the spectrum of learning agent in between the two extremes.

Aside from separate phases for learning, [34] suggest that there should be separate and disjoint sets for environment itself: training environments and testing environment. In this case, the agent spends time learning on an training environment and then gets evaluated on test environment that may or may not be similar to the one it trained for. [34] argues that any algorithm’s performance may drastically be overestimated if it is allowed to test on the same training set as it learned on. However, this assumption can be disputed as supervised machine learning on a data-set is very different than an agent learning new behavior in an environment, which is complex enough not to be fully explored in the limited learning phase. Therefore, we will argue that although it makes sense to separate learning phase from evaluation phase, it would depend on the complexity of the environment and the total number of environments available to be able to test that we use different disjoint sets of environments for training and testing.

6.2.6 Interactions with environment should be asynchronous

Several new learning agents and multi-agents show better results when they learn asynchronously from the environment. Everyday tasks are parallel and nature doesn’t wait till

agents actions are available. This will be tied to game environments that do not wait and have implicit option of 'pass' when agent is too late to act/react.

- Real life agents need multiple task unlike single RL goals. [36]
- Asynchronous learning algorithm outperforms current deep learning neural net based machine learning algorithms[37].
- Environment should not wait for actions necessarily (as in real life).
- Environments would incorporate time based rewards by not being synchronous.

6.2.7 Environments should be representative

Increasingly, there is an adoption of games as a test bed for general agents but unsurprisingly, games for the most part, do not represent the real world accurately due to additional uncertainties that real world entails. A block-world robot that can stack blocks is not prepared to handle the real life uncertainties in the environment that includes disturbances due to a mischievous agent that may interrupt the actions of the agent with unforeseen difficulties. That being said, there are several instances of the use of simulations for training humans for complex skills like aircraft pilot simulation training, training business professionals for complex economic and business decisions and so on. The goal for test environments should be to make the environment emulate real life as much as possible.

6.2.8 Adaptive complexity of environments

The problem with using a unbiased universal machine U to randomly generate environments is that environments with lower complexity has a lower Kolmogorov complexity value and hence higher probability to get generated. The agent then is faced with more simple environments than complex ones. Hibbard in [38] proposed a minimum complexity for environments. Ideally, the machine generating environments should be able to adapt the complexity of environments based on performance of the agent.

6.2.9 Incorporate physical time and speed of agent

The tests for agents can be time taking and resource intensive at times and therefore the efficiency of environment reproduction, time between observations, reward and next observation generation is an important factor. The speed of the agent has to be accounted for as a part of its performance either by incorporating the speed into the reward function or simply by increasing the exposure to earn more rewards the faster the agent progresses. In this context, it is desirable that:

- Environments should be episodic [35]
- Computational time should be considered for both environment and agent
 - The time the environment requires to generate the next state/observation, after each action.
 - The time the agent requires to decide upon an action.
- Resource consumption by agent can be a factored in the reward function itself.

6.2.10 Aggregate not average rewards

Several notions of an ideal test including [21] and [27] suggest calculation of a cumulative final score for the test subject (or agent in our case). This score can be used to compare different agents in their generality of intelligence. In order to make sure that the score is finite, the use of reward-bounded environment has been suggested. A reward-bounded environment generates a maximum total reward (in our example case, it is 1) in a possibly infinite number of interactions with the agent. This naturally translates to a discounting factor less than 1 for future rewards. Problems with such discounting rewards policies is that these may be easily exploited or may be even unfair on the agent. For instance, most of the bounded rewards would be typically disbursed during the initial phases of interactions. For an exploration focused agent, this would create a setting that is biased against the agent. At the same time, it would limit the tests for evaluation of far-sighted behavior of agents. It is a prevalent practice in reinforcement learning environment where there is a joint global performance metric that rewards are well distributed locally [39]. For a reward-bounded environment to evenly space its rewards and perhaps delay rewards

to later stages means that the description lengths or size of the environment will become quite large. To mitigate this issue, [27] suggests that environments contain the properties of symmetry and balance so that with shorter description length, the environment becomes fairer and more useful for evaluation. We discuss the properties of reward adjustment, symmetry and balance before we attempt to translate these into practical features.

- Adjusted rewards

Dowe et al in [27] device a new measure of cumulative score that incorporates practical aspects including the total number of interactions between agent and environment as well as the total number of tested environments. The proposed measure is shown as below denotes γ^{ii} as a function of the environments π , the machines used to generated the environments U , the total number of environments in the test set m and number of interactions in each environments which are n_i sensitive. The score is the average across rewards received V_μ^π in each of the environments. The problem with this averaging is that it fails to capture the inherent trade-off between m and n_i . As the number of environments in the set increases, the average complexity faced by the agent varies. Therefore, reward per environment must be adjusted against the complexity ξ and the time spend on that environment measured by number of cycles spent in the environment n .

$$\gamma^{ii}(\pi, U, m, n_i) := \frac{1}{m} \sum_{\mu \in S} V_\mu^\pi(n_i) \quad (13)$$

$$W_\mu^\pi := V_\mu^\pi f(n, \xi_\mu) \quad (14)$$

- Symmetric rewards

Bounded reward environment reduce the flexibility of environments in balancing exploration and exploitation trade-off policy. A solution to this problem has been suggested by [27] wherein the environments give equal measure of positive and negative rewards. $\forall i : -1 \leq r_i \leq 1$ In this scheme, there is no temporal bias for the agent

to explore new environments as each stage of the environment may reward or penalize the agent in equal measure. This is very similar to the way video games are designed where the player learns about the environment quickly by observing negative rewards at the same time, it is possible to overcome the penalty by accumulating positive rewards once enough learning has been achieved.

- **Balanced environments**

The notion of symmetry ties to the concept of balanced environments. A balanced environment is such that a random agent accumulates a total of 0 rewards in this environment. This means that a random policy limits to zero net rewards when the agent is allowed to interact with the environment for infinite number of cycles. If there are more negative than positive total rewards for a random agent then the environment can be considered to be hostile, on the other hand, if the random agent accumulated a positive value of reward over a sufficiently long period of them, then the environment is benevolent. This concept is very important in designing a bias-free evaluation benchmark as we will see when we evaluate Atari 2600 benchmark in the next chapter.

$$V_{\mu}^{\pi_r} = E\left(\sum_{i=1}^{\infty} r_i\right) = 0 \quad (15)$$

Example 3

Lets consider the robot and LED example once again with a modification. The LEDs $\{L_1 L_2 L_3\}$ are now colored in Red, Blue or White. Now the observation space $O = \{RWB, RBW, WRB, WBR, BRW, BWR\}$ where each letter represents the corresponding color of the corresponding LED. For instance, $O_i = RBW$ means $L_1 = Red, L_2 = Blue, L_3 = White$. The environment's μ 's reward system is defined as:

$$\mu(r_i | o_1 a_1 r_1 \dots o_{i-1} a_{i-1} r_{i-1}) = \begin{cases} 1 & \text{if } \left\{ \begin{array}{l} (a_{i-1} = K_1 \text{ and } o_{i-1} = Bxx) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = xBx) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = xxB) \text{ and } (r_i = +1) \end{array} \right. \\ 1 & \text{if } \left\{ \begin{array}{l} \{(a_{i-1} = K_1 \text{ and } o_{i-1} = Rxx) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = xRx) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = xxR)\} \text{ and } (r_i = -1) \end{array} \right. \\ 1 & \text{if } \left\{ \begin{array}{l} \neg\{(a_{i-1} = K_1 \text{ and } o_{i-1} = Bxx) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = xBx) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = xxB)\} \text{ and} \\ \neg\{(a_{i-1} = K_1 \text{ and } o_{i-1} = Rxx) \text{ or} \\ (a_{i-1} = K_2 \text{ and } o_{i-1} = xRx) \text{ or} \\ (a_{i-1} = K_3 \text{ and } o_{i-1} = xxR)\} \text{ and } (r_i = 0) \end{array} \right. \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

The robot is rewarded for choosing to correspond to blue light and penalized for red light and there is no effect on rewards by choosing the white light. Let's now analyze the effect of four distinct policies and their respective rewards to demonstrate the property of a balanced environment. An observation o_i is generated randomly with a uniform probability spread across the possible set of observations. Consider an agent that has an internal policy to always presses key K_1 , no matter what the observation is, the expected cumulative rewards over an infinite run of the game is:

$$E_{i \rightarrow \infty} \left(\sum_{k=1}^i r_k \right) = \frac{2}{6} \lim_{i \rightarrow \infty} \frac{i}{i} + \frac{2}{6} \lim_{i \rightarrow \infty} \frac{-i}{i} + \frac{2}{6} \lim_{i \rightarrow \infty} \frac{0}{i} = 0 \quad (17)$$

A random agent in this environment picks up each response randomly and so, over an infinite period of time, collects the following reward:

$$E_{i \rightarrow \infty} \left(\sum_{k=1}^i r_i \right) = \frac{3}{3} \left(\frac{2}{6} \lim_{i \rightarrow \infty} \frac{i}{i} + \frac{2}{6} \lim_{i \rightarrow \infty} \frac{-i}{i} + \frac{2}{6} \lim_{i \rightarrow \infty} \frac{0}{i} \right) = 0 \quad (18)$$

Since the random agent accumulated a net reward of 0 over infinite play of the game, this environment is balanced. Now, let's say the positive reward is increased to +2 while negative remains -1. In this case, the random agent accumulates a net positive reward over infinite turns of the game. In this case, the environment is unbalanced since rewards are not symmetric. The cumulative positive reward also makes the environment benevolent as opposed to hostile.

6.2.11 Incorporate multi-agent interaction into complexity measure

An important aspect of intelligence is the interaction with other agents to achieve a fruitful purpose or complete a specified task in collaborative or adversarial environment. However, both [21] and [27] leave out multi-agent interactions completely from their consideration. There is only a limited amount of discussion regarding multi-agent interaction with regard to general agent benchmark tests. For instance, [35] discusses about a fixed adversarial game (e.g. Deep Blue), and incorporation of the encoding length of the adversary complexity and computational time into the complexity measure of the game environment that evaluates a general agent.

Recent evaluations of general AI algorithms have happened mainly (and publicly [40]) through games such as the ancient Chinese game of Go. The implications of various types of interactions on performance on intelligence measures cannot be ignored. The use of adversarial search for example, helps AI community examine problems related to planning in the real world. Multi-agent games can be modeled based on well-defined states of games and agents are usually restricted to a fixed number of actions whose outcomes are determined by precisely defined rules. Of course, in the real world, many multi-agent games would have much more complicated descriptions and much larger range of possible actions. Due to the complexity of modeling required in physical games, these have not attracted much of attention from the AI research community.

To illustrate how independent and co-operative or adversarial interaction of agents can

be incorporated into practical benchmarks let's consider the following example which is derived from [41]:

Example 4

Let us consider a 10 by 10 grid matrix. There are two types of agents present in this grid world - hunters and prey. Every agent occupies a cell in the grid. More than one agent can occupy the same cell in the grid. A prey is captured if one or more hunters occupy the same cell as the prey or when two hunters are adjacent to a prey. The observation space for each agent is a subset of the grid world for each type of agent. The hunters' have an observation space of depth 2. This means that hunters cannot see beyond a 2 by 2 matrix centered at their current position.

$$\text{Observation } O_i = \begin{bmatrix} 0 & 0 & 0 & 0 & h_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_2 \end{bmatrix}$$

means that hunter h_1 and prey p_2 are in the top-right and bottom-right cell relative to the position (x) of the agent. Action space of agents is denoted by $A = \{up, down, left, right\}$ where each direction represents a move to the next cell in the corresponding direction. On each turn, each agent make a move. The observation is generated for each agent based on the location of other agent and assigned observation space of the agent.

Rewards can be defined to incorporate agent interactions. For instance, all hunters involved in a prey capture receive a reward of +1. For each move that they don't capture a prey, they receive a reward of -0.1. Since the rewards for catching a common prey are not shared, this environment will permit cooperating hunter agents.

In order to incorporate these types of multi-agent environments, the description string for the environment must extend to include the description of multi-agent interactions.

One of the several ways to account for multi-agent interaction is to extend the adjusted reward function discussed in section 7.2.10 to include complexity of interactions. This means that an agent that interacts with more than one other agent will be rewarded based on the complexity of their interaction. The other way is to expand the description of the environment to include the description of the interactions itself, but to mathematically formulate possibly complex interactions can be rather difficult and time consuming. To allow for a simple encoding of interaction complexity, while making a trade-off of losing some of the precision, we can experimentally calculate the relationship between interactions and environment complexity. This factor can then be used to weigh the reward disbursed to the agent. So, the rewards are now re-adjusted to incorporate multi-agent interactions as follows:

$$\gamma^{ii}(\pi, U, m, n_i, h) := \frac{1}{m} \sum_{\mu \in S} V_{\mu}^{\pi}(n_i, h) \quad (19)$$

$$W_{\mu}^{\pi} := V_{\mu}^{\pi} f(n, h, \xi_{\mu}) \quad (20)$$

Where h denotes the presence of other agents in the first equation and $f(n, h, \xi_{\mu})$ is the relationship between number of interactions n in the environment with number of agents h . Function $f(n, h, \xi_{\mu})$ is decreasing in n to account for learning, increasing in h to account for multi-agent interactions related complexity and increasing in ξ_{μ} which is the complexity of the base environment.

The correlation of multiple agents interaction to environment complexity can be estimated using a simple experimental method that is relevant to the structure of the environment. For instance, in the case of Example 4 that we discussed earlier, one way to find the added complexity is to compare random agents in single and joint tasks with multiple agents. This gives a baseline difficulty in terms of random policy-less behavior both for single and multiple agents in the environment. The same test can be repeated with

learning agents to give insights about the effect of interactions on performance.

In case of Example 4, [41] calculates the average number of steps for random hunters to capture a prey averaged over 200 trails. Table 1 displays the result of the experiment:

Table 1 - Individual and joint task performance in a multi-agent environment for Hunter-Prey game [41]

N-of-prey/N-of-hunters	1/1	1/2	1/2 (joint task)	2/2 (joint task)
Random hunter agents	123.08	56.47	354.45	224.92
Learning hunter agents	25.32	12.21	119.17	100.61

The first column with one prey and one hunter represents the baseline case of single agent task. It can be seen that in this particular game, the presence of multiple agent is positively correlated with the performance of individual agent even without explicit interaction. The third case of one prey and two hunters is a joint task with explicit interaction between agents. Since both the agents are random, the resultant outcome is due to random interaction which is negatively correlated with their performances. The last case shows how availability of more preys affects the random baseline performance.

The affect of multi-agent interaction on baseline performance is subjective to each environment. It is best calculated experimentally (as shown in our example) to adjust rewards received per environment. This insight is even more important for benchmarks involving legacy platforms that were not originally built for AI evaluations but need to be adapted for domain-independent agents performance benchmarking.

7 Case Study: ALE Platform for General AI Agents

“A game is a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, and an agent can affect the outcome through taking actions during the game based on full or partial knowledge of the game state.” – Schaul, Togelius and Schmidhuber, Measuring Intelligence through Games [35]

ALE platform has been introduced by [34] as an interface to Atari 2600 video games as a challenge for AI agents and also as a research methodology. The ALE platform has been gaining significant interest from the research community as a scalable, reliable test methodology and benchmark platform for general AI agents. Other more specific platforms to test intelligent agents includes physical robotics, cognitive robotics tasks such as path planning and blocks stacking, theorem proving, natural language understanding and the Turing test and its variations. In this section, we focus on the ALE platform as a benchmark for general agents and evaluate the platform against the criteria we listed out in the previous section.

7.1 Game-based Environment Features

Games have certain unique features that makes them a self-contained and viable environment for AI benchmarks:

- **Variable complexity:** There exist innumerable games suitable for one or the other aspect of human cognition such as perception, prediction, model-building, planning and learning. For each of the genres, the complexity of gaming environment as well as the skills required for human player are highly variable.
- **Flexible reward functions:** The reward functions can be precisely defined and computed in gaming environments making it easy to shape the rewards and control the structure of the games.
- **Control over agent’s observations:** Games range from partially to fully observable

states giving flexibility to test a wide range of Markov Decision Processes with varying degrees of control.

- **Input dimensionality:** Perceptual resolution and input parameters can be easily controlled in online games - leading to tests catered to an array of agents with different visual input processing capabilities.
- **Multiple levels of agent interaction:** Games allow us to encode varying degrees of interactions of varying nature as discussed in the Hunter-Prey game example earlier.
- **Comprehensible evaluation:** The evaluation can always be tweaked to include parameters including but not limited to processing speed, leaning time and exploratory aptitude of the agent based on the desired features expected of the agent.

7.2 Evaluation Metrics

The problem with using commercial games for evaluation of performance of our AI algorithms is that, each game has its own scale in terms of rewards and has its own difficulty level for learning to play the game as well as scoring high in the game. A score in one game cannot be compared directly with the other and therefore, there is no easy way to summarize scores and extract summary statistics from a variety of games. The use of off-the-shelf games calls for an improvised evaluation metrics. The other problem with games based evaluation is more fundamental.

Whiteson et al demonstrate in [42] that AI agents have a tendency to over-fit their evaluation metrics by over-specializing to the evaluation setting that the designer of the algorithm has tested it against. As a result these agents achieve misleadingly high scores in empirical evaluations such as games for example. They argue that not only supervised learning, but also reinforcement learning based agents are especially more prone to this problem. A supervised machine learning agent is prone to suffering from 'data overfitting' and in an analogous way, a reinforcement learning based agent is prone to what [42] calls is 'environment overfitting' where it specializes to its environment, which more often than not, are Markov Decision Process based environments that allow the agent to over-specialize since there exists an underlying optimal policy that performs the best that the

agent seeks actively.

In order to adjust for this over-specialization problem, the use of multiple environments for evaluation that are randomly sampled from a distribution has been advocated. The idea is to force randomness and uncertainty in evaluation process so that the agent cannot use prior knowledge that cannot be generalized to the target set of environments. On the surface this may seem to solve the evaluation problem, but there is a deeper underlying issue with overfitting that we will discuss next.

High performance or over-fit?

The most crucial performance metric for domain independent AI agents is the diversity and breadth of the chosen target environments. Theoretical research aims to discover algorithms that perform substantially well across the set of all discrete MDPs for example, the universal AIXI agent. At the same time, for all practical purposes, researchers make the trade-off between optimization of an algorithm for a specific setting versus its performance on several other applicable and general settings. This seems to be a successful strategy for most practical applications. In principle, this may not be a matter of real choice but a limitation whenever “no free lunch theorem” [38] [43] applies. The no free lunch theorems prove that for any general purpose optimization algorithm, an increase in performance for one class of problems comes with an equal measure of decrease in performance over the another class of problems.

Given that optimization algorithms are constrained in terms of their specialization and at the same time have to be of practical use, it may to be a rather a good thing for it to specialize in some environments at the cost of other classes that the designer doesn't care about. However, this notion may not be completely valid since specializing in a certain class of environment is desirable to the extent that the same algorithm should perform well in the same class of environments under various forms of uncertainty such as stochastic transition of states. The target set of environments may be large or small in size. On the other hand, an algorithm that has overfit its evaluation environment performs well in just the one that it has been developed and tested for, without scaling to other environments in the same class. What is worse is that, the algorithms that overfit a certain test environment tend to identify and use the optimal policy throughout while outperforming other

algorithm that actually learns from scratch.

The way to solve the problem of evaluation overfitting is: 1) predefined sample of environments with random sampling during test phase 2) secret sample of environment tested by third party 3) generalization and uncertainty is controlled.

7.3 Arcade Learning Environment

In this section, we study the most promising and extensive general AI benchmarking platform in current use. The Arcade Learning Platform was introduced in [34] as an experimental methodology for empirically assessing agents designed for general competency. Since then, it has gained massive attention from the AI research community as testing platform. Our goal is to analyze this platform and apply the features we have distilled from section 7.2 to assess current gaps and suggest meaningful extensions for the ALE framework.

Arcade Learning Environment (ALE) is a software framework designed to develop gaming environments to evaluate general AI agents. ALE currently provides a suite of games on Atari 2600 platform [34] that claims to present significant research challenge for domain-independent general AI agents in the areas of reinforcement learning, model-learning, model-based planning, imitation learning and several other AI domains for learning. The Atari 2600 video game console hosts more than 500 different individual games developed in 1977 for the general entertainment market. One of their most iconic games are PAC-MAN and SPACE INVADERS. Each of these games has it's own architecture and reward system so, each game differs substantially in terms of optimal cumulative reward policy or planning strategy.

7.3.1 Atari 2600

The Atari 2600 is a home video game console developed in 1977 and sold for over a decade. It came with a simple hardware - a general purpose 1.9 MHz CPU with game codes distributed in cartridges separately. The original pack of games were a set of 500 different games. It is interesting to note that new Atari 2600 games continue to be developed even today. The cartridge RAM can hold up to 4 kilobytes of game code and the console RAM

has only 128 bytes of memory. A single screen's resolution is only 160x210 pixels with 128-color palette. The input device is a console with 18 different actions - 3 positions of digital joystick on each axis and a separate button. The low complexity of the games offers a perfectly balanced environment that is challenging for emerging AI capabilities with manageable complexity.

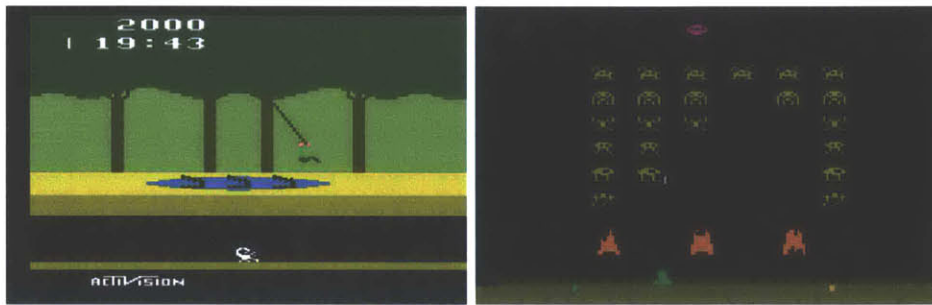


Figure 3 | **Screenshots of Pitfall! and Space Invaders** | Pitfall: The agent controls Pitfall Harry, navigating the jungle, collecting treasures, avoiding obstacles in a limited time | Space Invaders: One of the earliest successful games in Atari 2600 that involved shooting alien spaceships



Figure 4 | **Screenshots of Donkey Kong and Pac-man** | Donkey Kong: One of the earliest games in the platform game genre | Pac-man: One of the legendary games on Atari 2600 that involves the agent to navigate through a maze while avoiding adversarial entities

7.3.2 Scoring metrics

As every single game on ALE platform differs from the rest in its rules, challenges, goals and scoring scale, the ALE researchers use some way to normalize scores to compare and

aggregate the final scores for evaluation and comparison of general agent. In order to make the benchmark and scoring meaningful and useful, the AI community needs to agree on the method of normalization. [34] propose few ideas to create a baseline, we will summarize those and then add our own suggestions.

Choosing Normalization Baselines:

- Using predefined agents

$$\text{Percept are defined as: } x(t) = \begin{cases} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{cases} \quad \text{and actions are defined as: } \alpha(t) = \begin{cases} \alpha_1(t) \\ \alpha_2(t) \\ \dots \\ \alpha_D(t) \end{cases}$$

- Random agent as baseline

$$\alpha(t) = rnd$$

Several times in AI community, an algorithm's performance is compared to that of a random agent that does not follow any intelligible policy and does not imbibe any learning mechanism whatsoever. A random agent selects the next action purely at random without any consideration of its percepts. In AI community random agent are considered entirely unintelligent and are therefore often used as a baseline measure for an algorithm's performance - just like statistics and machine learning uses random choice and selection as baseline for performance measurements.

- Constant action agent

$$\alpha(t) = \alpha_{fixed}$$

A constant action agent chooses a single (fixed) action every cycle instead of a random action. [34] use the best score achieved by any fixed action as one of their baselines.

- Constant and random mixed agent

$$\alpha(t) = \begin{cases} \alpha_{fixed} & p = 0.95 \\ rnd() & p = 0.05 \end{cases}$$

This agent chooses a fixed action 95% of the time and a random action 5% of the time. Again, the policy that generates the best result is chosen as the baseline performance.

More than acting as a baseline agent, these agents also protect the evaluation process against trivial policies that may possibly exploit weakness in these games. In fact, their usage appears to be more of latter than former since these games are off the shelf entertainment games that have not be originally designed for the purpose of such a benchmark test and therefore may have some loopholes that could be exploited by such trivial agents.

- Human beginner to expert scale

Although a baseline using human competency levels appears too anthropomorphic, it serves the purpose well as these games were originally designed for human players. In order to have a fair comparison, it is essential that learning based algorithms be compared against human beginners average scores and afterward learning phase is over, the comparison can be moved to human experts.

- Inter-algorithm normalization

This method can be useful in measuring the relative performance of different algorithms. The normalization is done taking into account the minimum and maximum scores achieved among by all algorithms tested on the same set of games.

7.3.3 Benchmark Results

One of the major contributions of [34] in addition to proposing the ALE framework is sharing their benchmarking results on the new platform that they believe will serve as a point of comparison in the future. They also use the results to explain the empirical methodology involved in using the framework. In this brief discussion, we look at their results and use the insights to propose future extensions and modification based on our practical benchmarks.

Feature Extraction Mechanisms and Implication on Performance for Baseline Purpose

It turns out that designing general agents that can act near-optimally in a variety of unknown domains with stochastically changing states is a rather challenging task. This

task becomes even more difficult when the number of possible states in the environment is of a very high order (as in the natural world). For example, the number of states in the game of Go is 10^{38} . In such a situation, an important factor in designing a reinforcement learning based algorithm is to use linear function approximation [44]. Linear function approximation drastically reduces the number of parameters needed to be tracked by the algorithm by reducing the representation of a long-term utility function to a low dimension space. There are several popular reinforcement learning algorithms in the AI research community one of which is SARSA (λ) (State-Action-Reward-State-Action). SARSA (λ) is a well known model-free reinforcement learning technique that has been used by [34] to benchmark the ALE platform. A detailed explanation of SARSA (λ) can be found in [45]. Using this algorithm as their base, [34] test several feature selection methods for linear function optimization. The feature selection methods parse and extract different types of features from the screen such as colors, objects, movement and background. These features selection methods are analogous to visual sensory input for humans. The descriptions of these methods can be found in [34]. As it turns out, the choice of features make a significant difference in baseline game performance.

Evaluation Methodology

The evaluation method consists of agents playing two disjoint sets of games in a sequence. The first set is the training set and the second is the evaluation test. This is done in order to remove evaluation over-fitting. All games in both the sets were single player games. The training set consisted of 5 games as shown in Table 2. The test set was a random set of 50 games. The games begins after the game reset and ends in 5 minutes (or game over, whichever comes first). Each 5 minute trail is termed an 'episode'. The agent is allowed to act every 5 frames i.e. 12 times per second. The trail consists of 5000 training episodes while evaluation consists of 500 evaluation episodes.

Analysis of Baseline Measures

In order to understand the strength and weaknesses of the different baseline measures, we will analyze a sample result from [34] as shown in Table 2. The first two games ASTERIX and SEAQUEST are from the training set and the rest three are from test or evaluation set. It is clear from the representative test set that learning agents despite the type

of feature selection method outperform random agents consistently. This result is true of the next 50 games that the SARSA (λ) agents were tested with. This shows that there is significant learning opportunity possible in this platform for general purpose learning agents. These agents extract perceptual knowledge from the screen through visual input and the choice of feature extraction method affects agent performance from game to game. There is no single feature extraction method that performs consistently well across the array of games. This is a truly intriguing result that correlates the quality and dimensions of structural information extracted through visual input with learning performance.

Game	Basic	BASS	DISCO	LSH	RAM	Random	Const	Perturb	Human
ASTERIX	862	860	755	987	943	288	650	338	620
SEAQUEST	579	665	422	509	594	108	160	451	156
BOXING	-3	16	12	10	44	-1	-25	-10	-2
H.E.R.O.	6053	6459	2720	3836	3281	712	0	148	6087
ZAXXON	1392	2069	70	3365	304	0	0	2	820

Table 2: Initial Benchmark Results published by the ALE Platform

Recently, researchers at Google DeepMind Technologies [46] have demonstrated that their deep learning neural network based agent can learn successful game-playing policies through the visual sensory input from screen pixels and score data of the ALE platform to surpass all previously seen performance levels of AI agents. This single deep learning algorithm is able to learn and excel at a vast set of Atari 2600 games that is comparable to human expert performance. This is a significant achievement over not only above previous algorithms but over previously seen human expert performance levels on as many as 29 games out of the 49 games that were used to evaluate this agent. However, significant challenge in planning longer term strategies still remain. Games such as Montezuma’s revenge seems to require high-level planning that is beyond the capabilities of current domain-independent agents as pointed out in [34]. However, that challenge remains largely unsolved as seen from DeepMind’s performance on the game in Figure 6.

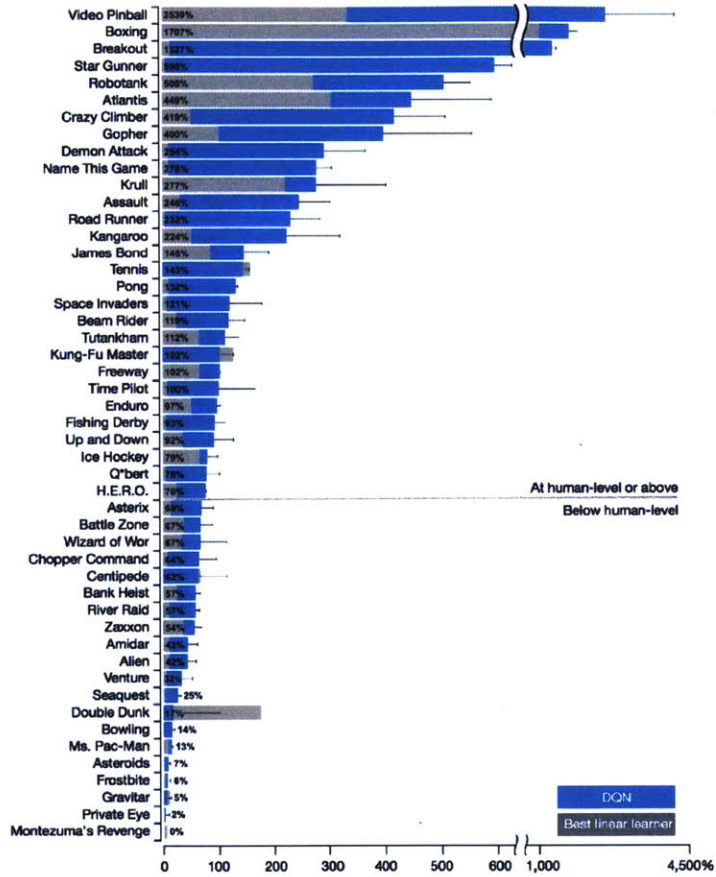


Figure 6 | Deepmind's DQN Results | Comparison of Deepmind's DQN algorithm with the other reinforcement learning methods in literature

Table 3 - Evaluation of the ALE framework on our criteria for a comprehensive general AI benchmark

Features	Requirements
	Met?
Finite set of randomly selected environments	Y
Environments should be reward-summable	Y
Set of selected environments should be diverse enough for generalization	N
Environments should be reward-sensitive	Y
Training and testing phases and environments	Y
Interactions with environment allowed to be asynchronous	N
Environments should be representative	N
Adaptive complexity of environments	N
Incorporate physical time and speed of agent	Y
Aggregate not average rewards	N
Incorporate multi-agent interaction into complexity measure	N

7.3.4 Comparison against proposed practical benchmark requirements

Table 3 shows how the Arcade Learning Environment’s Atari 2600 platform fare against the set of features that we have distilled from our survey. The most important contribution of the ALE platform is to create a useful and extensible software based platform for testing new agents against baseline benchmark data. The platform is open source and allows users to add new games and new features extensions to the platform. However, as the Table 3 shows, there are several criteria for testing general AI agents that still remain unsolved.

Finite set of randomly selected environments

The ALE platform offers an extensible but fixed number of games or environments. There is however, no current way to calculate or estimate complexity of individual games taking into consideration the encoding length of the environment, multi-agent interactions

and overall difficult estimate.

Environments should be reward-summable

The games played during evaluation period have a maximum possible score by design but the scores per games are normalized based on different criteria as discussed in Section 8.5.2. The process of normalization allows fair comparison based on specific criteria including comparison with human average, humans experts or inter-algorithm measures.

Set of selected environments should be diverse enough for generalization

In our discussion of diversity of environments, we excluded various subsets of the physical universe. Clearly, this is an unrealistic assumption for general agents that operate in the real world. Ignoring the physical tests aside, keeping in mind possible realistic simulations of real world, we are a long way in incorporating such features in current platforms including the ALE platform. As seen in the description of the ALE platform which is the most exhaustive and popular at the moment, there are several constraints in terms of resolution of visual inputs and memory storage for environments. Simulation games like mine-craft and half-life are good simulations of real life uncertainties. It should be possible to extend the ALE platform with more complex and diverse set of environments beyond the Atari 2600 games.

Environments should be reward-sensitive

Since most games on the Atari 2600 platform have a reasonable level of granularity in terms of final scoring, most games are n-action reward sensitive. It should be noted that when adding new simulation based games to the platform, this criteria should be enforced strictly to make sure the normalized scoring system is still valid specially when compared across algorithms.

Training and testing phases and environments

It has been suggested that within a given new environment, agents should get some 'learning' time before switching to the testing phase that would be the time for which the agent will be evaluated [35]. The other requirement especially for reinforcement learning agents is protection against environment over-fitting. The ALE platform uses different sets of randomly chosen games to let agents train on a set of games before evaluation on a disjoint set of randomly chosen games. This technique protects against evaluation over-fit.

It also gives the agent a chance to learn some policies on games that may or may not be useful directly on the testing set.

Interactions with environment allowed to be asynchronous

Due to the nature of the Atari games, this criteria is unmet. Asynchronous interactions require that the agent is able to act in a phase that may not match the environment's change of state. This is generally not possible for interactive games. It should be noted that this criteria is not in conflict with the reward-sensitive criteria since the environment can still be n-action reward sensitive, however, it would not need to wait for the agent to take an action before transitioning to another state.

Environments should be representative

Unfortunately, the complexity of current games on the ALE platform do not match real life complexity by any margin. However, by the current available performance of planning and search algorithms, Atari 2600 still presents significant challenges. These challenges may or may not hold given the pace of progress of domain-independent AI algorithms. However, it must be noted that no known domain-independent AI algorithm is able to perform equally well on every single game on the ALE platform as of now. We are still a long way in creating environments that closely mimic real life challenges both in terms of state transitions complexity as well as perceptual resolution.

Adaptive complexity of environments

Adaptive complexity of game based on progress of the test subject is currently not a available feature of the ALE platform. Unfortunately, there is no straightforward way to extend the ALE platform to implement this feature. In order to assess the complexity of games, there needs to be a consensus on the best way to encode game complexity. There are several options to encode games in standard formats for complexity measurement. It can be argued that all programming languages are a way to encode games and the length of the computer code to run the game can be taken as a measure of its complexity. It is certainly true that all varieties of games can be encoded on any programming language, but choosing an arbitrary format does not allow for a fair comparison of games.

None of the game description and encoding methods that exist today can encode every variety of game as observed by [47]. Game Description Languages (GDL) that are currently

available include the Stanford GDL, the Ludi language and several other proprietary languages. Each of them have their own shortcomings as studied in [47]. For instance, Stanford GDL is certainly biased toward board games. More recently though there have been attempts to find strong and extensible representations for two-dimensional video games. For instance, Tom Schaul in [48] developed a new language 'PyVGDL' to serve as a diverse benchmark for computational intelligence.

Incorporate physical time and speed of agent

The nature of games inherently allows for incorporation of agent's speed into the reward. For most of the benchmark tests, each game is limited based on cycles or physical time. The faster the agent learns, the better it scores in the array of games in Atari 2600 platform. The same time limitation and time-reward sensitivity should ideally apply to new games that are added to the platform.

Aggregate not average rewards

The limitations that we discussed regarding relative measurement of individual game complexity applies to calculation of aggregate rewards as well. There is no scope for reward or score adjustment if there is no uniform complexity parameter. Relative complexity based adjusted scores is a more important criteria that it may seem to appear at first. Currently there is no single algorithm that performs well on all types of games on the ALE platform. We have limited understanding as to what makes seemingly easy to master games for human players sometimes very difficult for planning and learning agents. The only explanation seems to be that the planning horizon of these difficult to master games are beyond the scale that current learning and planning agents can cope with. However, the difficulty could be more subtle than that. For example, in case of the notoriously difficult game of Montezuma's revenge, it is easy for a human to see that the end objective for the protagonist is to grab the key in the corner of the screen. For learning agents, this world knowledge is inaccessible and they make a lot of mistakes trying various actions until the time runs out. We currently have no parameters defined to measure this level of subtle cognitive complexity of games apart from the encoded length based complexity.

Incorporate multi-agent interaction into complexity measure

Taking the previous point to another level, there is currently no dimension to measure

different types of complexities relating to planning, learning and cognitive skills required to play the games on ALE platform. One of the missing dimensions is the multi-agent interaction related complexity. The level of currently integrated games on the ALE platform does not allow for multiple agents to interact. However, as the complexity and diversity of games grow, this parameter needs to be accounted for while evaluating benchmark results.

7.3.5 Proposed extensions

At this point, having evaluated the ALE platform for completeness as a benchmarking tool for domain-independent artificial intelligence, I would like to propose the following extensions to the platform to aim for further completeness as a benchmarking tool:

- **Training and testing phases for games during evaluation**

The ALE platform divides training and testing phases into disjoint sets of games to protect against environment over-fitting problem. However, as the diversity of games increases in the platform, it makes sense to further allow agents to train in a new and complex environment without being penalized before they start scoring. This will be especially true for real-world based simulation games like half-life and mine-craft that are too complex to be generalized by playing simpler games in the Atari 2600 suite. The feature can be made available on certain complex games that require cognitive complexity that is beyond the knowledge base for learning and planning agents.

- **Complexity measure for individual games**

The games on ALE platform currently run on Stella Video Computer System simulator. The underlying software structure of the games are mostly hidden since the games are run on top of an emulator for Atari 2600 cartridges on various modern operating systems. The first step for assessing complexity is to define the key dimensions that contribute to complexity of environments such as modified Kolmogorov complexity through length of the shortest game program, cognitive complexity and multi-agent interaction complexity. The games should then be ported to a standard game description language based software to add the required transparency for complexity related estimations.

- **Uncertainty incorporation**

The games provided on the ALE platform are not complex enough to model real-world complexity at this point. Real world based simulation games such as half-life have high degree of uncertainties embedded within the environment. It would be worthwhile to integrate complex real life role-playing based games to model high degree of uncertainties in the test environment.

- **Transparency for agent's policy and learning states**

This feature would be helpful on the agent side in order to add transparency regarding the learning status of the evaluated agents. [46] do this through a visualization of the last layer of their deep neural network as shown in Figure 7. The figure is helpful in understanding how a successful agent learns policies as reflected by the stored representations in the last layer of their deep neural network. They use a technique called "t-SNE" [49] for visual representation of high-dimensional data in order to illustrate how similar states in terms of expected rewards which are perceptually similar to humans are clustered together. The colors represent expected maximum rewards (with high rewards mapping to red color to low rewards mapping to blue color). This information is crucial to gain more understanding the conceptual basis of agent's performance especially for other members of the AI community.

- **Experimental benchmarks to establish the relation between number and complexity of games affecting the agent's score**

In order to understand the actual transferable learning gained by the agent, a feature that benchmarks the affect of playing more games with stated complexity and its affect on subsequent game performance would be very helpful. The affect can be measure as a delta increase in performance without a learning phase. Another way to gain this insight would be to order the play of games differently measuring the performance in various orders of the play. This relates to the requirement of finding a weighing factor $W_{\mu}^{\pi} := V_{\mu}^{\pi} f(n, \xi_{\mu})$ in Equation 14 that can be used to calculate an adjusted score based on the number and complexity of games encountered.

- **Asynchronous interactions**

It is important for the ALE platform allow agents of varying action speeds without

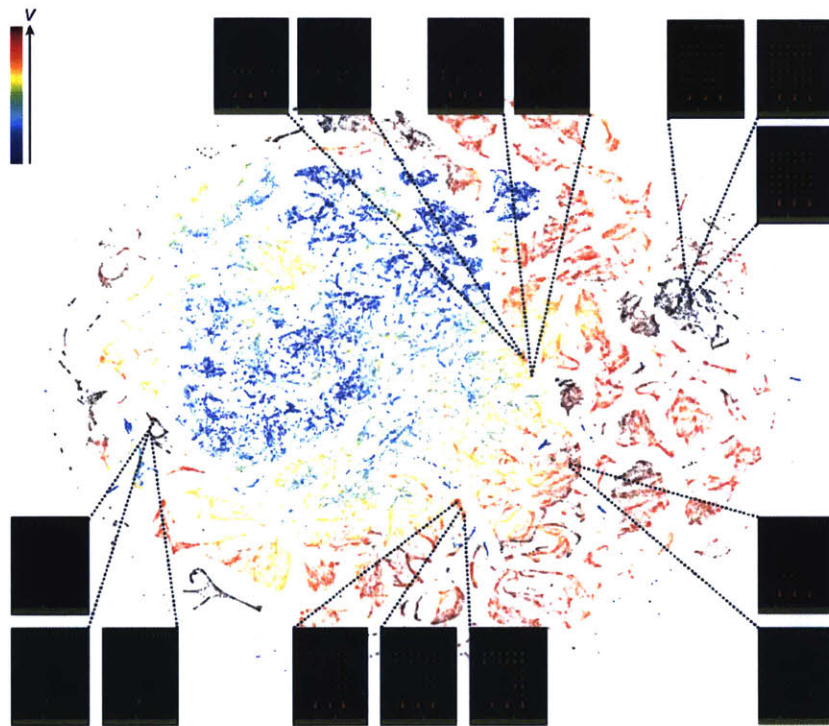


Figure 7 | Visual Representations for game states in the last layer of the DQN [34] |

losing the overall sensitivity of the gaming environment. This means that agents can choose to act at their own pace without being penalized for sticking to the time lines that the state of the game uses for transitioning. However, this does not mean that the games should allow completely inactive agents that may choose not to respond for long durations of the play essentially wasting time and resources. One way to do this would be to negotiate on the inactivity timeout with the agent during the start of the games. It has been seen in some of the benchmark games (such as Tennis and Boxing) that agents that choose not to act and collect negative points fare better in some games where exploration is very expensive. Such games should be able to negotiate timeout in the beginning of the game to save resources at the same time admit agents with varying speeds.

- **Enhancement to planning and goal based games**

The ALE platform serves very well for reinforcement learning based agents but planning agents that do not operate on intermediate rewards but rather work for a final goal are not well suited to the current set of games. It would be worthwhile adding planning based games that are solvable through a procedural programming language used in planning and searching.

- **Porting games to standardized format**

As discussed in earlier section on complexity measurement, it would be beneficial to port the games to a format such as Video Game Description Language (such as PyVGDL). This will not only allow for more visibility into game structure and complexity but also encourage modifications and enhancement of these games new features.

8 Conclusion

In a discussion about the nature of intelligence, Professor Gerald J. Sussman of MIT CSAIL compared intelligence as we know it to a sphere with several irregular surfaces much like our planet. There cannot be a single coordinate system to describe the entire sphere surface area, but we can only try to map out the flat patches that we can find using a known coordinate system. Intelligence as we know it at this point is a complex phenomenon that has several dimensions. Therefore a single definition and a single test would most certainly fail to capture all its properties. But that does not mean that we give up our quest to map the surfaces just as we explored our planet part by part since the middle ages until the Renaissance ending in the 16th century when the first maps of the entire planet started to appear.

There is a diverse community of computer scientists, roboticists , psychologists, neurologists, network theorists and several other communities of scientists and industry researchers trying to understand and advance artificial intelligence related technologies. Though, it is the need of the hour to bring a common understanding and tools for the diverse communities to be able to work toward a common purpose, this goal is not easy due to complex nature of intelligence. But this has to be an ongoing task in order to push the boundaries of inter-disciplinary research. Development of a common standard benchmark tests for various known and currently understood aspects of intelligence is therefore an important task for progressing this field. It is important to start bringing common understanding for various aspects of intelligence by discussing about benchmarks and features of intelligence.

The framework that has been proposed in this thesis tries to translate tests proposed by AI theorists into practical features. Several ideas has been taken from various sources and collated together to make the features as comprehensive as possible. This is a starting point and needs multiple extensions for other features related to other aspects of intelligence. Again, several features that relates to anthropomorphic aspects of intelligence has been left out. The psychophysical aspect of intelligence in human and animals have been left out as well. So are the physical, atheletic and dexterous movement related aspects of

intelligence that are pertinent for robotics. However, several core ideas about challenging and complex environments and learning are applicable in other aspects of intelligence tests. The features are biased toward reinforcement learning based agents and need to be expanded to incorporate planning and search based agents. This is not hard, the new tests can be added that replace reward based

We did a case-study of the ALE framework which is just a starting point as a benchmark. It has several limitations as we observed but the biggest limitation is perhaps physical. This framework is only defined in software making it extremely hard to incorporate physical tests including those used by the robotics community for instance. Perhaps virtual reality is a promising technology as the tests and environments can be extend to virtual physical space. We proposed several extensions to the platform deriving features from the framework that we synthesized earlier. We hope that the proposed extensions are meaningful to the ALE development and user community to further the development and adoption of their platform to the members of the AI community. That being said, these proposed extensions by no means provide an exhaustive test for general intelligence. There is no meta-learning for example, there is no agent that can essentially step out of the test environment and halt if necessary as we discussed in the MU problem. The usefulness of these extensions can only be thought about as a way to move a step forward. We are many such steps away from getting even close to success on a full-fledged Turing test. We also learned that the usefulness of mathematical generalizations is severely limited. According to the no free lunch theorem, a single agent cannot possibly find optimal solution for every set of environment without compromising performance on some classes of problems. The other limitation of mathematical generalization is its un-computability due to lack of physical constraints. This thesis tried to take the ideal features proposed mathematically and translated them to physical extensions into an upcoming test platform.

References

- [1] Shane Legg and Marcus Hutter. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and . . .*, pages 1–12, 2007.
- [2] Douglas R. Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., New York, NY, USA, 1979.
- [3] A. Binet and Th. A. Simon. Mthode nouvelle pour le diagnostic du niveau intellectuel des anormaux. *L'Anne Psychologique*, 11:191–244, 1905.
- [4] Kirk a Becker. Stanford-Binet Intelligence Scales , Assessment Service Bulletin Number 1 History of the Stanford-Binet Intelligence Scales : Content and Psychometrics. *Intelligence*, (1):14, 2003.
- [5] William Stern. Die psychologischen Methoden der Intelligenzprüfung und deren Anwendung an Schulkindern. *The Journal of Nervous and Mental Disease*, 43(4):388, 1912.
- [6] David Wechsler. The measurement and appraisal of adult intelligence . 1958.
- [7] John Raven. The raven’s progressive matrices: change and stability over culture and time. *Cognitive psychology*, 41(1):1–48, 2000.
- [8] Thomas R. Zentall. Animal intelligence. In Robert J. Sternberg and Scott Barry Kaufman, editors, *The Cambridge Handbook of Intelligence*, pages 309–327. Cambridge University Press, 2011. Cambridge Books Online.
- [9] Louis M. Herman, Adam A. Pack, and Amy M. Wood. Bottlenose dolphins can generalize rules and develop abstract concepts. *Marine Mammal Science*, 10(1):70–80, 1994.
- [10] Dana Princiotta and Sam Goldstein. Handbook of Intelligence. *Handbook of Intelligence: Evolutionary Theory, Historical Perspective, and Current Concepts*, pages 83–92, 2015.
- [11] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.
- [12] Ned Block. Psychologism and behaviorism. *The Philosophical Review*, 90(1):5–43, 1981.

- [13] John R Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(03):417–424, 1980.
- [14] Ayse Pinar Saygin, Ilyas Cicekli, and Varol Akman. Turing test: 50 years later. In *The Turing Test*, pages 23–78. Springer, 2003.
- [15] Robert J Sternberg and Elena L Grigorenko. *Dynamic testing: The nature and measurement of learning potential*. Cambridge university press, 2002.
- [16] Philip Nicholas Johnson-Laird and Peter Cathcart Wason. *Thinking: Readings in cognitive science*. CUP Archive, 1977.
- [17] Shane Legg and Marcus Hutter. Universal Intelligence : A Definition of Machine Intelligence. 2007.
- [18] Stuart Russell and Peter Norvig. Ai a modern approach. *Learning*, 2(3):4, 2005.
- [19] David L Dowe and José Hernández-Orallo. How universal can an intelligence test be? *Adaptive Behavior*, 22(1):51–69, 2014.
- [20] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.
- [21] Shane Legg. Machine super intelligence. *Doctoral Dissertation*, (June), 2008.
- [22] Jonathan Daniel Sorg. The Optimal Reward Problem : Designing Effective Reward for Bounded Agents by. 2011.
- [23] Alan M Turing. On Computable Numbers With an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society Series 2*, 42:230–265, 1936.
- [24] R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7(1):1–22, 1964.
- [25] Yann LeCun and John S Denker. Natural versus Universal Probability Complexity, and Entropy, 1992.

- [26] Shane Legg and Marcus Hutter. Universal Intelligence : A Definition of Machine Intelligence. 2007.
- [27] David L Dowe. Measuring Universal Intelligence : Towards an Anytime Intelligence Test. 2010.
- [28] Marcus Hutter. Open problems in universal induction and intelligence. *Algorithms*, 2(3):879–906, 2009.
- [29] LA Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.
- [30] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005.
- [31] Marcus Hutter. Universal Sequential Decisions in Unknown Environments. page 2, 2003.
- [32] Arkady Epshteyn, Adam Vogel, and Gerald DeJong. Active reinforcement learning. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 296–303, 2008.
- [33] Shimon Whiteson, Brian Tanner, Matthew E Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 120–127. IEEE, 2011.
- [34] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. 47:253–279, 2012.
- [35] Tom Schaul, Julian Togelius, and Jürgen Schmidhuber. Measuring Intelligence through Games. *Intelligence*, 2(1):1–19, 2011.
- [36] G M Hayes. Behavior-Based Reinforcement Learning. pages 1–28, 1992.

- [37] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. pages 1–28, 2016.
- [38] Bill Hibbard. Bias and No Free Lunch in Formal Measures of Intelligence. *Journal of Artificial General Intelligence*, 1(1):54–61, 2009.
- [39] Drew Bagnell and Andrew Ng. On Local Rewards and Scaling Distributed Reinforcement Learning. *Advances in Neural Information Processing Systems*, 18:91, 2006.
- [40] Lee Se-dol. Artificial intelligence: Google’s AlphaGo beats Go master Lee Se-dol. *BBC News*, pages 1–17, 2016.
- [41] Ming Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.
- [42] Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. *IEEE SSCI 2011: Symposium Series on Computational Intelligence - ADPRL 2011: 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 120–127, 2011.
- [43] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [44] Francisco S Melo, Sean P Meyn, and M Isabel Ribeiro. An analysis of reinforcement learning with function approximation. *Proceedings of the 25th international conference on Machine learning*, pages 664–671, 2008.
- [45] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.
- [46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King,

Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[47] Marc Ebner, John Levine, and SM Lucas. Towards a video game description language. *Dagstuhl Follow- ...*, 6:85–100, 2013.

[48] Tom Schaul. A video game description language for model-based or interactive learning. *IEEE Conference on Computational Intelligence and Games, CIG*, 2013.

[49] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.