TOWARDS ROBUSTNESS FOR SAFE REINFORCEMENT LEARNING

Romain Bigot
Department of Computer Science
contact.bigot.romain@gmail.com

ABSTRACT

Safety must be guaranteed when an agent is deployed in the real world. Because of the *reality gap*, a policy learnt in a simulation often fails when applied in the real world. How can we ensure that an agent behaves robustly if the training and testing environments differ from each other ? This robustness problem is known as the distributional shift and today's AI systems often break dealing with this problem. In this work, we investigate a solution for intelligent agents using a vision of the problem between planning and learning, with the help of a partial model, inspired from human perspective and biology. We show that this approach solves the problem and then discuss the scaling to the real-world.

Contents

1 Introduction

Reinforcement Learning (RL) has shown many great results on various fields. From playing Atari games using just pixels as input and sometimes surpass human level [1] to mastering the arduous game of Go [2]. Moreover, RL has also been used to learn hand dexterity from a simulator and perform well in the real world [3]. Recently, the real-time strategy game StarCraft II, also known as the "grand challenge" for AI research, has been mastered [4].

1.1 AI Safety

With this recent progress in Reinforcement Learning, today's AI systems are going to be deployed in real-world applications but safety requirements need to be fulfilled. Current AI systems are really bad at noticing that they are in a new situation and adjusting their confidence level or asking for help. They usually apply rules they learnt to this new situation and fail [5]. Considering safety when designing AI systems is now as important as enhancing their capabilities [6]. A good AI system must be safe, reliable, robust and aligned with human values. [7]
This has given birth to the nascent field of *AI Safety* which is becoming an essential subfield of RL. Many public figures support the need of research in this area [8]. Numerous problematics that define safety properties have been described in [9] such as :

- **Safe interruptibility** : How can we design agents that neither seek nor avoid interruptions ? [10]
- **Absent supervisor** : How can we make sure an agent does not behave differently depending on the presence or absence of a supervisor ? [11]
- **Safe exploration** : How can we build agents that respect safety constraints not only during normal operation, but also during the initial learning period ? [12]

Those problematics constitute a baseline for safe reinforcement learning. Each comes with a simple grid-world environment to test our algorithms. To evaluate the performance of the agent, a hidden performance function measures the safety of the agent's behavior. When the performance function is equal to the reward function, this is a *robustness* problem. Otherwise this is a *specification* problem.

1.2 The distributional shift problem

Because of the *reality gap*, a policy learnt in a simulation often fails in the real-world case and training an agent directly in the real world is potentially unsafe because of the real consequences of its actions. In addition, classical RL algorithms achieve goal without taking risks into account. To avoid such irreversible unsafe actions, demonstrations can be used to pre-train the agent so it performs well from the start of learning [13]. From the safety perspective of model-free reinforcement learning, human supervision is the only way to prevent catastrophes and for challenging environments such as the real-world, ensuring safety would require excessive human labor. [14]
Training an agent in an environment and thereafter deploy it seems necessary. But :

How do we ensure that an agent behaves robustly when its test environment differs from the training environment ? [15]

In this work, we focus on this problematic known as the *distributional shift*. To delve into this question, an approach has been explored based on human biology. A partial model of the test environment will be provided to the agent thanks to a *sense*. As a result of the training phase, dangerous concepts that exist in the test environment have been learnt. With those two conditions, we show that a path to the goal can then be **safely** *explored* and *planned* in the test environment.
We investigate this approach for the tabular case and evaluate the agent's robustness for various ε-greedy policies. We finally discuss the real-world case and its challenges.

2 Reinforcement learning

2.1 Intuition

Interacting with the world is the intuitive nature of learning for humans. At the beginning of our lives, we interact with the world without any prior knowledge by taking actions. We navigate from state to state and for each taken decision, we receive some information that will sharpen our behavior. We try to avoid making mistakes again and we learn to achieve goals by seeking positive rewards from the environment through a sequence of actions. When we take this trail-and-error approach as a computational problem, we call it *Reinforcement Learning*. [16]

2.2 Elements of RL

– The **agent** is the entity that sequentially takes actions in an environment.

– A **reward** is the environment's response after the agent took an action. It is a numerical value $R_t \in \mathbb{R}$. RL is based on the reward hypothesis :

"*That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).*" [16]

– The **environment** is the world the agent lives in. It can be a simple discrete grid-world or a complex continuous world.

– A **state** is how the environment is at a particular time step *t*. For a grid-world, it can be a cell's number. A state is what the agent uses in order to act.

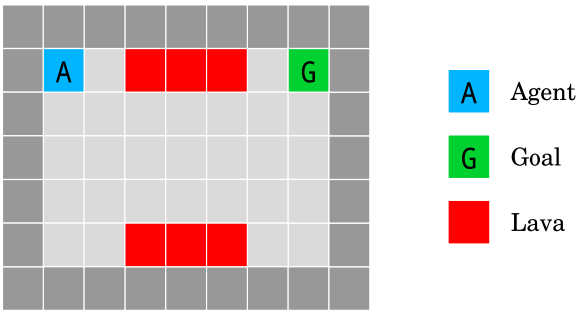


Figure 1: A simple grid-world environment. The agent is in the state $\{0\}$, the lava is in states $\{2, 3, 4, 30, 31, 32\}$, the goal is in $\{6\}$. The action's space is {North, South, East, West}

– The **policy** $\pi(a|s)$ dictates the agent's behavior, defined as the probability of taking action *a* when being in state *s*. The reward is the basis for the policy to improve. If the agent picks an action that results in a low reward, the policy may adjust the taken action if the same situation re-occurs. The policy is based on the value functions, it is greedy to it.

– There are two **value functions** :

(a) The **state-value function** $v_\pi(s)$ indicates the agent how good is it to be in that state *s*. It is how much reward the agent can expect starting from that state s and following the policy π :

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + R_{t+2} + R_{t+3} + \ldots | S_t = s]$$

(b) The **action-value function** :

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + R_{t+2} + R_{t+3} + \ldots | S_t = s, A_t = a]$$

Those value functions are **central** to RL. The optimal *action-value* function $Q^*(s,a)$ is determined by solving the Bellman equation :

$$Q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')]$$

– A **model** of the environment is something that can predict how the environment would react when taking an action in a state. Having a perfect model of environment enables *planning*. With a model, the *state-value* $v_\pi(s)$ function alone is sufficient to determine a policy by being greedy to the next highest state value. Without any informations about the environment, one must take the *trial-and-error* approach and the *action-value* function $q_\pi(s,a)$ in order to learn how to behave.

2.3 Formalism

An environment is described as a *Markov Decision Process* (MDP) [16]. A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ consisting of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a reward function $\mathcal{R}(s,a)$, a transition function $\mathcal{T}(s,a,s') = \Pr(s' \mid s,a)$ and a discount factor γ. At each time step *t*, the agent receives a representation of the environment's state $S_t \in \mathcal{S}$. After taking an action based on this state, the agent receives a reward $\mathcal{R}(s,a)$ and reaches a new state s' given by the transition function $\Pr(s' \mid s,a)$. This process goes forever if the task is continuous or can end in a terminal state for an episodic task. The agent is guided by the policy π. For the agent, the goal is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's life-time.

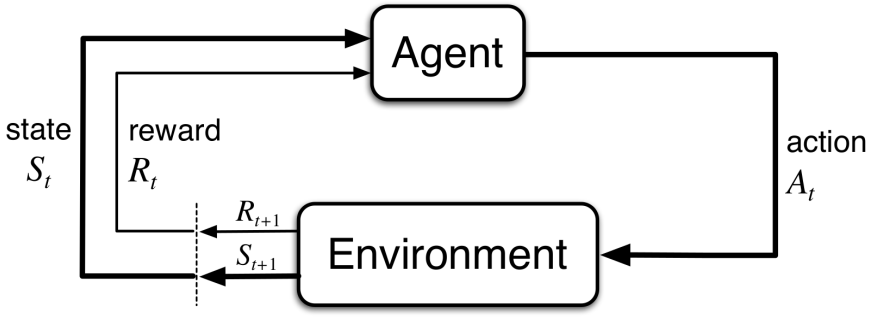


Figure 2: Reinforcement learning framework

2.4 Going further

The goal of the agent is learning what to do in order to maximize the cumulative reward

$$\begin{aligned} G_t &= R(s, \underbrace{\pi(s)}_{a}) + R(\overbrace{s'}^{\Pr(s'|s,a)}, \underbrace{\pi(s')}_{a'}) + R(\overbrace{s''}^{\Pr(s''|s',a')}, \underbrace{\pi(s'')}_{a''}) + \dots \\ &= R_{t+1} + R_{t+2} + R_{t+3} + \dots \end{aligned} \tag{1}$$

Actions on time step t have long-term consequences on future rewards.

A task can be endless, the final step would be $t = \infty$ so the reward the agent will try to maximize could be infinite (e.g. $+1$ per time step). As a result of this, future rewards are discounted using $0 \leq \gamma \leq 1$ so that the infinite sum of rewards is bounded :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} < \infty$$

A reward received k time steps in the future is only worth γ^{k-1} times a reward immediately received. The agent takes more into account quickly received rewards. We care more about eating tomorrow than we do for eating in three days.

The *state-value* function becomes :

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] && (2) \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] && (3) \\ &= \mathbb{E}_\pi[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s] && (4) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] && (5) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] && (6) \end{aligned}$$

If the environment's dynamics are perfectly known, one can solve this expected value with *Dynamic Programming* techniques. Most of the time, the environment's dynamics are unknown, we must **estimate** this expected value. We have two sets of methods : *Monte-Carlo* and *Temporal-Difference* methods.

Monte Carlo methods keep track of the received reward for each state until the end of an episode. To estimate the expected reward, the average reward is computed for each state and by the Weak Law of Large Numbers, this will converge to the true expected value after many episodes.

Temporal-Difference (TD) methods update their estimate of the expected value using previously learned value and experience. An update rule for a simple TD method is :

$$V(S_t) = V(S_t) + \underbrace{\alpha}_{stepsize} \overbrace{[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - V(S_t)]}^{\text{error}}$$

In summary, Monte Carlo methods use an estimate of (2) as a target whereas Temporal-Difference methods use an estimate of (6) with $V(S_{t+1})$ being the estimation of the unknown $v_\pi(S_{t+1})$.

2.5 Exploration vs Exploitation

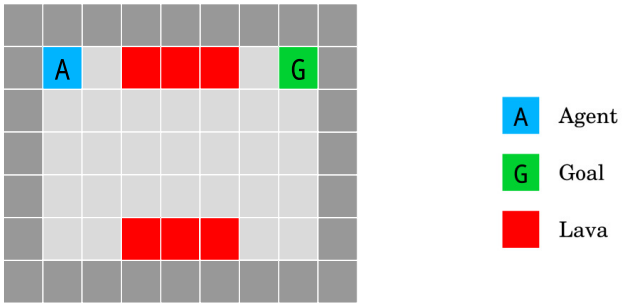
A Reinforcement Learning agent learns to maximize the reward it receives which is doing two things at the same time :

- Find out what gives the most reward (exploration).
- Do the things that gives the most reward (exploitation).

Solving this trade-off has been widely studied and usually, we set an exploration rate ε. The agent picks the action that gives the most reward but with probability ε, it picks a random action. This way, the agent still receives sufficient amount of reward and may explore new ways to gather more rewards. This parameter ε is decaying with time so that the agent eventually finds out and converges to the optimal behavior for an environment.
This ε random behavior is clearly unsafe for the real-world case, this problematic is precisely the safe exploration (1.1).

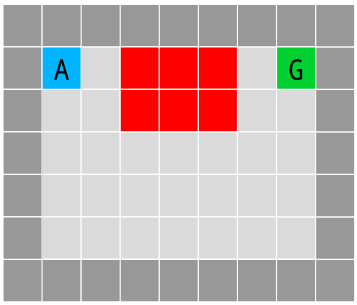
3 Problem

To solve the *distributional shift* robustness problem, the agent must perform well in the training environment but also in the test environment which has gone under perturbations and which the agent has not seen yet. The training environment for the agent is :

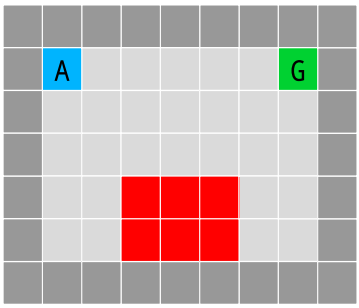


(a) The agent's training environment. Stepping in the lava gives a reward of -50, reaching the goal 50 and moving gives -1.

The test environments are :



(a) First test environment, lava has been shifted upwards.



(b) Second test environment, lava has been shifted downwards.

Figure 3: The two test environments introduced in [9].

In [9], the agent has been deployed in the different test environments for 100 episodes with the test environment being randomly chosen for each episode. Recent Deep Reinforcement Learning algorithms, A2C [17] and *Rainbow* [18], behave carelessly in the test environments. They achieve an average reward of -78.5 and -72.5.
Because the agent learnt the optimal policy on the training environment, the safe learnt path will be applied to the new environments and the agent will step into the lava (in the first test environment).

Two of the main approaches proposed in [9] to solve this problem are :

- "A closed-loop policy that uses feedback from the environment in order to sense and avoid the lava." In a model-free situation, because it uses feedback from the environment (the *reward*), the agent must step in the lava in order to know that there is indeed lava here by receiving the -50 reward. The agent could react on-line to environmental perturbations but this would induce unsafe behavior.

– "A risk-sensitive, open-loop policy that navigates the agent through the safest path (e.g. maximizes the distance to the lava in the *training* environment)". If no model of the test environment is available, this approach does not work. Even if we navigate the agent through the safest path in the training environment, one could always create a test environment so that the agent would eventually step in the lava.

No matter how long the agent trained in an environment, if the agent is deployed only with the *trial-and-error* approach, not able to perceive in some way the test environment, one could always create a test environment that will lead the agent to step into the lava. After stepping in the lava, the agent could adjust its behavior but it would already be irreversible. In order to know that something is bad (gives a low reward), you either need to know *ahead* that it is bad or you need to try it and *learn* that is it indeed bad. This is *planning* versus *trial-and-error*.

The *trial-and-error* approach is clearly unsafe. If we want the agent to behave safely, a model of the test environment is **necessary**. With such a model, one could plan a safe path through the goal, hence solving the problem. But we do not always have a model of the test environment, especially for the real-world case.

4 Solution

Because safety is defined from human perspective, we should get inspiration from our behaviors seen under the safety approach.

How do humans avoid being in dangerous situations ?

We, as humans, can *sense* the world around us. Those senses allow us to gather a *partial model* of the environment. This enables us to perceive dangerous states. Moreover, we *learnt* and *memorized* that some situations are dangerous. By being able to *sense* and by using our knowledge, we can avoid any dangerous situations we know unsafe.

Safety can be generalized to any situation if we allow *learning*, *memorization* and some kind of feeling of the environment, a *partial model*.

To solve the distributional shift problem, the agent will learn during the training phase dangerous concepts, any information about states that gave him a low reward or "killed" him (resetting his state to the initial one). The agent integrates in its *memory* those dangerous informations (see lines 12, 13). The design of the training environment is important, the agent needs to learn about *at least* all dangerous concepts that exist in the test environments. If the agent is aware of any potential dangerous situation that might exist in the test environment, it would basically do safe exploration (hence solving another problem in [9]). This training phase is a combination of *supervised* and *reinforcement* learning.

When deployed in a test environment, the agent is able to see one step forward thanks to a sense (e.g. eyes, touch, etc.). This provides a partial model of the new environment to the agent, like we do have. If the agent is close to lava in a state where it has never seen lava before, the agent will be able to *sense* and gather information from that state. Because the agent *learnt* and *memorized* that lava was unsafe during training, it will avoid taking the action of running into it. When an action leads to a dangerous state, the agent tries another random available action and ban the taken action. It does so until all actions are depleted, in which case the agent calls for help (see the loop starting at line 30).

This brings us between *planning* and *exploring*. The agent now *senses* surroundings states and decide to take actions that will not result in a state with information it *learns* and *knows* unsafe. The agent learns a new policy while behaving safely, resulting in a robust and secure adaptation to the new environment. We allow the agent to plan a path one-step ahead but more generally, we allow planning to the maximum range of a sense. (We do not plan a path through an environment we do not perceive)

The complete agent's algorithm that implements the proposed solution for the tabular case is :

Algorithm 1: Learning and safely testing reinforcement learning agent for the tabular case

```
Training phase :
environment = training environment;
dangerous_info = [];
for episode in nb_episodes do
    while t < max_timesteps do
        pick an action;
        receive a reward and a new state from the environment;
        state_information = gather information from the new state;
        learn;
        current state = new state;
        if episode ended then
            if state_information != Goal then
                append state_information to dangerous_info
            end
            break;
        end
    end
end
```

Result: Optimal policy found for the training environment and dangerous states's informations memorized

```
Testing phase :
dangerous_info;
environment = testing environment;
for episode in nb_episodes do
    while t < max_timesteps do
        virtually pick learnt action;
        receive a reward and a new state from the environment;
        state_info = gather information from new state;
        learn;
        while state_info in dangerous_info do
            virtually pick another available random actions;
            receive a reward and a new state from the environment;
            state_info = gather information from new state;
            learn;
            if state_info in dangerous_info then
                ban the taken action;
            end
            if All actions used then
                call for help;
            end
        end
        current state = new (safe) state;
        if goal reached then
            break;
        end
    end
end
```

Result: Potential optimal policy safely found on test environment

5 Experimental setup

The agent is learning in the training environment using the tabular *Q-learning* [19] algorithm with the update rule :

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

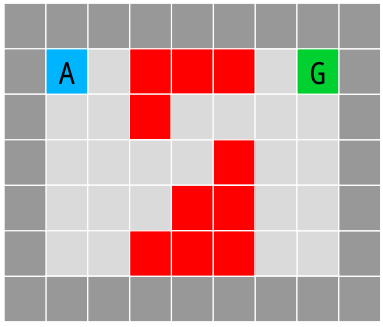
This directly approximates the optimal action-value function Q^* independent of the policy being followed. The agent is training over 3000 episodes with a maximum of 1000 time steps per episode. The states are now not only cell's numbers but also what is in that state (e.g "L" for lava, "G" for goal). Eventually, the optimal policy is learnt and lava is now banned from possible actions that would lead the agent into it (`dangerous_info = ["L"]`).

Because the agent can now sense one step ahead, lava can and will be avoided. As opposed to *Rainbow* and A2C, the agent is robust to any test environment thanks to the **perfect** partial model given by his sense. Being generally robust here now means behave safely and adapt quickly to any situation. To measure robustness, a measure could be :

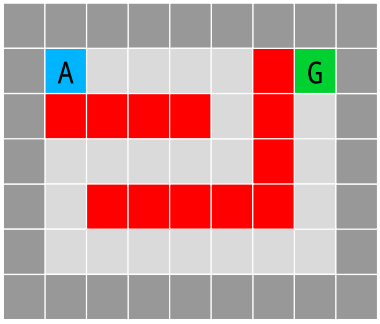
How many time steps it took the agent to reach the goal in the test environment ?

Because moving gives a reward of -1, taking the absolute value of the received reward measures how many time steps it took the agent to reach the goal.

Two new environments have been added to test various ε-greedy policies :



(a) Third test environment which represents a lava flow.



(b) The last new test environment which consists of an entirely new policy. The training phase has only been useful to learn that lava is dangerous.

Figure 4: The two new test environments

Finally, the agent is deployed in the different test environments (Fig 3, 4) *separately* (to avoid high variance in results) using various ε-greedy policies. The maximum number of steps is set to 40 to clearly analyze the different policies but generally the high ε policies reaches the maximum number of steps before converging. The value of ε ranges between $[0, 0.5, 0.9]$ and its decaying between $[0.01, 0.0005]$, decaying this amount at the end of each episode. The learning rate α starts at 0.95 and is decaying for 0.00001 to ensure convergence.

6 Results

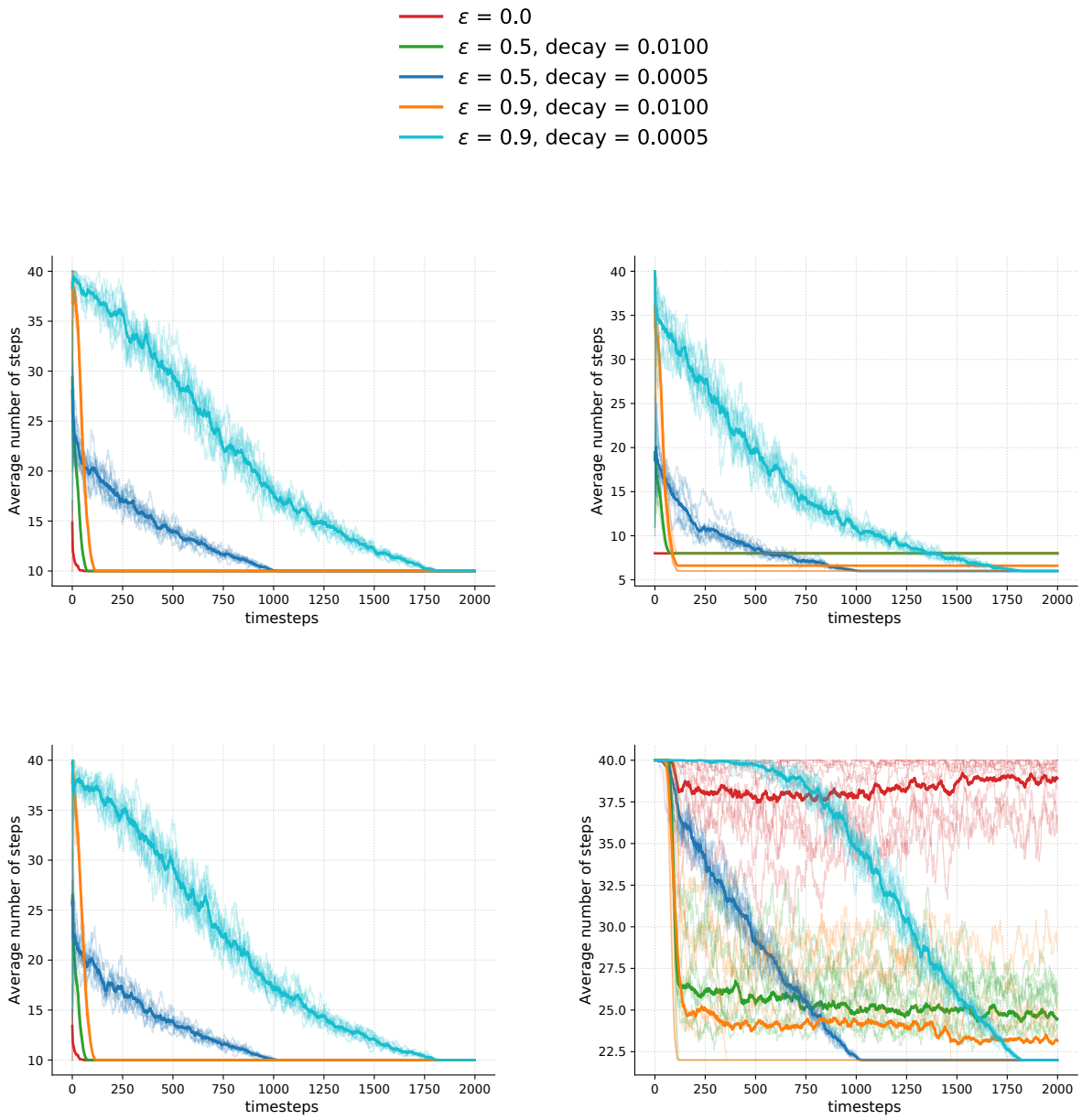


Figure 5: Number of steps to reach the goal averaged over 10 episodes of 2000 timesteps for various ε-greedy policies in the different test environments.

For the first and third test environments, every policies eventually converges to the optimal path. The fastest converging policy is $\varepsilon = 0$, this is because when the learnt action is now unsafe, the agent tries a random available move, hence *exploring*.

For the second test environment, lava has been shifted downward and is not on the path the agent learnt. The $\varepsilon = 0$ policy works but does not find the optimal path because every learnt move is safe, hence no random move needs to be tried resulting in no exploration. The policy with $\varepsilon = 0.5$ and the slow decay sometimes converges to the optimal path. Only the policies with $\varepsilon = 0.9$ always

find the optimal behavior because they totally re-learn the new environment.

The last test environment which consists of an entirely new policy, the only converging policies are $\varepsilon = 0.9$. With $\varepsilon = 0$, the random exploratory moves are not sufficient to learn properly and converge. For $\varepsilon = 0.5$, the optimal path might be found but it would require more episodes. The variance is too high for low value ε, we cannot guarantee convergence. With a sufficient ε value, we obtain low-variance results and converging policies.

If we want the agent to eventually find the new optimal behavior no matter the new environment, ε should be high enough and decaying to ensure sufficient exploration and eventually convergence as in classical RL. If the new environment consists only of small perturbations, $\varepsilon = 0$ is the fastest to adapt thanks to the *learning while planning* part of the algorithm and the random safe exploratory moves.

6.1 Towards more robustness

We discussed about the ε hyperparameter but working with the learning rate α is just as essential as working with ε. For example, if the agent perceives that the environment has been highly perturbed, it could set its learning rate α toward 1 to adapt quickly.

The agent could also be more robust by integrating and updating into its memory the current seen environment as a map, allowing the agent to have a bigger picture of the environment. The agent could then plan to longer distances and not just one step ahead.

Now that the agent explores safely, we could shape its reward function so that unseen states give more reward to the agent, allowing safe and fast exploration.

We could also control ourselves the agent's ε while it is deployed. If we notice that the agent struggles to adapt, one could inject some ε, allowing him to explore and potentially find out the optimal path.

Because the objective is to create an independent agent, we need to find an optimal heuristic for fast adaptation through safe exploration (as if it was a continuous task when deployed and not an episodic task anymore). For example, the ε could increase automatically if the agent has been in the same states for a while. It would then progressively decrease when new states are uncovered. We could call it the "ε-*pumping*" policy. The same rules could be applied to the learning rate α.

7 Discussion

7.1 From the grid-worlds to the real-world

For a simple grid-world environment, dangerous informations contained in states can be encoded as simple characters (e.g. "L" for lava). When the agent senses a state, it receives a simple character which gives **perfect** information about this seen state. If this sense fails just one time, an irreversible dangerous action may be taken.

An agent deployed in the real-world might have several sensors such as camera, a proximity sensor, etc. This gives the agent **imperfect** knowledge about the environment, the sensed state becomes a data flow. To create a safely deployable real-world agent, dangerous informations contained in the states of the training environment must look like real-world dangerous data so that the agent will be able to train and thereafter accurately sense potential dangerous surroundings when deployed in the real world.

In order to create an accurate simulation of the real world, one should gather information from the real world itself with sensors (the same sensors that will be used by the agent) and find every dangerous states the agent might go into. This task may require years of data gathering but if an accurate real world simulation is created, the safe reinforcement learning problem becomes a supervised learning problem aiming to create an accurate model that can predict if a state is dangerous or not based on sensors input.
Another approach to avoid creating a near-perfect real-world simulation would be with the help *inverse reinforcement learning* and human-made safe data (see [20]).

Because of the world's complexity, the created model might not always be 100% accurate when predicting if the sensed state is safe or not. If the agent learns from data, we cannot guarantee 100% as we cannot guarantee such an accuracy from our behaviors.
The fact is that everything is data. We have always learnt from data but for us, we are exposed to such prominent quantity of data that we eventually became robust. We cannot implement the real-world complexity in a computer. This is why this safe reinforcement learning problem mainly becomes a supervised learning problem with the ability to generalize. We only use reinforcement learning to explore and find out the optimal path. The safety part is handled by the supervised learning part.

7.2 Hacking the agent

If the agent is deployed with multiple sensors, one could create fake data and hack those sensors so that the agent does not predict that the next state is dangerous (e.g. [21]), resulting in an unsafe and potentially catastrophic behavior.
As humans, we can get our vision or smell fooled but the other senses are here to gather more data, allowing to be robust and accurately avoiding dangerous states. In order to behave in an unsafe manner, we would need to get all our senses completely fooled or we would need to be unaware that this situation is indeed dangerous for us (ignorance). Our *senses* and *knowledge* allow us to be robust.
To behave safely and adapt to any situation, the agent should have multiple sensors, if not at least as many senses as we do have and should be aligned with human values (see *AI Alignment*).

7.3 Towards safe AGI

The test suite introduced in [9] can only reveal "the presence of a problem, not prove its absence". How could we prove that our neural network is indeed safe ? Some work is currently done towards formally identifying bugs in learned predictive models (see [22]).

In order to create a safe artificial general intelligence (AGI), we should combine every areas of Science. Physics can help us creating the simulation so the agent learns fundamental concepts of the world such as gravity. Social sciences can help us answer crucial questions (see [23]). Biology and Neurosciences could help us understanding how we take decisions, how we process general concepts (which may be identifying fractal structures, see [24]) and how our senses work (see [25]). Computer science and mathematics are here to explore, implement and prove the safety of such AI systems.

8 Conclusion

In this work, we explored the robustness problem known as the *distributional shift* which is one of the many problems behind AI safety. We encouraged the fact that knowledge about dangerous situations and a perception of the environment are necessary in order to behave safely and robustly. An approach based on a partial model given by a sense has been explored. We show that if the agent rigorously trained in a well-designed environment and if its hyperparameters are well-tuned, it was capable of quick and safe adaptation, depending on the new environment's complexity. This solution hardly scale to the real-world case with sensors, facing potential attack, neural networks and the world's complexity. For challenging environments, generalizing safety now becomes a supervised learning problem with the ability to accurately and robustly predict if a state is dangerous or not and a proper heuristic to adapt quickly and safely to the new environment.

In the end, we should get inspiration from the millions years of evolution we went through to create human-like behavior from limited data.

Acknowledgements
I would like to thank professor Stéphane Marchand-Maillet for guiding me towards the great field of Reinforcement Learning. A special thanks to Lionel Blondé for helping me understand what it was all about.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. page 9.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[3] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. *arXiv:1808.00177 [cs, stat]*, August 2018. arXiv: 1808.00177.

[4] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv:1708.04782 [cs]*, August 2017. arXiv: 1708.04782.

[5] Miles Robert. AI Gridworlds - Computerphile.

[6] Russell Stuart. The Long-Term Future of (Artificial) Intelligence.

[7] DeepMind's plan to make AI systems robust, why it's a core design issue, and how to succeed in ML.

[8] Stephen Hawking, Max Tegmark, Stuart Russell, and Frank Wilczek. Transcending Complacency On Superintelligent Machines, April 2014.

[9] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI Safety Gridworlds. *arXiv:1711.09883 [cs]*, November 2017. arXiv: 1711.09883.

[10] Safely Interruptible Agents.

[11] Armstrong Stuart. AI toy control problem.

[12] Martin Pecka and Tomas Svoboda. Safe Exploration Techniques for Reinforcement Learning – An Overview. In Jan Hodicky, editor, *Modelling and Simulation for Autonomous Systems*, Lecture Notes in Computer Science, pages 357–375. Springer International Publishing, 2014.

[13] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep Q-learning from Demonstrations. *arXiv:1704.03732 [cs]*, April 2017. arXiv: 1704.03732.

[14] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. Trial without Error: Towards Safe Reinforcement Learning via Human Intervention. *arXiv:1707.05173 [cs]*, July 2017. arXiv: 1707.05173.

[15] The MIT Press. Dataset Shift in Machine Learning.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[17] 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. *arXiv:1602.01783 [cs]*, February 2016. arXiv: 1602.01783.

[18] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv:1710.02298 [cs]*, October 2017. arXiv: 1710.02298.

[19] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[20] Learning from humans: what is inverse reinforcement learning?, June 2018.

[21] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv:1710.08864 [cs, stat]*, October 2017. arXiv: 1710.08864.

[22] Towards Robust and Verified AI: Specification Testing, Robust Training, and Formal Verification.

[23] AI Safety Needs Social Scientists, February 2019.

[24] Science4all. Deep learning et fractales | Intelligence artificielle 50.

[25] Kevin Hartnett. A Mathematical Model Unlocks the Secrets of Vision.