# Risk Sensitivity with Exponential Functions in Reinforcement Learning: An Empirical Analysis

Eduardo Lopes Pereira Neto

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Master of Science

|  |  |
|---|---|
| Program: | Ciência da Computação |
| Advisor: | Prof<sup>a</sup>. Dr<sup>a</sup>. Karina Valdivia Delgado |

São Paulo
October, 2023

# Risk Sensitivity with Exponential Functions in Reinforcement Learning: An Empirical Analysis

Eduardo Lopes Pereira Neto

# Resumo

Eduardo Lopes Pereira Neto. **Sensibilidade ao Risco com Funções Exponenciais em Aprendizado por Reforço: Uma Análise Empírica**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

O Aprendizado por Reforço provou ser altamente bem-sucedido na resolução de problemas de decisão sequencial em ambientes complexos, com foco na maximização da recompensa acumulada esperada. Embora Aprendizado por Reforço tenha mostrado seu valor, os cenários do mundo real geralmente envolvem riscos inerentes que vão além dos resultados esperados, onde, na mesma situação, diferentes agentes podem considerar assumir diferentes níveis de risco. Nesses casos, o Aprendizado por Reforço Sensível ao Risco surge como uma solução, incorporando critérios de risco ao processo de tomada de decisão. Dentre esses critérios, métodos baseados em exponencial têm sido extensivamente estudados e aplicados. No entanto, a resposta de critérios exponenciais quando integrados com parâmetros de aprendizagem e aproximações, particularmente em combinação com Aprendizado por Reforço Profundo, permanece relativamente inexplorado. Essa falta de conhecimento pode impactar diretamente na aplicabilidade desses métodos em cenários do mundo real. Nesta dissertação, apresentamos um arcabouço que facilita a comparação de critérios de risco exponencial, como Utilidade Exponencial Esperada, Transformação Exponencial da Diferença Temporal e Transformação da Diferença Temporal com Soft Indicator considerando algoritmos de Aprendizagem por Reforço, como Q-Learning e Deep Q-Learning. Demonstramos formalmente que a Utilidade Esperada Exponencial e a Transformação Exponencial da Diferença Temporal convergem para o mesmo valor. Também realizamos experimentos para explorar a relação de cada critério de risco exponencial com o parâmetro de taxa de aprendizado, o fator de risco e os algoritmos de amostragem. Os resultados revelam que a Utilidade Esperada Exponencial apresenta estabilidade superior. Adicionalmente, esta dissertação analisa empiricamente problemas de estouro numérico. Uma técnica de truncamento para lidar com esse problema é analisada. Além disso, propomos a aplicação da técnica *LogSumExp* para mitigar este problema em algoritmos que utilizam a Utilidade Esperada Exponencial.

**Palavras-chave:** Processo de Decisão Markovianos. Aprendizado por Reforço. Sensivel a Risco. Utilidade Esperada Exponencial.

# Abstract

Eduardo Lopes Pereira Neto. **Risk Sensitivity with Exponential Functions in Reinforcement Learning: An Empirical Analysis** . Thesis (Master's). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Reinforcement Learning has proven to be highly successful in addressing sequential decision problems in complex environments, with a focus on maximizing the expected accumulated reward. Although Reinforcement Learning has shown its value, real-world scenarios often involve inherent risks that go beyond expected outcomes where, sometimes, in the same situation different agents could consider taking different levels of risk. In such cases, Risk-Sensitive Reinforcement Learning emerges as a solution, incorporating risk criteria into the decision-making process. Among these criteria, exponential-based methods have been extensively studied and applied. However, the response of exponential criteria when integrated with learning parameters and approximations, particularly in combination with Deep Reinforcement Learning, remains relatively unexplored. This lack of knowledge can directly impact the practical applicability of these methods in real-world scenarios. In this dissertation, we present a comprehensive framework that facilitates the comparison of exponential risk criteria, such as Exponential Expected Utility, Exponential Temporal Difference Transformation, and Soft Indicator Temporal Difference Transformation with Reinforcement Learning algorithms such as Q-Learning and Deep Q-Learning. We formally demonstrate that Exponential Expected Utility and Exponential Temporal Difference Transformation converge to the same value. We also perform experiments to explore the relationship of each exponential risk criterion with the learning rate parameter, the risk factor, and sampling algorithms. The results reveal that Exponential Expected Utility exhibits superior stability. Additionally, this dissertation empirically analyzes numerical overflow issues. A truncation technique to handle this issue is analyzed. Furthermore, we propose the application of the *LogSumExp* technique to mitigate this problem in algorithms that use Exponential Expected Utility.

**Keywords:** Markov Decision Process. Reinforcement Learning. Risk Sensitive. Exponential Expected Utility.

# List of Abbreviations

| | |
|---|---|
| ANN | Artificial Neural Network |
| CNN | Convolutional Neural Network |
| CVaR | Conditional Value-at-Risk |
| DP | Dynamic Programming |
| DQN | Deep Q-Network |
| EEU | Exponential Expected Utility |
| ETD | Exponential Temporal Difference Transformation |
| MDP | Markov Decision Process |
| QL | Q-Learning |
| RSMDP | Risk Sensitive Markov Decision Process |
| SITD | Soft Indicator Temporal Difference Transformation |
| TD | Temporal Difference |
| VaR | Value-at-Risk |
| VI | Value Iteration |

# List of Figures

# List of Algorithms

# Contents

# Chapter 1

# Introduction

Reinforcement Learning has emerged as a powerful framework for training intelligent agents and combined with Deep neural networks has revolutionized the field of artificial intelligence enabling agents to learn complex decision-making policies to make sequential decisions in complex environments (Sutton and Barto, 1998). Reinforcement Learning consists mainly in how an agent interacts with an environment described by a Markov Decision Process(MDP) taking actions that maximize expected rewards.

Although Reinforcement Learning has shown its value, real-world scenarios often involve inherent risks that go beyond expected outcomes where, sometimes, in the same situation different agents could consider taking different levels of risk. For example, consider an agent with two available actions: (i) the first deterministic action offering a guaranteed reward of $r$, and (ii) the second stochastic action with a 25% chance of winning a reward of $r - 1$, 25% chance of winning $r + 1$, and a 50% chance of receiving the same reward $r$. For a risk-neutral agent, both actions yield the same expected reward. However, in such circumstances, individuals often tend to be risk-averse, leading them to favor the option of selecting the deterministic action which has the lowest variation and a guaranteed reward of $r$ (Ralph L Keeney and Howard Raiffa, 1993). Conversely, when the agent is risk-prone, she would opt for selecting the stochastic action which has the chance of winning the greatest reward.

To tackle these challenges, Safe Reinforcement Learning employs two main approaches (Garcia and Fernández, 2015). One of the approaches focuses on adapting the exploration process to avoid actions that could lead the learning system into undesirable or catastrophic situations by incorporating external knowledge (Driessens and Džeroski, 2004; Abbeel *et al.*, 2010; Garcia and Fernández, 2012; Dalal *et al.*, 2018; Kidambi *et al.*, 2020; Swazinna *et al.*, 2021) or the use of risk-directed exploration (Gehring and Precup, 2013; Andersen *et al.*, 2020). The other approach involves modifying the optimality criterion to include the concept of risk. These optimization criteria can be divided into three main groups: (i) worst-case criterion (Heger, 1994; Littman and Szepesvári, 1996; Nilim and El Ghaoui, 2005), (ii) constrained criterion (Kadota *et al.*, 2006; Moldovan and Abbeel, 2012; HasanzadeZonuzy *et al.*, 2021), and (iii) risk sensitive criterion (Ronald A. Howard and Matheson, 1972; Chung and Sobel, 1987; Bouakiz and Sobel, 1992; Mausser and Rosen, 1999; Mihatsch and Neuneier, 2002; Morimura *et al.*, 2010; Cavazos-Cadena and Hernández-Hernández, 2011; Chow and Ghavamzadeh, 2014; Bäuerle and Rieder, 2014; Wood and Khosravanian, 2015; Tang *et al.*, 2019; Stanko and Macek, 2019; Fei, Yang, Yudong Chen, Wang, and Xie, 2020; Jin *et al.*, 2020; Delétang *et al.*, 2021; Fei, Yang, and Wang, 2021; Fei, Yang, Yudong Chen, and Wang, 2021; Du *et al.*, 2022; Xu *et al.*, 2023).

The application of Risk Sensitive Criteria in Reinforcement Learning has garnered significant attention, with exponential criteria being a subject of extensive research and analysis among various risk measures. Leveraging the expected utility theory, exponential utility criteria offers advantages, such as robustness to uncertainty, as it can effectively capture both high and low probabilities of rare events (Wood and Khosravanian, 2015).

Within the realm of exponential criteria, several studies have explored its application in different forms. Some have utilized the exponential function in the context of exponential expected utility (Ronald A. Howard and Matheson, 1972), while others have incorporated the exponential function into Temporal Difference transformations (Shen *et al.*, 2014). Additionally, some works have combined Temporal Difference transformations with a related exponential function known as the Soft Indicator function (Delétang *et al.*, 2021), revealing the close relationship between this function and the classical exponential function.

Despite the advantages of using exponential utility functions, there are two main drawbacks when trying to apply this function to real-world domains:

- Scalability and Computational Complexity: When exponential risk criteria are introduced to Reinforcement Learning, the computational complexity further increases and as the values grow exponentially it becomes a hard task to prevent numeric overflow (Bäuerle and Rieder, 2014; Shen *et al.*, 2014; Freitas *et al.*, 2020).
- Risk-Aware Policy Optimization: Calibrating the risk sensitivity parameter in exponential risk criteria presents a challenge in Reinforcement Learning. The choice of this parameter governs the trade-off between risk and reward and directly impacts the behavior of the agent.

Numerical overflow has been a focus of some studies, with efforts to address the issue through mathematical techniques such as *LogSumExp* (Naylor *et al.*, 2001; Freitas *et al.*, 2020). However, the application of this technique to Reinforcement Learning algorithms has not been extensively explored. Additionally, truncating techniques have been applied to prevent large numbers (Shen *et al.*, 2014).

Moreover, the incorporation of exponential utility functions with Deep Reinforcement Learning holds the potential to enhance real-world applications.

## 1.1 Motivation

While the application of the exponential function in Reinforcement Learning for risk modeling has been extensively explored in various studies (Ronald A. Howard and Matheson, 1972; Chung and Sobel, 1987; Cavazos-Cadena and Hernández-Hernández, 2011; Shen *et al.*, 2014; Delétang *et al.*, 2021), there is a lack in the literature that comprehensively investigates the key distinctions among the different methods of employing this function. Specifically, understanding how additional parameters influence the learning process in each approach remains relatively unexplored. By delving into these nuances, we aim to provide valuable insights into the behaviors and dynamics of various exponential-based risk modeling techniques, facilitating the selection of the most suitable method for specific Reinforcement Learning and Deep Reinforcement Learning algorithms.

Furthermore, the existing literature lacks a comprehensive examination of the primary challenges associated with implementing these exponential-based methods in real-world scenarios. To effectively apply these approaches in practical situations, a thorough understanding of their fundamental characteristics and stability is critical. Identifying potential obstacles and devising solutions to address them is crucial in ensuring the successful integration of risk modeling techniques based on exponential functions. Through this research, we seek to clarify these challenges, contributing to the development of more robust and efficient risk sensitive systems and adaptable to the uncertainties and complexities of real-life environments.

## 1.2 Objectives

The main objective is to explore the application of exponential functions to model risk attitudes in Reinforcement Learning and Deep Reinforcement Learning algorithms, focusing on the following goals:

- Explore different exponential risk criteria: The first goal is to establish a flexible framework that allows the comparison of exponential functions in different forms to model risk attitudes in Reinforcement Learning and Deep Reinforcement Learning algorithms. We explore different exponential criteria and different exponential functions (exponential utility and soft indicator), to accommodate diverse risk attitudes. This will enable decision-makers to express risk-averse, risk-neutral, or risk-prone behaviors, offering a richer spectrum of risk modeling options.

- Analysis of criteria and learning parameters: The second goal is to conduct an analysis of how different risk modeling criteria, and different exponential functions, interact with approximation and learning parameters in various Reinforcement Learning and Deep Reinforcement Learning algorithms. Understanding how these criteria influence learning dynamics, convergence properties, and algorithmic stability is crucial for designing efficient and robust risk-sensitive algorithms. By systematically exploring these interactions, we can identify the strengths and limitations of different risk modeling approaches.

- Combining Deep Reinforcement Learning and exponential risk criteria: In this work, we aim to propose an intuitive and more stable way of combining Deep Reinforcement Learning and exponential risk criteria and analyze how those exponential criteria interact with deep neural network approximation.

- Overcoming numerical challenges: The third goal is to address numerical challenges, such as overflow issues, that may arise when employing exponential functions, especially in high-dimensional and complex Reinforcement Learning and Deep

Reinforcement Learning environments. We aim to analyze the truncating technique and introduce a technique to apply *LogSumExp* across different classes of algorithms. These approaches ensure seamless integration of exponential-based risk modeling, promoting the reliability and scalability of risk-sensitive algorithms.

## 1.3 Organization

This work is organized as follows. In Chapter 2, we introduce the key concepts of Markov Decision Processes, Reinforcement Learning, and Deep Reinforcement Learning. Additionally, we present the main Reinforcement Learning algorithms.

In Chapter 3 the Safe Sensitive Reinforcement Learning area is presented with the main approaches to model risk. In Chapter 4 we focus on Risk Sensitive Reinforcement Learning with Exponential Functions. We present risk attitudes and introduce the exponential utility function. We demonstrate its application across Value Iteration, Q-Learning, and Deep Q-Learning algorithms. We modify these algorithms to integrate Exponential Expected Utility. Then, we apply the utility function to Temporal Difference, incorporating both the exponential function and a related function known as the soft indicator. Notably, our analysis reveals that applying the exponential function to both expected utility and Temporal Difference yields identical values. Concluding the chapter, we explore strategies to mitigate overflow issues.

In Chapter 5 the domains used and the results obtained in the experiments performed are presented. Finally, in Chapter 6 the final considerations and future works are presented.

# Chapter 2

# Reinforcement Learning

In this chapter, an overview of Reinforcement Learning and some of the main implementations that use this learning method is presented.

Reinforcement Learning consists of the interaction between an agent and an environment in a way that maximizes rewards over time. The agent does not have prior knowledge about the result of executing actions and must find out which actions generate more reward by interacting with the environment. In other words, the agent has knowledge of the current state ($s_t$) and the actions that can be taken, and when interacting with the environment taking a certain action ($a_t$) she receives a reward ($r_t$) and goes to a new state ($s_{t+1}$), as shown in Figure 2.1 (Sutton and Barto, 1998).

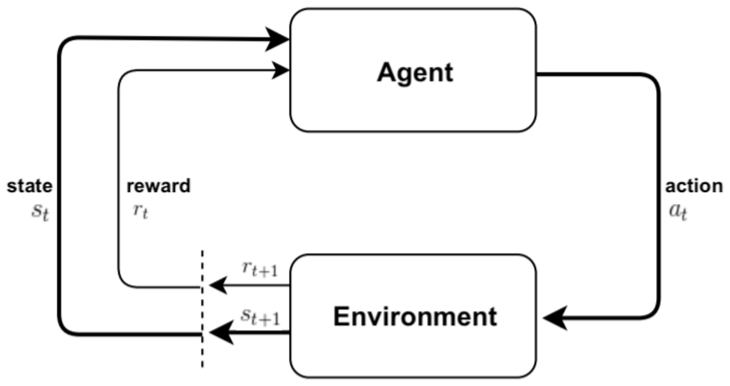


**Figure 2.1:** *Interaction between agent and environment.*

The agent's goal is to maximize the reward received throughout the execution regardless of the initial state $s_0$. This interaction between the agent and the environment can be modeled as a Markov Decision Process.

## 2.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical model widely used in decision-making problems and provides a mathematical framework to represent the interaction between an agent and an environment.

An MDP can be formally defined by a tuple $\langle S, A, R, T, s_0 \rangle$ (Puterman, 1994), where:

- $S$ is a finite set of states;
- $A$ is a finite set of actions that can be performed by the agent;
- $R : S \times A \rightarrow \mathbb{R}$ is the reward function that defines the reward when an action is taken in a state;
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function that defines the state transition probability; It is
- $s_0$ is a finite set of initial states which is a subset of $S$.

The agent's goal is to find the actions that, when executed in certain states, maximize the accumulated rewards. This function that relates states and actions to be performed is called a policy. This policy is represented by $\pi$ and can be stochastic or deterministic.

In a stochastic policy, for each state, there is a probability for each action to be taken while in a deterministic policy, for each state there is a single action to be taken.

Furthermore, a policy can be stationary or non-stationary. A stationary policy is a policy where the chosen action does not depend on the step at which the action is taken. In turn, in a non-stationary policy, the action chosen in a given state depends on the step at which the action is being taken.

An MDP can be classified according to its *horizon*, which is the number of stages through which the agent will make decisions, as being: (i) finite horizon when there is a certain number of steps to complete the process; (ii) infinite horizon, in this case, the MDP never complete its process; and (iii) undefined horizon when the number of steps is unknown, but the process can be finished when reaching some goal state.

Knowing the MDP horizon type, it is possible to find the value of a policy by calculating the amount of utility that we can expect from the execution of this policy (Mausam and Kolobov, 2012). Considering a discount factor $\gamma \in [0, 1)$ to ensure that the rewards are limited to a finite value for an MDP with an infinite horizon, we can define the value of a policy $\pi$ as the expected reward accumulated discounted:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t} | s_0 = s\right]. \tag{2.1}$$

The value function can be calculated by solving the system of equation (Bellman, 1957):

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s'). \tag{2.2}$$

By defining the value of a policy, it is possible to define the concept of an optimal solution for an MDP. The optimal solution of an MDP is an optimal policy $\pi^*$ such that this policy is at least as good as any other policy in all states given the same utility criteria, that is:

$$V^*(s) \geq V^\pi(s), \forall s \in S. \tag{2.3}$$

## 2.2 Dynamic Programming

Algorithms based on Dynamic Programming (DP) can be used to calculate optimal policies for a Markov Decision Process. Classical DP algorithms for MDPs assume a perfect model of the environment in which the transition function $T$ and the reward function $R$ are known. Two Dynamic Programming algorithms widely used to solve MDPs are Value Iteration (Bertsekas, 1995) and Policy Iteration (Ronald A Howard, 1960).

In the Value Iteration algorithm (Algorithm 1) the objective is to iteratively calculate the value of the Optimal Value function $V^*$ from the equation:

$$V^{t+1}(s) \leftarrow \max_a \left[ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^t(s') \right]. \tag{2.4}$$

Formally, the update step must be executed infinite times so that $V$ converges to $V^*$. In practice, iterations are suspended as soon as $V$ is updated by at most a small value defined by the parameter $\theta$. The maximum difference in the $V$ function is represented by $\Delta$ in the algorithm. Finally, the optimal policy $\pi^*$ is obtained through:

$$\pi^*(s) \leftarrow arg \max_a \left[ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right]. \tag{2.5}$$

**Algorithm 1:** Value Iteration

```
1  begin
       input  : MDP $\langle S, A, R, T, s_0 \rangle$, $\gamma$, $\theta$
       output : $\pi$
2      Initializes V with arbitrary values (e.g., $V(s) = 0$ for all $s \in S$);
3      repeat
4          $\Delta \leftarrow 0$;
5          foreach $s \in S$ do
6              $v \leftarrow V(s)$;
7              $V(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')\right]$;
8              $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
9      until $\Delta < \theta$;
10     return $arg \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')\right] \forall s \in S$;
```

In the Policy Iteration algorithm, the goal is to find an optimal policy through changes in the policy itself and not indirectly through the Value function, as is done in the Value Iteration algorithm.

The Policy Iteration algorithm (Algorithm 2) first calculates the Value function of a policy, this step is called *Policy Evaluation* and the calculation of the function can be done by solving a system of equations or through approximation by an iterative method. You can define the step of *Policy Evaluation* through approximation by an interactive method as being:

$$V^{t+1}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^t(s'), \forall s \in S. \tag{2.6}$$

After obtaining the function Value $V^\pi$, a second step called *Policy Improvement* is performed by the Policy Iteration algorithm. The objective of *Policy Improvement* is to find a new policy maximizing rewards using the Value function $V^\pi$ calculated before:

$$\pi'(s) \leftarrow arg \max_a \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right]. \tag{2.7}$$

The two steps of the algorithm are executed until $\pi = \pi'$ and thus find the optimal policy $\pi^*$.

**Algorithm 2:** Policy Iteration

```
1  begin
       input  : MDP $\langle S, A, R, T, s_0 \rangle$, $\gamma$, $\theta$
       output : $\pi$
2      Initializes $\pi'$ with arbitrary values;
3      repeat
4          $\pi \leftarrow \pi'$;
5          1: Evaluate policy $\pi$;
6          $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'), \forall s \in S$;
7          2: Improve policy $\pi'$;
8          $\pi'(s) \leftarrow arg \max_a \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right]$
9      until $\pi = \pi'$;
10     return $\pi$
```

## 2.3 Monte Carlo method

Reinforcement Learning methods differ from Dynamic Programming methods because they are methods in which the model of the Markov Decision Process is not known, that is, the agent knows the actions $a \in A$ but does not know the transition function $T$ and the reward function $R$. In these methods, an environment able to produce samples of the transitions and rewards is necessary and there is no need to know all probability distributions, as is necessary for Dynamic Programming.

In the Monte Carlo Reinforcement Learning method, the goal is to learn Value functions from experiments on samples. Rather than using the model to calculate the value of each state, the sampled returns for each state present in a run episode are averaged. An episode $(s_1, a_1, r_1, s_2, a_2, r_2, ...)$ is generated by the execution of a policy $\pi$ from an initial state until a terminal (target) state is reached and the execution is finished. An iteration of the Monte Carlo method can be described as (Sutton and Barto, 1998):

$$V\left(s_t\right) \leftarrow V\left(s_t\right) + \alpha\left[G_t - V\left(s_t\right)\right], \tag{2.8}$$

where $\alpha$ is the learning rate and $G_t$ the accumulated rewards in an episode from time $t$:

$$G_t \leftarrow \sum_{k=0}^{T} \gamma^k r_{t+k}, \tag{2.9}$$

where $T$ is the last time of the episode and each episode proceeds in a discrete number of steps $t$ so that an episode starts at $t = 0$ and ends at $t = T$. This way, the function $V\left(s_t\right)$ can only be updated at the end of the episode, when it is possible to obtain the value of $G_t$.

The Monte Carlo method is an estimation method since the value of $G_t$ is an estimate of the expected value. This feature gives the Monte Carlo methods three advantages when compared to Dynamic Programming methods. First, they can be used to learn optimal behavior directly from interacting with the environment without the need for a model of the environment. Second, they can be used with simulation or sampling models. Third, it is easy and efficient to apply Monte Carlo methods on subsets of states (Sutton and Barto, 1998).

## 2.4 Temporal Difference

In Reinforcement Learning there is a central idea called Temporal Difference (TD). This method, similar to the Monte Carlo methods can learn without prior knowledge of the environment and similar to Dynamic Programming methods updates the estimated value without waiting for the final result of the episode. The fact of not waiting for the end of the episode to update the estimated value can be very advantageous when applied in environments where the episode is very long or even in ongoing problems where the episode may never finish (Sutton and Barto, 1998).

As in the Monte Carlo method, the Temporal Difference methods update the estimated value of the V-value function for all non-terminal states present in the sampled episode, however, they update at each step the value $V(s_t)$ starting from the value $r_{t+1}$ observed and the value of $V(s_{t+1})$ estimated. One can describe the Temporal Difference method as being (Sutton and Barto, 1998):

$$V\left(s\right) \leftarrow V\left(s\right) + \alpha\left[r + \gamma V\left(s'\right) - V\left(s\right)\right]. \tag{2.10}$$

The Temporal Difference called $TD(0)$ (Sutton and Barto, 1998) (Algorithm 3) begins by taking a policy $\pi$ and a number of episodes for the evaluation process. During each episode, $TD(0)$ selects an action $a$ based on the policy $\pi$ and the current state $s$. $TD(0)$ then takes this action and observes the resulting reward $r$ and the next state $s'$. The current state value $V(s)$ is then updated using Equation 2.10. The episode continues until the current state becomes a terminal state.

**Algorithm 3:** TD(0)

```
1  begin
       input  : $\pi, s_0, \alpha, \gamma$
       output : $V$
2      Initializes V with arbitrary values (e.g., $V(s) = 0$ for all $s \in S$);
3      foreach episode do
4          $s \leftarrow s_0$;
5          repeat
6              $a \leftarrow \pi(s)$;
7              Take action $a$ and observe $r$ and $s'$;
8              $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$;
9              $s \leftarrow s'$;
10         until $s$ being a terminal;
11     return $V$
```

The Temporal Difference method is considered an estimation method as it takes samples to calculate the estimated value and uses the current estimate of $V$ to make the updates.

### 2.4.1 SARSA

In the SARSA algorithm (Rummery and Niranjan, 1994) (Sutton and Barto, 1998) the policy is learned at runtime, that is, the same policy that is used to interact with the environment has the value updated by the algorithm.

Unlike the algorithms presented so far, SARSA instead of estimating the Value $V$ function aims to estimate the Value-Action function $Q$. The Value-Action function $Q^*$ is defined as the sum of the reward received by the state-action pair $(s, a)$ with the Value-Action function of the next state and action $Q^*(s', a')$:

$$Q^*(s,a) \leftarrow r + \gamma Q^*\left(s', a'\right). \tag{2.11}$$

The update of the Value-Action function $Q$ in the SARSA algorithm can be represented as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q\left(s', a'\right) - Q(s,a)\right]. \tag{2.12}$$

This interaction with the tuple $(s, a, r, s', a')$ gives SARSA its name (*State-Action-Reward-State-Action*). To choose actions $a$ and $a'$, the SARSA algorithm must use an exploration or exploitation strategy. This strategy aims to perform an already known action to increase the rewards or choose a random action so that the reward can be improved over a longer period. A widely used strategy for choosing actions is $\epsilon$-greedy. In this strategy, the agent selects a random action with probability $\epsilon$ or executes the best action with probability $1 - \epsilon$.

The SARSA algorithm, described in Algorithm 4, takes the learning rate $\alpha$, exploration rate $\epsilon$, and the number of episodes as input parameters. During each episode, the algorithm starts with an initial state $s_0$, and an action $a$ is chosen based on a policy derived from the state-action value function $Q$ for the current state $s$. For example, an action can be randomly selected following the $\epsilon$-greedy policy.

After taking action $a$, the algorithm observes the reward $r$ and the next state $s'$. The next action $a'$ for state $s'$ is selected using the same process as in the previous step. The state-action value function $Q(s,a)$ is updated for the current state-action pair using the SARSA update rule (Equation 2.12). Then, the next state $s'$ and the next action $a'$ is set as the current state and action, respectively.

**Algorithm 4:** SARSA

```
1  begin
       input  : $s_0, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initializes $Q(s,a)$ with arbitrary values for all $s \in S$, $a \in A$ ;
3      foreach episode do
4          $s \leftarrow s_0$;
5          Selects $a$ from a policy derived from $Q$ (e.g., $\epsilon$-greedy) ;
6          repeat
7              Take an action $a$ and observe $r$ and $s'$;
8              Selects $a'$ from a policy derived from $Q$ (e.g., $\epsilon$-greedy) ;
9              $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') - Q(s,a)]$;
10             $s \leftarrow s'$;
11             $a \leftarrow a'$;
12         until s being a terminal;
13     return $arg\max_a [Q(s,a)]\ \forall s \in S$;
```

### 2.4.2 Q-Learning

One of the best-known Temporal Difference algorithms is the Q-Learning algorithm defined as (Watkins, 1989):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_a Q\left(s',a\right) - Q(s,a) \right]. \tag{2.13}$$

Q-Learning also uses the $\epsilon$-greedy policy like SARSA, but when updating the Q-value for a state-action pair, Q-learning considers the action with the highest Q-value for the next state $s'$. In this way, the Value-Action function $Q$ approaches the optimal function $Q^*$ regardless of the policy being followed as long as all state-action pairs are updated (Watkins, 1989).

The Q-Learning algorithm (Algorithm 5) works similarly to SARSA algorithm only removing the selection of the next action $a'$ and replacing the Q-value for the next state ($Q(s',a')$) by the maximum Q-value for the next state $s'$ ($\max_a Q(s',a)$).

**Algorithm 5:** Q-LEARNING

```
1  begin
       input  : $s_0, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initializes $Q(s, a)$ with arbitrary values for all $s \in S$, $a \in A$ ;
3      foreach episode do
4          $s \leftarrow s_0$;
5          repeat
6              Selects $a$ from a policy derived from $Q$ (e.g., $\epsilon$-greedy) ;
7              Take an action $a$ and observe $r$ and $s'$;
8              $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_a Q(s', a) - Q(s, a)]$;
9              $s \leftarrow s'$;
10         until $s$ being a terminal;
11     return $arg \max_a [Q(s, a)] \forall s \in S$;
```

## 2.5 Approximation of the Value Function

Applying Reinforcement Learning to real-world problems can pose significant challenges due to the large state space involved. The limitations of using tabular methods (all methods presented in the previous sections) extend beyond memory constraints. It can be practically impossible to fill and compute values for all states, resulting in most states being visited for the first time. In such cases, it becomes crucial to generalize from past experiences when states exhibit similarities.

Fortunately, generalization techniques have been extensively studied in the field of supervised learning, and the use of artificial neural networks has demonstrated remarkable success in addressing this challenge. Numerous successful studies have explored the combination of Artificial Neural Networks and Reinforcement Learning. For instance, TD-gammon (TESAURO, 1994), a backgammon-playing agent, achieved a human level of play by exclusively learning through reinforcement learning and self-play. Multi-layer Artificial Neural Networks were utilized for function approximation in reinforcement learning, automating the feature design process while learning to play 46 different Atari 2600 games (MNIH *et al.*, 2013). Additionally, AlphaGo (SILVER, SCHRITTWIESER, *et al.*, 2017) and AlphaGo Zero (SILVER, SCHRITTWIESER, *et al.*, 2017) utilized Deep Artificial Neural Networks, supervised learning, Monte Carlo tree search, and Reinforcement Learning to master the game of Go, achieving remarkable results.

In Reinforcement Learning, there are two main scenarios where function approximation can be applied: (i) when the policy is fixed, and only the value function is approximated, and (ii) when function approximation is employed for the optimal policy function.

Applying function approximation to the policy function is notably different and more challenging than applying it to the value function (SUTTON and BARTO, 1998). The theoretical guarantees and empirical results for the former method are not as strong or satisfactory as those for the latter. Considering this, the focus of this section is on applying function approximation to the value function. More specifically applying neural networks to approximate the value function.

### 2.5.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They have revolutionized the field of machine learning and have become a fundamental tool in various domains, ranging from computer vision and natural language processing to finance and healthcare.

At its core, an artificial neural network consists of interconnected neurons organized into layers. Each neuron receives inputs, performs a computation, and produces an output. The connections between neurons are represented by weights, which determine the strength and influence of each input. The arrangement of these neurons and their connections forms the network's architecture.

Training an artificial neural network involves an iterative process of adjusting the weights to minimize the difference between the network's predicted outputs and the desired outputs. This process, known as supervised learning, relies on presenting to the network a set of expected results. The most common algorithm used for training ANNs is backpropagation, which calculates the gradient of the network's error and updates the weights.

One of the key strengths of artificial neural networks is their ability to learn complex patterns and extract meaningful features from raw data. This capability is especially prominent in deep neural networks, which are ANNs with multiple hidden layers. Deep learning (LeCun, Bengio, *et al.*, 2015) has led to significant breakthroughs in areas such as image and speech recognition, natural language processing, and autonomous systems.

Convolutional Neural Networks (CNNs) (LeCun, Bottou, *et al.*, 1998) is a specialized type of ANN commonly used for analyzing visual data. By employing convolutional layers, pooling operations, and nonlinear activations, CNNs can effectively capture spatial and hierarchical features in images or videos. The use of this class of deep neural networks has shown impressive results when applied to reinforcement learning applications (Mnih *et al.*, 2013; Silver, Huang, *et al.*, 2016; Silver, Schrittwieser, *et al.*, 2017).

### 2.5.2 Deep Q-Network

One outstanding result was achieved by combining a convolutional neural network and the Q-Learning algorithm (Mnih *et al.*, 2013). They presented a proposal capable of automating the feature design and generalizing among 46 different Atari 2600 games. Their proposal was divided into two: (i) the network architecture called Deep Q-Network and (ii) the algorithm called Deep Q-Learning.

The proposed network architecture, named Deep Q-Network (DQN), is designed to process four images, each with dimensions of $84 \times 84$ pixels. It begins with an input layer of size $84 \times 84 \times 4$, followed by three hidden convolutional layers, then a fully connected hidden layer, and finally the output layer.

To add more stability to the learning process two identical networks with the described architecture were added to the Deep Q-Learning algorithm (Van Hasselt *et al.*, 2016). One network is called the learning network $w$ and the other is called the target network $\hat{w}$. The target network $\hat{w}$ is used to estimate while the learning network $w$ learns from experience replays. Experience replay is a method that is incorporated into the algorithm, which involves storing the agent N last experience in a queue called replay memory $D$. An experience is a tuple $(s, a, r, s')$ where $s$ is the current state, $a$ the action, $r$ the reward, and $s'$ next state.

The Deep Q-Learning algorithm (Algorithm 6) operates similarly to Q-Learning, utilizing $\epsilon$-greedy to select action $a$ during each episode. However, instead of finding the action from a Value-Action function $Q$, it employs a Value-Action prediction $\hat{Q}$ based on the learning network $w$ and the current state $s$. Once the action $a$ is chosen, the algorithm observes the reward $r$ and the next state $s'$.

The tuple $(s, a, r, s')$ is stored in the replay buffer $D$. Deep Q-Learning randomly samples batches of tuples (minibatch) from $D$. For each tuple in the minibatch, it employs a Value-Action prediction $\hat{Q}$ using the target network $\hat{w}$ and the next state $s'$. Deep Q-Learning then calculates the target value $y$ based on the Value-Action function equation.

Using the target value $y$, a gradient descent is performed to update the weights of the learning network $w$.

After a certain number of steps ($C$ steps), the weight of the learning network is copied to the target network ($\hat{w} = w$).

**Algorithm 6:** Deep Q-Learning

```
1  begin
       input  : $s_0, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initialize $w$, $\hat{w} = w$, $t = 0$ , $D = 0$;
3      foreach episode do
4          $s \leftarrow s_0$;
5          Selects $a$ from a policy derived from $\hat{Q}(s, a, w)$ (e.g., $\epsilon$-greedy) ;
6          Take an action $a$ and observe $r$ and $s'$;
7          Store tuple $(s, a, r, s')$ in replay queue $D$ ;
8          $s \leftarrow s'$;
9          Sample random minibatch from $D$ ;
10         foreach tuple $(s, a, r, s')$ in minibatch do
11             $y \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a', \hat{w})$;
12             Do gradient descent step on $(y - \hat{Q}(s, a, w))^2$;
13         if mod(t, C) = 0 then
14             $\hat{w} = w$;
15         $t = t + 1$;
16     return $arg \max_a \left[\hat{Q}(s', a', \hat{w})\right] \forall s \in S$;
```

While the algorithm is commonly referred to as Deep Q-Learning when combined with the Deep Q-Network, it is also quite prevalent in the literature to simply refer to it as the Deep Q-Network (Sutton and Barto, 1998). This work applies the same terminology.

# Chapter 3

# Safe Reinforcement Learning

Reinforcement Learning has proven to be a powerful approach for learning optimal decision-making policies across various domains. However, in real-world applications, safety concerns pose a significant challenge, as the selection of incorrect actions or policies can result in harmful consequences.

In safety-critical domains like autonomous vehicles, healthcare, and robotics, it becomes imperative to ensure that Reinforcement Learning agents not only maximize expected rewards but also adhere to predefined safety constraints (Mihatsch and Neuneier, 2002). To address these challenges, Safe Reinforcement Learning adopts two main approaches. One approach focuses on adapting the exploration process to avoid actions that could lead the learning system into undesirable or catastrophic situations (Garcia and Fernández, 2015). The other approach involves modifying the optimality criterion to incorporate the concept of risk (Garcia and Fernández, 2015). This allows the learning process to take into account potential hazards and uncertainties, ensuring policies that prioritize safety over pure expected reward maximization.

## 3.1 Exploration Process

In Reinforcement Learning Algorithms, it is common to start learning without any prior knowledge of the environment or the desired goal. In such cases, exploration strategies like $\epsilon$-greedy are employed. However, this random exploration may lead the agent to states where she can encounter potential harm or injury (Garcia and Fernández, 2015).

To address this issue, two methods for modifying the exploration process are commonly employed:

- The incorporation of external knowledge involves several approaches. In this group, the learning algorithm is provided with additional knowledge to enhance the learning process. This additional knowledge can be in the form of initializing the Q-function with previous demonstration recorded by a human teacher (Driessens and Džeroski, 2004), a policy derived from a predefined and limited number of demonstrations (Abbeel *et al.*, 2010; Kidambi *et al.*, 2020; Swazinna *et al.*, 2021) or even adding a teacher or an additional layer capable of providing advice to the agent (Garcia and Fernández, 2012; Dalal *et al.*, 2018).

- The use of a risk-directed exploration. In these approaches, a risk measure is employed to assess the probability of selecting different actions during the exploration process (Gehring and Precup, 2013; Andersen *et al.*, 2020).

## 3.2 Optimization Criterion

Classical Reinforcement Learning primarily focuses on maximizing the expected reward, but this optimization criterion may not always be the most suitable approach for dangerous or risky tasks (Mihatsch and Neuneier, 2002). Recognizing the need to consider risk in decision-making, several alternative optimization criteria have been proposed. These criteria can be categorized into three main groups (Garcia and Fernández, 2015):

- The Worst-Case Criterion: This group of optimization criteria, considers a policy to be optimal if it maximizes the worst-case reward. This approach minimizes the variability of a policy that can be caused by the stochastic nature of the system (Heger, 1994; Littman and Szepesvári, 1996) or the uncertainty of a not known MDP parameter (Nilim and El Ghaoui, 2005).

- The Constrained Criterion: This group aims to maximize the reward while ensuring that other expected measures satisfy specific bounds (Kadota *et al.*, 2006; Moldovan and Abbeel, 2012; HasanzadeZonuzy *et al.*, 2021).

- The Risk Sensitive Criterion: This group represents a different approach, involving the addition of a parameter to control the sensitivity to risk. This parameter allows for the fine-tuning of risk perception in the learning process. To achieve this, the optimization criterion is transformed into either an exponential utility function (Ronald A. Howard and Matheson, 1972; Chung and Sobel, 1987; Bouakiz and Sobel, 1992; Cavazos-Cadena and Hernández-Hernández, 2011; Bäuerle and Rieder, 2014; Wood and Khosravanian, 2015; Fei, Yang, Yudong Chen, Wang, and Xie, 2020; Fei, Yang, Yudong Chen, and Wang, 2021) or utilizing implicit definitions within dynamic programming functions, aiming to incorporate modifiers that transform metrics like the temporal difference (Mihatsch and Neuneier, 2002; Shen *et al.*, 2014; Jin *et al.*, 2020; Delétang *et al.*, 2021; Fei, Yang, and Wang, 2021). Other approaches are rooted in financial engineering and utilize optimization criteria like value-at-risk (VaR) (Mausser and Rosen, 1999), conditional value-at-risk (CVaR) (Chow and Ghavamzadeh, 2014; Tang *et al.*, 2019; Stanko and Macek, 2019; Du *et al.*, 2022; Xu *et al.*, 2023), or the density of the return (Morimura *et al.*, 2010).

The application of the Risk Sensitive Criterion in Reinforcement Learning has garnered significant attention , and among various criteria, the exponential utility approach stands out as the most popular and extensively analyzed risk sensitive control framework in the literature. This work focuses on Risk Sensitive Reinforcement Learning and it will be further explored in Chapter 4.

# Chapter 4

# Risk Sensitivity with Exponential Functions in Reinforcement Learning

In the Risk Sensitive Reinforcement Learning literature the combination of the exponential function with the expected utility theory constitutes the most popular and best-analyzed risk sensitive control framework (Garcia and Fernández, 2015). As this work focuses on utilizing exponential functions to model risk, we explore the implementation of Exponential Expected Utility alongside other proposals that make use of exponential functions but apply it to the Temporal Difference.

In the context of Risk Sensitive Reinforcement Learning with exponential functions, we can model the environment as a Risk Sensitive Markov Decision Process (RSMDP) (Ronald A. Howard and Matheson, 1972). An RSMDP extends the traditional Markov Decision Process (MDP) by incorporating a risk factor that allows us to define the agent's attitude toward risk. Formally, an RSMDP can be represented by the tuple $\langle S, A, R, T, s_0, \lambda \rangle$, where $S, A, R, T, s_0$ are defined in the same way as in MDPs, and $\lambda$ represents the risk factor.

This chapter is structured as follows: Firstly, we present the risk attitudes. Subsequently, we introduce the exponential utility function and its application in risk modeling across different algorithms, including Value Iteration, Q-Learning, and Deep Q-Learning. We adapt these algorithms to incorporate Exponential Expected Utility.

Moving forward, we explore an alternative approach that applies the utility function to the Temporal Difference across Value Iteration, Q-Learning, and Deep Q-Learning. Then, we examine how both the exponential function and a related function known as the soft indicator can be used with these algorithms.

We show that when applying the exponential function to both expected utility and Temporal Difference transformations, the values calculated are the same. Finally, we explore strategies aimed at mitigating overflow issues, which frequently arise when dealing with exponential functions.

## 4.1 Risk and Certainty Equivalent

The execution of a policy $\pi$ and the dynamics of a process defines a random variable $C^{\pi}$ representing the total rewards obtained by policy $\pi$. This random variable is defined as follows:

$$C^{\pi}(s) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t \,\middle|\, \pi, s_0 = s\right]. \tag{4.1}$$

Based on these considerations, the value of a policy $\pi$ at state $s$ is $U(V^{\pi}(s))$ and can be defined as follows [1] :

$$U(V^{\pi}(s)) = \mathrm{E}[U(C^{\pi}(s))]. \tag{4.2}$$

Since $C^{\pi}$ is a random variable, we may consider three general attitudes regarding risk (R L Keeney and Raiffa, 1976): neutral, prone, and averse. First, we need to define the certainty equivalent of a policy $\pi$.

Intuitively, the certainty equivalent is the reward that an agent would prefer to receive with certainty rather than taking a chance on an uncertain outcome. If $U(V^{\pi}(s)) < \infty$ and there exists the inverse function $U^{-1} : \mathbb{R} \to \mathbb{R}^{+}$, the certainty equivalent $V^{\pi}(s)$ of a policy $\pi$ is defined by (Bäuerle and Rieder, 2014; Freitas *et al.*, 2020):

$$V^{\pi}(s) = U^{-1}(U(V^{\pi}(s))) = U^{-1}(\mathrm{E}[U(C^{\pi}(s))]). \tag{4.3}$$

We also define the expected reward $\widetilde{C}^{\pi}(s)$ of a policy $\pi$ by:

$$\widetilde{C}^{\pi}(s) = \mathrm{E}[C^{\pi}(s)]. \tag{4.4}$$

An agent is risk prone if $V^{\pi}(s) > \widetilde{C}^{\pi}(s)$, risk averse if $V^{\pi}(s) < \widetilde{C}^{\pi}(s)$, and risk neutral if $V^{\pi}(s) = \widetilde{C}^{\pi}(s)$ for every state $s \in \mathcal{S}$ and policy $\pi \in \Pi$.

Intuitively, if you are risk prone (optimistic), you would prefer to receive more than the expected reward to not play the lottery because you are focused on better results (high reward) of the lottery (i.e. certainty equivalent is bigger than the expected reward). If you are risk-averse (pessimistic), you would prefer to receive less than the expected reward to not play the lottery because you are focused on worse results (low reward) of the lottery (i.e. certainty equivalent is smaller than the expected reward).

Figure 4.1 shows a Risk Sensitive MDP example (Example presented in Chapter 1) with four states ($s_0, s_1, s_2, s_3$) where $s_0$ is the initial state and the others are terminals. The agent can take two different actions: (i) a deterministic action (selecting Arm 0) and (ii) a stochastic action (selecting Arm 1). When the agent selects Arm 0 she goes to terminal state $s_1$ receiving a reward $r$. When the agent selects Arm 1 she has a probability of 0,5 to

[1] In this work we make use of the notation $U(V^{\pi}(s))$ to represent the value of a policy $\pi$ at state $s$ . This facilitates comparing the different exponential criteria.

go to terminal state $s_1$ receiving the same reward $r$. However, there is also a probability of 0,25 to go to terminal state $s_2$ and receiving a reward of $r - 1$ as well as a probability of 0,25 to go to terminal state $s_3$ and receiving a reward of $r + 1$.

In the context of the example in Figure 4.1, if you are risk-prone, you will be more focused on the chances of receiving the higher reward $r + 1$ when selecting Arm 1, leading you to take that action. On the other hand, if you are risk-averse, you will be more concerned about the chances of receiving the lower reward $r - 1$ when choosing Arm 1, and as a result, you will opt for the deterministic action (select Arm 0) instead.

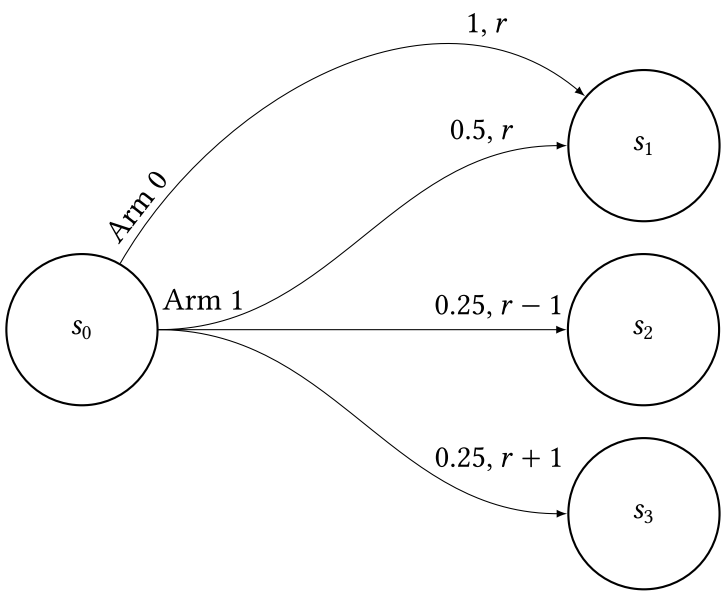


**Figure 4.1:** *Risk Sensitive MDP example*

## 4.2 Exponential Utility Function

An RSMDP can use an exponential utility function to define the risk attitude. The exponential function is used since, although it grows very quickly, it has mathematical properties that allow the use of Dynamic Programming (Ronald A Howard, 1960).

The exponential utility function for an RSMDP can be defined by (Ronald A. Howard and Matheson, 1972):

$$U(x) = sign(\lambda)e^{\lambda x}, \tag{4.5}$$

and model arbitrary risk attitude by considering a risk-attitude factor $\lambda$. If $\lambda < 0$ the agent considers a risk-averse attitude, if $\lambda > 0$ the agent considers a risk-prone attitude and, in the limit, if $\lambda \to 0$ the agent considers a risk-neutral attitude. Figure 4.2 illustrate the exponential utility function for a risk-averse parameter ($\lambda = -1$) and for a risk-prone parameter ($\lambda = 1$).

The inverse exponential utility function $U^{-1}$ is defined as follows [2]:

[2] This dissertation utilizes the notation log to represent the natural logarithm.

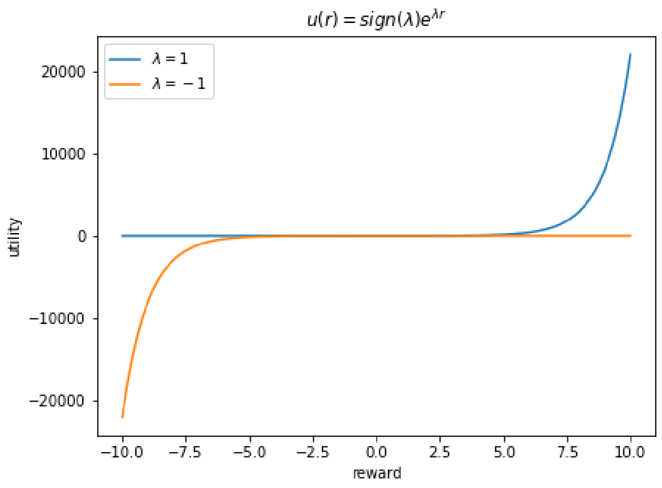


**Figure 4.2:** *Behavior of the exponential utility function.*

$$U^{-1}(x) = \frac{\log(sign(\lambda)x)}{\lambda}. \tag{4.6}$$

## 4.3 Exponential Expected Utility Criterion

In the Expected Utility Theory, the main idea is to transform the cumulative rewards by applying utility functions (Ronald A. Howard and Matheson, 1972) and seek optimal policies concerning this utility measure. In this work, we refer to the criterion that applies the exponential function (Equation 4.5) to rewards as Exponential Expected Utility (EEU) criterion. In the next subsections, we explain how this criterion is employed in Value Iteration, Q-Learning, and Deep Q-Learning algorithms.

### 4.3.1 Value Iteration with EEU

By leveraging the principles of expected utility theory, we directly apply the exponential function to the target value $R(s,a) + \gamma V^i(s')$. This allows us to reformulate Value Iteration as follows:

$$U\left(V^{i+1}(s)\right) \leftarrow \max_a \left[\sum_{s' \in S} T\left(s'|s,a\right) U\left(R(s,a) + \gamma V^i(s')\right)\right]. \tag{4.7}$$

An important distinction arises from the applied update, the update now returns the exponential utility of $V^{i+1}(s)$, i.e., $U\left(V^{i+1}(s)\right)$. To calculate the certainty equivalent, we need to apply the inverse utility function $U^{-1}$.

The Value Iteration algorithm with the incorporation of the EEU is shown in Algorithm 7. A key aspect of Algorithm 7 is that the certainty equivalent $V(s)$ is preserved (line 8). This aspect is particularly beneficial when comparing this method to other transformations. Additionally, the use of $\Delta$ in the stop criterion remains unchanged from the original Value

Iteration.

**Algorithm 7:** VALUE ITERATION WITH EEU

```
1  begin
       input  : RSMDP $\langle S, A, R, T, s_0, \lambda\rangle, \gamma, \theta$
       output : $\pi$
2      Initializes V with arbitrary values (e.g., $V(s) = 0$ for all $s \in S$);
3      repeat
4          $\Delta \leftarrow 0$;
5          foreach $s \in S$ do
6              $v \leftarrow V(s)$;
7              $u \leftarrow \max_a \left[\sum_{s' \in S} T(s'|s,a) U(R(s,a) + \gamma V(s'))\right]$;
8              $V(s) \leftarrow U^{-1}(u)$;
9              $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
10     until $\Delta < \theta$;
11     return $arg \max_a \left[\sum_{s' \in S} T(s'|s,a) U(R(s,a) + \gamma V(s'))\right] \forall s \in S$;
```

## 4.3.2 Q-Learning with EEU

To incorporate EEU into Q-Learning, we apply the same principles used in Value Iteration. The Q-Learning with EEU update equation has the following form:

$$U\left(Q^{i+1}(s,a)\right) \leftarrow U\left(Q^{i}(s,a)\right) + \alpha\left[U\left(R(s,a) + \gamma \max_a Q^i(s',a)\right) - U\left(Q^i(s,a)\right)\right]. \quad (4.8)$$

In this work, the Q-Learning algorithm modified to add EEU is called Q-Learning with EEU (Algorithm 8).

**Algorithm 8:** Q-LEARNING WITH EEU

```
1  begin
       input  : $s_0, \lambda, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initializes $Q(s,a)$ with arbitrary values for all $s \in S$, $a \in A$ ;
3      foreach episode do
4          $s \leftarrow s_0$;
5          repeat
6              Select $a$ from a policy derived from $Q$ (e.g. $\epsilon$-greedy) ;
7              Take and action $a$ and observe $r$ and $s'$;
8              $u \leftarrow U(Q(s,a)) + \alpha [U(r + \gamma \max_a Q(s',a)) - U(Q(s,a))]$;
9              $Q(s,a) \leftarrow U^{-1}(u)$;
10             $s \leftarrow s'$;
11         until s being a terminal;
12     return $arg \max_a [Q(s,a)] \forall s \in S$;
```

Similar to the modification made in Value Iteration, our objective is to retain the Value-Action function $Q(s, a)$ as the certainty equivalent of $U(Q(s, a))$. Thus, in line 9 of Algorithm 8 the inverse utility $U^{-1}$ is applied to calculate the certainty equivalent of the value $u$.

### 4.3.3 Deep Q-Learning with EEU

The proposed Deep Q-Learning algorithm with EEU is shown in Algorithm 9. To integrate EEU into the Deep Q-Learning algorithm, we maintain the complete Q-Learning with EEU update (Equation 4.8) but replacing $Q$ to the network prediction $\hat{Q}$ that receives a state, an action and a network parameter $w$ to make the prediction. This update is applied on Line 11 of Algorithm 9.

In the case of Value Iteration with EEU and Q-Learning with EEU, using the certainty equivalent value was a deliberate choice to facilitate further analyses in these algorithms. However, in the Deep Q-Learning with EEU, this change also proved to be crucial to the learning process.

Applying the exponential utility function to the Value-Action function $Q$ can lead to significantly large or small values, making it challenging for the neural network to learn efficiently. In line 12 of Algorithm 9, $u$ is transformed to its certainty-equivalent $y$ by applying $U^{-1}$. This transformation normalizes the values that the network needs to approximate, resulting in a faster and more stable learning process.

**Algorithm 9:** Deep Q-Learning with EEU

```
1  begin
       input  : $s_0, \lambda, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initialize $w$, $\hat{w} = w$, $t = 0$ , $D = 0$;
3      foreach episode do
4          $s \leftarrow s_0$;
5          Selects $a$ from a policy derived from $\hat{Q}(s, a, w)$ (e.g., $\epsilon$-greedy) ;
6          Take an action $a$ and observe $r$ and $s'$;
7          Store tuple $(s, a, r, s')$ in replay queue $D$ ;
8          $s \leftarrow s'$;
9          Sample random minibatch from $D$ ;
10         foreach tuple $(s, a, r, s')$ in minibatch do
11             $u \leftarrow U\left(\hat{Q}(s', a, w)\right) + \alpha\left[U\left(r + \gamma \max_{a'} \hat{Q}(s', a', \hat{w})\right) - U\left(\hat{Q}(s', a, w)\right)\right]$;

12             $y \leftarrow U^{-1}(u)$;
13             Do gradient descent step on $(y - \hat{Q}(s, a, w))^2$;
14         if $mod(t, C) = 0$ then
15             $\hat{w} = w$;
16         $t = t + 1$;
17     return $arg \max_a \left[\hat{Q}(s', a', \hat{w})\right] \forall s \in S$;
```

## 4.4 Temporal Difference Transformation

The concept of defining risk directly in the dynamic programming equation has been extensively explored, and one of the most studied approaches involves applying a utility function to transform the Temporal Difference. This transformation is as follows (SHEN *et al.*, 2014):

$$Q^{i+1}(s,a) \leftarrow Q^{i}(s,a) + \alpha \left[ U\left( R(s,a) + \gamma \max_{a} Q^{i}(s',a) - Q^{i}(s,a) \right) - x_0 \right]. \tag{4.9}$$

Analogous to the Value Iteration, when the transition function $T$ is known the Value Iteration with Temporal Difference Transformation can be defined as follow:

$$Q^{i+1}(s,a) \leftarrow Q^{i}(s,a) + \alpha \left[ \sum_{s' \epsilon S} T\left(s'|s,a\right) U\left(R(s,a) + \gamma V^{i}(s') - Q^{i}(s,a)\right) - x_0 \right]. \tag{4.10}$$

Where $V^i(s)$ is defined as follow:

$$V^{i}(s) \leftarrow \max_{a} Q^{i}(s,a), \tag{4.11}$$

$x_0$ is the acceptance level (SHEN *et al.*, 2014) and $y_0$ is its certainty equivalent:

$$x_0 = U(y_0). \tag{4.12}$$

When $U(x) = sign(\lambda)e^{\lambda x}$, the acceptance level is defined as $x_0 = sign(\lambda)$ and $y_0 = 0$ (SHEN *et al.*, 2014).

### 4.4.1 Temporal Difference Transformation Algorithms

In this section, we introduce the adaptation of Value Iteration, Q-Learning, and Deep Q-Learning to use Equation 4.9 and Equation 4.10.

To modify the Value Iteration for the Temporal Difference Transformation, we use Equation 4.10, as shown in line 10 of Algorithm 10. In line 3, we set the acceptance level $x_0 = sign(\lambda)$. In line 11 we apply Equation 4.11 to obtain $V(s)$.

Adapting the Q-Learning algorithm is a straightforward process, as illustrated in Algorithm 11.

To maintain consistency with the other proposed methods we adapted the Deep Q-Learning algorithm by maintaining the complete Q-Learning with TD Transformation update, resulting in Algorithm 12.

**Algorithm 10:** Value Iteration with TD Transformation

```
1  begin
       input  : RSMDP $\langle S, A, R, T, s_0, \lambda \rangle$, $\alpha$, $\gamma$, $\theta$
       output : $\pi$
2      Initializes V with arbitrary values (e.g., $V(s) = 0$ for all $s \in S$);
3      Initializes $Q(s,a)$ with arbitrary values (e.g., $Q(s,a) = 0$ for all $s \in S$, $a \in A$) ;
4      $x_0 \leftarrow sign(\lambda)$;
5      repeat
6          $\Delta \leftarrow 0$;
7          foreach $s \in S$ do
8              $v \leftarrow V(s)$;
9              foreach $a \in A$ do
10                 $Q(s,a) \leftarrow Q(s,a) + \alpha \left[\sum_{s' \in S} T(s'|s,a) U(R(s,a) + \gamma V(s') - Q(s,a)) - x_0\right]$;
11             $V(s) \leftarrow \max_a Q(s,a)$;
12             $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
13     until $\Delta < \theta$;
14     return $arg \max_a \left[\sum_{s' \in S} T(s'|s,a) U(R(s,a) + \gamma V(s'))\right] \forall s \in S$;
```

### 4.4.2 Exponential Utility and Soft Indicator Functions with Temporal Difference Transformation

The Temporal Difference Transformation algorithms have the flexibility to work with various utility functions. Previous research (Mihatsch and Neuneier, 2002; Shen *et al.*, 2014; Delétang *et al.*, 2021) has shown their incorporation into the TD transformation framework. In this study, we specifically concentrate on two different approaches for applying exponential criteria: (i) the exponential utility function (Shen *et al.*, 2014) and (ii) the soft indicator function (Delétang *et al.*, 2021).

The exponential utility function (Equation 4.5) could be used in the Temporal Difference Transformation algorithms (Shen *et al.*, 2014). We call this modification Exponential TD Transformation (ETD).

Another exponential function that could be used in the Temporal Difference Transformation algorithms is the soft indicator function (Delétang *et al.*, 2021):

$$U(x) = \frac{2x}{1 + e^{-\lambda x}}. \tag{4.13}$$

The Soft Indicator function serves as an approximation of the exponential utility function (Delétang *et al.*, 2021). Figure 4.3 displays the Soft Indicator function for a risk-averse parameter ($\lambda = -1$) and for a risk-prone parameter ($\lambda = 1$). Its behavior is similar to the use of exponential utility (Figure 4.2). While these functions exhibit some similarities in their behavior, the Soft Indicator method produces entirely different results in value function as we show in the Experiments chapter. We refer to this approach as the Soft Indicator TD Transformation (SITD).

**Algorithm 11:** Q-LEARNING WITH TD TRANSFORMATION

```
1  begin
       input  : $s_0, \lambda, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initializes $Q(s, a)$ with arbitrary values for all $s \in S$, $a \in A$ ;
3      $x_o \leftarrow sign(\lambda)$;
4      foreach episode do
5          $s \leftarrow s_0$;
6          repeat
7              Select $a$ from a policy derived from $Q$ (e.g. $\epsilon$-greedy) ;
8              Take and action $a$ and observe $r$ and $s'$;
9              $Q(s, a) \leftarrow Q(s, a) + \alpha [U(r + \gamma \max_a Q(s', a) - Q(s, a)) - x_0]$;
10             $s \leftarrow s'$;
11         until $s$ being a terminal;
12     return $arg \max_a [Q(s, a)] \forall s \in S$;
```

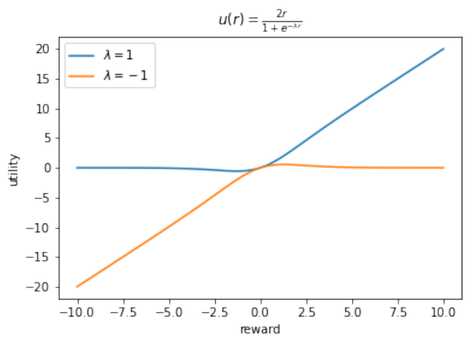


**Figure 4.3:** *Behavior of the Soft Indicator function.*

## 4.5 Relationship between EEU and ETD

In sections 4.3 and 4.4, we explored the application of the Exponential Utility function (Equation 4.5) in two distinct criteria: EEU and ETD. An important observation from applying the Exponential Utility in both criteria is that they converge to the same value.

**Theorem 1** (EEU and ETD converge to the same state value.). *When $y_0 = 0$ and then $x_0 = sign(\lambda)$, the ETD and EEU converge to the same state value.*

*Proof.* Let's assume the convergence of the Value Iteration with Temporal Difference Transformation given in Equation 4.10:

$$\sum_{s' \epsilon S} T\left(s'|s, a\right) U\left(R(s, a) + \gamma V^i(s') - Q^i(s, a)\right) - x_0 = 0. \tag{4.14}$$

**Algorithm 12:** Deep Q-Learning with TD Transformation

```
1  begin
       input  : $s_0, \lambda, \alpha, \gamma, \epsilon$
       output : $\pi$
2      Initialize $w$, $\hat{w} = w$, $t = 0$ , $D = 0$;
3      $x_o \leftarrow sign(\lambda)$;
4      foreach episode do
5          $s \leftarrow s_0$;
6          Selects $a$ from a policy derived from $\hat{Q}(s, a, w)$ (e.g., $\epsilon$-greedy) ;
7          Take an action $a$ and observe $r$ and $s'$;
8          Store tuple $(s, a, r, s')$ in replay queue $D$ ;
9          $s \leftarrow s'$;
10         Sample random minibatch from $D$ ;
11         foreach tuple $(s_e, a_e, r_e, s'_e)$ in minibatch do
12             $y \leftarrow \hat{Q}(s', a, w) + \alpha \left[U\left(r + \gamma \max_{a'} \hat{Q}(s', a', \hat{w}) - \hat{Q}(s', a, w)\right) - x_0\right]$;
13             Do gradient descent step on $(y - \hat{Q}(s, a, w))^2$;
14         if mod(t, C) = 0 then
15             $\hat{w} = w$;
16         $t = t + 1$;
17     return $arg\max_a \left[\hat{Q}(s', a', \hat{w})\right] \forall s \in S$;
```

Using Equation 4.5 and $x_0 = sign(\lambda)$ in Equation 4.14, we obtain:

$$\begin{aligned}
sign(\lambda) &= \sum_{s' \epsilon S} T\left(s'|s,a\right) sign(\lambda) e^{\lambda\left(R(s,a)+\gamma V^i(s') - Q^i(s,a)\right)} \\
sign(\lambda) &= \sum_{s' \epsilon S} T\left(s'|s,a\right) sign(\lambda) e^{\lambda\left(R(s,a)+\gamma V^i(s')\right) - \lambda Q^i(s,a)} \\
sign(\lambda) &= \frac{1}{e^{\lambda Q^i(s,a)}} \sum_{s' \epsilon S} T\left(s'|s,a\right) sign(\lambda) e^{\lambda\left(R(s,a)+\gamma V^i(s')\right)} \\
sign(\lambda) e^{\lambda Q^i(s,a)} &= \sum_{s' \epsilon S} T\left(s'|s,a\right) sign(\lambda) e^{\lambda\left(R(s,a)+\gamma V^i(s')\right)} \\
U\left(Q^i(s,a)\right) &= \sum_{s' \epsilon S} T\left(s'|s,a\right) U\left(R(s,a) + \gamma V^i(s')\right).
\end{aligned}$$

By applying Equation 4.11 to find $V^i(s)$ in the system of equations, we obtain:

$$U\left(V^i(s)\right) = \max_a \left[\sum_{s' \epsilon S} T\left(s'|s,a\right) U\left(R(s,a) + \gamma V^i(s')\right)\right]. \tag{4.15}$$

The system of equations is the same as Equation 4.7 that corresponds to Value Iteration with EEU, demonstrating that both converge to the same state value.

□

In the Experiments chapter, we illustrate these aspects of the convergence of EEU and ETD criteria.

## 4.6 Exponential criteria and overflow

Although there are various advantages of using the exponential utility function to model risk attitudes, there is a significant drawback related to potential underflow or overflow issues when computing intermediate calculations due to the rapid decrease or growth of this function. To address these problems, this section presents two different approaches: (i) truncating the utility function, and (ii) applying LogSumExp during utility calculations. These techniques aim to mitigate the numerical instability associated with the use of the exponential function and improve the overall performance of risk-sensitive algorithms.

### 4.6.1 Truncated Exponential Utility

The Q-Learning with Temporal Difference transformation (Equation 4.9) converges when the Lipschitz condition is satisfied by the utility function (Shen *et al.*, 2014). Although the exponential utility function does not satisfy the global Lipschitz condition, truncating it within specific lower and upper bounds guarantees convergence (Shen *et al.*, 2014). This truncating can also handle some overflow errors.

The first crucial assumption that must be made for this truncation technique is that we have prior knowledge of the upper bound for the absolute value of rewards:

$$\overline{R} = sup \left| R(s,a) \right| . \tag{4.16}$$

Considering $y_0 = U^{-1}(x_0)$, the lower and upper bounds $\underline{x}$ and $\overline{x}$ are defined by (Shen *et al.*, 2014):

$$\underline{x} = y_0 - \tfrac{2\overline{R}}{1-\gamma} \quad \text{and} \quad \overline{x} = y_0 + \tfrac{2\overline{R}}{1-\gamma} \ . \tag{4.17}$$

Assuming positive constants $\epsilon, L \in \mathbb{R}^+$ such that $0 < \epsilon \leq \frac{U(x)-U(y)}{x-u} \leq L$, for all $x \neq y \in [\underline{x}, \overline{x}]$, the truncated utility $U'$ is defined by truncating the utility function when U outside the interval $[\underline{x}, \overline{x}]$ (Shen *et al.*, 2014):

$$U'(x) = \begin{cases} U\left(\underline{x}\right) + \epsilon\left(x - \underline{x}\right), & x \in (-\infty, \underline{x}) \\ U\left(x\right), & x \in [\underline{x}, \overline{x}] \\ U\left(\overline{x}\right) - \epsilon\left(x - \overline{x}\right), & x \in (\overline{x}, \infty) \end{cases} \tag{4.18}$$

The truncate strategy can be applied to the Exponential Utility function on EEU and ETD criteria. The truncate strategy will be further examined and explored in detail in the Experiments chapter.

### 4.6.2 LogSumExp

The numeric overflow problem is becoming more common in machine learning algorithms (Robert, 2014). There are techniques that have been used to deal with this type of problem such as *LogSumExp* (Naylor *et al.*, 2001), *Gordian-L* (Sigl *et al.*, 1991) and *Lp-Norm* (Kennings and Markov, 2000). In particular, the *LogSumExp* technique, has been used successfully in several works (Nielsen and Sun, 2016; Yi Chen and Gao, 2016). This type of problem also happens in Hidden Markov Models and some techniques used are a scaling technique (Rabiner, 1990) and the *LogSumExp* technique (Mann, 2006). We chose the *LogSumExp* (Logarithm of the Sum of Exponential) strategy because this strategy uses a logarithmic function on an exponential function.

The *LogSumExp* instead of calculating the sum of exponential, which can lead to numerical overflow, uses the properties of logarithms to perform the calculation as follow:

$$\log(e^A + e^B) = \log(e^A(1 + \frac{e^B}{e^A})) = A + \log(1 + e^{B-A}). \tag{4.19}$$

When using Equation 4.19 it is necessary to identify the largest exponent and designate it as A. By doing so, the exponential function is applied to a negative number $(B - A)$, preventing overflow issues.

The integration of the EEU criterion with the *LogSumExp* has demonstrated promising results (Freitas *et al.*, 2020), and its application on EEU is simple. Next, we present how *LogSumExp* can be employed in Value Iteration with EEU (Freitas *et al.*, 2020). As a valuable contribution from this work, we apply the *LogSumExp* technique to the update function of Q-Learning with EEU. By incorporating *LogSumExp*, we extend its applicability to both Q-Learning with EEU and Deep Q-Learning with EEU algorithms.

#### Value Iteration with EEU and LogSumExp

Modifying the Value Iteration with EEU from Equation 4.7 to iterate over the function $Q^{i+1}(\cdot)$ instead of $V^i(\cdot)$, we have:

$$U\left(Q^{i+1}(s,a)\right) = \sum_{s' \in S} T\left(s'|s,a\right) U\left(R\left(s,a\right) + \gamma V^i(s')\right) \tag{4.20}$$

Applying the exponential utility function (Equation 4.5) in Equation 4.20, we obtain:

$$sign(\lambda) e^{\lambda Q^{i+1}(s,a)} = \sum_{s' \in S} T\left(s'|s,a\right) sign(\lambda) e^{\lambda\left(R(s,a)+\gamma V^i(s')\right)} \tag{4.21}$$

Eliminating $sign\,(\lambda)$ and applying log in both sides of Equation 4.21 to became a log of exponential sum, we have:

$$\begin{aligned}
Q^{i+1}(s,a) &= \frac{1}{\lambda}\log\left[\sum_{s'\in S} T\left(s'|s,a\right) e^{\lambda\left(R(s,a)+\gamma V^i(s')\right)}\right] \\
&= \frac{1}{\lambda}\log\left[\sum_{s'\in S} T\left(s'|s,a\right) e^{\lambda R(s,a)} e^{\lambda\gamma V^i(s')}\right] \\
&= R(s,a) + \frac{1}{\lambda}\log\left[\sum_{s'\in S} T\left(s'|s,a\right) e^{\lambda\gamma V^i(s')}\right] \\
&= R(s,a) + \frac{1}{\lambda}\log\left[\sum_{s'\in S} e^{\log\left(T\left(s'|s,a\right)\right)+\lambda\gamma V^i(s')}\right].
\end{aligned} \tag{4.22}$$

The two auxiliary function $k_{s,s'}^{a,i}$ and $K_s^{a,i}$ are defined with the objective to identify the largest exponent (Freitas *et al.*, 2020):

$$k_{s,s'}^{a,i} = \log\left(T\left(s'|s,a\right)\right) + \lambda\gamma V^i(s') \tag{4.23}$$

$$K_s^{a,i} = \max_{s'\in S}\left(k_{s,s'}^{a,i}\right). \tag{4.24}$$

Introducing $k_{s,s'}^{a,i}$ and $K_s^{a,i}$ in Equation 4.22, we have:

$$\begin{aligned}
Q^{i+1}(s,a) &= R(s,a) + \frac{1}{\lambda}\log\left[\sum_{s'\in S} e^{k_{s,s'}^{a,i}}\right] \\
&= R(s,a) + \frac{1}{\lambda}\log\left[\sum_{s'\in S} e^{k_{s,s'}^{a,i}-K_s^{a,i}} e^{K_s^{a,i}}\right] \\
&= R(s,a) + \frac{1}{\lambda}\log\left[e^{K_s^{a,i}} \sum_{s'\in S} e^{k_{s,s'}^{a,i}-K_s^{a,i}}\right] \\
&= R(s,a) + \frac{1}{\lambda}K_s^{a,i} + \frac{1}{\lambda}\log\left[\sum_{s'\in S} e^{k_{s,s'}^{a,i}-K_s^{a,i}}\right].
\end{aligned} \tag{4.25}$$

Note that, in this last formulation the exponential function is applied to a negative number, preventing overflow issues.

Finally, the value $V^{i+1}(s)$ is calculated with Equation 4.11.

### Q-Learning with EEU and LogSumExp

We now apply the *LogSumExp* on the Q-Learning with EEU update function (Equation 4.8).

First Equation 4.8 is rewritten as follows:

$$U\left(Q^{i+1}(s,a)\right) = (1-\alpha)U\left(Q^{i}(s,a)\right) + \alpha\left(U\left(R(s,a) + \gamma\max_{a} Q^{i}(s',a)\right)\right). \tag{4.26}$$

Applying the exponential utility function (Equation 4.5) on Equation 4.26:

$$sign(\lambda)e^{\lambda Q^{i+1}(s,a)} = (1-\alpha)\left(sign(\lambda)e^{\lambda Q^{i}(s,a)}\right) + \alpha\left(sign(\lambda)e^{\lambda\left(R(s,a)+\gamma\max_a Q^{i}(s',a)\right)}\right). \tag{4.27}$$

Eliminating $sign(\lambda)$ and applying log in both sides of Equation 4.27 to became a log of exponential sum, we have:

$$\begin{aligned} Q^{i+1}(s,a) &= \frac{1}{\lambda}\log\left[(1-\alpha)\left(e^{\lambda Q^{i}(s,a)}\right) + \alpha\left(e^{\lambda\left(R(s,a)+\gamma\max_a Q^{i}(s',a)\right)}\right)\right] \\ &= \frac{1}{\lambda}\log\left[\left(e^{\lambda Q^{i}(s,a)+\log(1-\alpha)}\right) + \left(e^{\lambda\left(R(s,a)+\gamma\max_a Q^{i}(s',a)\right)+\log(\alpha)}\right)\right]. \end{aligned} \tag{4.28}$$

The *LogSumExp* strategy first identifies the largest term of an exponential sum. Note that, differently from the application of *LogSumExp* to Value Iteration with EEU, we have only two terms instead of a sum of many terms. Thus, we define the two terms as being $A$ and $B$:

$$A = \lambda Q^{i}(s,a) + \log(1-\alpha). \tag{4.29}$$

$$B = \lambda\left(R(s,a) + \gamma\max_{a} Q^{i}(s',a)\right) + \log(\alpha). \tag{4.30}$$

When $A > B$:

$$Q^{i+1}(s,a) = \frac{1}{\lambda}\left[A + \log\left(1 + e^{B-A}\right)\right]. \tag{4.31}$$

Otherwise:

$$Q^{i+1}(s,a) = \frac{1}{\lambda}\left[B + \log\left(1 + e^{A-B}\right)\right]. \tag{4.32}$$

# Chapter 5

# Experiments

Experiments were performed in two domains and three different algorithms: (i) Value Iteration (VI); (ii) Q-Learning (QL); and (iii) Deep Q-Network (DQN). For each one of the domains and each algorithm, three exponential criteria were used: (i) Exponential Expected Utility (EEU); (ii) Exponential TD Transformation (ETD); and (iii) Soft Indicator TD Transformation (SITD).

With the proposed experiments, we aim to answer the following questions:

- How does the risk factor, $\lambda$, affect the exponential criteria?
- How do sampling algorithms (e.g.: Q-Learning and Deep Q-Network) influence the use of exponential criteria?
- Does the learning rate, $\alpha$, directly affect the learning process and how much is each exponential criterion affected?
- What is the relationship between exponential criteria and the use of Deep Q-Network?
- How overflow issues are handled? Can they be avoided in all scenarios?
- How difficult is calibrating risk sensitivity parameters in exponential risk criteria?
- Among the EEU, ETD, and SITD criteria, which one is more stable and easier to apply in different scenarios?

The experiments were done on a machine with a processor of 4 cores at 2.3 GHz and 16 GB of memory [1].

## 5.1 Domains and setup

In this section, we first describe the domains used in the experiments and the experimental setup to run each algorithm.

[1] The source code of the implementation is available at https://github.com/dulpneto/exponential_rs_rl

### 5.1.1 Two Arm Bandit Domain

The Two Arm Bandit domain (adapted from (Delétang *et al.*, 2021)) is a simple environment to evaluate how much risk the agent is willing to take by setting two different arms, Arm 0 (called Deterministic Arm) and Arm 1 (called Stochastic Arm). Arm 0 always returns a deterministic reward $r_0$ while Arm 1 retrieves a stochastic reward with mean $r_1$ with precision $p$. For example, setting $p = 2$ makes Arm 1 return rewards $r_1 - 1$ with 0.25 of chance, return $r_1$ with 0.5 of chance, and return $r_1 + 1$ with 0.25 of chance. Figure 5.1 shows rewards distribution from the Two Armed Bandit instance with $r_0 = 0$ and $r_1 = 0$ with precision $p = 2$.

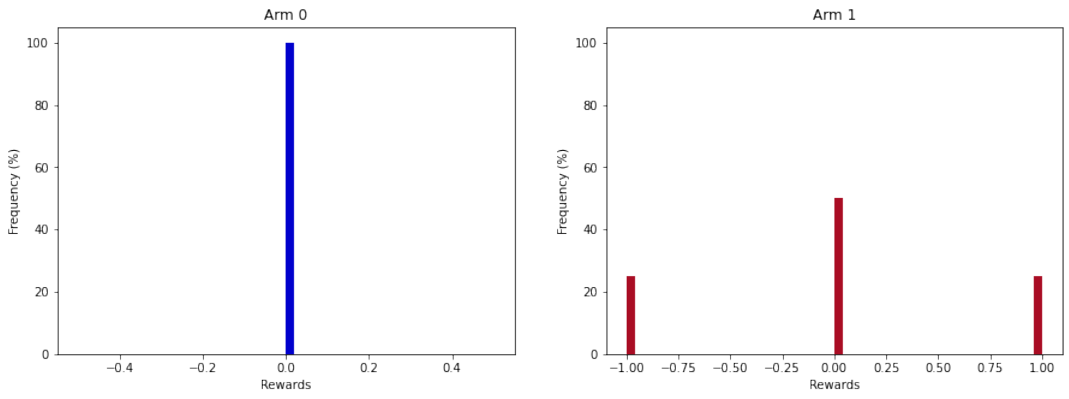


**Figure 5.1:** *Rewards distribution of the Two Armed Bandit instance with $r_0 = 0$ and $r_1 = 0$ with precision $p = 2$.*

#### Value Iteration and Q-Learning Setup

We run tests on the Two Armed Bandit by setting $r_0$ and $r_1$ from -0.5 to 0.5 with 0.025 intervals and $p = 2$. The risk factor values used are -5, -2, -0.5, -0.1, 0.1, 0.5, 2, and 5. Thus, the number of different instances created (called, configurations) is $30 \times 30 \times 8$.

As Value Iteration does not involve sampling, we execute it only once because it produces the same result for all runs with the same configuration. For the Q-learning algorithm, we executed 100 runs of experiments for each configuration and we set each run to finish after 150 episodes.

In the Value Iteration experiments, each point of the chart (that corresponds to one configuration) is painted in blue when the policy selects Arm 0 and red when it selects Arm 1.

In the experiments with Q-learning, for each configuration, we compute the number of times that the obtained policies select Arm 1 minus the number of times that selects Arm 0 considering the 100 runs. With these values, a blue-read heat map is created (Figure 5.2). The point will be blue when the agent selects Arm 0 for all runs and red when it selects Arm 1 for all runs. The point will be colored in white when the agent selects Arm 0 in 50% of the runs.

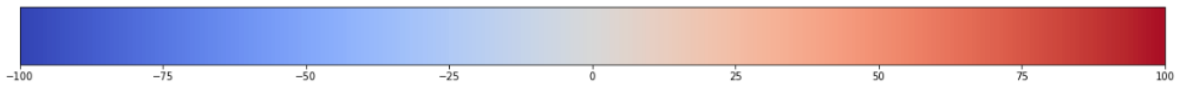


**Figure 5.2:** *Heat map used to color the Two Armed Bandit policies.*

### 5.1.2 River Crossing Domain

The River Crossing domain (Freire and Delgado, 2017) consists of a grid ($N_x \times N_y$) with the lower left corner being the initial state and the lower right corner being the final state. In this domain, the agent can perform 4 actions: moving north (↑), south (↓), east (→), or west (←).

The agent's goal is to reach the final state and for that, the agent can cross the river or walk along the river margin until reaching the bridge at the top of the grid. When the agent walks along the margin or over the bridge, she has a probability 1 of performing the chosen action. However, when the agent is in a state that represents the river, she has a chance of 75% to perform the selected action and a chance of 25% to be pushed by the river to the next state to the south.

For each action performed, the agent receives a reward of -1 and if the agent reaches the waterfall, she returns to the initial state. Figure 5.3 shows an instance of the River domain with a grid size 10 × 10.

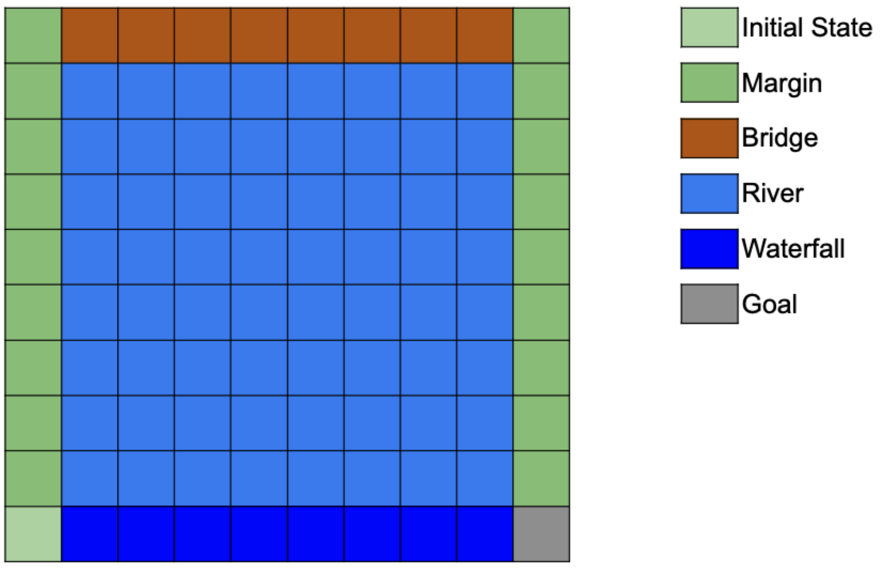


**Figure 5.3:** *A* $10 \times 10$ *instance of the River Crossing Domain.*

The number of steps on the left margin the agent will perform before turning right can be use to analyze the risk attitude in this domain. In this work, we call it the *number of safe steps* and this will be computed for all the instances analyzed. Specifically, in an instance with a shape of 10 × 10, we have nine distinct risk attitudes by considering the number of safe steps. In Figure 5.4 we show four of them, from the most risk-prone to the most risk-averse. The number of safe steps is 1,3,6,9, respectively (from left to right).

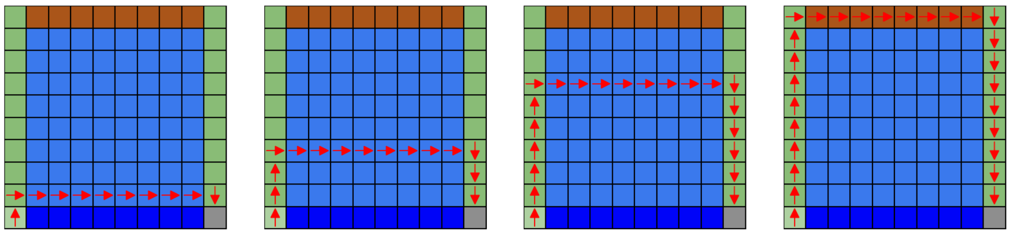


**Figure 5.4:** *Number of safe steps in the most risk-prone to the most risk-averse policies.*

## Value Iteration, Q-Learning and Deep Q-Network Setup

A $10 \times 10$ instance of the River Crossing Domain was used in the experiments and the risk factor was varying between -1.5 and 1.5 with intervals of 0.1.

While Value Iteration was executed once for each configuration, Q-Learning was executed 15 times for each configuration and each run was set to finish after 300 episodes.

To apply the Deep Q-Network algorithm, we first change the River Crossing environment to return an image of the current observation instead of returning the state number. This image is a plot from the environment similar to Figure 5.3 with size $100 \times 100$ pixels in gray scale where the river and waterfall are marked with hatches.

The input layer of the Convolutional Neural Network has shape $100 \times 100 \times 1$ followed by:

- Conv2D: 6 filters, kernel size 7x7, stride 3, ReLU activation.
- MaxPooling2D: Pool size 2x2.
- Conv2D: 12 filters, kernel size 4x4, ReLU activation.
- MaxPooling2D: Pool size 2x2.
- Flatten.
- Dense: 216 units, ReLU activation.
- Dense: 4 units, with no activation representing the total number of actions.

Figure 5.5 shows the network architecture along with one example of an observation image from the River Crossing domain in the left side.

The mean squared error loss and the Adam optimizer with a learning rate of 0.001 are used.

Deep Q-Network was executed 15 times for each configuration and each run was also set to finish after 300 episodes.

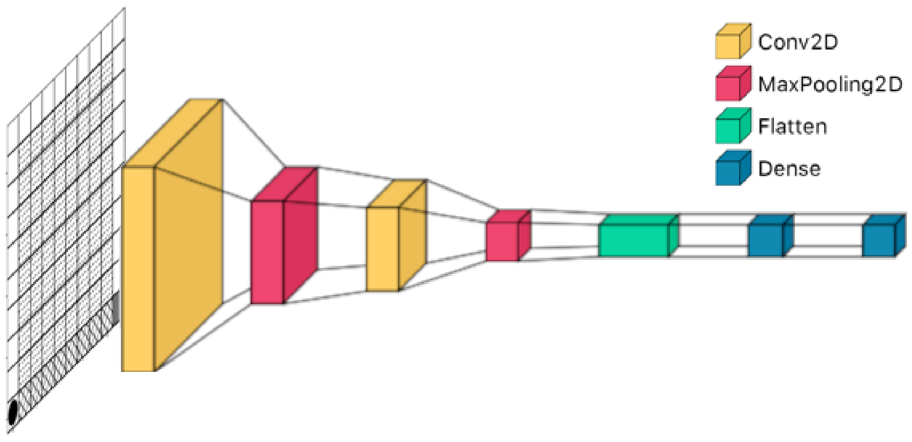


**Figure 5.5:** *DQN network architecture used in the experiments for the River Crossing domain.*

## 5.2 Results for the Two Arm Bandit Domain

In this section, experiments were performed varying the risk factor and the learning rate for the Two Arm Bandit Domain.

### 5.2.1 Varying the risk factor

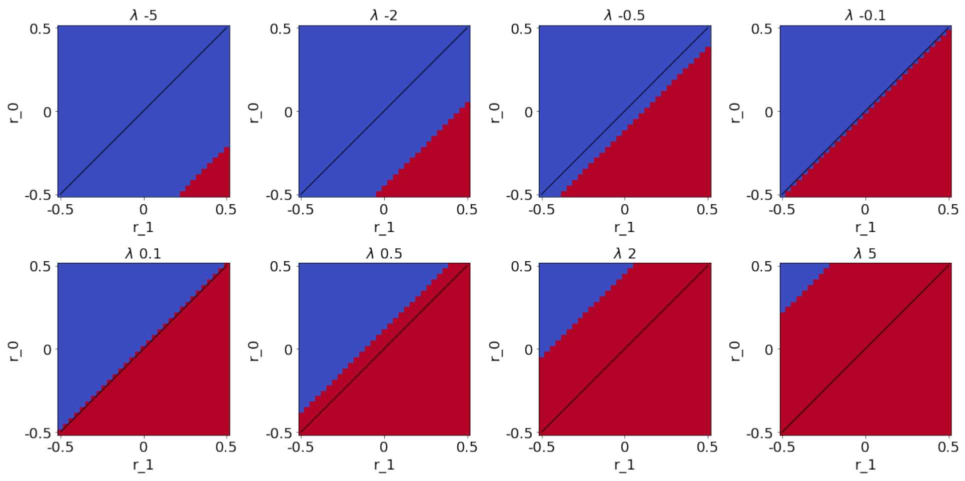
For this experiment, we varied the risk factor and fixed the learning rate $\alpha = 0.1$. The Value Iteration and Q-Learning algorithms were executed with the three exponential criteria: (i)EEU; (ii) ETD; and (iii) SITD. The objective of this experiment is to analyze how each exponential criteria behave for each risk factor and how they are influenced by the use of the sampling algorithms.

#### Exponential Expected Utility

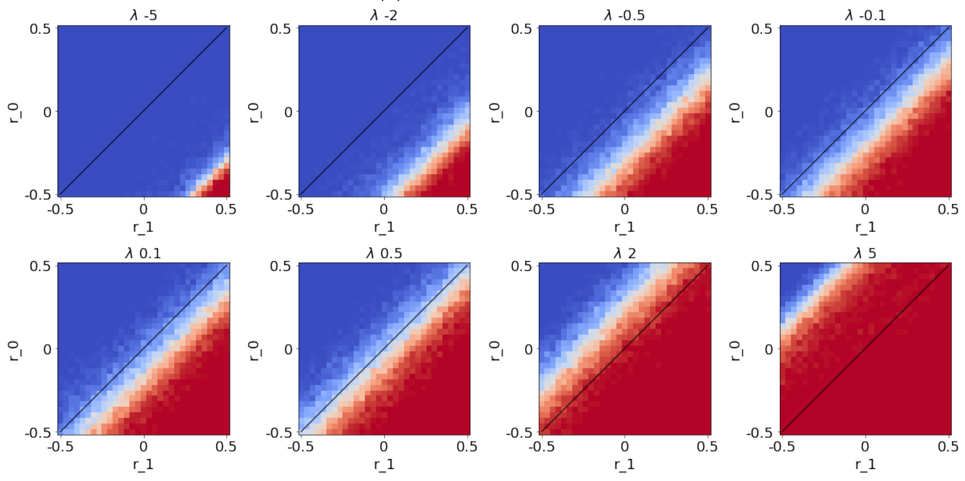
The color representation of the policies obtained by the Value Iteration and Q-Learning algorithms with EEU criterion are shown in Figure 5.6(a) and 5.6(b), respectively. In the last one, we consider the 100 runs for each configuration as described in Section 5.1.1.

As expected, for the Value Iteration algorithm with EEU criterion, for $\lambda = -5$ the agent is more risk averse and selects the Deterministic Arm for almost all the configurations. As the risk factor gets closer to zero, she approximates to the black diagonal line that represents the risk-neutral policy. And finally when $\lambda = 5$ the agent chooses the Stochastic Arm for almost all the instances.

Q-Learning with EEU has almost the same behavior as Value Iteration with EEU regarding the decisions for each risk factor. The main difference is the area where the agent changes the policy, in this part appears a white area showing for some cases the agent selects the Deterministic Arm in 50% of the runs and the Stochastic Arm in the rest of the runs although the area is still parallel to the risk neutral-policy.



(a) Value Iteration



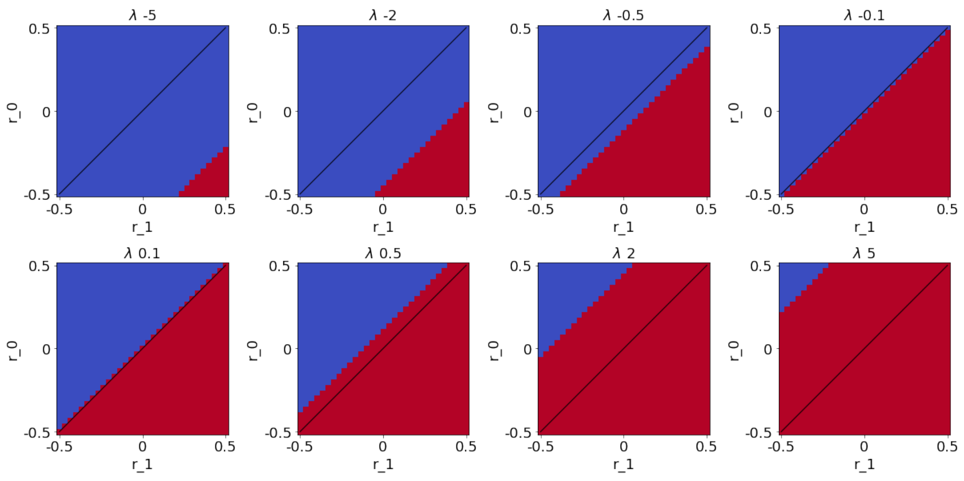
(b) Q-Learning

**Figure 5.6:** *Policies obtained by the Value Iteration and Q-Learning algorithms with EEU criterion for Two Armed Bandit domain.*

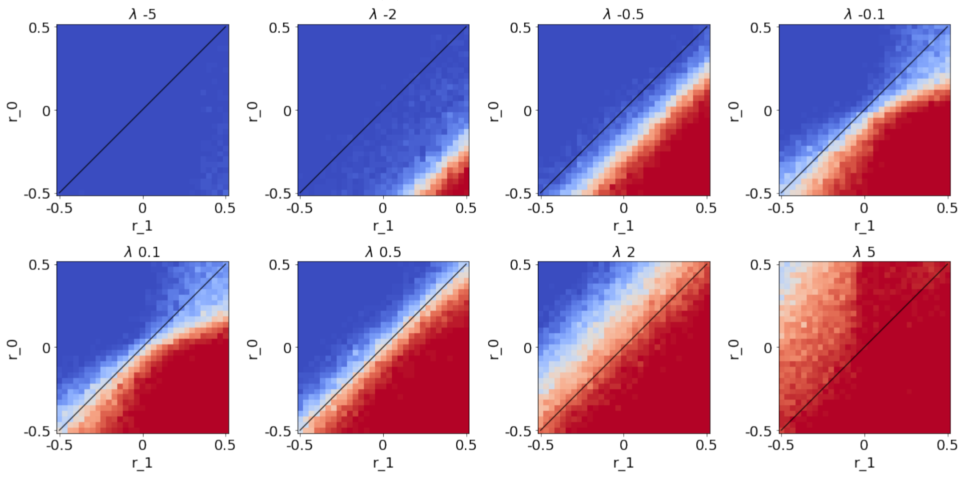
## Exponential TD Transformation

The color representation of the policies obtained by the Value Iteration and Q-Learning algorithms with ETD criterion are presented in Figure 5.7(a) and 5.7(b), respectively. In the last one, we consider the 100 runs for each configuration as described in Section 5.1.1.

As expected, the policies obtained by the Value Iteration algorithm with EEU criterion for each configuration (Figure 5.6(a)) are equal to the ones obtained by the Value Iteration algorithm with ETD criterion (Figure 5.7(a)). This follows Theorem 1 that states that EEU and ETD converge to the same state value. However, the policies obtained by the Q-Learning algorithm with EEU criterion (Figure 5.6(b)) are different from the ones obtained by the Q-Learning algorithm with ETD criterion (Figure 5.7(b)) for some configurations. For $\lambda = -0.1$ and $\lambda = 0.1$ and using Q-Learning with the ETD criterion, the white area is not more parallel to the line that represents the risk-neutral policy. For $\lambda = 5$ the white area for Q-Learning with ETD criterion is bigger than for the Q-Learning with EEU criterion. Finally, for $\lambda = -5$, the Q-Learning with ETD criterion must have a red area in the bottom right corner, but it was not able to find the correct policy in this part.



(a) Value Iteration



(b) Q-Learning

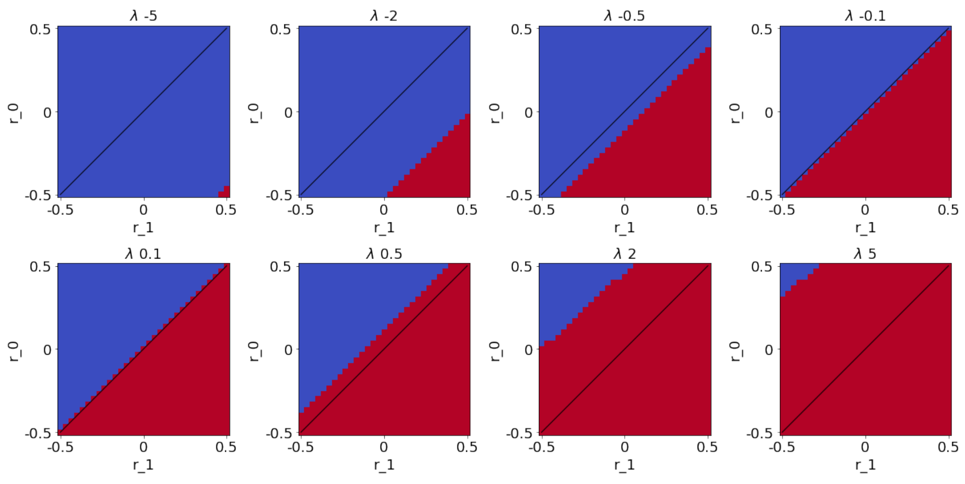
**Figure 5.7:** *Policies obtained by the Value Iteration and Q-Learning algorithms with ETD criterion for Two Armed Bandit domain.*

## Soft Indicator TD Transformation

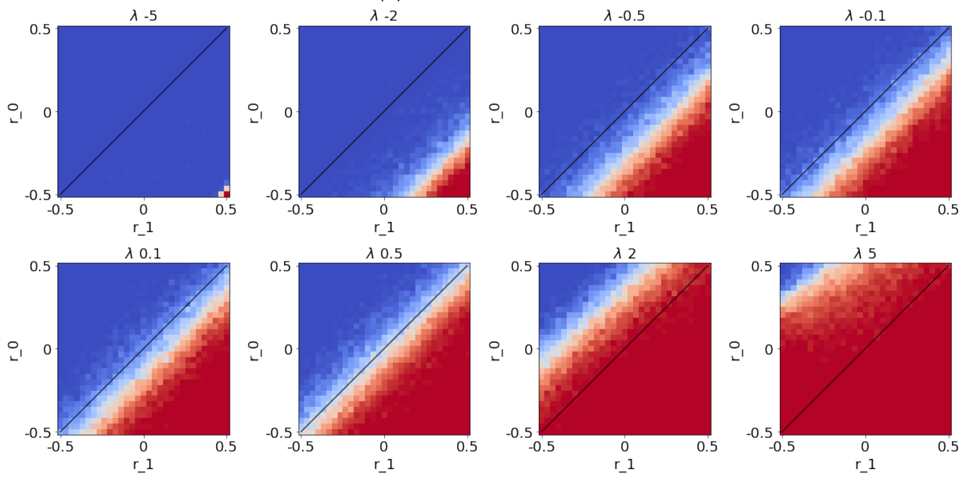
The color representation of the policies obtained by the Value Iteration and Q-Learning algorithms with SITD criterion are presented in Figure 5.8(a) and 5.8(b), respectively. In the last one, we consider the 100 runs for each configuration as described in Section 5.1.1.

SITD criterion converges to different values than EEU and ETD criteria. This is expected since SITD uses a different transformation function. For $\lambda = 2$ and $\lambda = 5$ the line where the agent changes from Arm 0 to Arm 1 is not parallel with the line that represents the risk-neutral policy. This behavior was not observed with the other criteria.

Figure 5.8 shows that Q-Learning with SITD has almost the same behavior as Value Iteration with SITD regarding the decisions for each risk factor. The main difference is the area where the agent changes the policy (white area).



(a) Value Iteration



(b) Q-Learning

**Figure 5.8:** *Policies obtained by the Value Iteration and Q-Learning algorithms with SITD criterion for Two Armed Bandit domain.*

### 5.2.2 Varying the learning rate

In the experiments of Section 5.2.1 we notice some stability of the Q-Learning algorithm with EEU criteria but the two TD transformations (ETD and SITD) displayed some unexpected results, mainly when looking for scenarios with $\lambda = 5$. Thus, in this section, we analyze how each criterion is affected by the learning rate. We fixed $\lambda = 5$ and used the following learning rates 0.2, 0.1, 0.05, and 0.01.

Figure 5.9 shows the color representation of the policies obtained by the Q-Learning algorithm with EEU, ETD, and SITD criteria in the first, second, and third lines, respectively.

EEU is the one with less variation while varying the learning rate. On the other hand, the other two criteria are showing problems depending on the value of the learning rate. The ETD works better as we decrease the learning rate, while the SITD works better as we increase the learning rate. This behavior can make parameter calibration and convergence more difficult in bigger scenarios. This will be also analyzed in the next domain experiments.

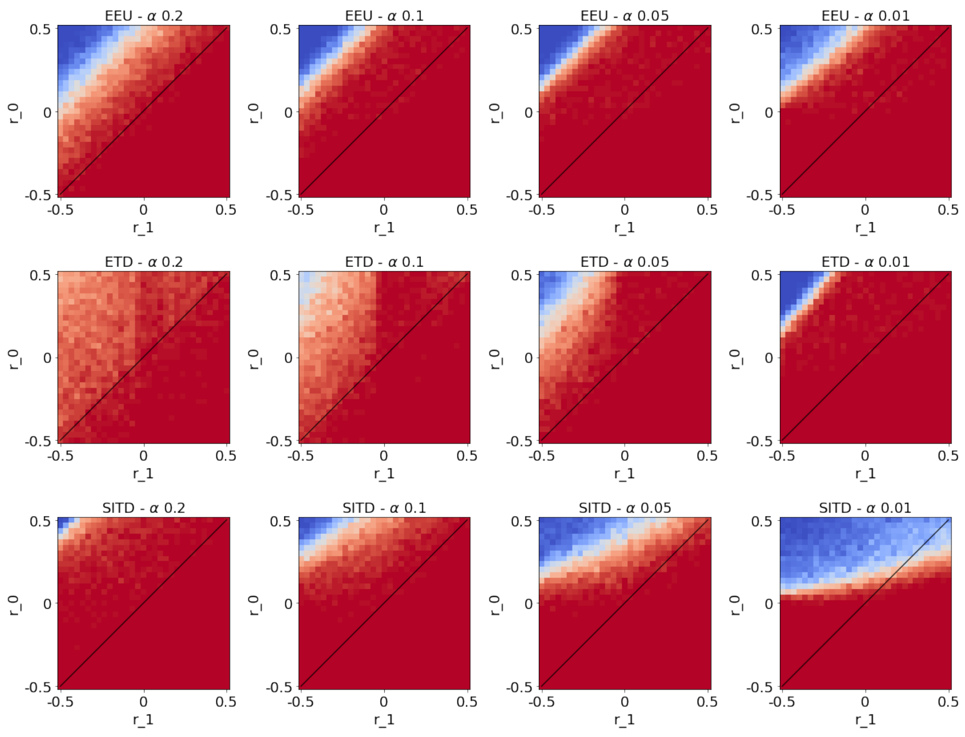


**Figure 5.9:** *Policies obtained by Q-Learning with EEU, ETD, and SITD criteria and varying the learning rate for Two Armed Bandit domain.*

## 5.3 Results for the River Crossing Domain

In this section, experiments were performed using Value Iteration and Q-Learning in the River Crossing domain. Additionally, we incorporate the Deep Q-Network algorithm to verify how all exponential criteria interact with deep learning methods.

### 5.3.1 Varying the risk factor

As stated before, the number of safe steps can be used to analyze the risk attitude. Specifically, for the $10 \times 10$ instance of the River Crossing domain, we have nine possible policies considering this number. To investigate how the sampling algorithms respond to this range of possible policies, we varied the risk factor while maintaining the learning rate $\alpha = 0.1$.

We analyze if the policies encountered by the sampling algorithms are equal to the policy encountered by the Value Iteration algorithm. In these figures, the policies found by Value Iteration are represented by orange points, and Q-Learning or Deep Q-Network are represented by blue points. When these policies are the same the orange points overlay the blue ones.

Furthermore, we examined the distribution of policies that each sampling algorithm converged to, as well as the scenarios in which they encountered difficulties in converging to a proper policy (a policy that successfully guides the agent to the goal state). Convergence issues and overflow problems will be thoroughly analyzed in Sections 5.3.3 and 5.3.4, respectively, providing more detailed insights into these aspects.

### Exponential Expected Utility

The left side of Figure 5.10 shows the number of safe steps in the policies computed by Value Iteration with EEU and the average number of states for the Q-Learning algorithm with EEU considering 25 runs for each configuration.

The right side of Figure 5.10 shows the distribution of safe steps for the Q-Learning algorithm with EEU criterion.

Value Iteration with EEU effectively captures various risk attitudes based on the risk factor, and this behavior is consistent when applying the same criterion to the Q-Learning algorithm, except when the risk factor is close to zero. In such cases, there are configurations where the Q-Learning algorithm converges to policies that differ from those obtained when running the Value Iteration algorithm. For instance, when running Q-Learning with $\lambda = -0.2$, the agent converged to policies with 4, 5, 6, 7, and 8 safe steps. This observation highlights the challenge of calibrating the risk parameter to achieve the desired risk attitude while running sampling algorithms.

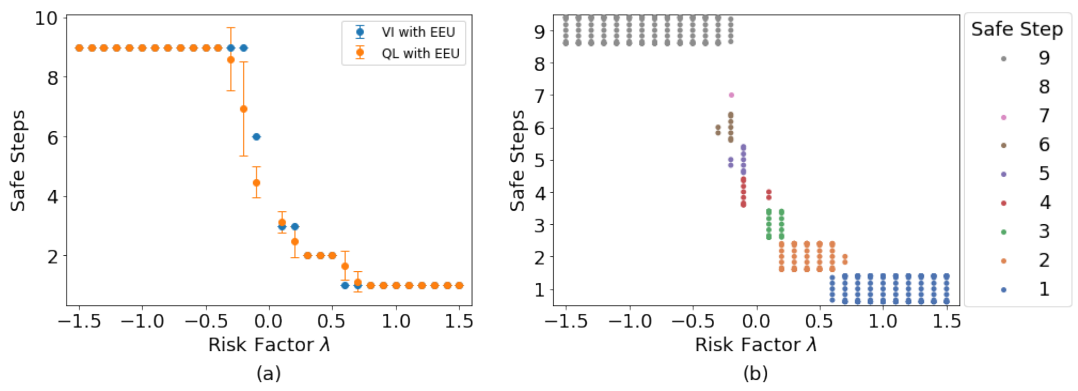


**Figure 5.10:** *Safe steps in the policies for the* $10 \times 10$ *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Q-Learning algorithms with EEU; and (b) distribution of the number of safe steps computed by Q-Learning with EEU.*

Figure 5.11 shows the same experiments but with Deep Q-Network over 15 runs for each configuration. The results obtained from Q-Learning and Deep Q-Network are quite similar in most cases, except for some increased variability in the policies obtained for certain risk factors. For instance, when examining the results for $\lambda = 0.8$, we obtained stable results when running the Q-Learning algorithm (the number of safe steps is 1). However, the same stability was not observed when running the Deep Q-Network algorithm.

The observed change in behavior highlights the increasing challenge of calibration as we introduce additional components to the learning processes. In this case, the first change occurs when we transition from Value Iteration to Q-Learning, where the addition of sampling contributes to the complexity. Furthermore, the introduction of the neural network learning process in the shift from Q-Learning to Deep Q-Network further complicates the calibration. However, it is noteworthy that despite these challenges, the application of EEU still provides some level of stability in the learning process.

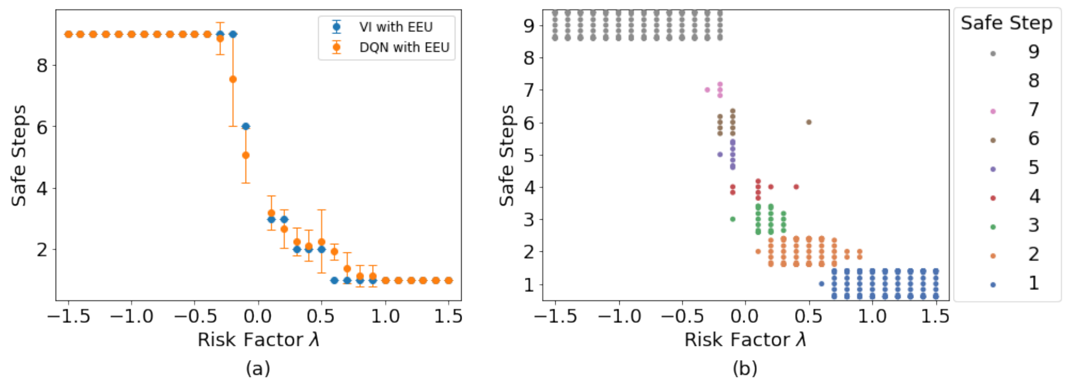


**Figure 5.11:** *Safe steps in the policies for the $10 \times 10$ instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Deep Q-Network algorithms with EEU; and (b) distribution of the number of safe steps computed by Deep Q-Network with EEU.*

## Exponential TD Transformation

In this section, we analyze the ETD criterion. In Figure 5.12, the results obtained for Value Iteration and Q-Learning algorithms are presented. The policies found by the Value Iteration algorithm with ETD criterion are the same as EEU. However, for Q-Learning with ETD, calculation problems arise when the risk factor is lower than -0.7 or greater than 1.2. The behavior observed for Q-Learning with ETD when the risk factor is close to zero remains similar to Q-Learning with EEU.

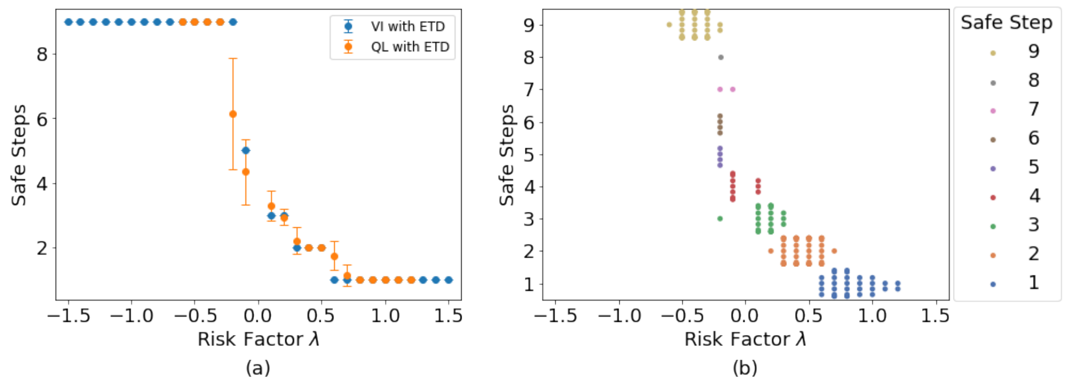


**Figure 5.12:** *Safe steps in the policies for the $10 \times 10$ instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Q-Learning algorithms with ETD; and (b) distribution of the number of safe steps computed by Q-Learning with ETD.*

By applying ETD to Deep Q-Network, we obtained interesting results that can be observed in Figure 5.13. Notably, the ETD criterion performed better with the Deep Q-Network algorithm compared to its performance on Q-Learning. Although Deep Q-Network with ETD still faced calculation problems, these occurred in fewer configurations than in the Q-Learning with ETD.

This finding highlights the difficulty of applying ETD criterion to sampling algorithms such as Q-Learning and Deep Q-Network. However, it also suggests a good relationship between the characteristics of the TD transformation and the neural network learning process.

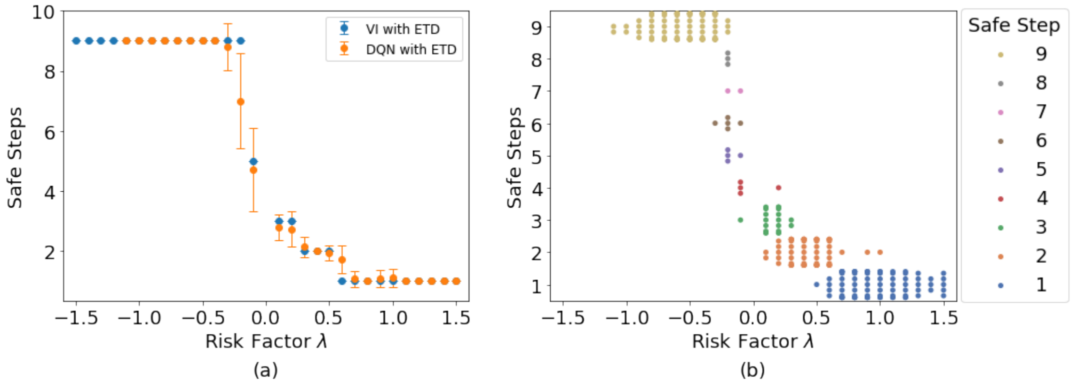


**Figure 5.13:** *Safe steps in the policies for the* 10 × 10 *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Deep Q-Network algorithms with ETD; and (b) distribution of the number of safe steps computed by Deep Q-Network with ETD.*

## Soft Indicator TD Transformation

In this section, we analyze the SITD criterion. The number of safe steps in the policies computed by the Value Iteration algorithm with SITD criterion (Figure 5.14) is different than the ones obtained by the Value Iteration algorithm with ETD and EEU criterion for some configurations.

Figure 5.14 shows that differently from Q-Learning with ETD, Q-Learning with SITD was able to find proper policies for risk factors lower than -0.7. However, unlike EEU, Q-Learning with SITD faced problems in calculating policies for risk factors greater than 1.2.

Figure 5.15 shows the number of safe states in the policies computed by Value Iteration and Deep Q-Network algorithms with SITD. The experiments show that Deep Q-Network with SITD has better behavior than Q-Learning with SITD. In cases where Q-Learning with SITD encountered convergence problems, Deep Q-Network with SITD was able to converge to proper policies. Although convergence problems persisted for risk factors greater than 1.2, Deep Q-Network with SITD exhibited the ability to find proper policies in some runs for these risk factor values, unlike Q-Learning with SITD.

The recurring patterns observed in these experiments further reinforce the good relationship between the characteristics of the TD transformation and the neural network learning process.

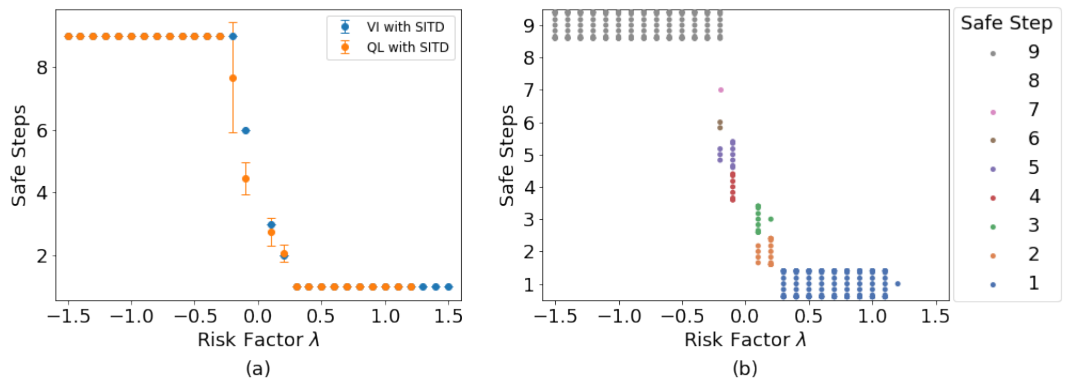


**Figure 5.14:** *Safe steps in the policies for the* 10 × 10 *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Q-Learning algorithms with SITD; and (b) distribution of the number of safe steps computed by Q-Learning with SITD.*

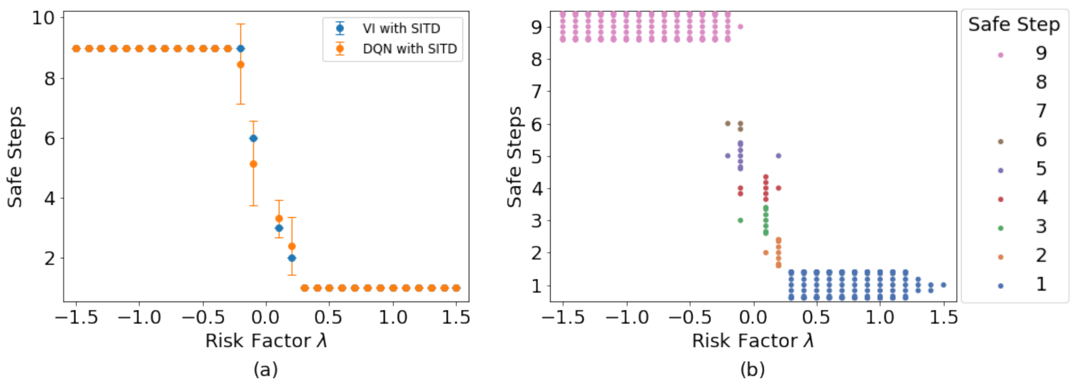


**Figure 5.15:** *Safe steps in the policies for the* 10 × 10 *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Deep Q-Network algorithms with SITD; and (b) distribution of the number of safe steps computed by Deep Q-Network with SITD.*

### 5.3.2 Varying the learning rate

In this section, we analyze the influence of the learning rate when sampling algorithms are used considering different exponential criteria in the River Crossing domain. Specifically, we execute Q-Learning and Deep Q-Network with each criterion considering 15 runs. Each run finishes after 300 episodes. Additionally, we delve into the calculation problems encountered for this domain to identify their underlying causes.

Each run is classified as Proper Policy, Not Proper Policy, Underflow Error, and Overflow Error. If the policy calculated after 300 episodes successfully guides the agent to the goal state, the run is classified as *Proper Policy*. If the calculated policy does not lead the agent to the goal state, we assess the number of numerical underflow errors in the last episode. The run is labeled as *Not Proper Policy* if it encounters less than 20% underflow errors in the total of updates, indicating a high likelihood of convergence with more episodes. If it encounters more or equals 20% of underflow errors, it falls under the category of *Underflow Error*, signifying that it will likely never converge to a proper policy, even with more episodes. If a numerical overflow error occurred while computing the utility function, the run is classified as *Overflow Error*. The percentage of each type of policy found in the runs is shown in the figures in this section.

### Exponential Expected Utility

Upon comparing the three criteria in the Two Armed Bandit experiment, we observed that EEU exhibited the highest level of stability when varying the learning rate, as depicted in Figure 5.9. To further validate this, Figures 5.16(a) and 5.16(b) show the percentage of each type of policy returned by Q-Learning with EEU and Deep Q-Network with EEU, respectively. We observe in both figures that reducing the learning rate leads to some convergence problems within the considered number of episodes. This issue becomes more pronounced when transitioning from Q-Learning to Deep Q-Network for $\alpha = 0.01$. However, EEU continues to provide some degree of stability, as demonstrated previously in the Two Armed Bandit domain.

### Exponential TD Transformation

Figures 5.17(a) and 5.17(b) show the percentage of each type of policy returned by Q-Learning with ETD and Deep Q-Network with ETD, respectively.

Figure 5.17(a) shows that the Q-Learning algorithm with ETD criterion faces overflow problems. As we decrease the learning rate, the percentage of overflow errors decreases but we start to find more convergence problems when running with risk factors close to zero (i.e., the percentage of not proper policies increases). The convergence problem can be fixed by increasing the number of episodes but as the objective of this experiment is to show the stability of each of the methods, we keep it as 300.

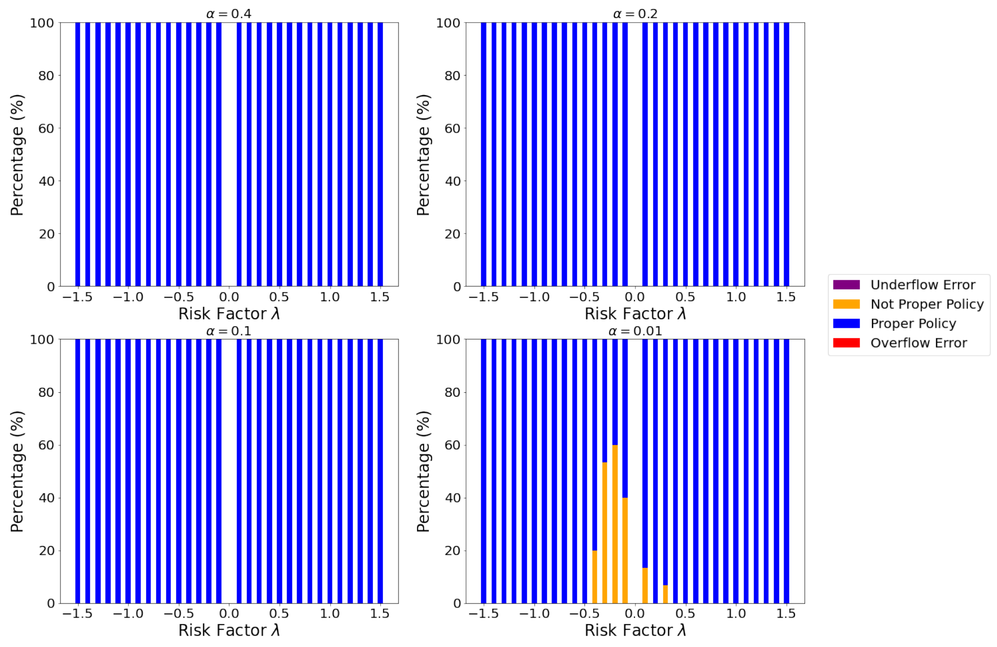
Figure 5.17(b) shows that Deep Q-Network algorithm with ETD encounters slightly fewer overflow problems when compared to Q-Learning algorithm with ETD. However, the observed behavior regarding the decrease in learning rate remains similar. As the learning rate decreases, the occurrence of overflow problems diminishes. However, this also leads to an increase in convergence problems for risk factors close to zero.

This behavior highlights the augmented challenge of calibrating sampling algorithms with ETD criterion when compared to EEU criterion. The observed increase in calibration difficulty underscores the need for careful parameter tuning and adjustment when utilizing ETD in conjunction with sampling learning algorithms.

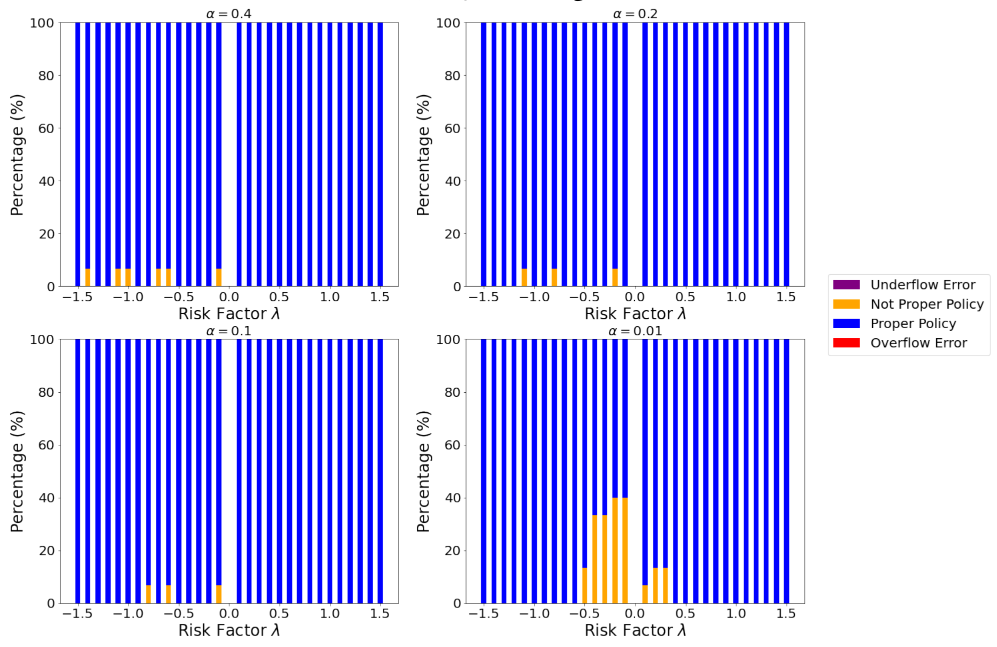
### Soft Indicator TD Transformation

The analysis conducted on the Two Armed bandit, as depicted in Figure 5.9, shown that the Q-Learning with SITD performed better with higher learning rates but poorly with smaller ones. This behavior is further confirmed in Figure 5.18(a), where we observe that Q-Learning with SITD achieved more proper policies with higher learning rates.

Furthermore, Deep Q-Network with SITD criterion resulted in an improvement in the percentage of convergence to proper policies when compared to Q-Learning with SITD criterion (Figure 5.18). The utilization of SITD criterion led to a higher number of proper policies in the Deep Q-Network algorithm as happened with the other TD transformation (ETD criterion).



(a) Q-Learning



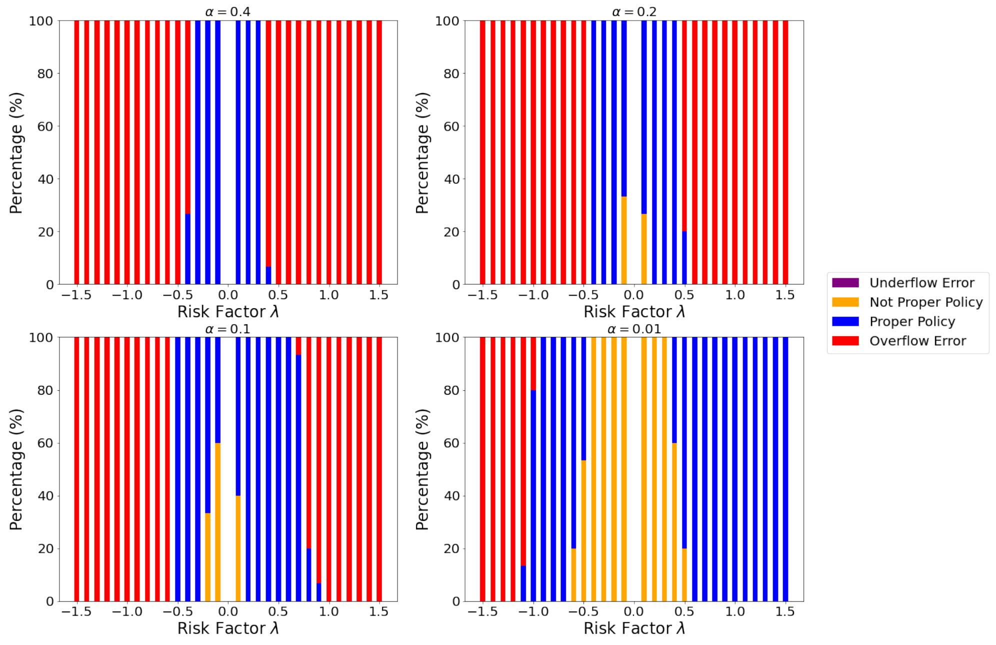
(b) Deep Q-Network

**Figure 5.16:** *Policies obtained running Q-Learning and Deep Q-Network algorithms with EEU criterion for the* $10 \times 10$ *instance of the River Crossing domain.*

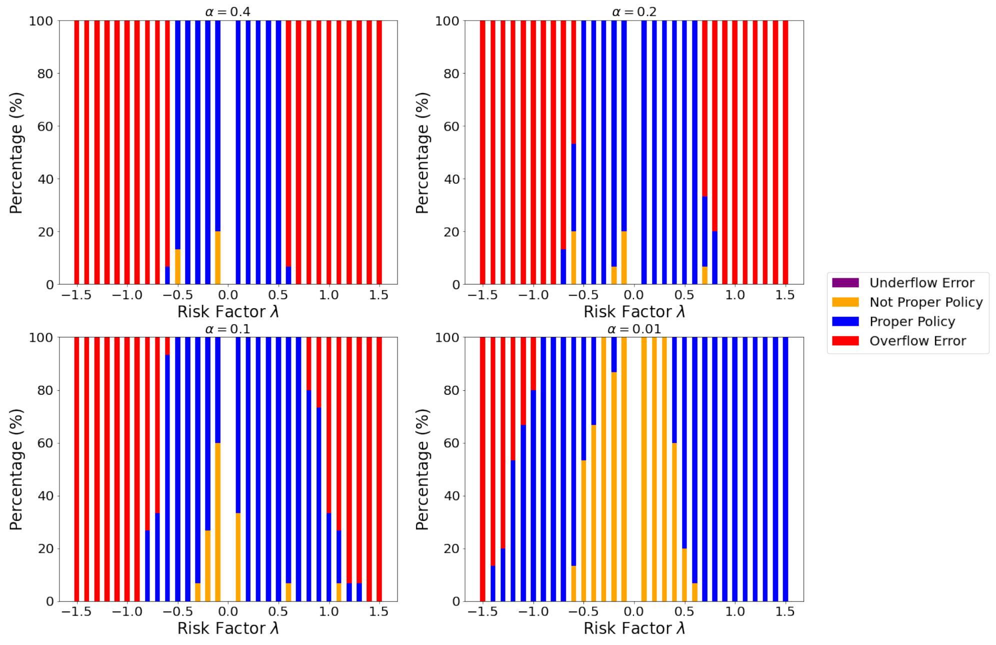
### 5.3.3 Convergence analysis

Through the various experiments conducted in the River Crossing domain, two key differences emerged when running the three exponential risk criteria.

The first difference is related to convergence problems. We observed that the conver-



(a) Q-Learning

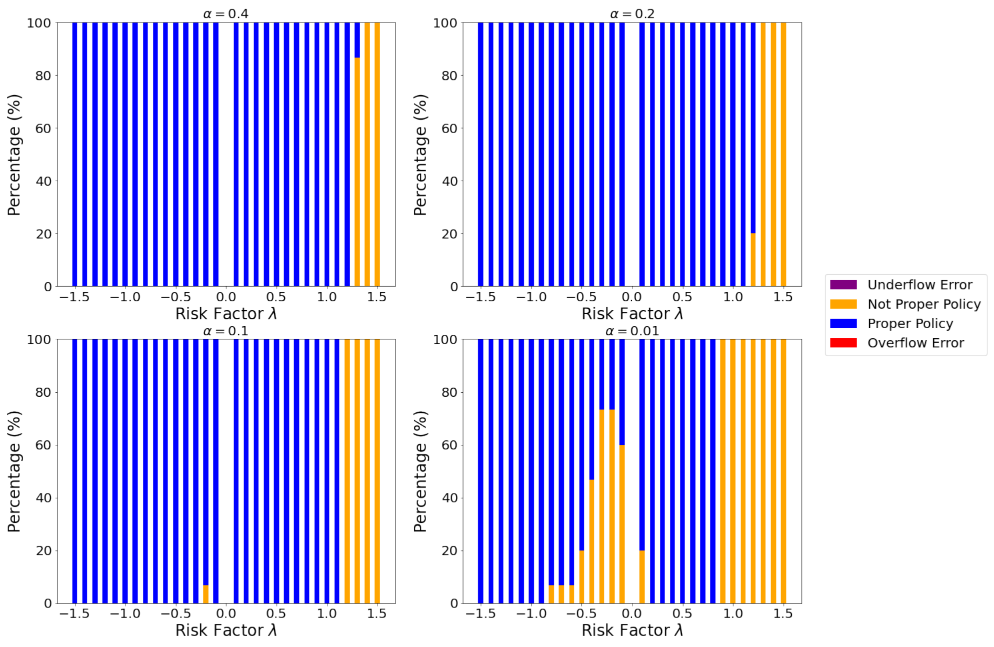


(b) Deep Q-Network

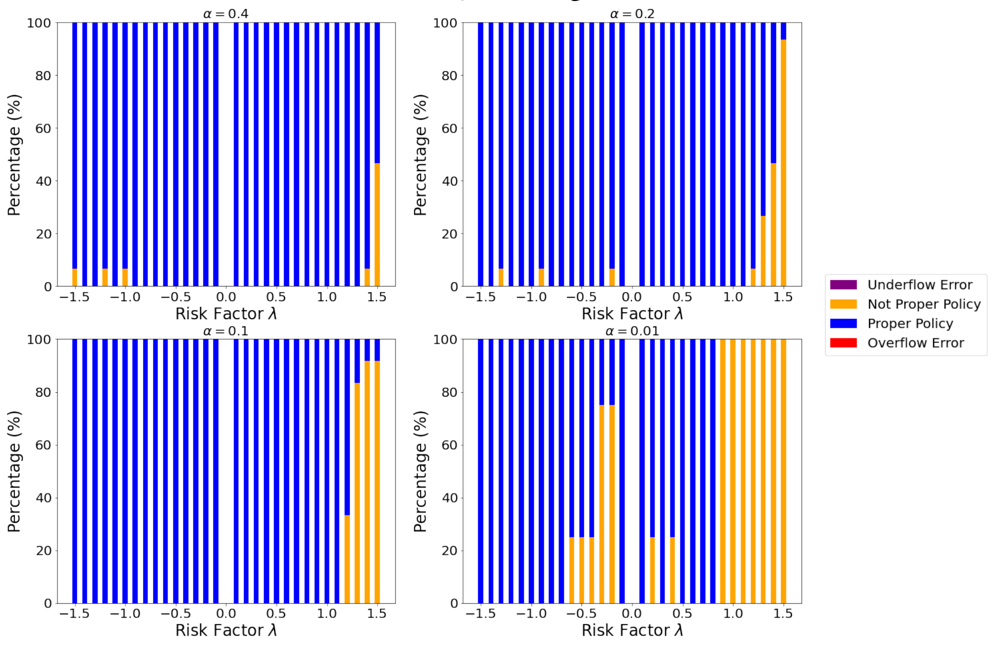
**Figure 5.17:** *Policies obtained running Q-Learning and Deep Q-Network algorithms with ETD criterion for the* $10 \times 10$ *instance of the River Crossing domain.*

gence of the algorithms varied based on the learning rate and the specific risk criterion employed. Different risk criteria exhibited different behaviors in terms of convergence, with some criteria leading to more pronounced convergence problems at certain learning rates.

The second difference is associated with overflow issues encountered during the



(a) Q-Learning



(b) Deep Q-Network

**Figure 5.18:** *Policies obtained running Q-Leaning and Deep Q-Network algorithms with SITD criterion for the* $10 \times 10$ *instance of the River Crossing domain.*

learning process. These overflow problems arose due to the complex nature of the learning algorithms and the risk criteria being used. The overflow problems will be analyzed in Section 5.3.4.

In this section, our focus is on the convergence problems encountered in the River Crossing domain experiments. To investigate this further, Figure 5.19 shows the number of iterations required for convergence when running the three exponential criteria (EEU, ETD, SITD) with the Value Iteration algorithm. The results are showcased for four different learning rates (0.4, 0.2, 0.1, and 0.01).

Figure 5.19 shows that EEU exhibits less variance in the number of iterations required for convergence as the risk factor is varied. On the other hand, both TD Transformations (ETD and SITD) display a peak in the number of iterations for risk factors closer to zero.

This finding carries significant importance, as it directly impacts the application setup and introduces additional complexity in adjusting the learning parameters for larger problems. The need to carefully calibrate the risk factor becomes crucial to ensure proper convergence and desired learning outcomes. It highlights the challenges involved in parameter tuning and emphasizes the importance of understanding the implications of different risk criteria in learning algorithms for real-world applications.

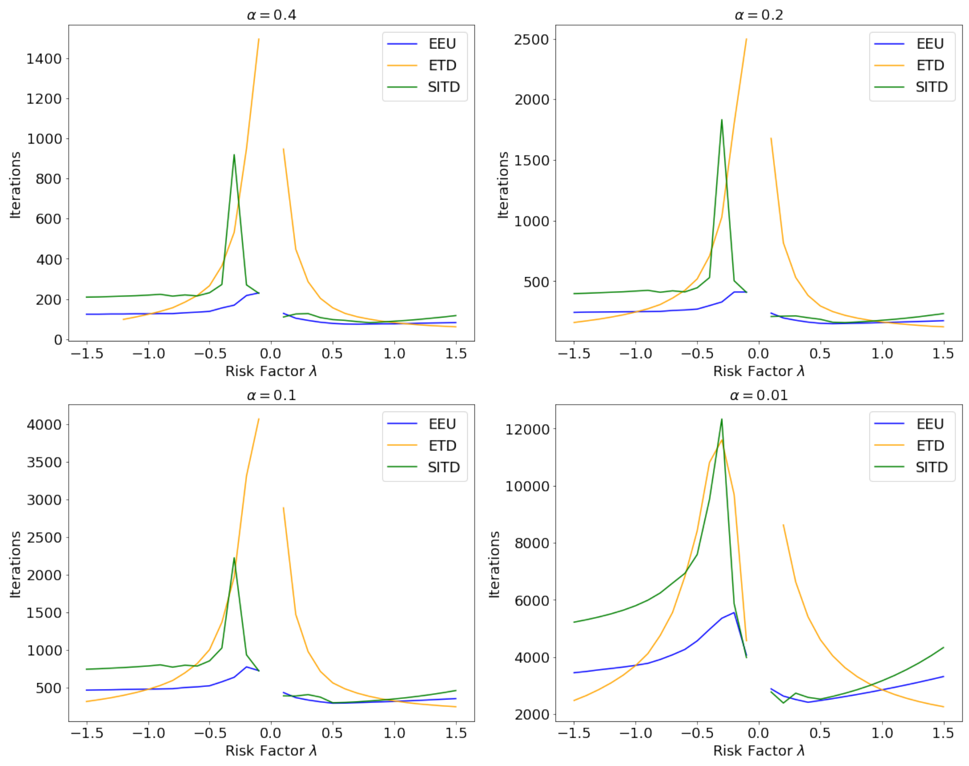


**Figure 5.19:** *Iterations to converge for the* $10 \times 10$ *instance of the River Crossing domain running Value Iteration.*

Theorem 1 demonstrates that EEU and ETD criteria converge to the same value for each state. Figure 5.20 shows the initial state value ($V_{s_0}$) per iteration running Value Iteration and different criteria. As expected, EEU and ETD criteria converge to the same value for $s_0$. Additionally, the SITD criteria converge to a completely different value in certain cases, as it applies a distinct transformation function. This distinction highlights the unique characteristics and effects of each risk criterion on the convergence behavior of the learning algorithms.

Upon closer examination, it became apparent that SITD encountered difficulties in

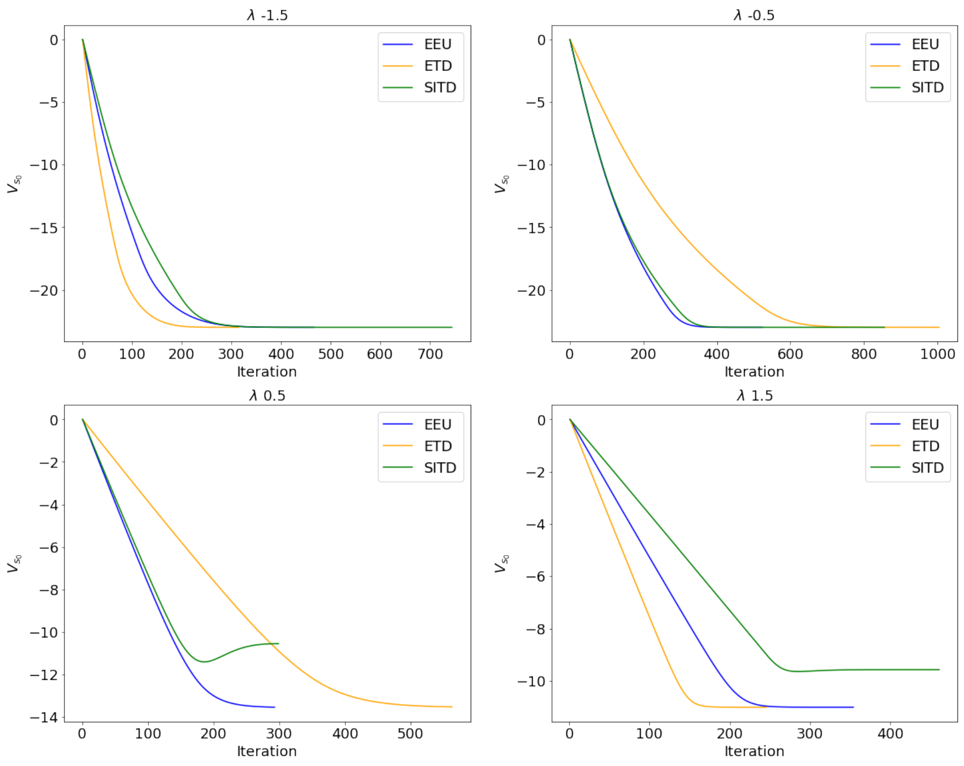


**Figure 5.20:** *$V_{s0}$ per iteration of the $10 \times 10$ instance of the River Crossing domain running Value Iteration.*

converging when the risk factor was set to 1.5 across various experimental setups involving sampling learning algorithms. Figure 5.21 displays the Q-Learning convergence curve for different risk factors. When the risk factor is set to 1.5, the SITD criterion requires significantly more episodes to converge compared to the other exponential criteria. This issue was not observed in the Value Iteration convergence results shown in Figure 5.19, but it emerged as a recurring problem in the Q-Learning and Deep Q-Network setups. This highlights a specific challenge associated with the convergence of the SITD criterion in the context of reinforcement learning algorithms.

### 5.3.4 Overflow analysis

One significant factor that added complexity to the experiment setup was the occurrence of overflow problems while running the ETD criterion. As both the EEU and ETD criteria converged to the same values for all states, the overflow issues were primarily attributed to the internal calculations involved in the learning process. To investigate this further, we compute the maximum absolute value of each utility calculation performed by Q-Learning with different criteria and additionally, we implement the truncated version of the ETD criterion (Shen *et al.*, 2014) that was explained in Section 4.6.1 for Q-Learning and Deep Q-Network.

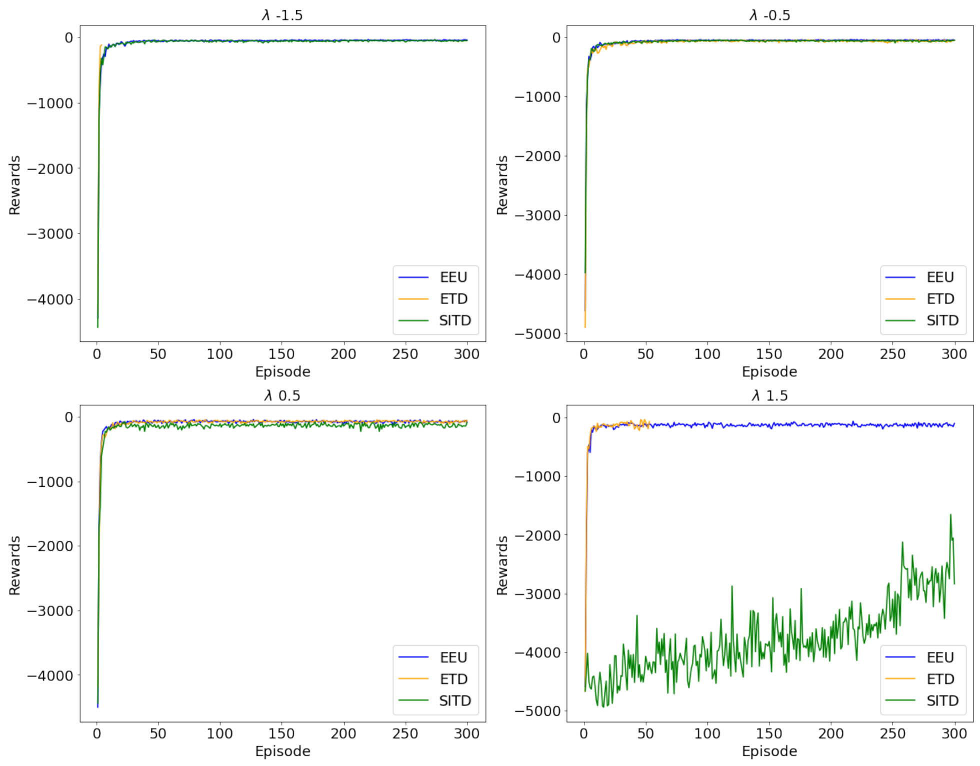


**Figure 5.21:** *Q-Learning convergence curve of the* $10 \times 10$ *instance of the River Crossing domain.*

## Maximum absolute value of each utility

Figure 5.22 depicts the maximum absolute value of each utility calculation per episode computed by Q-Learning with different criteria. ETD curves exhibit spikes, indicating the presence of calculation values that approached or exceeded the limits of the numerical representation. Although these spikes were captured without triggering overflow errors, even larger spikes were likely responsible for the overflow issues encountered.

The overflow problems encountered in the ETD criterion underscore the challenge of managing numerical precision and the potential impact on the convergence and stability of the learning algorithms.

## Truncated version of the ETD criterion.

Figure 5.23 displays the policies obtained by the Value Iteration and Q-Learning algorithms with the truncated version of the ETD criterion for different risk factors. Truncation effectively resolves overflow problems in scenarios with negative risk factors. However, for most positive risk factor values, the truncated ETD still struggles to converge to a proper policy. Additionally, even for risk factors such as 1.0, where the agent was able to converge successfully with the non-truncated version, the truncated ETD now exhibits convergence issues.

An interesting observation is that the truncated version of the ETD criterion performed worse than the non-truncated version when applied to the Deep Q-Network (compare Figures 5.24 and 5.13). Despite addressing overflow problems, the truncated version of the

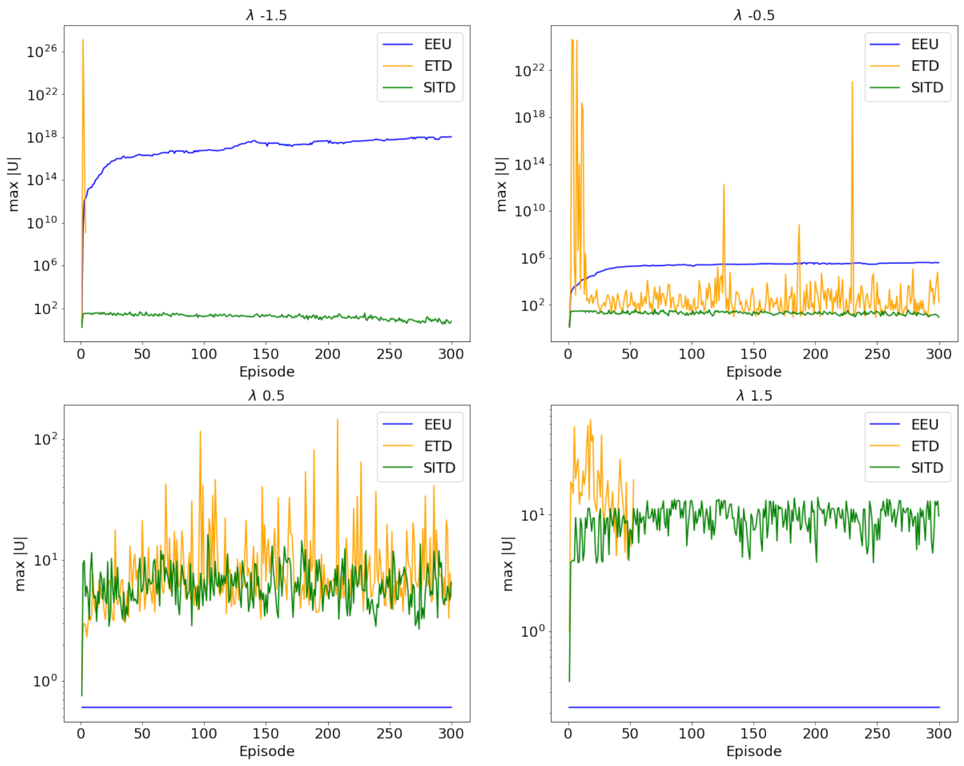


**Figure 5.22:** *Max* $|U|$ *per episode with Q-Learning for the* $10 \times 10$ *instance of the River Crossing domain.*

ETD criterion exhibited poorer performance in terms of finding proper policies compared to the non-truncated version.

Figures 5.25(a) and 5.25(b) show the percentage of each type of policy returned by Q-Learning with the truncated version of ETD and Deep Q-Network with the truncated version of ETD, respectively, for different values of $\alpha$.

Figure 5.25(a) shows that Q-Learning with the truncated version of ETD no longer encountered overflow problems. However, new issues emerged which are underflow errors. Note that the not proper policy issues were also encountered in the ETD experiment without truncation technique (Figure 5.17 (a)).

These results suggest that the Q-Learning with the truncated version successfully mitigates the numerical instability caused by spikes. However, the truncated ETD method begins encountering more underflow errors as the risk factor or learning rate is increased. It highlights the trade-off between addressing overflow issues and underflow errors when applying the truncated version of ETD

In contrast to Q-Learning with the truncated version of ETD, Figure 5.25(b) reveals that the truncated version still encountered overflow problems when executed with the Deep Q-Network algorithm.

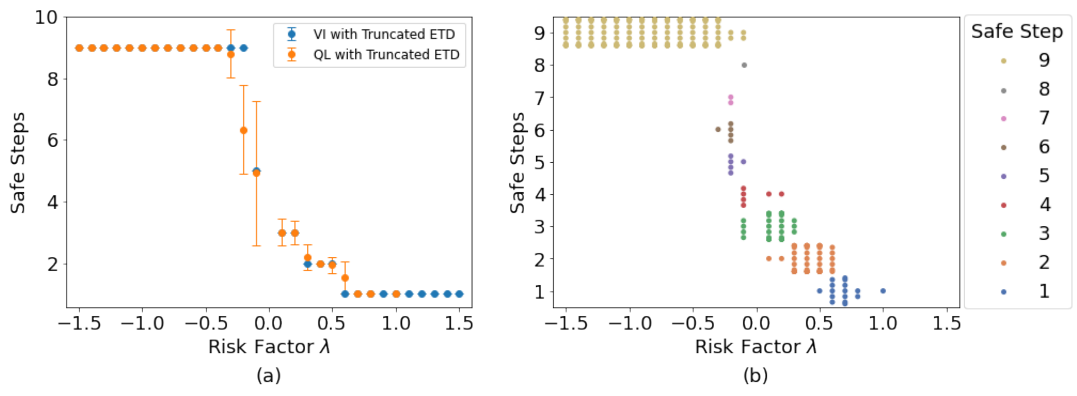


**Figure 5.23:** *Safe steps in the policies for the* 10 × 10 *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Q-Learning algorithms with the truncated version of ETD; and (b) distribution of the number of safe steps computed by Q-Learning with the truncated version of ETD.*

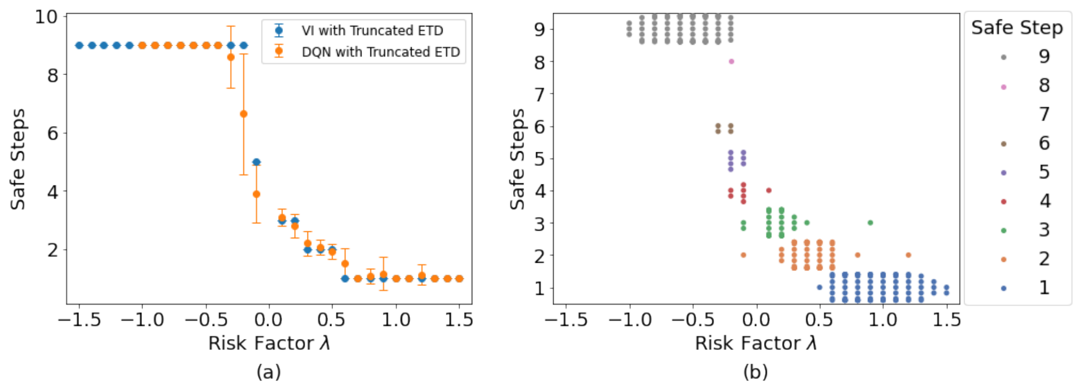


**Figure 5.24:** *Safe steps in the policies for the* 10 × 10 *instance of the River Crossing domain. (a) Number of safe states computed by Value Iteration and Deep Q-Network algorithms with the truncated version of ETD; and (b) distribution of the number of safe steps computed by Deep Q-Network with the truncated version of ETD.*

### Overflow analysis when the reward is increased.

In the experiments presented before we observed that EEU and SITD criteria did not encounter overflow errors with Q-Learning and Deep Q-Network for River Crossing domain instances with size 10x10 and reward of -1. While running the ETD criterion it faced overflow errors for both algorithms, the Truncated ETD criterion presented overflow errors only when combined with the Deep Q-Network algorithm. With the focus on observing how the criteria behave with greater rewards, we now change the reward of -1 that the agent receives at each action on the River Crossing domain to -100. In this experiment, we fixed the learning rate to 0.1.

Figure 5.26 shows that the EEU criterion started to present overflow errors for risk factors lower than -0.2 on the Q-Learning algorithm and for risk factors lower than -0.1 on Deep Q-Network. The overflow error became an issue in both ETD and Truncated ETD for most of the scenarios on Q-Learning and all scenarios on Deep Q-Network. The SITD also started to face overflow errors for negative risk factors.

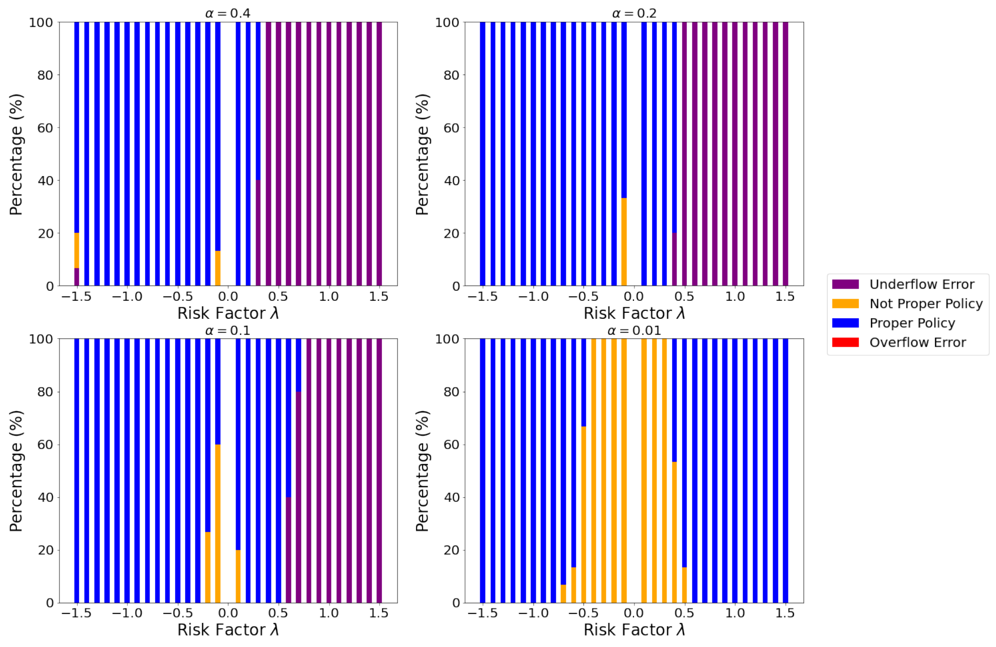
Figure 5.26(a) also shows that even though all the other criteria are facing overflow errors, the EEU with LogSumExp on the Q-Learning algorithm was able to find the proper policy for all risk factors. In Figure 5.26(b) we observe that the EEU with LogSumExp and Deep Q-Network also produced a better result than the other criteria.

### 5.3.5 Scalability Analysis

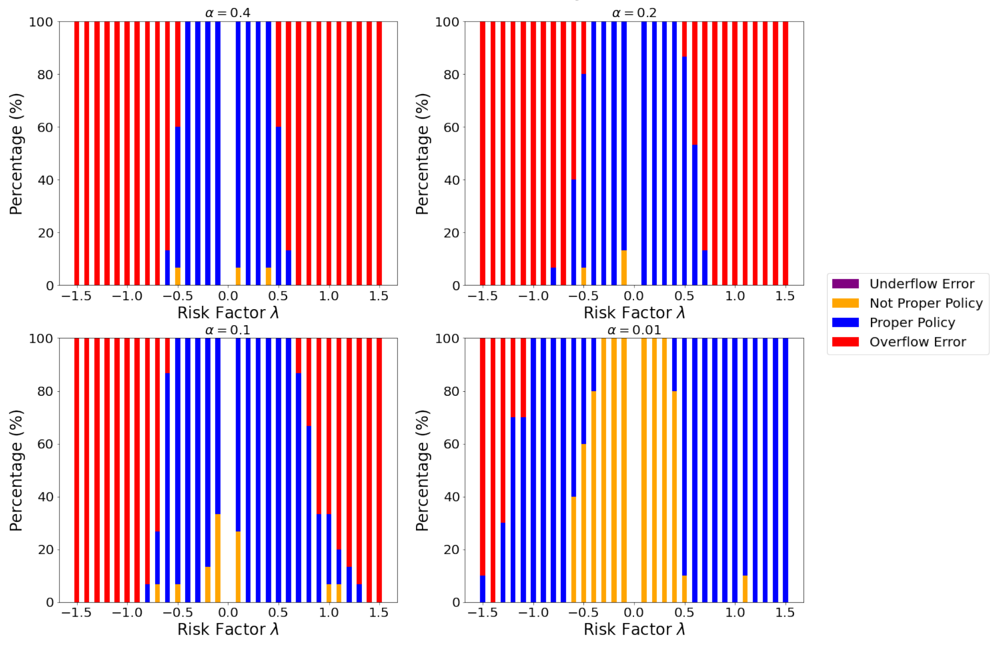
In this section, we run the Value Iteration algorithm with 5 River Crossing domain instances with different sizes ($6 \times 6$, $8 \times 8$, $10 \times 10$, $12 \times 12$, and $14 \times 14$). The results are showcased for four different risk factors (-1.5, -0.5, 0.5, and 1.5), a fixed learning rate of 0.1, and the reward is set back to -1.

The number of iterations to converge for EEU is always between 209 and 766 for all domain instances and risk factors, as shown in Figure 5.27. Conversely, the number of iterations to converge for SITD varies more (between 212 and 1281), and for ETD varies even more (between 148 and 1471).

This finding and the other results presented before in this chapter indicate that the algorithms with EEU criterion are more stable than the algorithms with ETD and SITD criteria and it also indicates that algorithms with EEU criterion are easier to calibrate than the other algorithms.

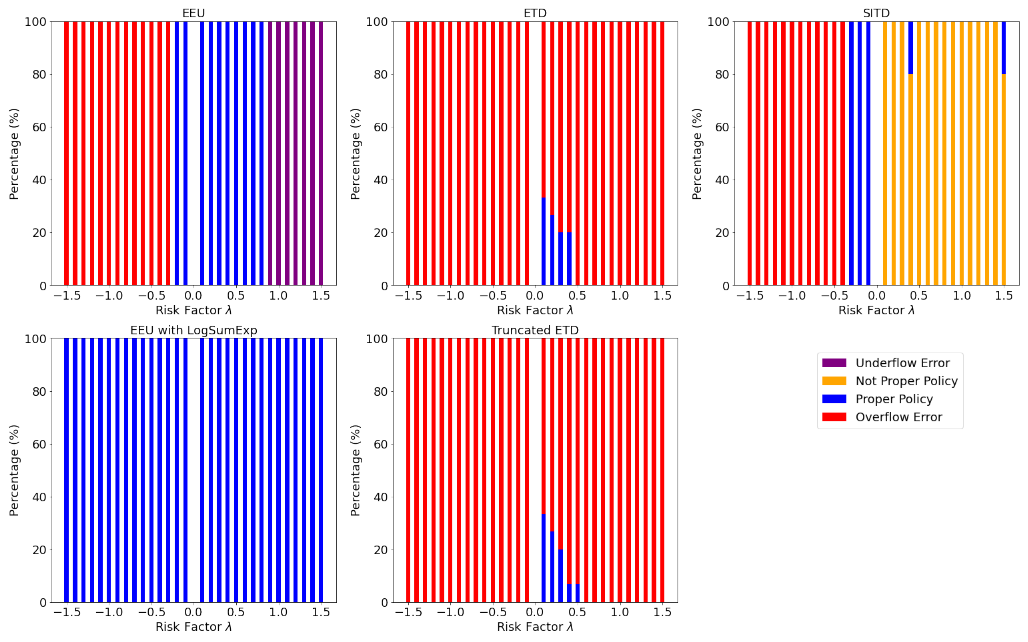


(a) Q-Learning

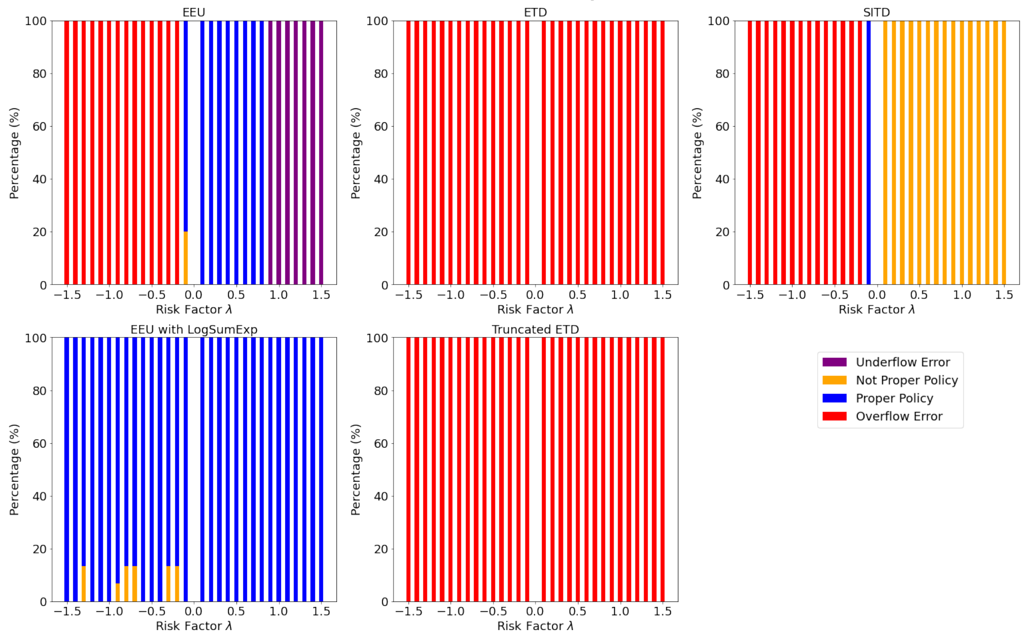


(b) Deep Q-Network

**Figure 5.25:** *Policies obtained running Q-Leaning and Deep Q-Network algorithms with the truncated version of ETD for the* $10 \times 10$ *instance of the River Crossing domain.*



(a) Q-Learning



(b) Deep Q-Network

**Figure 5.26:** *Policies obtained running Q-Leaning and Deep Q-Network algorithms with all criteria for the* $10 \times 10$ *instance of the River Crossing domain with rewards of -100.*

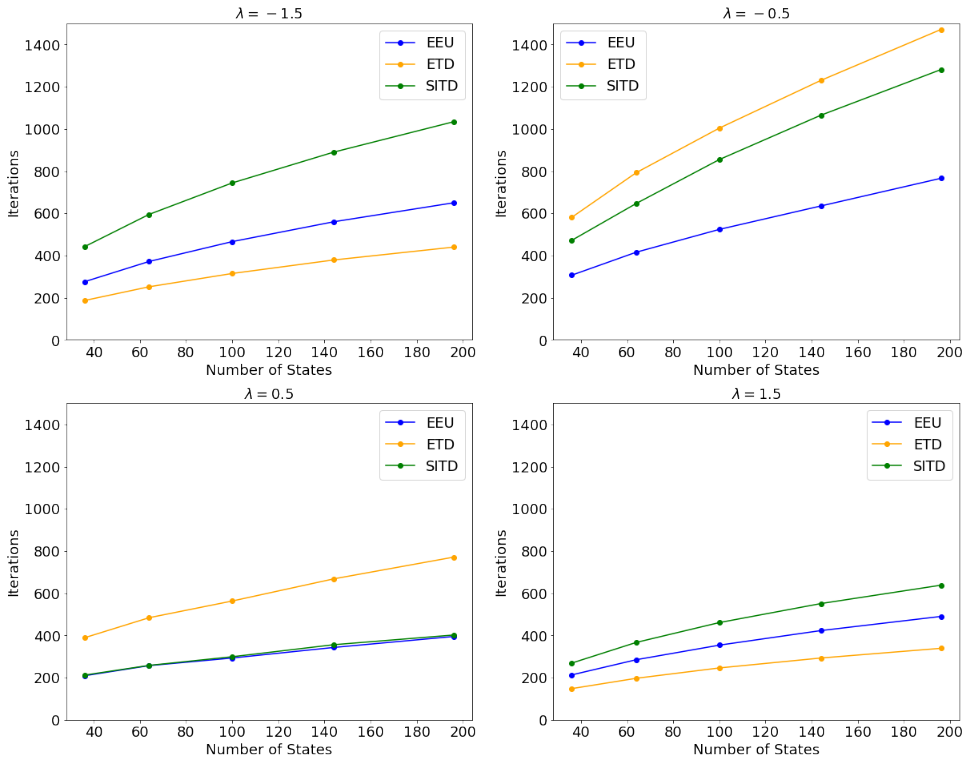


**Figure 5.27:** *Number of iterations to converge for the* $6 \times 6$, $8 \times 8$, $10 \times 10$, $12 \times 12$ *and* $14 \times 14$ *instances of the River Crossing domain running Value Iteration for different criteria.*

# Chapter 6

# Conclusions and future work

Reinforcement Learning has been a groundbreaking advancement in artificial intelligence, empowering agents to learn intricate decision-making policies for sequential actions. However, in many real-world scenarios, the consideration of risk attitudes becomes crucial when applying this technique. Risk-Sensitive Reinforcement Learning emerges as a solution, incorporating risk criteria into the decision-making process. Among these criteria, exponential-based methods have been extensively studied and applied.

While the adoption of exponential criteria has garnered significant attention in the literature, a notable gap exists in the comprehensive analysis of how exponential criteria respond when integrated with learning parameters and approximations, especially in conjunction with Deep Reinforcement Learning. The absence of comprehensive exploration poses a challenge in calibrating and selecting the most suitable technique. In this research, we investigate the applicability of exponential utility in Deep Reinforcement Learning. By understanding how to effectively combine exponential criteria with Deep Reinforcement Learning and addressing their interactions with learning parameters and approximations, we seek to unlock new possibilities for more robust and adaptive risk-sensitive decision-making in complex environments.

## 6.1 Contributions

In this section, we highlight the contributions that this dissertation brings to the area of Risk Sensitive Reinforcement Learning.

- **Exponential criteria and Reinforcement Learning Framework:** This research introduces a framework that facilitates the use of exponential criteria in various forms to model risk attitudes in both Reinforcement Learning and Deep Reinforcement Learning algorithms. The framework includes three distinct criteria: (i) Exponential Expected Utility, (ii) Exponential TD Transformation, and (iii) Soft Indicator TD Transformation.

- **Deep Reinforcement Learning with exponential risk criteria**: In this work, we proposed an intuitive and more stable way of combining Deep Reinforcement

Learning and exponential risk criteria. The proposed algorithm maintains the complete Q-Learning with EEU and Q-Learning with TD Transformation updates. By transforming values into their certainty equivalents, the algorithm normalizes the values required for network approximation. This normalization avoids significantly large or small values resulting in a faster and more stable learning process

- **Fit between Exponential criteria and Deep Reinforcement Learning:** Our experiments yielded significant results, indicating that the combination of EEU and Deep Learning has ensured stable learning with outcomes comparable to those obtained through Reinforcement Learning without artificial neural networks. Additionally, the utilization of ETD and SITD, when integrated with artificial neural networks, outperformed the results achieved by using a Q-Table (Q-Learning), surpassing our expectations.

- **EEU and ETD converge to the same value:** In this study, we demonstrate that when applying the EEU and ETD, both criteria converge to the same value (Theorem 1).

- **EEU robustness:** During our investigation, we encountered an undesirable outcome when using ETD and SITD. These criteria exhibited higher levels of instability as we manipulated the learning rate and risk attitude in both domains used in the experiments. In contrast, the EEU criterion consistently demonstrated superior stability across all experiments, exhibiting less variance in the number of iterations required to converge in the Value Iteration with EEU algorithm. This confirmed the robustness and effectiveness of the extensively studied EEU criteria, showcasing its potential for risk modeling in Reinforcement Learning and Deep Reinforcement Learning scenarios.

- **Difficulty when applying sampling and defining risk attitude:** A significant drawback observed in this work while applying the three exponential criteria was the undesirable policies when utilizing sampling algorithms (Q-Learning and Deep Q-Network) in the River Crossing Domain. Particularly, for certain cases (where $\lambda$ is close to zero), the agent converged to policies that differ from those obtained through Value Iteration algorithms. This highlights the challenge of calibrating the risk parameter when combining exponential risk criteria with sampling algorithms.

- **Handling overflow with truncate technique:** An additional contribution of this research was verifying the effectiveness of the truncate technique in addressing overflow problems. The truncate technique was demonstrated to significantly mitigate overflow issues during internal calculations within the algorithms. Particularly, its application to ETD was crucial, as this method tends to encounter peaks of utility values during the convergence process. By employing the Truncate technique, we were able to ensure stable computations and prevent numerical instability, enhancing the overall performance of ETD. However, for most positive risk factor values, the truncated ETD still struggles to converge to a proper policy since it started to encounter underflow errors.

- **Handling overflow with LogSumExp:** In addition, this research also addresses the issue of overflow problems when using the EEU criteria. A theoretical contribution is made by proposing the application of the *LogSumExp* technique to mitigate numerical overflow in Q-Learning with EEU and Deep Q-Learning with EEU algorithms. Its effectiveness is also demonstrated in the experiments.

## 6.2 Publication

### 6.2.1 Risk Sensitive Markov Decision Process for Portfolio Management

The work *Risk Sensitive Markov Decision Process for Portfolio Management* (Neto *et al.*, 2020) proposes a novel strategy for modeling the Portfolio Management problem, specifically targeting day trade operations. The objective is to leverage dynamic programming algorithms, such as Value Iteration, to efficiently find solutions for this problem. In addition, a new risk attitude measure based on Conditional Value-at-Risk (CVaR) is introduced to assess and evaluate the level of risk tolerance within the portfolio management framework.

The conducted experiments involve applying Value Iteration with EEU criteria to the portfolio management problem. The results demonstrate the effectiveness of the proposed modeling strategy, as it accurately represents the dataset and enables Risk Sensitive Value Iteration to derive policies with varying risk attitudes.

Furthermore, the experiments highlight the significance of employing robust risk attitude measures, such as the one based on CVaR.

This work was published at the Mexican International Conference on Artificial Intelligence 2020 (MICAI 2020).

### 6.2.2 Risk Sensitive with Exponential Functions in Reinforcement Learning: An Empirical Analysis

The contributions of this dissertation are being compiled into a new paper that is being prepared to be submitted.

## 6.3 Future work

Throughout the development of this research, several ideas and gaps surfaced, presenting promising avenues for future investigations and improvements. These potential areas for future work include:

- During the experiments conducted for this work, we opted to assign a fixed learning rate to each experiment configuration. This decision was made to facilitate comparisons among different methods with varying convergence behaviors. However, it is worth noting that in the literature, there are techniques that demonstrate higher stability during the learning process, such as learning rate decay. Conversely, some studies suggest that certain techniques, like changing the batch size in Deep Reinforcement Learning methods, should be applied instead (Smith *et al.*, 2017). A further investigation would involve exploring the application of both learning rate decay and batch size adjustments in Risk Deep Reinforcement Learning.

- An intriguing observation from the analysis of EEU and ETD criteria is that they both converge to the same state value. However, despite this similarity, these criteria exhibit distinct learning processes and respond differently to the learning rate $\alpha$. A compelling area for further investigation would be to explore the possibility of combining both criteria, leveraging the strengths of each, to develop a more stable convergence criterion.
- Value Iteration with EEU, Q-Learning with EEU, and Deep Q-Learning with EEU could use Soft Indicator as the utility function $U$. However, further analysis is required to investigate the potential advantages of combining Soft Indicator criteria with these algorithms.
- The experiments revealed that combining exponential criteria with sampling algorithms (Q-Learning and Deep Q-Network) can result in the agent converging to undesirable policies in certain scenarios. This sensitivity to risk parameter calibration highlights the necessity for further investigation into how to prevent such behavior.
- Two techniques to handle overflow were presented in this work: (i) truncating exponential utility function, and (ii) applying *LogSumExp* on EEU criteria. However, further analyses of each technique and how they can be related can contribute to preventing numerical overflow and adding more stability to the learning processes.

# References

[Abbeel *et al.* 2010] Pieter Abbeel, Adam Coates, and Andrew Y Ng. "Autonomous helicopter aerobatics through apprenticeship learning". *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639 (cit. on pp. 1, 17).

[Andersen *et al.* 2020] Per-Arne Andersen, Morten Goodwin, and Ole-Christoffer Granmo. "Towards safe reinforcement-learning in industrial grid-warehousing". *Information Sciences* 537 (2020), pp. 467–484 (cit. on pp. 1, 18).

[Bäuerle and Rieder 2014] Nicole Bäuerle and Ulrich Rieder. "More risk-sensitive markov decision processes". *Mathematics of Operations Research* 39.1 (2014), pp. 105–120 (cit. on pp. 2, 18, 20).

[Bellman 1957] R. E. Bellman. *Dynamic Programming*. USA: Princeton University Press, 1957 (cit. on p. 6).

[Bertsekas 1995] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Vol. 1. 2. Athena Scientific Belmont, MA, 1995 (cit. on p. 7).

[Bouakiz and Sobel 1992] Mokrane Bouakiz and Matthew J Sobel. "Inventory control with an exponential utility criterion". *Operations Research* 40.3 (1992), pp. 603–608 (cit. on pp. 1, 18).

[Cavazos-Cadena and Hernández-Hernández 2011] Rolando Cavazos-Cadena and Daniel Hernández-Hernández. "Discounted approximations for Risk-sensitive average criteria in Markov decision chains with finite state space". *Mathematics of Operations Research* 36.1 (2011), pp. 133–146 (cit. on pp. 1, 2, 18).

[Yi Chen and Gao 2016] Yi Chen and David Y Gao. "Global solutions to nonconvex optimization of 4th-order polynomial and log-sum-exp functions". *Journal of Global Optimization* 64.3 (2016), pp. 417–431 (cit. on p. 30).

[Chow and Ghavamzadeh 2014] Yinlam Chow and Mohammad Ghavamzadeh. "Algorithms for cvar optimization in mdps". *Advances in neural information processing systems* 27 (2014) (cit. on pp. 2, 18).

[Chung and Sobel 1987] Kun-Jen Chung and Matthew J Sobel. "Discounted MDP's: distribution functions and exponential utility maximization". *SIAM journal on control and optimization* 25.1 (1987), pp. 49–62 (cit. on pp. 1, 2, 18).

[Dalal *et al.* 2018] Gal Dalal *et al.* "Safe exploration in continuous action spaces". *arXiv preprint arXiv:1801.08757* (2018) (cit. on pp. 1, 18).

[Delétang *et al.* 2021] Grégoire Delétang *et al.* "Model-free risk-sensitive reinforcement learning". *arXiv preprint arXiv:2111.02907* (2021) (cit. on pp. 2, 18, 26, 34).

[Driessens and Džeroski 2004] Kurt Driessens and Sašo Džeroski. "Integrating guidance into relational reinforcement learning". *Machine Learning* 57 (2004), pp. 271–304 (cit. on pp. 1, 17).

[Du *et al.* 2022] Yihan Du, Siwei Wang, and Longbo Huang. "Risk-sensitive reinforcement learning: iterated cvar and the worst path". *arXiv preprint arXiv:2206.02678* (2022) (cit. on pp. 2, 18).

[Fei, Yang, Yudong Chen, and Wang 2021] Yingjie Fei, Zhuoran Yang, Yudong Chen, and Zhaoran Wang. "Exponential bellman equation and improved regret bounds for risk-sensitive reinforcement learning". *Advances in Neural Information Processing Systems* 34 (2021), pp. 20436–20446 (cit. on pp. 2, 18).

[Fei, Yang, Yudong Chen, Wang, and Xie 2020] Yingjie Fei, Zhuoran Yang, Yudong Chen, Zhaoran Wang, and Qiaomin Xie. "Risk-sensitive reinforcement learning: near-optimal risk-sample tradeoff in regret". *Advances in Neural Information Processing Systems* 33 (2020), pp. 22384–22395 (cit. on pp. 2, 18).

[Fei, Yang, and Wang 2021] Yingjie Fei, Zhuoran Yang, and Zhaoran Wang. "Risk-sensitive reinforcement learning with function approximation: a debiasing approach". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 3198–3207 (cit. on pp. 2, 18).

[Freire and Delgado 2017] Valdinei Freire and Karina Valdivia Delgado. "Gubs: a utility-based semantic for goal-directed Markov Decision Processes". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 741–749 (cit. on p. 35).

[Freitas *et al.* 2020] Elthon Manhas de Freitas, Valdinei Freire, and Karina Valdivia Delgado. "Risk sensitive stochastic shortest path and logsumexp: from theory to practice". In: *Intelligent Systems: 9th Brazilian Conference, BRACIS 2020, Rio Grande, Brazil, October 20–23, 2020, Proceedings, Part II 9*. Springer. 2020, pp. 123–139 (cit. on pp. 2, 20, 30, 31).

[Garcia and Fernández 2012] Javier Garcia and Fernando Fernández. "Safe exploration of state and action spaces in reinforcement learning". *Journal of Artificial Intelligence Research* 45 (2012), pp. 515–564 (cit. on pp. 1, 18).

[Garcia and Fernández 2015] Javier Garcia and Fernando Fernández. “A comprehensive survey on safe reinforcement learning”. *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480 (cit. on pp. 1, 17–19).

[Gehring and Precup 2013] Clement Gehring and Doina Precup. “Smart exploration in reinforcement learning using absolute temporal difference errors”. In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems.* 2013, pp. 1037–1044 (cit. on pp. 1, 18).

[HasanzadeZonuzy *et al.* 2021] Aria HasanzadeZonuzy, Archana Bura, Dileep Kalathil, and Srinivas Shakkottai. “Learning with safety constraints: sample complexity of reinforcement learning for constrained mdps”. In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 35. 9. 2021, pp. 7667–7674 (cit. on pp. 1, 18).

[Heger 1994] Matthias Heger. “Consideration of risk in reinforcement learning”. In: *Machine Learning Proceedings 1994.* Elsevier, 1994, pp. 105–111 (cit. on pp. 1, 18).

[Ronald A Howard 1960] Ronald A Howard. “Dynamic programming and markov processes.” (1960) (cit. on pp. 7, 21).

[Ronald A. Howard and Matheson 1972] Ronald A. Howard and James E. Matheson. “Risk-sensitive Markov Decision Processes”. *Management Science* 18.7 (1972), pp. 356–369. issn: 00251909, 15265501. url: http://www.jstor.org/stable/2629352 (cit. on pp. 1, 2, 18, 19, 21, 22).

[Jin *et al.* 2020] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. “Provably efficient reinforcement learning with linear function approximation”. In: *Conference on Learning Theory.* PMLR. 2020, pp. 2137–2143 (cit. on pp. 2, 18).

[Kadota *et al.* 2006] Yoshinobu Kadota, Masami Kurano, and Masami Yasuda. “Discounted markov decision processes with utility constraints”. *Computers & Mathematics with Applications* 51.2 (2006), pp. 279–284 (cit. on pp. 1, 18).

[R L Keeney and Raiffa 1976] R L Keeney and H Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* New York: Wiley, 1976 (cit. on p. 20).

[Ralph L Keeney and Howard Raiffa 1993] Ralph L Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value trade-offs.* Cambridge university press, 1993 (cit. on p. 1).

[Kennings and Markov 2000] Andrew A Kennings and Igor L Markov. “Analytical minimization of half-perimeter wirelength”. In: *Proceedings of the 2000 Asia and South Pacific Design Automation Conference.* ACM. 2000, pp. 179–184 (cit. on p. 30).

[Kidambi *et al.* 2020] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. “Morel: model-based offline reinforcement learning”. *Advances in neural information processing systems* 33 (2020), pp. 21810–21823 (cit. on pp. 1, 17).

[LeCun, Bengio, *et al.* 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. *nature* 521.7553 (2015), pp. 436–444 (cit. on p. 13).

[LeCun, Bottou, *et al.* 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 13).

[Littman and Szepesvári 1996] Michael L Littman and Csaba Szepesvári. “A generalized reinforcement-learning model: convergence and applications”. In: *ICML*. Vol. 96. 1996, pp. 310–318 (cit. on pp. 1, 18).

[Mann 2006] Tobias P. Mann. “Numerically stable hidden markov model implementation”. In: *An HMM scaling tutorial.* 2006, pp. 1–8 (cit. on p. 30).

[Mausam and Kolobov 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012 (cit. on p. 6).

[Mausser and Rosen 1999] Helmut Mausser and Dan Rosen. “Beyond var: from measuring risk to managing risk”. In: *Proceedings of the IEEE/IAFE 1999 Conference on Computational Intelligence for Financial Engineering (CIFEr)(IEEE Cat. No. 99TH8408).* IEEE. 1999, pp. 163–178 (cit. on pp. 1, 18).

[Mihatsch and Neuneier 2002] Oliver Mihatsch and Ralph Neuneier. “Risk-sensitive reinforcement learning”. *Machine learning* 49.2-3 (2002), pp. 267–290 (cit. on pp. 1, 17, 18, 26).

[Mnih *et al.* 2013] Volodymyr Mnih *et al.* “Playing atari with deep reinforcement learning”. *arXiv preprint arXiv:1312.5602* (2013) (cit. on pp. 12, 13).

[Moldovan and Abbeel 2012] Teodor Mihai Moldovan and Pieter Abbeel. “Safe exploration in markov decision processes”. *arXiv preprint arXiv:1205.4810* (2012) (cit. on pp. 1, 18).

[Morimura *et al.* 2010] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. “Nonparametric return distribution approximation for reinforcement learning”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* 2010, pp. 799–806 (cit. on pp. 1, 18).

[Naylor *et al.* 2001] William C Naylor, Ross Donelly, and Lu Sha. *Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer.* US Patent 6,301,693. 2001 (cit. on pp. 2, 30).

[Neto *et al.* 2020] Eduardo Lopes Pereira Neto, Valdinei Freire, and Karina Valdivia Delgado. "Risk sensitive markov decision process for portfolio management". In: *Mexican International Conference on Artificial Intelligence*. Springer. 2020, pp. 370–382 (cit. on p. 61).

[Nielsen and Sun 2016] Frank Nielsen and Ke Sun. "Guaranteed bounds on information-theoretic measures of univariate mixtures using piecewise log-sum-exp inequalities". *Entropy* 18.12 (2016), p. 442 (cit. on p. 30).

[Nilim and El Ghaoui 2005] Arnab Nilim and Laurent El Ghaoui. "Robust control of markov decision processes with uncertain transition matrices". *Operations Research* 53.5 (2005), pp. 780–798 (cit. on pp. 1, 18).

[Puterman 1994] Martin L. Puterman. *Markov Decision Processes*. Wiley Series in Probability and Mathematical Statistics. New York: John Wiley and Sons, 1994 (cit. on p. 6).

[Rabiner 1990] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". *Readings in speech recognition* (1990) (cit. on p. 30).

[Robert 2014] Christian Robert. *Machine Learning, a Probabilistic Perspective*. Taylor & Francis, 2014 (cit. on p. 30).

[Rummery and Niranjan 1994] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994 (cit. on p. 10).

[Shen *et al.* 2014] Yun Shen, Michael J Tobia, Tobias Sommer, and Klaus Obermayer. "Risk-sensitive reinforcement learning". *Neural computation* 26.7 (2014), pp. 1298–1328 (cit. on pp. 2, 18, 25, 26, 29, 51).

[Sigl *et al.* 1991] Georg Sigl, Konrad Doll, and Frank M Johannes. "Analytical placement: a linear or a quadratic objective function". In: *28th ACM/IEEE Design Automation Conference*. 1991, pp. 427–432 (cit. on p. 30).

[Silver, Huang, *et al.* 2016] David Silver, Aja Huang, *et al.* "Mastering the game of go with deep neural networks and tree search". *nature* 529.7587 (2016), pp. 484–489 (cit. on p. 13).

[Silver, Schrittwieser, *et al.* 2017] David Silver, Julian Schrittwieser, *et al.* "Mastering the game of go without human knowledge". *nature* 550.7676 (2017), pp. 354–359 (cit. on pp. 12, 13).

[Smith *et al.* 2017] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. "Don't decay the learning rate, increase the batch size". *arXiv preprint arXiv:1711.00489* (2017) (cit. on p. 61).

[Stanko and Macek 2019] Silvestr Stanko and Karel Macek. “Risk-averse distributional reinforcement learning: a cvar optimization approach.” In: *IJCCI*. 2019, pp. 412–423 (cit. on pp. 2, 18).

[Sutton and Barto 1998] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998 (cit. on pp. 1, 5, 9, 10, 12, 15).

[Swazinna *et al.* 2021] Phillip Swazinna, Steffen Udluft, and Thomas Runkler. “Overcoming model bias for robust offline deep reinforcement learning”. *Engineering Applications of Artificial Intelligence* 104 (2021), p. 104366 (cit. on pp. 1, 17).

[Tang *et al.* 2019] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. “Worst cases policy gradients”. *arXiv preprint arXiv:1911.03618* (2019) (cit. on pp. 2, 18).

[Tesauro 1994] Gerald Tesauro. “Td-gammon, a self-teaching backgammon program, achieves master-level play”. *Neural computation* 6.2 (1994), pp. 215–219 (cit. on p. 12).

[Van Hasselt *et al.* 2016] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016 (cit. on p. 14).

[Watkins 1989] Christopher Watkins. “Learning from delayed rewards” (Jan. 1989) (cit. on p. 11).

[Wood and Khosravanian 2015] David A Wood and Rassoul Khosravanian. “Exponential utility functions aid upstream decision making”. *Journal of Natural Gas Science and Engineering* 27 (2015), pp. 1482–1494 (cit. on pp. 2, 18).

[Xu *et al.* 2023] Wenhao Xu, Xuefeng Gao, and Xuedong He. “Regret bounds for markov decision processes with recursive optimized certainty equivalents”. *arXiv preprint arXiv:2301.12601* (2023) (cit. on pp. 2, 18).