POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Unsupervised Pre-Training for Reinforcement Learning via Recursive History Encoders

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Author: **Pietro Maldini**

Student ID: 940414
Advisor: Prof. Marcello Restelli
Co-advisors: Dott. Mirco Mutti
Academic Year: 2020-21

Abstract

Reinforcement Learning focuses on Sequential Decision-Making problems, where an agent interacts with an environment to learn how to reach a goal. The *Markov Decision Process (MDP)* framework is used to model these problems. A solution to an MDP is a behaviour that optimizes the performance of an agent. The performance is measured using a reward signal which is feedback from the environment to the agent. This is based on the idea of Reinforcement (rewards and punishments) which is used in training animals. This learning process is determined by a learning algorithm that starts from an agent with an initial behaviour and updates it according to experience to maximize its performance. The traditional formulation of an MDP comprises an environment that is fixed, stationary and out of the control of the agent.

In this work, we consider an extension of standard MDPs, Multiple Environments MDPs. The agent finds itself in one within a group of environments, and it has to maximize its performance for all of them.

In this thesis, we consider the problem of unsupervised pre-training in RL using non-Markovian policies. The typical learning process starts from an initial behaviour, often random, and learns interacting with the environment and collecting rewards. This process can be inefficient when the reward is sparse and difficult to collect. The goal of unsupervised pre-training is to learn skills in the environment to define a starting behaviour without collecting rewards from the environment. An initial behaviour defined in this way can make the learning process of a subsequent task faster and more efficient.

This thesis proposes a new neural architecture used to represent non-Markovian behaviours, which depends on the story and not just on the current state of the environment. The policy that our algorithm, History Mepol (HMepol) learns, retains a limited size representation of the story in which it encodes only the information needed for future decisions. Having introduced this architecture, we then provide empirical results to show its performance and then we will compare those results with other state-of-the-art algorithms and highlight the advantages of HMepol.

Keywords: Reinforcement Learning, Policy Gradient, Maximum State Entropy, Unsupervised RL, Non-Markovian Policy

Abstract in lingua italiana

L'*Apprendimento per Rinforzo* è incentrato sui problemi sequenziale di decisione, dove un agente interagisce con un ambiente per raggiungere un obiettivo. Il formalismo dei *Processi decisionali di Markov (MDP)* è usato per modellare questi problemi. Una soluzione di un MDP è un comportamento di un agente che ottimizza le proprie performance. Le performance di un agente sono misurate usando un segnale retroattivo di premio fornito dall' ambiente. Questo apprendimento trova le sua fondamenta nell'idea del *Rinforzo* (premio o punizione) che viene utilizzata nell'adderstramento degli animali. Il processo parte da un agente con un comportamento iniziale che viene aggiornato e migliorato al fine di massimizzare le proprie performance. La formulazione tradizionale prevede la presenza di un solo ambiente, fisso e stazionario, fuori dal controllo diretto dell'agente.

All'interno di questa tesi verrà considerata l'estensione dei MDP multi ambiente. L'agente si trova in uno tra numerosi ambienti, e il suo compito è massimizzare le sue performance per ognuno di essi.

In questa tesi lavoreremo nel sottocampo dell'apprendimento non supervisionato per l'apprendimento per rinforzo usando politiche non Markoviane. Il tipico processo di apprendimento parte da un comportamento iniziale, spesso casuale, che viene migliorato interagendo con l'ambiente e raccogliendo segnali di rinforzo. Questo processo può risultare molto inefficiente quando il segnale è sparso o difficile da trovare. L'obiettivo dell'apprendimento non supervisionato è imparare diverse abilità utili per esplorare un ambiente definendo così un compotamento iniziale senza segnali di rinforzo. Un tale comportamento permette un successivo apprendimento più rapido ed efficiente.

Questa tesi propone una nuova architettura neurale usata per rappresentare comportamenti non Markoviani, le cui scelte dipendono dalla storia e non solo dalle condizioni attuali dell'ambiente. La politica, imparata dal nostro algoritmo HMepol, mantiene una rappresentazione limitata della storia al suo interno in cui codifica solo le informazioni utili per scelte future. Una volta introdotta la nostra architettura, forniremo risultati sperimentali delle performance ottenute confronteremo poi tali risultati con altri algoritmi che compongono lo stato dell'arte.

Parole chiave: Apprendimento per Rinforzo, Policy Gradient, Massimizzazione dell'entropia degli stati, Apprendimento non supervisionato, Politiche non Markoviane

Contents

1 | Introduction

Many problems can be formalized as a sequence of decisions to reach a goal. Reinforcement Learning (RL) aims to solve sequential decision problems usually formalized as Markov Decision Process (MDP). A decision-maker is placed inside an environment and performs actions to interact with it, this process leads to changes and evolution in the current condition of the environment. We usually refer to the decision-maker as an *agent*, we call its behaviour, i.e., its action selection strategy, a *policy*. We call the subset of the current condition of the environment, available to the agent, the *state* of the environment. A Markovian policy chooses the action to perform only using the information of the current state. A non-Markovian policy uses the information of all the states the agent visited before.
The interaction process is usually divided into discrete time-steps. At each time-step, the agent observes the state, performs an action according to its policy, receives a reward and the state of the environment after the action is performed. The general formulation of an MDP assumes that the next state of the environment depends only on the current state and the current action performed, this is referred to as the *Markov Property*. The goal of a traditional RL agent is to maximize the cumulative reward of a group of subsequent interactions, this means that the agent must learn to maximize the reward not only in the immediate future but in the long term as well.

THE IMPORTANCE OF THE REWARD The learning process usually starts with a random policy. This policy is used to interact with the environment and uses the information gathered in the interactions to modify its behaviour. The reward, that guides the learning process, is usually defined by a human designer, this design process is difficult and different tasks may require reward signals that can be arbitrarily different and complex. A mistake in the design phase may also lead to undesired behaviours.
To overcome this complex phase usually a sparse signal is chosen,i.e. the agent receives a reward when it performs a specific action but has no incentive in intermediate steps leading to it. The information gathering is critical, if the agent is not able to collect any reward it can't learn. Even if the agent can collect reward if it is sparse and rarely found

the learning process can become long and the agent may not learn an optimal behaviour. Thus, the process is very sensitive to the shape of the reward and the dimensionality of the environment.

UNSUPERVISED PRE-TRAINING Mimicking unsupervised pre-training of supervised learning models, the idea of unsupervised pre-training in RL has been proposed. The agent is placed inside an environment without a reward function, so there is no signal from the environment which guides the agent toward a specific goal. The agent needs to learn a policy which can explore the environment, reach a wide range of states and perform complex actions. Using such a policy as the initial policy to learn a subsequent task would allow for an easier collection of sparse rewards and more efficient learning.

STATE OF THE ART In the literature, different algorithms are available to perform an unsupervised pre-training of a policy. Some algorithms [24, 28, 34] provide an *intrinsic* bonus to the agent. This signal is used to incentivize the agent to explore unseen or less predictable states. These approaches have been successfully scaled also in high-dimensional and continuous environments[2]. These approaches suffer from a non-systematic exploration of the environment. This intrinsic bonus is consumable and degrades with time. This leads to an unbalanced exploration of the environment.
To overcome this limitation, new algorithms [9, 10, 12–14, 18, 19, 30, 36, 39, 40], that motivates the agent towards an uniform exploration of the environment, have been proposed.
These algorithms were then extended to handle high-dimensional and vision-based environments [22, 30]. Expanding on the ideas provided by those works extensions to multi environments MDP has been proposed. Using a different objective function [22] or defining a new intrinsic bonus [23] it was possible to learn better policies to facilitate subsequent tasks. A common factor of all those algorithms is that they learn one or a mixture of Markovian policies.

MOTIVATION Mutti et al. in [21] showed theoretical proofs of the advantage of Non-Markovian policies in unsupervised pre-training for RL. Learning a policy to explore an environment poses several challenges to a Markovian policy which has no memory of the past. When the agent finds itself twice in front of a choice, which require different actions with different past interactions, it will always randomize. A non-Markovian policy, which retains the memory of the past, instead can perform deterministically and in a different way every time. This problem is aggravated in the multi environments setting where different environments may share states with different optimal actions. In this setting, a

Markovian policy can't discriminate the environments so it must randomize and choose a suboptimal action. A non-Markovian policy can choose an optimal action as it can learn to discriminate the environments using the history of the interactions.

CONTRIBUTION In this thesis, we present a practical algorithm History Mepol (HMepol) to learn a Non-Markovian policy. Our main contribution is in the policy architecture learned by HMepol that recursively encodes a compact representation of the history of previous states and the current states to obtain a new compact representation of the history. This learning process allows to reduce the size of the representation of the history and shows the advantages of Non-Markovian behaviours in unsupervised pre-training in the multiple environments setting. The use of deep recursive encoders allows for retaining the relevant components of the history for the subsequent decisions of the agent.

OVERVIEW The contents of this thesis are organized as follows. In Chapter 2, we describe Markov decision processes, reinforcement learning and information theory concepts useful in the description of the state of the art. In Chapter 3, we describe the state of the art of the problem of unsupervised pre-training in RL. In Chapter 4, we present the limitations of markovian policies in the unsupervised RL problem, and a new non-markovian policy architecture which overcomes these limitations. Chapter 5 presents the experimental evaluation of the algorithms, initially on a small class of environments to later show the generalization to bigger classes of environments. We conclude the thesis in Chapter 6, where we propose some possible extensions for future work. In Appendix A, we provide additional experimental results and the hyperparameter used in the experimental settings of Chapter 5.

2 | Essential Background

This chapter introduces the essential background for the remainder of this thesis. In section 2.1 we present the framework of Markov Decision Processes used to formalize problems addressed in RL. In section 2.2 we present reinforcement learning, present its goal and describe some of the dichotomies which characterize RL algorithms. We then focus on Policy Gradient algorithms among which HMepol gets categorized. Lastly, in section 2.3, we describe other mathematical notions required in the description of State of the Art and that will be used in the following chapters.

2.1. Markov Decision Processes

A *Markov Decision Process (MDP)* [25] is the framework used for modelling decision-making problems. A decision-maker, usually called *agent*, is placed inside an environment. The agent can interact with the environment by performing actions and modifying its current condition. A representation of the current conditions of the environment, which is usually referred to as *state*, is available to the agent. The interaction process happens in a sequence of discrete time-steps, $t = 0, 1, 2, 3, ..., T$. The agent at each time-step t uses the current state s_t to choose an action a_t to perform. After the action is performed, the agent receives the representation of the next state s_{t+1} and a numerical reward signal r_{t+1} from the environment as seen in Figure 2.1.

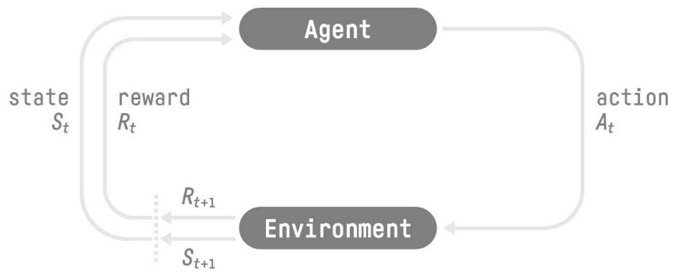


Figure 2.1: Interaction between the agent and the environment in an MDP (from [35]).

The agent interacting with the MDP thus generates a sequence of States, Actions and

Rewards usually called a *trajectory*. A discrete-time Markov Decision Process can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, d_0, T \rangle$, where:

- $\mathcal{S} \in \mathbb{R}^{n_s}$ is an n_s-dimensional continuous space, usually called *state space*. The state space represents what the agent can perceive in the environment i.e. measurements from sensors in a robot;
- $\mathcal{A} \in \mathbb{R}^{n_a}$ is an n_a-dimensional continuous space, usually called *action space*. The acion space represents the set of actions available to the agent i.e. the range of possible torque which a motor can output;
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, is a *Markovian transition model*, where $\Delta(\mathcal{S})$ represents the simplex over the state space. $\mathcal{P}(s'|s, a)$ is the conditional probability of reaching the next state s' , given the current state s and the selected action a. This function defines the dynamics of the environment;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is the *reward function* where $\mathcal{R}(s, a)$ is the expected scalar reward collected after taking action a in state s. The reward function can also be defined as $\mathcal{R} : \mathcal{S} \to \mathbb{R}$, in this case $\mathcal{R}(s)$ is the scalar reward given when reaching state s . This signal has the role of guiding the learning process and encodes the task that the agent must learn;
- $d_0 \in \Delta(\mathcal{S})$ is the *initial state distribution* and $d_0(s)$ defines the probability that the starting state s_0 is s;
- T is the *Time Horizon* which is the maximum length of a trajectory.

We defined continuous action and state spaces, but they can be alternatively formulated as discrete sets. The transition model $P(s'|s, a)$ satisfies the Markov property: The transition depends only on the current state and the current action and there is no dependence on previously visited states or previously taken actions. In the remainder of the thesis, we will consider MDPs which are stationary, both the transition model and reward function don't change over time.

2.1.1. Multiple Environments MDP

An extension of traditional MDPs which will be analysed and treated in this thesis. Multiple environments MDPs are characterised by the presence of several environments. We define $\mathcal{M} = \mathcal{M}_1, ..., \mathcal{M}_I$ the set of environments with which the agent can interact each defined as $\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, \mathcal{P}_i, \mathcal{R}_i, d_0)$. We consider sets of environments with state and action spaces which are homogeneous. At the beginning of each trajectory, an environment is chosen following a distribution $p_{\mathcal{M}}$ defined over $\mathcal{M}$. We can consider the generic MDP formulation as a specific case of Multiple Environements MDP with a single environment $\mathcal{M}_1$ with $p_{\mathcal{M}}(\mathcal{M}_1) = 1$.

2.1.2. Policy

The behaviour of an agent inside an MDP is called a *policy* and is usually stochastic. A non-Markovian policy is a function:

$$\pi : \mathcal{H} \to \Delta(\mathcal{A}),$$

that maps every history $h \in \mathcal{H}$ to a probability distribution $\pi(\cdot|h)$, over the action-space $\mathcal{A}$. We define the history at a time-step t in a trajectory as $h_t = \langle s_t, s_{t-1}, ..., s_0\rangle$. When the Markov property holds, the policy can be simplified as a function of the last state rather than the full histroy:

$$\pi : \mathcal{S} \to \Delta(\mathcal{A}).$$

The resulting Markovian policy maps a state s to a probability distribution $\pi(\cdot|s)$. We call the policy *deterministic* if it selects an action with probability 1, we call it *stochastic* otherwise.

Starting from the formalization of the MDP and the policy's definition, we can describe the general interaction of the agent with the MDP. The interaction starts at time $t = 0$ in a state s_0 which is determined by the distribution d_0. At each time step the agent samples an action a_t from $\pi(\cdot|s_t)$. The environment evolves according to its transition model presenting a new state s_{t+1} to the agent, and the reward function gives a signal r_{t+1} to the agent. This interaction continues until the horizon T is reached.

A trajectory $\tau \in \mathcal{T}$ is defined as a sequence $\langle s_t, a_t, r_t\rangle_{t=0,\dots,T-1}$ of states, actions and reward collected by the agent. Finally, we define the probability that the policy π induces a trajectory τ in the environment as:

$$\rho(\tau|\pi) = d_0(s_0) \prod_{i=0}^{T-1} \mathcal{P}(s_{i+1}|s_i, a_i)\pi(a_i, s_i). \tag{2.1}$$

.

2.1.3. State Distribution

The interaction between an agent following a policy π and an MDP induces a distribution over the states at each time step t which can be defined as:

$$d_t^\pi(s) = \mathbb{P}(s_t = s|\pi).$$

Where $\mathbb{P}(s_t = s|\pi)$ indicates the probability of being in state s at time-step t following the policy π. The notion of state distribution can be rewritten considering the set of possible trajectories $\mathcal{T}$:

$$d_t^\pi(s) = \int_{\mathcal{T}} \mathbb{P}(\tau|\pi, s_t = s)\, d\tau. \tag{2.2}$$

Where $\mathbb{P}(\tau|\pi, s_t = s)$ is the probability of being in a certain trajectory following the policy π and being in state s at time t in that trajectory.

2.2. Reinforcement Learning

Reinforcement Learning [35] is a sub-field of Machine Learning. The term is used to refer at the same time to the field, the problems, and the algorithms used. Reinforcement learning focuses on sequential decision-making problems, where an agent interacts with an environment to learn how to reach a goal. The MDP is a framework used to model sequential decision-making problems. The goal is encoded in the reward signal that guides the learning process of the agent. The goal of the agent is to maximize the cumulative reward received from the environment by modifying its strategy according to the information gathered from several trajectories.

2.2.1. Performance of a Policy

The goal of RL algorithms is to train an agent to learn an optimal policy for an MDP. This concept is encoded in the cumulative reward function, also called *return* [35], can be defined as:

$$G(\tau) = \sum_{k=0}^{T-1} R_{k+1}(\tau).$$

It is the sum of rewards collected in trajectory τ from time t_i up to the horizon T. For some tasks, the current reward may have a different value compared to a future reward so giving equal weight to all rewards can lead to misleading results. These considerations lead to creating a different objective, a discounted cumulative reward function that weights the reward at each time t with a value γ^t. The discounted cumulative reward of a trajectory, is usually defined as:

$$G(\tau) = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}(\tau).$$

We can define the performance of a policy π as:

$$J^{\pi} = \mathop{\mathbb{E}}_{\tau \sim \pi} G(\tau). \tag{2.3}$$

This measure represents the quality of π. We can compare two policies using this measure, a higher performance means a better policy and if the reward function is defined correctly it means also an agent that performs the required task better. The concept of expected reward has some limitations, some tasks have no time horizon and this can lead to infinite expected reward.

Using this measure we can define the optimal policy π^* as the policy:

$$\pi^* \in \arg\max_{\pi \in \Pi} J^{\pi},$$

which maximizes the performance over the space Π of policies. In the remainder of this thesis, we will mainly focus on parametric policies, so we will treat Π as the set of parametric policies.

2.2.2. Classes of MDP solutions

Reinforcement Learning solutions are characterized by several dichotomies that arise from different algorithms' aspects. The main aspects and distinctions between solutions described in [35] are:

- Model of the environment:
  - *Model-based*: The agent has a model of the environment or tries to estimate it to then compute the optimal policy under the estimated model;
  - *Model-free*: The agent learns without an explicit model of the environment.
- Length of trajectories:
  - *Infinite-horizon*: The experience is presented as a single and infinite trajectory to the agent;
  - *Episodic*: The experience is divided in finite-length portions that are called episodes.

- When the learning happens:
  - *On-line*: The agent learns while the experience is collected ;
  - *Off-line*: The agent learns through experience that has been previously collected (possibly by other agents).
- The policy used to collect the samples:
  - *On-policy*: The agent modifies the policy used to interact with the environment during the learning process ;
  - *Off-policy*: The agent uses experience collect using another policy to learn an optimal policy.
- How the policy is represented:
  - *Value-based*: The agent doesn't learn directly a policy but learns a value function;
  - *Policy-based*: The agent builds an explicit representation of the policy.

In the remainder of this thesis, we will focus on Policy-based algorithms and in particular in *Policy Search* which is the family of algorithms within which our solution lies.

2.2.3. Policy Search

Policy search is a family of RL algorithms that prescribe to look for the optimal policy through direct navigation of the policy space. In this thesis, we will build upon policy gradient methods [38], where the navigation is done via gradient ascent. We restrict the policy space to the stochastic parametric policies, which can be defined as:

$$\Pi_\Theta = \left\{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^k\right\}, \tag{2.4}$$

where $\boldsymbol{\theta}$ is a vector of parameter that defines the policy. The learning process is guided by the gradient of the policy performance w.r.t. the parameter vector, thus the policy must be stochastic and differentiable in $\boldsymbol{\theta}$. The process starts from a parameter vector $\boldsymbol{\theta_0}$, which might be randomly initialized. The, at each step, it is updated following the gradient of the performance metric:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t),$$

where $\alpha_t \geq 0$ is the learning rate. This process is guaranteed to converge to a stationary point of the performance function under common regularity assumptions. Starting from the definition of performance given in Equation (2.3), we can obtain the gradient of the policy as [4]:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int_T \rho(\tau, \pi) G(\tau)\, d\tau \\
&= \int_T \nabla_{\boldsymbol{\theta}}(\rho(\tau, \pi)) G(\tau)\, d\tau \\
&= \int_T \rho(\tau, \pi) \frac{\nabla_{\boldsymbol{\theta}}(\rho(\tau, \pi))}{\rho(\tau, \pi)} G(\tau)\, d\tau \\
&= \mathop{\mathbb{E}}_{\tau \in T} \frac{\nabla_{\boldsymbol{\theta}}(\rho(\tau, \pi))}{\rho(\tau, \pi)} G(\tau) \\
&= \mathop{\mathbb{E}}_{\tau \in T} \nabla_{\boldsymbol{\theta}}(\log(\rho(\tau, \pi))) G(\tau).
\end{aligned}$$

This last formula with the definition of ρ we gave in Equation (2.1) allows us to find a Monte-Carlo approximation of the gradient.

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathop{\mathbb{E}}_{\tau \in T} \nabla_{\boldsymbol{\theta}}(\log(\rho(\tau, \pi))) G(\tau) \\
&\approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}}(\log \pi(a_t | s_t)) G(\tau_i).
\end{aligned} \tag{2.5}$$

In this approximation we are considering N trajectories $\tau_0, \tau_1, ..., \tau_{N-1}$. For each trajectory, we are summing the gradient of the log-likelihood of the action performed under the policy π in the current state s_t and we multiply it with the reward of the corresponding trajectory, we then take an average of this measure w.r.t. the set of collected trajectories. With this gradient, we are changing the parameters of the policy to increase the likelihood of actions performed in trajectories with higher performance and reduce the likelihood of actions performed in less promising trajectories.

2.3. Information Entropy

In this section, we introduce the measure of information entropy, starting from the concept of the information content of an outcome of a random variable, we derive the discrete entropy. Then, we describe generalizations and estimators that will be used in the following chapters.

2.3.1. Information Content

Given a random variable or a random vector X, with possible outcomes $x \in \{x_1, ..., x_n\}$ we can define the *information content* associated with X assuming a value x_i as:

$$\begin{aligned} I(x_i) &= \log_b \mathbb{P}\left(\frac{1}{x_i}\right) \\ &= -\log_b \mathbb{P}(x_i). \end{aligned}$$

Where $P(x_i)$ is the probability of X assuming the value x_i. The Information content gives a value that indicates how unlikely an event is, or how we would be surprised to see a given realization of the random vector X. For the rest of this thesis we will consider the base b to be e, the Euler's number.

2.3.2. Discrete Entropy

To measure the level of information contained in a random variable/vector we define the discrete entropy [31]. The discrete entropy is the expected information content of a random vector X:

$$\begin{aligned} H(X) &= \mathop{\mathbb{E}}_{x_i \in X} I(x_i) \\ &= -\sum_{i=1}^{N} \mathbb{P}(x_i) \log \mathbb{P}(x_i), \end{aligned}$$

An higher entropy value is associated with a less predictable variable, which thus holds more information. The maximum value for discrete entropy is obtained when the distribution of outcomes is uniform so that no outcome is more likely than others. Considering the fact that $0 \leq P(x_i) \leq 1$ we get that $\log P(x_i) \leq 0$, every term in the summation is negative or zero, so the overall discrete entropy is always positive. The Discrete Entropy is measured in natural unit of information (nats), one nat is the information of an event with probability $1/e$.

2.3.3. Differential Entropy

We can extend the concept of discrete entropy to random vectors with continuous support. Let X be a p-dimensional random vector with values in $\mathbb{R}^p$, we can define the differential entropy of X:

$$h(X) = -\mathbb{E}\left[\log(f(x))\right] = -\int_X f(x) \log(f(x))\, dx,$$

where $f(x)$ is the probability density function of X. This quantity can be shown to be similar to its discrete counterpart if we consider a discretization into bins of width δ with a difference of a quantity $\log(\delta)$. This difference diverges in the limit of $\delta \to 0$, this can be explained by the fact that to specify a continuous distribution more information is required. Differently from Discrete Entropy, the Differential Entropy can assume negative values.

2.3.4. Non-Parametric Differential Entropy Estimator

Up to now, we considered the case where the density f of the random vector X was known. In many problems this is not true, e.g. in the model-free setting. In those problems, we can use sample-based estimates for the entropy. In this thesis, we will use a non parametric, k-Nearest Neighbors (knn) entropy estimator proposed by Singh et al. in [33] of the form:

$$\hat{H}_k(f) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{k}{NV_i^k} + \log k - \Psi(k), \tag{2.6}$$

where Ψ is the digamma function and V_i^k is the volume of the hyper-sphere of radius $R_i = |x_i - x_i^{knn}|$ that represents the euclidean distance between x_i and its knn x_i^{knn}. In this formula, we can distinguish two parts: $\log k - \Psi(k)$ is a bias correction term for the estimate, and $\log \frac{k}{NV_i^k}$ accounts for the density of samples close to the i-th sample. The use of the k^{th} nearest neighbour mitigates the effects of pure chance in the estimation of the density. The second term is then averaged for all samples to provide an estimation of how they are distributed. The estimator in Equation (2.6) is known to be asymptotically unbiased and consistent [33].

In an off-policy scenario, the samples can be obtained from a distribution f which may differ from the target distribution f'. We can use an Importance Weighted (IW) knn estimator [15] to compute the entropy under f' starting from samples collected using f as follows:

$$\hat{H}_k(f'|f) = -\sum_{i=1}^{N} \frac{W_i}{k} log \frac{W_i}{V_i^k} + logk - \Psi(k). \tag{2.7}$$

In this formula $W_i = \sum_{j \in N_i^j} w_j$ is the sum of the normalized importance weights w_j computed over the set N_i^k of knn samples of x_i. The normalized importance weights can be defined as:

$$w_j = \frac{f'(x_j)/f(x_j)}{\sum_{n=1}^{N} f'(x_n)/f(x_n)}.$$

Starting from the normalized importance weights it is also possible to compute an estimate of the Kullback-Leibler (KL) divergence [1]:

$$\hat{D}_{KL}(f'||f) = \frac{1}{N}\sum_{i=1}^{N} log\frac{k/N}{\sum_{N_i^k} w_i}. \tag{2.8}$$

This measure is closely related to the entropy, indeed it is also called relative entropy, and is a divergence measure between two distributions.

3 | Unsupervised Reinforcement Learning: State of the Art

In this chapter, we provide an overview of the state of the art in unsupervised reinforcement learning. In particular, in Section 3.1 we introduce the problem of unsupervised pre-training of exploration policies, and we explain why it has a central role in RL. In Section 3.2 we introduce more recent methods that use mainly intrinsic rewards. Then, in Section 3.3 we introduce methods that try to motivate the agent towards a systematic exploration of the environment. We then provide, in Section 3.4, an introduction to algorithms proposed to tackle the unsupervised pre-training problem in multiple environments. Lastly, in Section 3.5 we report early results on the advantages of non-Markovian policies for unsupervised reinforcement learning.

3.1. Unsupervised Pre-Training for Reinforcement Learning

The typical setting of reinforcement learning assumes the existence of a reward signal which is used to guide the learning process of an agent. With a rich enough reward signal, RL has been shown to have outstanding performance in a variety of complex tasks [16, 32]. Designing a rich reward signal is very hard in practice, and the shape of this reward function varies depending on the environment and the task to be solved. Usually, this design phase can be complex, often leading to sparse reward signals, i.e., a reward that is given only when a specific action is performed and no reward is given in all the other cases. This kind of reward is hard to collect, especially if the environment is huge. When the agent is unable to collect the reward in this kind of scenario the learning process cannot even start, since, without a signal which tells the agent what to do, the agent has no direction of improvement. Following the success of unsupervised pre-training in supervised learning tasks, a new paradigm of unsupervised pre-training for RL has been proposed. In supervised learning tasks different pre-training techniques exist, what they have in common is that the supervised target is hidden and a different task is given to the algorithm. This task is used to insert knowledge about the domain of the problem and understand the statistics of the input data in the model. After this phase of initial pre-training, the supervised target is restored and the model uses the information gained in the unsupervised phase to learn the task more efficiently, this process helps also the agent to find hidden characteristics of the dataset that a normal supervised approach would neglect as it deemed them as not important or are rare characteristics of outliers which a supervised approach may have ignored, leading to an inefficient learning process for those samples. In RL an initial random policy may be inefficient in learning a specific task especially when the reward signal associated with it is sparse. A parallel approach to unsupervised pre-training in RL can be taken, the reward signal is hidden and the agent is presented with a different task, learning to understand the environment, learning useful skills in the environment and exploring it to reach a wide range of states. This unsupervised phase allows learning a policy which can be more efficient in learning a task even in presence of a sparse reward. The learning process of a subsequent task is helped not only by the reward signal but also by the knowledge and skills learned in the pre-training phase.

3.2. Traditional Intrinsic Rewards

In this section, we will provide an overview of methods that defines intrinsic rewards to learn an exploration policy in reward-free and sparse reward settings. An intrinsic reward is an artificial reward which is used to incentivize exploration beyond the fixed task reward.

3.2.1. Prediction-Error Methods

In prediction-error methods [24, 28], the agent tries to predict the next state with its model of the environment, and it receives a reward that is proportional to the prediction error. The agent should explore more where it is more uncertain, so where the error is higher. This kind of approach defines an intrinsic reward function of the form:

$$R_{intrinsic}(s_t, s_{t+1}) = ||f(s_{t+1}) - M(f(s_t), a_t)||_2,$$

where we want to improve improve M which is the model of the environment, which is typically encoded as a neural network, and $f(\cdot)$ is a representation of the feature space, which consists only of the information that is related to the agent's choice of action. This filtered feature space representation allows to remove local stochasticity which would stall agent exploration.

3.2.2. Count-Based Methods

The count-based methods were originally proposed in discrete settings with a finite number of states [34]. In these settings, the agent can keep the counts of each state visitation. Then, an intrinsic reward is computed in a way that provides higher bonuses to less visited states. The bonus can be defined as [34] :

$$R_{intrinsic}(s_t) = \frac{1}{\sqrt{N(s_t)}} \tag{3.1}$$

where $N(s_t)$ represents the number of times s_t has been visited. Directly applying this idea in a continuous environment would lead to several issues. However, one can emply a density model in place of the counts $N(s_t)$, which allows to compute pseudocounts for every state [2].

3.2.3. Limitations of Intrinsic rewards

Intrinsic rewards are limited and consumable, the longer the agent explores the environment the lower those rewards become. This makes them more suitable for exploration in sparse reward settings, rather than to perform unsupervised pre-training of an exploration policy. In [5] Ecoffet et al. showed how these consumable intrinsic rewards cause the problem of *detachment*. As the agent consumes the reward it gets detached from rewarding states, but without intrinsic rewards to reach those states the agent is no longer incentivized to move towards unexplored regions of the environment.

3.3. Maximum State Entropy Methods

Maximum state entropy methods find an objective that can lead to learning an exploratory policy that can be later fine-tuned to solve many subsequent tasks. The ideal objective is a uniform visitation of the state space, which is equivalent to maximizing the entropy of the state distribution induced by the policy. Hazan et al. in [10] proposed this objective :

$$\pi^* \in \arg\max_{\pi \in \Pi} H(d_\pi),$$

where Π is the policy space and d_π is the state distribution induced by π and $H(d_\pi)$ is the entropy of the state distribution. Several methods [9, 10, 12–14, 18, 19, 30, 36, 39, 40] have been proposed to optimize this objective. Hazan et al. in [10], proposed an algorithm that uses the Frank-Wolfe algorithm to approximate gradients used to learn an optimal mixture of policies. In [12] the agent is given a target distribution d^* that represents the target distribution to be induced. The exploration objective is rewritten as:

$$\min_{\pi\in\Pi} D_{KL}(d_\pi || d^*) = \max_{\pi\in\Pi} \mathbf{E}_{d^\pi(s)} \log d^* + H(d_\pi).$$

Where D_{KL} is the KL divergence between the distribution induced by the policy π and the target distribution. This objective is used to optimize a mixture of policies using a density model to estimate the entropy of the state distribution. Other algorithms have been presented, such as [40] in which the agent learns a single policy that is obtained by optimizing the Rényi entropy of the discounted state distribution, [9] that introduces a similarity function on the state space and uses it to define a geometry-aware entropy objective. In [19] a new approach which uses a KNN estimator for the entropy of the state distribution induced by a policy has been proposed, expanding from this idea [30] showed an extension for environments with visual inputs using the estimate on the reduced dimensionality obtained by randomly initialized convolutional encoders.

3.3.1. Mepol

Mutti et al. in [19] proposed an algorithm that outputs a single policy and which uses a non-parametric estimate of the entropy, Maximum Entropy POLicy optimization (Mepol). This algorithm uses the importance weighted entropy estimate previously described (2.7) to compute the entropy of the state distribution induced by π over the state space. Mepol combines this estimator with an importance sampling offline optimization [15], which allows the efficient reuse of samples collected with previous policies, and an update via gradient ascent within a trust-region boundary [29] which limits the distance of the agent's policy to the one used to collect the samples. This distance is computed as the KL divergence of the distributions induced by the two policies using the estimate (2.8), which can be obtained as a byproduct of the IW knn entropy estimate.

Algorithm 3.1 Mepol

```
Input: Horizon $T$, sample-size N,trust-region threshold $\delta$, learning rate $\alpha$, nearest neighbors k
initialize $\theta$
for $epoch = 0, 1, 2, ...$ until convergence do
    draw a batch of $\left\lceil \frac{N}{T} \right\rceil$ trajectories of length $T$
    build a dataset of particles $D_\tau = \{(\tau_i^t, s_i)\}$
    $\theta' \leftarrow \theta, g \leftarrow 0$
    while $D_{KL}(\hat{d}_T(\theta)||\hat{d}_T(\theta')) \leq \delta$ do
        $\theta' = \theta' + \alpha \nabla_{\theta'} \hat{H}_k(\hat{d}_T(\theta')|\hat{d}_T(\theta))$
    end while
    $\theta \leftarrow \theta'$
end for
return task-agnostic policy $\pi_\theta$
```

This algorithm collects batches of trajectories and uses these batches to compute an entropy estimate of the state distribution, then uses this estimate to update its policy with off-policy updates, the samples collected are then reused until the policy reaches a value higher than a given threshold δ of the estimated KL divergence. This process is then repeated until convergence.

3.4. Unsupervised Pre-Training in Multiple Environments

The algorithms proposed up to now are environment-specific. To overcome this limitation some recent works extended previously presented ideas to new algorithms that can learn exploratory policies that generalize over multiple environments. In this section, we will give an overview of two algorithms tackling the multiple-environments setting [22, 23].

3.4.1. Change-Based Exploration Transfer

In [23] Parisi et al. proposed an algorithm called Change-Based Exploration Transfer (C-BET), which finds a task-agnostic policy that seeks to produce interesting changes in the environment. The intrinsic reward signal is defined as:

$$r_i(s, a, s') = 1/(N(s') + N(c(s, s'))), \tag{3.2}$$

where N represents the (pseudo)counts of the state visitations, and changes and $c(s, s^{'})$ represents the changes in the environment occured during the transition from s to $s^{'}$. This can be seen as an extension of (3.1). To overcome the limitations of this intrinsic reward the counts are randomly reset with a given probability instead of resetting at the start of each trajectory as proposed in [26]. This choice allows to have better exploratory capacities among different trajectories, as preserving counts allows remembering where the agent already went and rewards unvisited states.

3.4.2. αMepol

One straightforward approach to extending a single environment algorithm to find an exploratory policy in multiple environments is averaging the entropy obtained for each environment by performing a training phase in each one of those environments. Given a set $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_I\}$ of environments, with different transition model P_i but homogeneous action and state spaces, and a probability function $p_{\mathcal{M}}$ over it, we can consider the problem to be multi-objective, as one can choose any possible preference among the environments in $\mathcal{M}$ by giving different weights in the computation of the average entropy. The average of the entropy among trajectories collected in the different environments does not account for the tail of the entropies of trajectories obtained from unfavourable or rare environments. To overcome this limitation αMepol [22] has been proposed as an extension of Mepol. αMepol introduces a new objective inspired by the risk-averse literature [27], which uses the Conditional Value-at-Risk (CVaR) of the entropies of the collected trajectories:

$$CVaR_{\alpha}(H_{\tau}) = \mathop{\mathbb{E}}_{\substack{\mathcal{M} \sim p_{\mathcal{M}} \\ \tau \sim p_{\pi, \mathcal{M}}}} \left[H_{\tau} | H_{\tau} \leq VaR_{\alpha}(H_{\tau}) \right], \tag{3.3}$$

where H_{τ} represents the entropy of the state distribution which is estimated using(2.7) with the states visited in trajectory τ. This objective allows for improving the entropy obtained in the worst case, as the objective can account for rare environments and unfavourable transition functions.

3.5. Non-Markovian Policies for Unsupervised Reinforcement Learning

Mutti et al. in [21] showed that a Markovian policy has limitations in the unsupervised pre-training problem, as, a Markovian stochastic policy can learn to explore an environment in the average of infinite trials but can perform poorly on any single trial. A

Markovian policy can learn to explore an environment or a class of environments within the limit of infinite trials. In some environments, a Markovian policy reaches multiple times a given state with multiple possible actions which can lead to exploring different parts of the environment. With only the information of the current state, a Markovian policy can only randomize, leading to exploring all the choices on average. Considering the single trial exploration the Markovian policy would not be optimal, as the randomization prevents it from avoiding to explore again an already visited state. Using the history instead of the current state allows the agent to retain information about the environment and thus disambiguate the action to perform. The agent can avoid randomization, and by remembering the states it visited it can choose a different action to explore the environment better in a single trial.

4 | Recursive History Encoders for Non-Markovian Policies

In this chapter, we describe the limitations of Markovian policies in the unsupervised pre-training problem and present our main contribution, an architecture to represent non-Markovian policies using a limited representation of the history.

4.1. Limiting Factors for a Markovian Policy

Algorithms currently available in the state of the art for the unsupervised pre-training of RL agents learns one or mixture of markovian policies.

Mutti et al. in [20, 21] showed how in the infinite samples setting a Markovian policy is enough for the maximum state entropy objective while in a finite sample regime this is no longer true. In the latter setting, whereas a Markovian policy is usually randomized, a non-Markovian policy can act deterministically to achieve a superior overall exploration. A markovian policy, having only the information of the current state, is required to randomize actions and often make suboptimal choices in different scenarios. The choice of a suboptimal action can be seen as a consequence of the partial observability of the MDP and the history. We can recognize different possible elements that may be partially or not observable by a Markovian policy:

- History: By definition, this information is not available to a markovian policy, this leads to suboptimal choices in states that require taking different actions when visited multiple times. [21] used as an example two rooms connected by a corridor (see Figure 4.1), and an agent which starts in this corridor. Suppose the time horizon is enough to explore both rooms. The optimal Markovian policy is deterministic within a room, but must be randomized when the agent is in the corridor, leading to several trajectories exploring twice the same room. Retaining the history an agent is able to choose deterministically the room not previously visited to get a better exploration in the single trial;

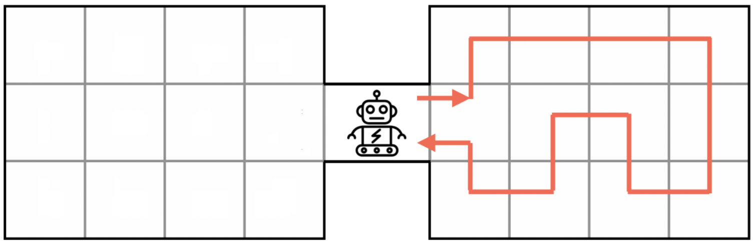


Figure 4.1: With no information of the history(the red trace) a Markovian agent must randomize in which room to explore once returned in the initial position (from [21]).

- Environment: An agent placed in one of multiple environments with similar state spaces is not able to recognize and retain the information of the environment. This can be seen as the epistemic partial observability presented in [6]. Even if the agent reaches some states, available only for a specific environment, once it moves outside these states it can no longer know in which environment it is located. If the state spaces coincide it is even worse, the agent must randomize so that it takes an action which is good for all the environments on average, thus leading to a suboptimal exploration. A non-Markovian policy can use the information from history to understand in which environment it is located. It can even choose to perform "bad" actions at first, which can help to identify the environment and to select the best environment-specific action thereafter.

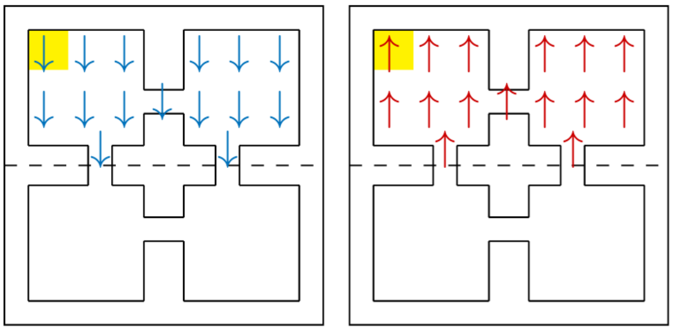


Figure 4.2: A set of environments with different dynamics. A different force(coloured arrow) is added to the agent's action. A different optimal action is required in each state of the two environments. A Markovian policy cannot choose the optimal action but must randomize and get a suboptimal action for both (from [22]).

- State: Partial observability of the environment's state can make Markovian policies insufficient. For example, in the mountain car environment [17] for example knowing only the position of the car but not the speed of the car makes the problem harder. If we observe Figure 4.3 we cannot tell the direction of the car or its speed. Without this information it is impossible to correctly choose in which direction we should accelerate. A non-Markovian policy can understand the direction of movement of the car from the sequence previous states, and it can approximate its speed and choose the action using this additional information.

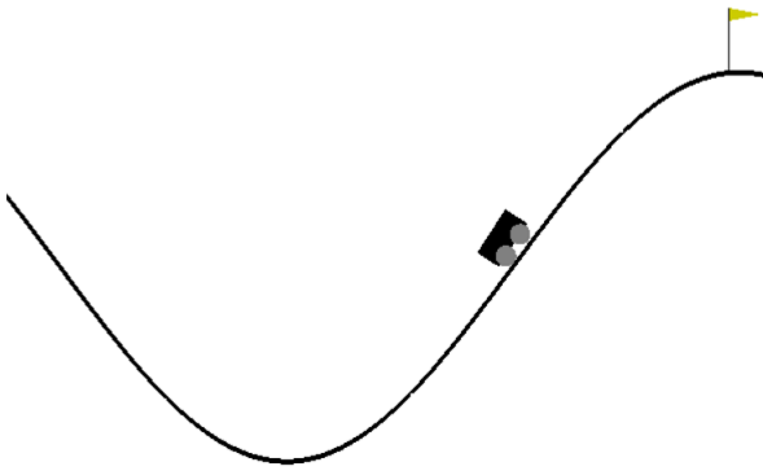


Figure 4.3: With only knowledge about the current position a markovian policy can't understand if the car is going up or down.

4.2. History Mepol

In this section, we present the architecture of the non-markovian policy that HMepol learns. HMepol is a policy gradient algorithm which searches policy in the space of stochastic parametric policies differentiable with respect to their parameter vector as defined in (2.4). HMepol searches a policy which optimizes the non-parametric IW knn entropy estimate (2.6) where we consider the sample based estimate of the distribution $\overline{d}^{\pi_\theta}$ induced by π_θ as our distribution f. In the following sections, we will describe the parts that compose this architecture and then we will describe the training process and how these components are used.

4.2.1. Recursive History Encoder

The policy HMepol learns is made of two components. In this section, we describe the first component which we call a recursive history encoder. This component allows us to compactly represent the history as a limited size vector at each time-step, which we will call h'_t. At the beginning of each trajectory, the history embedding is initialized with a zero vector, namely $h'_0 = \mathbf{0}$. At each step t, we encode together s_t and h'_{t-1} and the resulting encoding becomes the new history vector h'_t. Calling $g_{\boldsymbol{\theta}}$ the function computed by the encoder we can define the history recursively as:

$$\begin{cases} h'_0 = \mathbf{0} \\ h'_t = g_{\boldsymbol{\theta}}(s_t, h'_{t-1}). \end{cases}$$

The vector h'_t depends on the full history of states visited up to time t, and it can retain part of the information about the visited states without requiring to save the complete history. A gradient ascent procedure allows to learn a function $g_{\boldsymbol{\theta}}$ that retains only the most important information from the history and to be encoded in h'_t.

4.2.2. Diagonal Gaussian Policy

The second component of the policy is a diagonal Gaussian policy that can be defined as a multivariate Gaussian distribution with mean μ_θ and covariance matrix Σ:

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi^n |\Sigma|}} \exp\left(-\frac{1}{2}(a-\mu_\theta)^T \Sigma^{-1} (a-\mu_\theta)\right),$$

where $|\Sigma|$ is the determinant of the covariance matrix and $n = n_a$ is the dimension of the action vector. The mean of this distribution is parameterized with a neural network that, at each time-step t, takes as input the representation of the history h_t given by the history encoder and outputs the mean action vector μ_θ. The covariance matrix can be defined in different ways, ranging from a constant value to a function of the history representation at each step. It is useful to directly represent the logarithm of the standard deviation instead of the covariance matrix since $\log \sigma$ has values in the range $(-\infty, +\infty)$ which makes the learning process easier.

Combining these two components we obtain the policy which is learned by HMepol (see Figure 4.4).

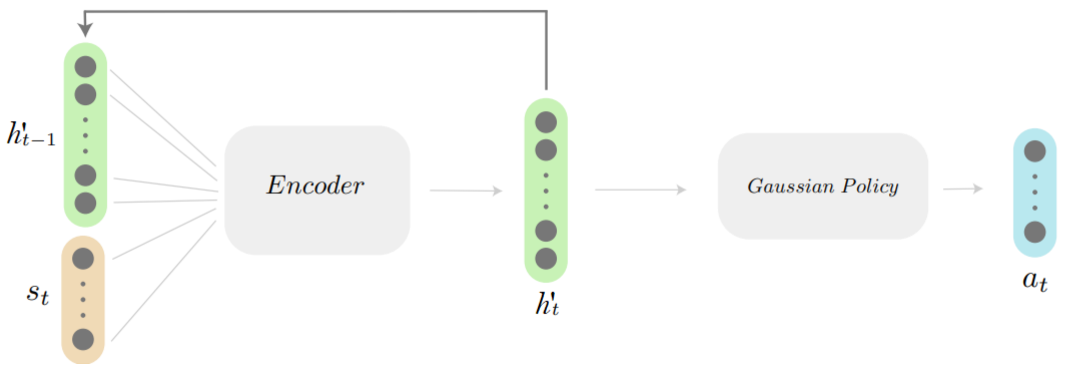


Figure 4.4: A representation of the policy architecture learned by HMepol.

From here we will denote the policy as $\pi_{re\theta}$ where re stands for Recursive Encoder, differently from a general non-Markovian policy we will define it as:

$$\pi_{re\theta} : \mathcal{H}' \times \mathcal{S} \rightarrow \mathcal{H}' \times \Delta(\mathcal{A}),$$

where $\mathcal{H}'$ represents the image of the function computed by the recursive history encoder. The policy gets as input the previous encoded story and the current state, uses them to get the new encoded story and lastly uses this story to compute the action to perform. As output, it produces the current encoded story which will be used in the next interaction and the action to perform.

4.2.3. The Training Process

The training process of $\pi_{re\theta}$ starts from sampling the environment $\mathcal{M}_i$ with which it interacts according to $p_{\mathcal{M}}$. The agent interacts with the environment generating B trajectories of length T, where B is the size of each mini-batch. The addition of this parameter is motivated by the fact that the entropy estimate 2.7 is asymptotically unbiased, thus it requires a significant amount of samples to give a reliable estimate. This particular parameter is crucial: in an infinite-samples regime a non-Markovian policy has no advantages over a Markovian policy [21], so increasing the number of samples used in the entropy estimation reduces the gap in performance between the two classes of policies. We compute the entropy of each mini-batch $\hat{H}_{\tau_j}$ using 2.7 and we obtain a set $\mathcal{D}$ of N/B mini-batches. Then, we compute the gradient as seen in 2.5, where we consider for each trajectory the entropy estimate obtained in the corresponding mini-batch. The update of the policy is done off-policy as in [19, 22]. This process continues until the updated policy $\pi'_{re\theta}$ is within a trust-region bound [29], computed as the KL divergence between the current policy and the sampling policy, or until the number of off-policy iterations exceeds a specific threshold. The overall structure of the algorithm resembles the one seen in Algorithm 3.1 considering several batches of trajectories instead of a single batch. The main difference lies in the policy architecture. At the beginning of each trajectory, the policy resets the last encoded history to $\mathbf{0}$ and at each step of the interaction, it saves the new encoded history which is used by the gaussian policy in the current step and by the recursive encoder in the step. We can represent these operations with a computational graph from the first step of a trajectory up to the last one through multiple passes of the recursive encoder which for simplicity can be unfolded as seen in 4.5.

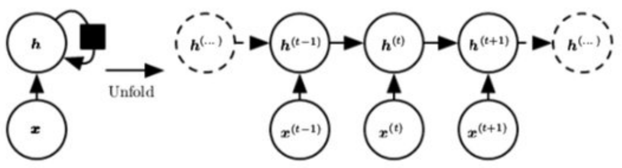


Figure 4.5: Unfolding of a recurrent computational graph (from [7])

Several approaches to back-propagate the gradient are available:

- back-propagating through the entire graph so through the entire trajectory. Unfortunately, this can lead to several known issues like exploding and vanishing gradients [7] and easily becomes computationally intractable with the length of a trajectory.
- back-propagating through "n" steps of the recursive computation which adds a new hyper-parameter while mitigating the aforementioned issues;
- back-propagating just one step, by considering each step of the trajectory independently for the gradient computation. This removes the issues of exploding and vanishing gradients of recurrent and recursive policies, but it can sometimes lead to slower convergence as the optimization process is not aware of the recursive nature of the encoder inside the policy.

In the next chapter, we will describe experimental results obtained by using the last approach for training.

5 | Experimental Analysis

In this chapter, we present experimental analysis to show a comparison of HMepol with other state-of-the-art algorithms, mainly Mepol [19] and αMepol [22]. In Section 5.1 we illustrate the advantages of the non-Markovian policy learned by HMepol in the experimental setting presented in [22]. In Section 5.2 we introduce a new setting which allows us to highlight the cost that is associated with the identification of an environment and how the policy learned by HMepol can take actions targeting the environment identification to later exploit the environment-specific optimal strategy. Lastly, in Section 5.3 we show how the presented results handily scale to larger classes of environments. The values of all the parameters and additional visualizations are provided in Appendix A.

5.1. Grid World with Slope

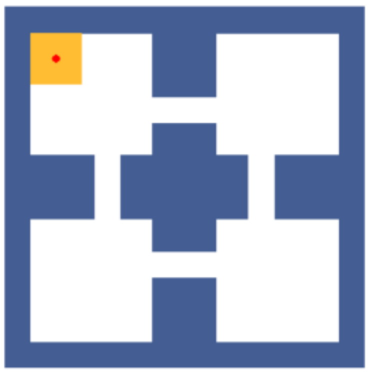


Figure 5.1: Visual representation of the Grid World environment. The agent (red circle) starts in a random position in the area highlighted in yellow (from [19]).

The Grid World with Slope environment, represented in Figure 5.1, is a continuous domain that is composed of four rooms connected by hallways. The state is the current position of the agent, represented as its (x, y) coordinate. The action represents the movement of the agent on both axes. Both the action and state spaces are subsets of $\mathbb{R}^2$. The State space is limited within a square with a side of length 2 centred in the origin, while the action for both axes is limited in the range $[-0.2, 0.2]$. On top of this Grid World configuration, we define two environments with different dynamics. One with a slope that moves the agent downwards and the other that moves the agent upwards. The slope s is inserted in the environment as a stochastic force sampled at each step from a Gaussian $s \sim \mathcal{N}(0.1,\ 0.01)$ truncated in the range $[0, 0.2]$. In such a set of environments a unique Markovian policy, that is optimal over the whole class, does not exist. To make the task even harder following [22] we set an unbalanced probability over this set $p_{\mathcal{M}} = (0.8, 0.2)$, similarly as in [22]. This sampling probability makes the setting with a south-facing slope easier to learn and, at the same time, increases the impact of this configuration on the average entropy. In Figure 5.2 we can see the entropy obtained in the two configurations

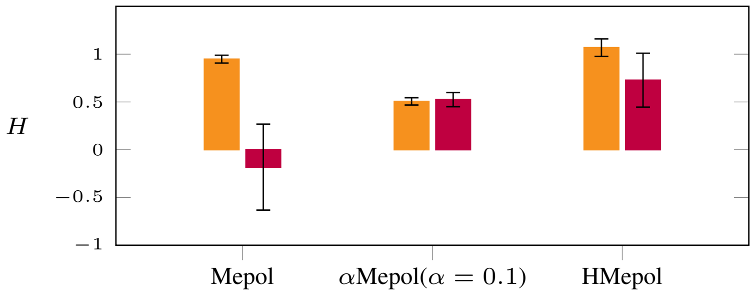
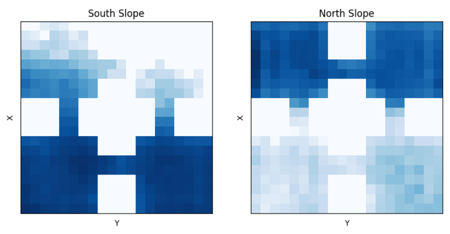
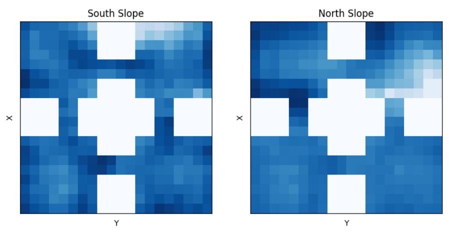


Figure 5.2: Entropy obtained by Mepol,αMepol and HMepol. On the left the result for the favorable configuration (south-facing slope) on the right the unfavorable configuration(north-facing slope). We provide average and 95% confidence intervals over eight runs.

by a policy learned with HMepol compared with Mepol and αMepol. We can see that the non-Markovian policy can reach optimal results in the favourable configuration(on the left) without sacrificing the performance in the less visited configuration (right). In Figure 5.3, we can see the difference in the coverage of the state space that we can get from 100 trajectories collected with a Markovian policy learned by αMepol and a non-Markovian policy learned by HMepol. We can see that the Markovian policy is forced to trade-off the performance between the two environments, resulting in suboptimal in both the configurations. The non-Markovian policy doesn't show such a trade-off, both environments are explored optimally covering the whole state space.



(a) αMepol



(b) HMepol

Figure 5.3: Heatmap of log probability of state visitations, computed from 100 trajectories of 400 steps collected using policies learned by AlphaMepol and HMepol.

5.2. Grid World with Corridor

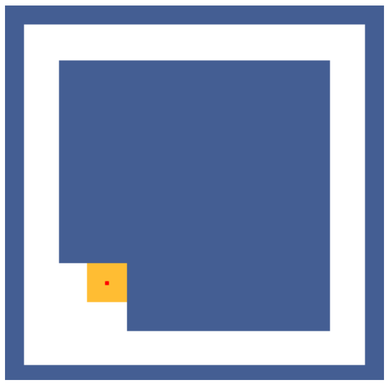


Figure 5.4: Visual representation of the Grid World with Corridor environment. The agent (red circle) starts in a random position in the area highlighted in yellow.

In this section, we will provide a comparison between Mepol and HMepol in a new environment which can be seen in Figure 5.4.

This environment may seem similar to the previous one but it is 36 times bigger. In this setting, we trained both Mepol and HMepol with 100 trajectories of 400 steps. We estimate the entropy independently for each trajectory so that to effectively reproduce a finite-trial setting as presented in [20]. On top of this Grid World domain, we define two environments with different dynamics. One with a wind that goes clockwise in the corridor and one that goes counterclockwise. The two configurations are sampled with equal probabilities. The wind is simulated as a fixed force added to the action of the agent. In this setting we consider a wind which adds an increment of 0.4 to the agent's action in the specified direction, this wind is twice as strong as the higher action the agent can make. With a wind this strong even if the agent tries to go against it would not be able to counter its effect. The wind is only present in the corridors and an agent cannot know in which of the environments it is placed unless it reaches a corridor. A Markovian policy learned with Mepol, is not able to grasp this information as it is only aware of the current state. This leads the agent to move around in front of the corridor entrance if it goes towards a corridor and it is the wrong one. In Figure 5.5, we can see that a Markovian policy leads the agent to explore that with a Markovian policy the agent ends up exploring only the corridor in one of the two configurations, while in the other one it just moves around the same entrance as it is pushed back by the wind. Depending on the initialization, the chosen entrance is different. The policy learns to enter using the first seen entrance and hardly explore the rest of the room. A non-Markovian policy, as we can instead see in Figure 5.6, can discriminate between the two settings. Through several runs of HMepol, we can see that once the agent reaches an entrance it makes several

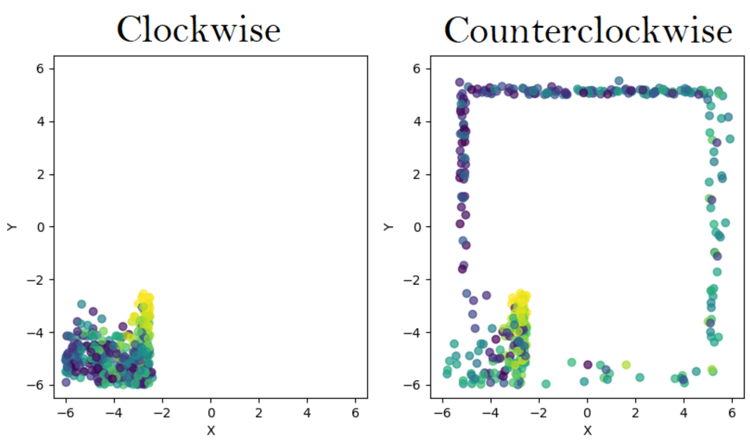


Figure 5.5: Representation of 10 trajectories collected from a policy learned with Mepol showing the position of the agent every 10 actions. Lighter colours indicate samples closer to the beginning of a trajectory.

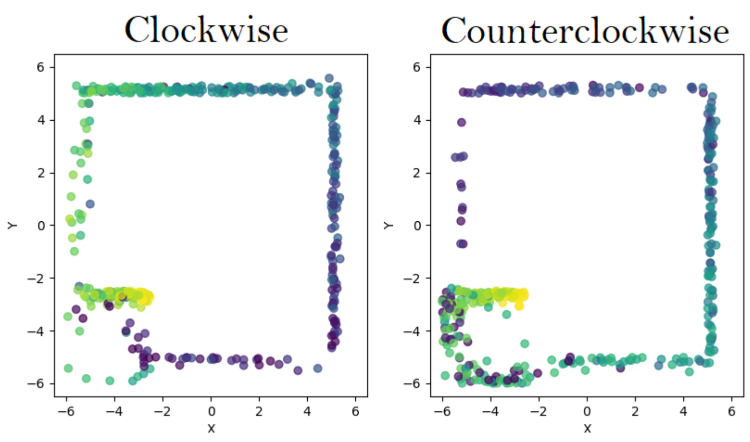


Figure 5.6: Representation of 10 trajectories collected from a policy learned with HMepol showing the position of the agent every 10 actions. Lighter colors indicates samples closer to the beginning of a trajectory.

attempts at entering the corridor, if those attempts fail the agent changes behaviour and goes towards the other entrance. This shows how the agent learned to identify the environment and how it exploits this information to act optimally. In Figure 5.7, we can see the entropy obtained in the two configurations by a policy learned with HMepol compared with Mepol. In Figure 5.8 we can see the difference in the coverage of the state space in 100 trajectories collected using a Markovian policy learned using Mepol and a non-Markovian policy learned by HMepol.

Lastly, in Figure 5.9 we show results obtained by fine-tuning using TRPO [29] the policies pre-trained with Mepol and HMepol. The subsequent task considered is reaching a specific area of the environment, we repeated this fine-tuning process for five random target areas.

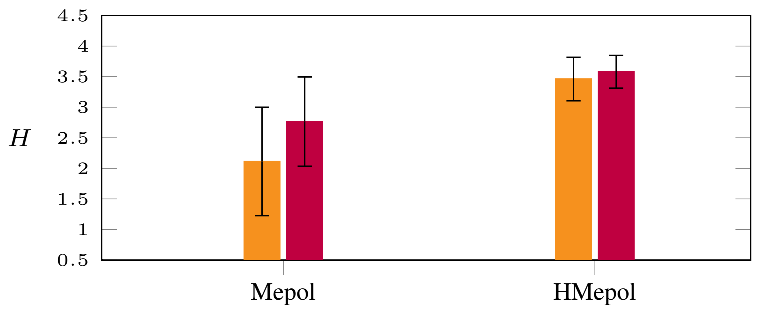
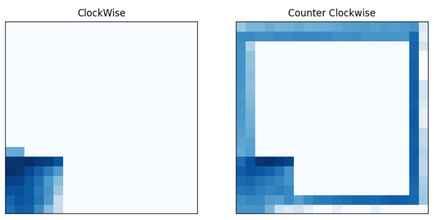
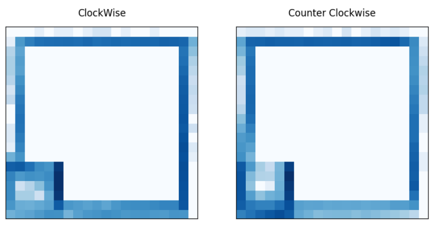


Figure 5.7: Entropy obtained by Mepol and HMepol. The left bar refers to the Grid World with clockwise wind and right bar to the one with counterclockwise wind. We provide average and 95% confidence intervals over eight runs.



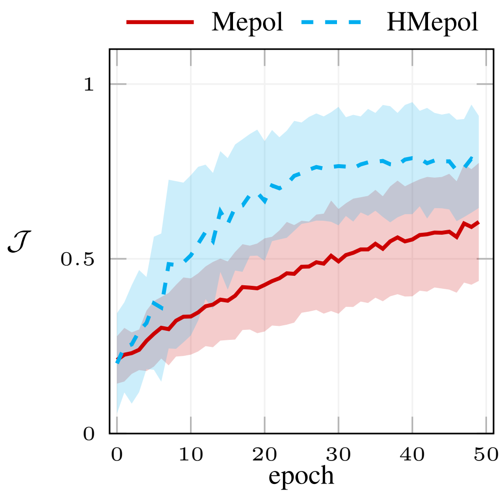
(a) Mepol



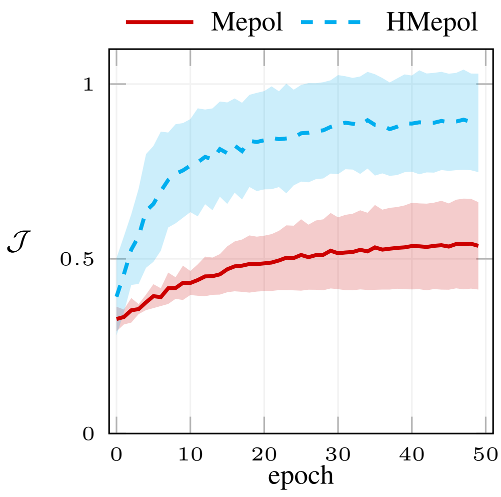
(b) HMepol

Figure 5.8: Heatmap of log probability of state visitation, among 100 trajectories with T=400, collected using policies learned using Mepol and HMepol.

We can see how the advantage in entropy reflects into a faster fine-tuning process, in the counterclockwise configuration in particular this difference is more evident, with HMepol reaching in less than 10 epochs the same return reached by Mepol in 50 epochs.



(a) Clockwise



(b) Counterclockwise

Figure 5.9: Average returns obtained by fine-tuning the trained policies using TRPO [29] with 5 random objectives for both clockwise (a) and counterclockwise (b) configurations. We provide average and 95% confidence intervals over eight runs.

5.2.1. The effects of different wind forces

Up to now, we considered a wind which completely overwhelmed the moving capabilities of the agent whenever it is faced in the wrong direction. By changing the value of the wind it is possible to tune the hardness of this set of environments. Intuitively, without the wind, a policy can just learn to explore the corridor in one direction with no risk. Increasing the wind, exploring the corridor in the wrong direction becomes harder while the exploration in the correct direction receives a boost. Identifying in which environment the agent is located has a cost. If this cost is higher than the loss in entropy of mistaking the corridor then the agent has no advantage in performing identification, making a Markovian and a non-Markovian policy equal. With a strong wind, this is no longer true as the agent is not able to enter the corridor in case of a mistake. This motivates the non-Markovian policy to identify the environment and act differently in the two configurations. In Figure 5.10 we provide a comparison of the average entropy obtained by policies learned with αMepol and HMepol. The gap is small with small wind forces and is maximum when we reach the highest wind which can't be handled effectively by a Markovian policy. Another interesting aspect of this comparison is that the non-Markovian policy can reach almost the same entropy whatever the force of the wind it finds in the environment. However, the performance of the Markovian policy clearly degrades once the wind becomes too strong.

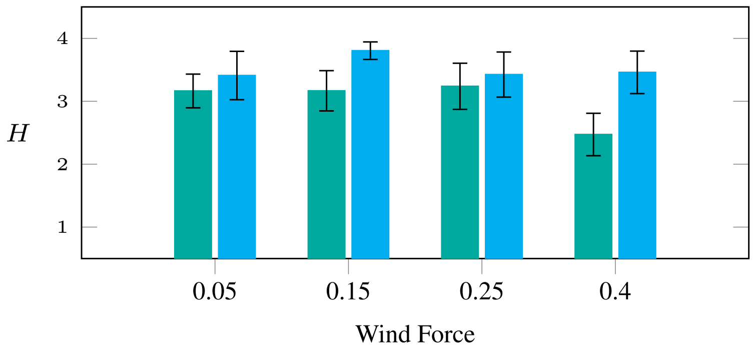


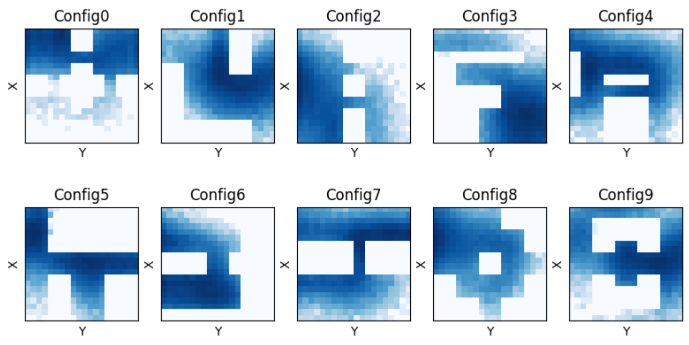
Figure 5.10: Average entropy at convergence (500 epochs) with different wind forces, on the left αMepol on the right HMepol. We provide average and 95% confidence intervals over eight runs.

5.3. Scaling to Larger Classes of Environments

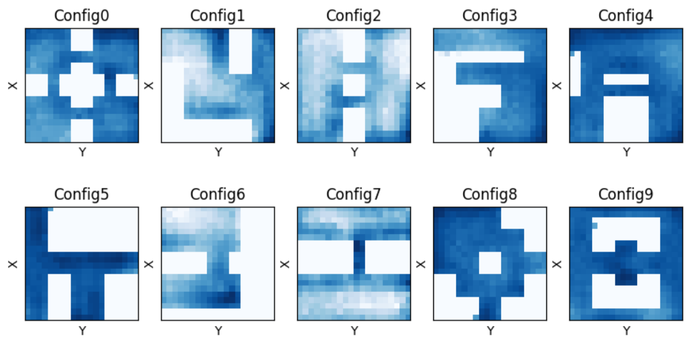
Up to now, we considered settings with two environments. In these experimental settings we were able to show the results which were expected from [20, 21]: in an unsupervised pre-training setting a non-Markovian policy can outperform a Markovian policy. Increasing the size of the set of environments the limitations of a Markovian policy is even more evident. At the same time a non-Markovian policy faces an increasing cost to identify the current environment and act accordingly. In this section, we show the ability of HMepol to learn a policy that can cope with a larger class of environemnts.

Our experimental setting is a class $\mathcal{M}$ of ten different gridworlds, i.e. the *MultiGrid* domain presented in [22]. One of the environments is the Grid World with Slope pointing to the north seen in Section 5.1, with a slope with a reduced slope $s \sim \mathcal{N}(0.077, 0.01)$. The other 9 gridworlds have different configurations and slopes.

Two configurations have slopes toward the south, three have an east-facing slope, one has a slope pointing towards the south-east and the remaining environments have no slope. The structure of the walls can be seen in the heatmaps in Figure 5.11. From Figure 5.12, we can see a clear advantage of the non-Markovian policy learned by HMepol. In Figure 5.13, we can see how the learning process starts with a minor difference between the Markovian and the non-Markovian policies, but the gap slowly increases as the policy learns to identify the environment. From the heatmaps we can also see how HMepol learns to explore all the environments adapting to each of them, reaching states that are not seen by αMepol.



(a) αMepol



(b) HMepol

Figure 5.11: Heatmap of log probability of state visitations, computed from 100 trajectories of 400 steps collected using policies learned by αMepol and HMepol.

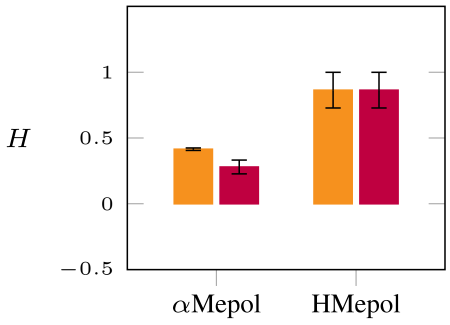


Figure 5.12: Entropy obtained by αMepol and HMepol. The left bar represents the average entropy, the right bar represents the entropy computed only in the first configuration (the most difficult one).

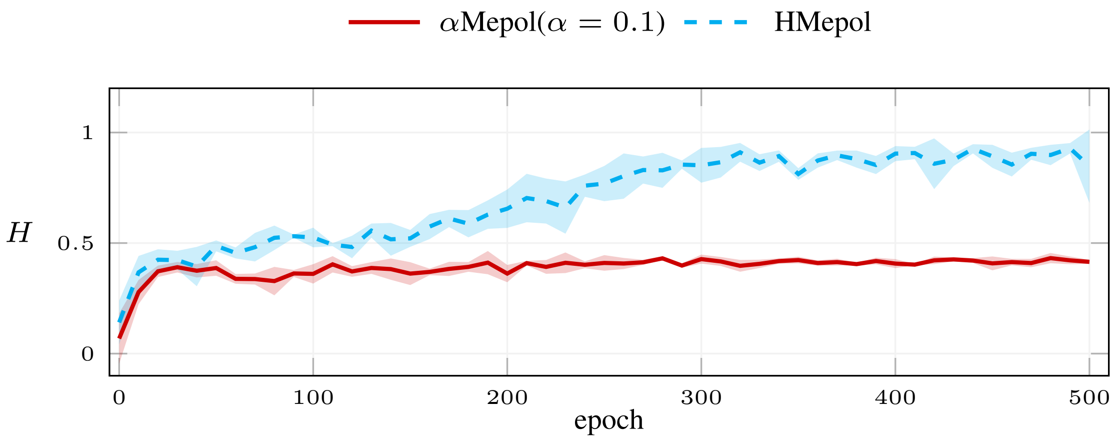


Figure 5.13: Average entropy at each epoch from policies learned with αMepol and HMepol. We provide average and 95% confidence intervals over four runs.

6 | Conclusions

In this thesis, we proposed HMepol, a policy-search algorithm for unsupervised pre-training in multiple environments. We explained the motivations that led our attention towards non-Markovian policies. Then, we showed the advantage of the proposed algorithm over state-of-the-art algorithms. We have also suggested the ability of the algorithm to scale to large classes of environments. We now propose some directions for possible future research. HMepol, as it is proposed in the thesis, is unable to handle vision-based domains. In the field of unsupervised pre-training for reinforcement learning some work [22, 30] uses compact state representation resulting from randomly initialized convolutional encoders. One possible approach would be to introduce the recursive history encoder after these convolutional encoders, but alternative architectures might be considered as well. Another direction consists in finding a more efficient and effective architecture to represent a non-Markovian policy. The current architecture is composed of two neural networks, which are quite small in comparison with other deep models. At the same time, scaling to high dimensional environments may require increasing the dimension of the history vector, which should be big enough to keep all the required information about the past. One possible architectural change that can be done to the current policy is to combine the two networks into one single network with two outputs, the action and the history vector. Other interesting architectures may also use gated recurrent neural networks with Long Short-Term Memory (LSTM) [11] or Gated Recurrent Unit (GRU) [3], which are able to control what to remember, and can better learn long-term dependencies. The use of more complex architectures, like neural Turing machines [8] and transformers [37] may also lead to overall better results, at the cost of additional computational resources and computational complexity.

Finally, we believe that this work motivates the study of non-Markovian architectures for unsupervised reinforcement learning, and that it represents a further step towards the development of artificial general intelligence.

Bibliography

[1] J. Ajgl and M. Šimandl. Differential entropy estimation by particles. *IFAC Proceedings Volumes*, 44(1):11991–11996, 2011.

[2] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[3] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[4] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403, 2013.

[5] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

[6] D. Ghosh, J. Rahme, A. Kumar, A. Zhang, R. P. Adams, and S. Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25502–25515. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/d5ff135377d39f1de7372c95c74dd962-Paper.pdf`.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[8] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[9] Z. D. Guo, M. G. Azar, A. Saade, S. Thakoor, B. Piot, B. A. Pires, M. Valko, T. Mesnard, T. Lattimore, and R. Munos. Geometric entropic exploration. *arXiv preprint arXiv:2101.02055*, 2021.

[10] E. Hazan, S. Kakade, K. Singh, and A. Van Soest. Provably efficient maximum

entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[12] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

[13] H. Liu and P. Abbeel. Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pages 6736–6747. PMLR, 2021.

[14] H. Liu and P. Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34, 2021.

[15] A. M. Metelli, M. Papini, F. Faccio, and M. Restelli. Policy optimization via importance sampling. *Advances in Neural Information Processing Systems*, 31, 2018.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[17] A. Moore. Efficient memory-based learning for robot control. Technical report, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[18] M. Mutti and M. Restelli. An intrinsically-motivated approach for learning highly exploring and fast mixing policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5232–5239, 2020.

[19] M. Mutti, L. Pratissoli, and M. Restelli. Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9028–9036, 2021.

[20] M. Mutti, R. De Santi, P. De Bartolomeis, and M. Restelli. Challenging common assumptions in convex reinforcement learning. *arXiv preprint arXiv:2202.01511*, 2022.

[21] M. Mutti, R. De Santi, and M. Restelli. The importance of non-markovianity in maximum state entropy exploration. *arXiv preprint arXiv:2202.03060*, 2022.

[22] M. Mutti, M. Mancassola, and M. Restelli. Unsupervised reinforcement learning in multiple environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[23] S. Parisi, V. Dean, D. Pathak, and A. Gupta. Interesting object, curious agent: Learning task-agnostic exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[25] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[26] R. Raileanu and T. Rocktäschel. RIDE: rewarding impact-driven exploration for procedurally-generated environments. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rkg-TJBFPB`.

[27] R. T. Rockafellar, S. Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[28] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[30] Y. Seo, L. Chen, J. Shin, H. Lee, P. Abbeel, and K. Lee. State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, pages 9443–9454. PMLR, 2021.

[31] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[32] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[33] H. Singh, N. Misra, V. Hnizdo, A. Fedorowicz, and E. Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

[34] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8): 1309–1331, 2008.

[35] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[36] J. Tarbouriech and A. Lazaric. Active exploration in markov decision processes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 974–982. PMLR, 2019.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[39] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pages 11920–11931. PMLR, 2021.

[40] C. Zhang, Y. Cai, L. Huang, and J. Li. Exploration by maximizing rényi entropy for reward-free rl framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10859–10867, 2021.

A | Experimental Setup

A.1. Parameters

In this Section, we provide the hyperparameters used in the experimental analysis.

A.1.1. HMepol

Parameters	**Slope**	**Corridor**	**MultiGrid**
Number of epochs	500	500	500
Samples collected for each	200	100	200
Horizon(T)	400	400	400
Batch Size	5	1	5
KL threshold (δ)	15	0.1	15
Learning rate	10^{-4}	$4 * 10^{-5}$	10^{-5}
Max off policy iterations	30	30	30
Number of neightbors (k)	30	30	30
recursive encoder layer sizes	(300,300)	(300,300)	(300,300)
history vector length	100	100	100
Policy hidden layers sizes	(300,300)	(300,300)	(300,300)
History encoder output act. function	ReLU	ReLU	ReLU
Policy and History encoder act. function	ReLU	ReLU	ReLU
Number of seeds	8	8	4

Table A.1: Hyperparameter HMepol

A.1.2. Mepol

Parameters	Slope	Corridor
Number of epochs	500	500
Samples collected for each	200	100
Horizon(T)	400	400
Batch Size	5	1
KL threshold (δ)	15	0.1
Learning rate	10^{-5}	$4 * 10^{-5}$
Max off policy iterations	30	30
Number of neighbors (k)	30	30
Policy hidden layers sizes	(300,300)	(300,300)
Policy hidden layers act. function	ReLU	ReLU
Number of seeds	8	8

Table A.2: Hyperparameter Mepol

A.1.3. AlphaMepol

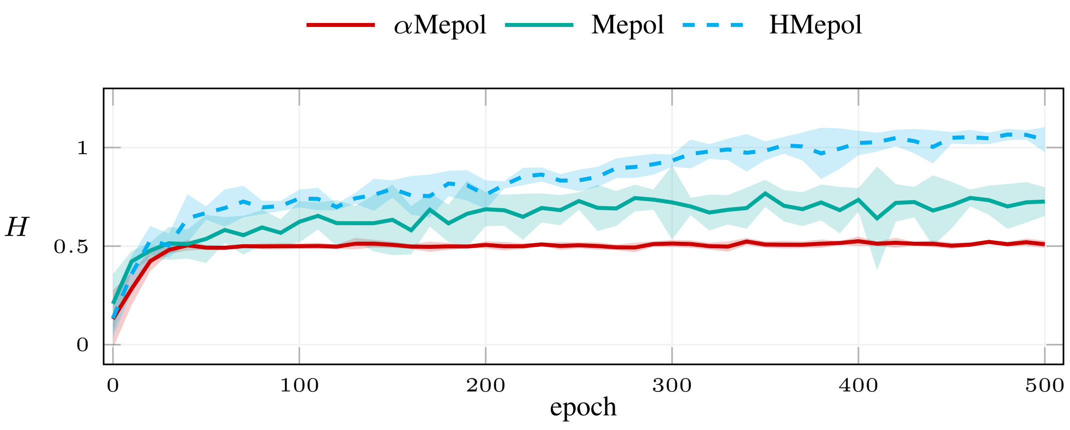
Parameters	Slope	MultiGrid
Number of epochs	500	500
Samples collected for each	200	200
Horizon(T)	400	400
Batch Size	5	5
KL threshold (δ)	15	15
Learning rate	10^{-5}	10^{-5}
Max off policy iterations	30	30
Number of neighbors (k)	30	30
α	0.35	0.1
Policy hidden layers sizes	(300,300)	(300,300)
Policy hidden layers act. function	ReLU	ReLU
Number of seeds	8	4

Table A.3: Hyperparameter AlphaMepol

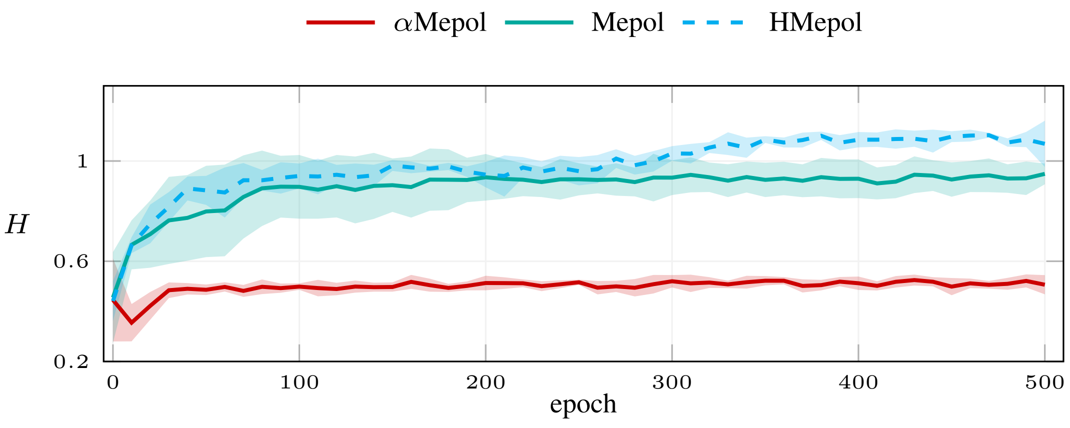
A.2. Additional Experimental Results

A.2.1. Grid World with Slope

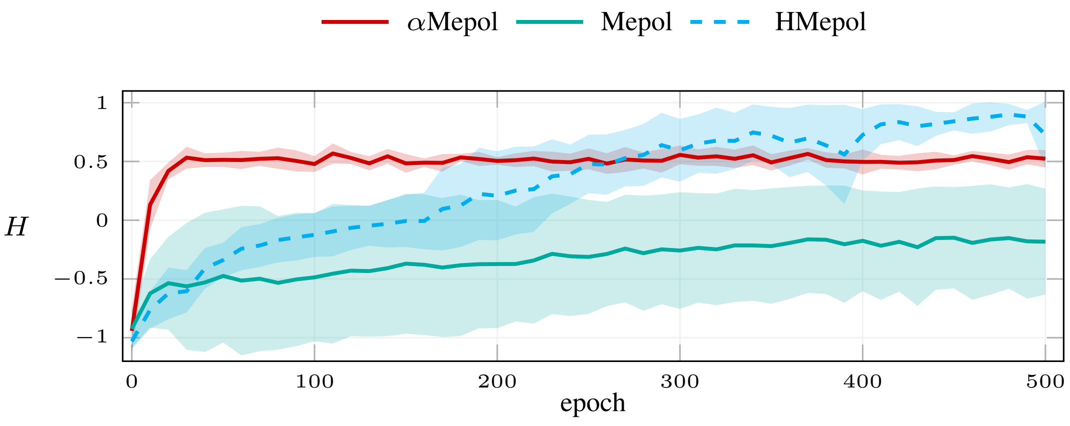
In this Section, we further analyze the results obtained in the Grid World with Slope environment. In Figure A.1a we provide the average entropy obtained at each epoch of the training process using Mepol, αMepol and HMepol. We can see that HMepol has better average entropy after the first fifty epochs and the gap between it and the other baselines increases during the whole training process, reaching values close to $H = 1$ which is the optimal entropy score obtainable in this environment. A more detailed analysis of this result can be done using Figure A.1b and Figure A.1c. The first one represents the average entropy, obtained by considering only trajectories sampled from the configuration with southwards wind, at each step of the learning process. In this configuration which is easier to learn, we can see that the obtained results are close to those of Mepol and superior to those of αMepol. In the other configuration, we see that HMepol takes a lot of epochs to surpass αMepol, but can surpass Mepol on average within the first fifty epochs. These separated graphs allow showing how a non-Markovian policy can learn to explore even harder or rare environments, without the need for a risk-averse objective. The learning process of HMepol in those environments is slower than that of a risk-averse algorithm but can reach better performance at convergence thanks to the environment identification that allows a non-Markovian policy to act optimally according to the chosen configuration.



(a) Average entropy



(b) Entropy contribute first configuration (southwards wind)

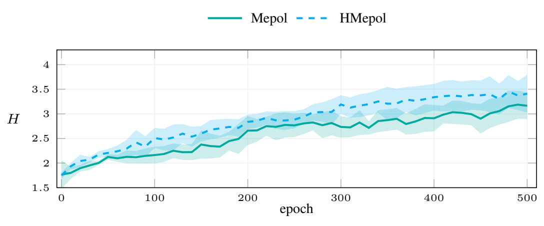


(c) Entropy contribute second configuration (northwards wind)

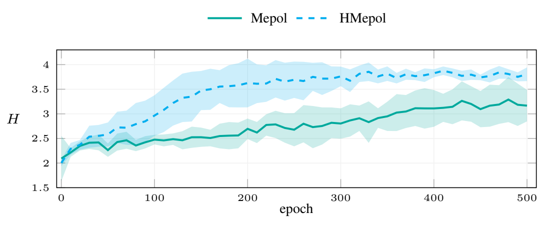
Figure A.1: Average entropy (a) obtained during the training process of Mepol, αMepol and HMepol. We also provide the average entropy obtained by considering only the first configuration (b) and the second configuration (c). We provide average and 95% confidence intervals over eight runs.

A.2.2. Grid World with Corridor

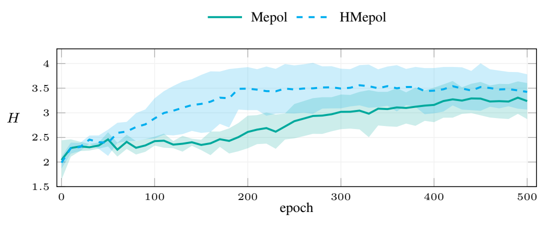
In this Section, we further analyze the results obtained in the Grid World with Corridor environment. In Figure A.2 we provide the average entropy, considering different wind forces, obtained at each epoch of the training process using Mepol and HMepol. In each graph we can see that initially there is almost no difference in obtained entropy, these initial epochs are those where the agent learns to explore the room and has not found the corridor yet, once the corridor is found both algorithms have a quick improvement. In Figure A.2a we can see that this improvement continues and the difference between the two algorithms is small. With stronger wind forces we can see that after 30-40 epochs the improvement of Mepol slows down and almost stops with a wind force of 0.4. After those epochs, the gap between a Markovian policy learned using Mepol and a non-Markovian policy learned using HMepol increases until the latter reaches a plateau at the optimal entropy value.



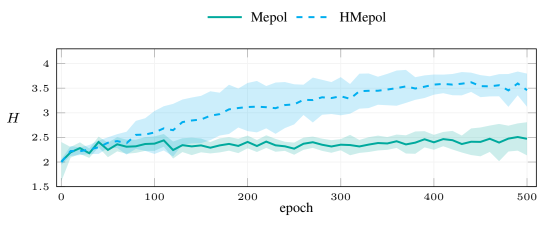
(a) wind force 0.05



(b) wind force 0.15



(c) wind force 0.25



(d) wind force 0.4

Figure A.2: Average entropy (a) obtained during the training process of Mepol and HMepol with different wind forces. We provide average and 95% confidence intervals over eight runs.

.

List of Figures

List of Tables

List of Symbols

Variable	**Description**
$\mathcal{S}$	State Space
$\mathcal{A}$	Action Space
$\mathcal{P}$	Transition Model
$\mathcal{R}$	Reward Function
$\mathbb{R}$	Set of Real Numbers
$\Delta(\cdot)$	Simplex over the specified set
T	Time Horizon
$\mathcal{H}$	Set of Histories
$\mathcal{T}$	Set of Trajectories
$\mathbb{P}(\cdot)$	Probability of the specified event
H	Entropy function
$D_{KL}(a\|\|b)$	KL divergenge between distribution a and b
$\mathcal{M}$	Set of Environments
Σ	Covariance matrix of a multivariate Gaussian distribution

Acronyms

C-BET	Change-Based Exploration Transfer 19
CVaR	Conditional Value-at-Risk 20
HMepol	History Mepol i, 3, 5, 25, 26, 29, 31, 32, 34, 35, 39, 47–49, 51, 52
IW	Importance Weighted 13, 18, 25
KL	Kullback-Leibler 13, 19, 27
knn	k-Nearest Neighbors 13, 18, 25
MDP	Markov Decision Process i, 1, 2, 5–8, 23, 51
Mepol	Maximum Entropy POLicy optimization 18, 29, 31, 32, 34, 35, 47–49, 51, 52
nat	natural unit of information 12
RL	Reinforcement Learning i, 1–3, 5, 8–10, 15, 23