Executive Summary of the Thesis

Game-theoretic policy optimization for non-stationary environments characterized by abrupt changes and drifts

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Valerio Colombo

Advisor: Prof. Manuel Roveri

Co-advisor: Prof. Marcello Restelli

Academic year: 2020-2021

1. Introduction

Non-stationary environments are challenging scenarios for Reinforcement Learning(RL) algorithms, due to the changing nature of the environment. Many work address such environments making strong assumptions on the type of non-stationarity[1, 2, 7] or on the distribution of changes that can be encountered[5, 8]. The objective of this work is to provide an algorithm capable of training an agent to be resilient to non-stationary environments while dropping as many assumptions as possible. In particular, the environment considered in this work can have both abrupt and gradual changes in either the transition or reward function, unlike the work done by [1, 2, 7].
The environment is composed of a sequence of Markov Decision Process(MDP), each of which will be called context. To make the environment as realistic as possible, no assumptions on the number of contexts, duration of contexts, and distribution of contexts have been considered as opposed to the work done by [5, 8].
The algorithm developed in this work, Game-MBCD, tries to address all these requirements to construct an agent able to act in such complex environments. Following the work done by Alegre et al.[1], the main idea of the proposed algorithm is to construct a mixture model M, where each element represents the context-specific environment model and policy. The mixture model can act as a library of encountered contexts to avoid the *catastrophic forgetting* phenomenon.
The main contribution of this work is to reduce performance loss when new contexts are encountered for the first time. Based on the work done by Rajeswaran[7], Game-MBCD incorporates game theory into a Dyna-style policy optimization algorithm. This new policy learning procedure shows promising results from both a sample-efficiency and asymptotic performance point of view, while handling different types of non-stationarity.

2. Problem formulation

The non-stationary environment considered can be formulated as a concatenation of MDPs $\{\mathcal{M}_z(t_z)\}$ with $z \in \mathbb{N}^+$ defining the context and $t_z \in \mathbb{N}^+$ indicating the timestep inside the context z. Each MDPs is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}_z(t_z), \mathcal{R}_z(t_z), \mu_0, \gamma)$, where $\mathcal{S}$ is a continuous state space, $\mathcal{A}$ is a continuous action space, $\mathcal{T}_z(t_z)$ is the transition function. $\mathcal{R}_z(t_z)$ describes the reward function. μ_0 is the starting state distribution and γ is the discount factor. The transition and reward function are modeled as a parametrized joint conditional probability distribution over next-state and reward defined as $p_{\theta_z}(s', r|s, a, t_z)$, where θ_z are latent parameters unknown to the agent. The sequence $S_M = (\mathcal{M}_1(t), \mathcal{M}_2(t), \dots)$ of MDPs is randomly sampled from an unknown distribution. The only assumption made, is that before and after a drifting context there are only stationary contexts. Another random distribution generates the sequence of changing points timesteps $S_C = (C_1, C_2, \dots)$, which indicates at which timestep the environment switches from one context to another. Furthermore, the exact number of possible contexts is unknown at the start of the training.

As stated before, the environment is characterized from both abrupt changes and drifts. Abrupt changes are modeled as a transition from a stationary context $\mathcal{M}_{z_i}$ to a stationary context $\mathcal{M}_{z_j}$ with a different transition or reward function. Instead, drifts are modeled as slowly changing contexts. The drifting contexts have a starting transition and reward function, which are respectively equal to the transition and reward function at the end of the previous context. During the drifting context execution, either the transition or reward function change gradually. The time index t_z is necessary, in the case of drifting environments, to define the trend of the drift.

3. Environment modeling

As introduced in Section 2, every context $\mathcal{M}_z$ is characterized by a distribution over the next state and reward $p_{\theta_z}(s', r|s, a)$. To handle more than one context, each model $p_{\theta_z}(s', r|s, a)$ is added to a mixture model M. This mixture model works as a library of encountered environments. The objective is to recollect a model $p_{\theta_z}(s', r|s, a)$ and the relative policy π_{ψ_z} when the corresponding context z is recognized. Otherwise, if a new environment is encountered, a new model will be added to the mixture model M, to avoid the phenomenon of catastrophic forgetting. The mechanism enabling the recognition of the various contexts will be explained in Section 4. Game-MBCD assumes that every model $p_{\theta_z}(s', r|s, a)$ represents a multivariate Gaussian distribution with diagonal covariance matrix, modeling the true dynamic and reward function of context $\mathcal{M}_z$. Based on the work of Chua et al. [3], Game-MBCD models each $p_{\theta_z}(s', r|s, a)$ as a bootstrap ensemble of probabilistic neural networks parametrized by θ_z. From the analysis in [3] it emerges that this methodology is capable of handling both aleatoric and epistemic uncertainty. To train each ensemble element $p^n_{\theta_z}(s', r|s, a)$, the loss is:

$$\ell_p(\theta^n_z, D) = \mathbb{E}_{s',r,s,a \sim D}[-log\, p_{\theta^n_z}(s', r|s, a)], \quad (1)$$

where D represents the memory of the agent containing transitions coming from a given context.

In Game-MBCD, the environment model is used for two purposes: a) to generate simulated rollouts used to train the policy in a Dyna-style approach and b) to create the mixture model M for context recollection. These two objectives have contrasting requirements. For this reason, it is useful to use two different networks. One for rollout generation (p_{θ_z}) and the other for context recognition ($p^C_{\theta_z}$) trained in different ways. In particular, they differ in the definition of set D. For p_{θ_z} only the last $\mathcal{G}$ most recent samples are used.

For $p^C_{\theta_z}$ all the data acquired in a given context are used. In the case of the context recognition network $p^C_{\theta_z}$, the objective is to create the most exhaustive representation of the current context, covering as much state space as possible. For this reason, it is advisable to use all available samples[1] originated from context z. In the case of the rollout generator network p_{θ_z}, the requirements are almost complementary.

The network has to adapt as fast as possible to changes in the environment, and should create a good representation of the states that are visited the most during the normal behavior of the agent, avoiding to model states that are rarely encountered or unreachable. Furthermore, after the environment transition from a stationary to a drifting context, using all the samples during the detection delay period could lead to two main problems. First, the samples from the past stationary environment could hurt the performance of the model because they are sampled from a dynamic that is different from the current one. Second, the more time it passes between the start of the stationary context, and the start of the drift, the bigger is the number of samples relative to the stationary part compared to the one originating from the drift. This implies that the learning is not invariant to the duration of the various contexts, leading to different training results.

4. Change detection algorithm

To fully use the mixture model, Game-MBCD needs an online change point detection algorithm(CPD). In general, CPDs are designed to detect when the underlying parameters of a stochastic process change. In the case of Game-MBCD, the CPD detects when the parameters θ_z of $p^C_{\theta_z}(s', r|s, a)$ change due to a context switch. Following the work in [1], let's take as an example an abrupt change between two contexts $\mathcal{M}_0$ and $\mathcal{M}_1$. Let's consider the case where both contexts have already been encountered. The online CPD algorithm task is to output a precise estimate Ω of the true change point $\mathcal{C}$. Being a random variable, Ω will have some inherent noise caused by the missing information about the prior of the change point sequence, and by the stochasticity of the MDPs where the agent acts. In particular, the CPD defined in [1] is based on a multivariate version of CUSUM. The CUSUM statistics $W_{k,t}$ is calculated between the current context model and every entry in the mixture model, plus an additional entry representing a new, unseen context. In particular, a CUSUM statistic will be calculated for every model k as:

$$L_{k,t} = log\frac{p^C_{\theta_k}(s', r|s, a)}{p_{\theta^C_{z_t}}(s', r|s, a)} \tag{2}$$

$$W_{k,t} = max\Big(0, W_{k,t-1} + L_{k,t}\Big), \quad \forall k \in [1, K] \cup [new] \tag{3}$$

where z_t represents the index of the current model in the mixture model. $W_{new,t}$ can be interpreted as a measure on the likelihood that a completely new environment is encountered. This measure is based on whether the likelihood of all known contexts k is smaller than $p_{\theta_{new}}$. The new model likelihood can be written as

$$\hat{Y}_t = Y_t + \delta\, diag(\Sigma)_{\theta_{z_t}}(X_t)) \tag{4}$$

$$p_{\theta_{new}}(Y_t, X_t) = \mathcal{N}(\hat{Y}_t, \Sigma_{\theta_{z_t}}(X_t)) \tag{5}$$

where $Y_t = (s', r)$, $X_t = (s, a)$. In particular, δ indicates the smallest variation in the mean of p that is worth detecting.

Finally, after all CUSUM statistics $W_{k,t}$ have been calculated, the most likely context is selected as follows:

$$z_t = \begin{cases} \underset{k}{argmax}\, W_{k,t}, & \exists k \in [1, K] \cup [new] \\ & s.t.\, W_{k,t} > h, \\ z_{t-1}, & otherwise. \end{cases} \tag{6}$$

When a new context is detected($z_t \neq z_{t+1}$), the new context is initialized as follows. $p_{\theta_{z_{t+1}}}$ is initialized as $p_{\theta_{z_t}}$ in order to have continuity in case of drifts. The same happens with the policy $\pi_{\psi_{z_{t+1}}} = \pi_{\psi_{z_t}}$. The context recognition network $p^C_{\theta_{z_{t+1}}}(s', r|s, a)$ is randomly initialized.

5. Game-theoretic Dyna-style policy learning

Game-MBCD context-specific policies are learned through a Dyna-style approach. This approach consists of using the model of the environment to generate imaginary rollouts, used as training data for some model-free algorithm. One of the advantages of Dyna-style approaches is the sample efficiency compared to their model-free counterpart, still maintaining comparable asymptotic performance. The sample efficiency derives from the ability to generate a virtually infinite amount of training data for policy learning. The quality of the training data is heavily dependent on the performance of the environment modelization.

Dyna-style approaches rely on the cooperation of two components: model learning and policy learning. This cross-dependency has to be addressed properly to construct algorithms capable of extracting the most from the usage of environment models. To address this co-dependency, Rajeswaran[7] developed a methodology to cast MBRL algorithms into a game-theory framework. The main idea is to consider the interaction between the model and policy optimizer as a cooperative game. The class of games studied to construct this framework is Stackelberg games, which are asymmetric sequential games where players make decisions in a pre-specified order. In particular, there is a leader player who takes a decision first, then a follower player who takes a decision, called "response", based on the leader player's move. More formally, if A is the leader player, and B is the follower player the game formulation can be written as follows:

$$\min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*(\theta_A))$$
$$s.t.\ \theta_B^*(\theta_A) = \underset{\tilde{\theta}}{argmin} \mathcal{L}_B(\theta_A, \tilde{\theta}),$$

where $\mathcal{L}_A$ and $\mathcal{L}_B$ are respectively the loss of player A and B. θ_A and θ_B are players' parameterizations. In the case of Game-MBCD, the objective is to use the MBRL framework developed by [7], and use it on a Dyna-style algorithm. The two players of Game-MBCD are:

1. Policy player: This player is in charge of finding a policy for a given context. It maximizes the reward in the model learned by the model player.
2. Model player: This player tries to find a good representation of the environment. It minimizes the prediction error of data collected by the policy player.

Due to the asymmetric nature of Stackelberg games, the game can take two forms, depending on the choice of the leader player. Game-MBCD considers the variant Policy player As a Leader(PAL). This choice comes from the fact that this version is more easily adaptable to the baseline used, and from the non-stationary analysis done in the work by Rajeswaran[7]. The PAL formulation for the single context z can be described as:

$$\max_{\psi} \big\{ J(\psi, D^{sim}) \quad s.t.\ D^{sim} \sim p_\theta(\cdot|\cdot) = \underset{\theta}{argmin}\ \ell(\theta, D^{sub}) \big\}, \tag{7}$$

where J is the loss of policy π parameterized by ψ. D^{sim} is the set of simulated rollouts sampled from the learned environment model $p_\theta(\cdot|\cdot)$. D^{sub} is the subset of D containing only the S most recent data of D. To avoid making the notation too complicated, all the z subscripts are omitted from now on, because the discussion now considers only a single context. In practice, Game-MBCD uses the following procedure:

1. **Policy player move: Policy optimization**. The policy player (leader player) optimizes the policy using the simulated rollouts generated from the environment model.
2. **Policy player move: True data gathering**. The agent will act in the true environment following the newly updated policy π_ψ. By doing so, it will gather a fixed number of new transitions and rewards.
3. **Model player move: Environment model optimization**. This newly acquired data will be used by the model player to update the environment model.
4. **Model player: Simulated rollouts generation**. The newly updated model is then used to generate the simulated rollouts.

Let's see in details the procedure.

Policy optimization One of the positives of using a Dyna-style approach is its modularity in policy learning. This approach can be adapted to virtually every model-free policy learner because it can generate datasets from the environment model . For this work, the Soft Actor Critic(SAC)[4] algorithm has been used as model-free policy learner. The loss used to optimize the policy can be written as:

$$J(\psi, D^{sim}) = \mathbb{E}_{s\sim D^{sim}} \big[\mathbb{E}_{a\sim\pi_\psi} [\beta log(\pi_\psi(a|s))] - q_{\pi_\psi}(s,a) \big]. \tag{8}$$

Following the Dyna-style approach, the data used by SAC are coming from D^{sim} and so are entirely simulated.

Environment model optimization After the agent has gathered new data D^{sub} from the real environment, the model player optimizes the environment model $p_\theta(\cdot|\cdot)$ with those data. Because the model player is the follower player, the update of the network consists of a response to the policy player. In particular, following the formulation of the Stackelberg game, the model player should update the parameter θ till convergence. This type of response is called *best response*, because it minimizes the loss relative to the parameters. To achieve this, the true data D^{sub} acquired by π_ψ are split in a training set and a validation set sampled uniformly from D^{sub}. The validation set is used to perform early stopping. This early stopping procedure becomes more and more important the less data we have in the training set. Game-MBCD manages a trade-off on the number of samples $|D^{sub}|$ used for training. The smaller is $|D^{sub}|$ the more the training procedure will be able to follow changes in the environment.

This is because the elements of D^{sub} would be very recent, and so they are more representative of the current dynamic. On the other hand, a small training set could be very sensitive to overfitting and to noise in the data gathered. Because of this, the early stopping procedure is crucial for a good learning procedure, because it allows Game-MBCD to use smaller training sets and so be more resistant to changes in the environment, still maintaining a good generalization performance.

Simulated rollouts generation Game-MBCD makes use of the approach developed by Pan[6], called Masked Model-based Actor-Critic(M2AC), to generate better and longer simulated rollouts. The objective of M2AC is to construct a score u, which measures the quality of each simulated rollout. The rollouts are ranked based on this score, and only the top-performing rollouts are used for training. More in detail, the score u is measuring the uncertainty on the prediction of the next state and reward. To do so, M2AC measures the disagreement of the elements of the ensemble modeling $p_\theta(s', r|s, a)$. Following a One versus the Rest approach, at each rollout simulation step, a random network n is chosen from the ensemble. Then, using the Kullback-Leibler divergence, the estimated Gaussian distribution of network n is compared with the distribution estimated by all the other networks in the ensemble. More formally:

$$u_n(s, a) = D_{KL}\big(p_\theta^n(\cdot|\cdot)||p_\theta^{-n}(\cdot|\cdot)\big). \qquad (9)$$

Algorithm 1: Policy learning procedure

```
Input      : Non-stationary environment $E$,
             Number of network in ensemble $N$
Initialize: Model $p_\theta$; CUSUM model $p_\theta^C$; Policy $\pi_\Psi$;
             Datasets $D$, $D^{sim}$ and $D^{sub}$
for $t = 0, \dots, \infty$ do
    if $t \, mod \, F = 0$ then
        $D^{Sub} \leftarrow \{D_i\}, \quad i \in \{|D| - \mathcal{G}, \dots, |D|\}$
        // Update till convergence using $\mathcal{D}^{Sub}$
        $\theta \leftarrow \underset{\theta}{argmin}\big(\mathbb{E}\big[\ell(\theta, D^{sub})\big]\big)$
        // Update till convergence using $\mathcal{D}$
        $\theta^C \leftarrow \underset{\theta}{argmin}\big(\mathbb{E}\big[\ell(\theta, D)\big]\big)$
        $D^{sim} \leftarrow \{\}$
        $S \leftarrow \{s_i\}_i^B \sim D$ with replacement
        for $h = 0, \dots, H_{max} - 1$ do
            for $i = 1, \dots, B$ do
                $a_i \leftarrow \pi_\Psi(s_i)$, $s_i \in S$
                Randomly choose $n$ in $[1, N]$
                $(s_i', \tilde{r_i}) \leftarrow p_{\theta^n}(r, s^{'}|s_i, a_i)$
                $u_i \leftarrow D_{KL}[\mathcal{N}(\mu_\theta^n(\cdot), \Sigma_\theta^n(\cdot)) \,\|$
                  $\mathcal{N}(\mu_\theta^{-n}(\cdot), \Sigma_\theta^{-n}(\cdot))]$
            end
            Rank samples by $u_i$
            Get first $\lfloor wB \rfloor$ samples, $\{i_j\}_j^{wB}$
            $r_{i_j} \leftarrow \tilde{r_{i_j}} - \alpha u_{i_j}$
            $D^{sim} \leftarrow D^{sim} \cup \{(s_{i_j}, a_{i_j}, r_{i_j}, s^{'}_{i_j})\}$
            $S \leftarrow \{s^{'}_i\}_i^B$
        end
        for $s = 1, \dots, \eta F$ do
            $\Psi \leftarrow \Psi - \lambda_\pi \nabla \mathbb{E}_{D^{sim}}\big[J(\psi, D^{sim})\big]$
        end
    end
end
```

6. Experiments

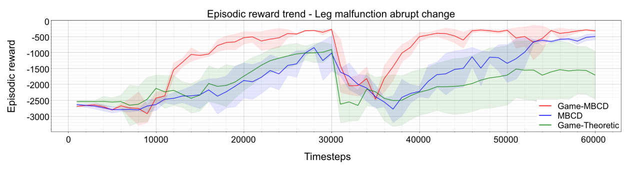
Game-MBCD has been evaluated on various non-stationary environments to test its resistance to various types of non-stationarity. In particular, the algorithm was tested on the continuous-state, continuous-action half-cheetah environment taken from OpenAI Gym. Game-MBCD was designed to extend the capabilities of the baseline used, in the scenario of unseen contexts. The experiments were designed to test how quickly the agent can adapt to these situations. The non-stationarity considered in the experiments involves both changes in the dynamics and changes in the goals. This is done to study the effects of changes on the transition function and the reward function. In particular:

- Dynamics changes: two actuators of the back leg may malfunction. In particular, the agent actions relative to these two joints are rescaled based on a factor.
- Goal changes: the target velocity can change during the execution.

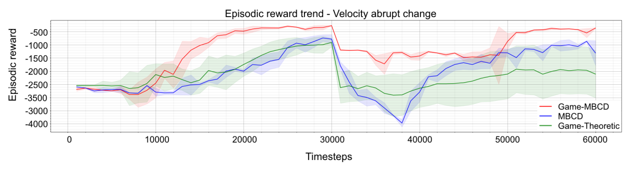
These two types of changes can be applied to the environments in two ways:

- Abrupt changes: either the back leg actuators are completely disabled or the target velocity changes with a jump
- Drifts: either the back leg actuators go linearly from completely functional to completely disabled in the span of a context or the velocity changes linearly

The first set of experiments studies the behavior of Game-MBCD in case of abrupt changes in the environment. In Figure 1, the environment is formed by two contexts, both stationary but with different transition or reward functions. From the episodic reward plot, it is clear how

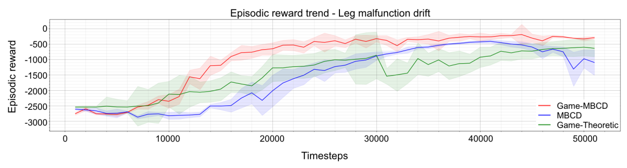


(a) Abrupt back leg malfunction

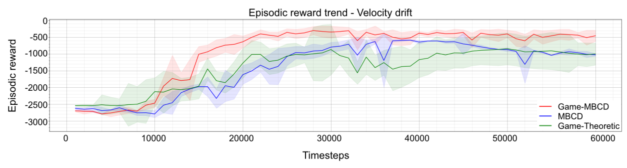


(b) Abrupt target velocity change

Figure 1: Half-Cheetah episodic reward in a two contexts environment. A stationary context with no anomalies from 0 to 30K timesteps. A stationary context with an anomaly from 30K to 60K timesteps.



(a) Leg actuator drift



(b) Target velocity drift

Figure 2: Half-Cheetah episodic reward in a two contexts environment. A stationary context with no anomalies from 0 to 30K timesteps. A drifting context from 30K to 50K for the leg actuator experiment, and a drifting context from 30K to 60K for the target velocity experiment

Game-MBCD outperforms both MBCD[1] and the game-theoretic framework[7] in sample efficiency during the first stationary context. Furthermore, when the context changes by either disabling the back leg of the agent or changing abruptly the target velocity, Game-MBCD is faster at recovering, and reaches a new asymptotic regime earlier compared to the other two algorithms.
The second set of experiments studies the behavior of Game-MBCD in case of drifts in the environment. In Figure 2, the environment is composed of two contexts: one stationary, and the following characterized by a drift in either transition (linearly disable back leg actuators) or reward function (linearly change target velocity). It can be seen how in both experiments, MBCD is not able to follow the changes in the environment. The performance drops as the drift unfolds, and the back leg is more and more inactive or the target velocity is different from the starting velocity. On the other hand, the two methods based on game theory can follow the changes without having any drop in performance. Furthermore, Game-MBCD has overall a higher performance compared to the two other methodologies. In the stationary part, Game-MBCD has a higher sample efficiency, and during the drift, it has a higher asymptotic performance. These experiments show how Game-MBCD is a step-up in both sample efficiency and resilience to unseen context changes.

7. Conclusions

This work introduces Game-MBCD, a hybrid model-based/model-free approach based on game theory, capable of handling non-stationary environments affected by both abrupt changes and drifts. This work explores the effects of applying game theory to hybrid RL algorithms. In particular, its usage enhances the algorithm performance in presence of drift in the transition function or reward function, and its sample efficiency when new unseen contexts are encountered. From the experiments conducted, Game-MBCD is capable of handling both abrupt changes and drifts in the environment associated with the transition and reward function. Furthermore, its sample efficiency surpasses recent MBRL algorithms like [7].
This work can be expanded by implementing a mechanism to transfer the knowledge between contexts, to further reduce the adaptation type in unseen contexts. Some interesting research directions include meta-learning and context encoding approaches to reuse the knowledge acquired in the past to make learning faster.

References

[1] Lucas N. Alegre, Ana L. C. Bazzan, and Bruno C. da Silva. Minimum-delay adaptation in non-stationary reinforcement learning via online high-confidence change-point detection, 2021.

[2] Yash Chandak, Georgios Theocharous, Shiv Shankar, Martha White, Sridhar Mahadevan, and Philip S. Thomas. Optimizing for the future in non-stationary MDPs, 2020.

[3] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[5] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. 2020.

[6] Feiyang Pan, Jia He, Dandan Tu, and Qing He. Trust the model when it is confident: Masked model-based actor-critic, 2020.

[7] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning, 2021.

[8] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables.

Game-theoretic policy optimization for non-stationary environments characterized by abrupt changes and drifts

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Valerio Colombo, 10499221

Advisor:
Prof. Manuel Roveri

Co-advisors:
Prof. Marcello Restelli
Giuseppe Canonaco, PhD

Academic year:
2020-2021

Abstract: Non-stationary environments are challenging scenarios for Reinforcement Learning algorithms, due to the changing nature of the transition and reward functions. The setting studied by this work considers an infinite random sequence of Markov Decision Processes (MDPs), each of which is drawn from some unknown distribution. To consider the most realistic setting possible, the algorithm does not make assumptions on the existence of a pre-training phase, a priori knowledge about the number, or boundaries between contexts. This work introduces Game-MBCD, a hybrid model-based/model-free approach based on game theory, capable of handling non-stationary environments affected by both abrupt changes and drifts. Game-MBCD does not require a pre-training phase. In particular, one of the objectives of this work is to improve the performance of the state-of-the-art when a new unseen context is encountered. Policy learning for every context is carried out with a procedure based on game theory, which accounts for the cross-dependency between environment modelization and policy optimization in Dyna-style RL algorithms. Furthermore, the baseline used for the algorithm development has been enriched with an approach based on the KL-divergence, to improve the quality of the simulated rollout dataset used for policy training. The experiments conducted show that Game-MBCD is more resilient to the various classes of non-stationary environments compared with model-based algorithms and non-stationary RL state-of-the-art algorithms.

Key-words: Reinforcement Learning, Non-Stationary, Change detection, Drift, Abrupt change

1. Introduction

Reinforcement Learning (RL) has been applied with success to many sequential decision problems with high-dimensional state spaces. Most RL algorithms start from the assumption that the environment in which the agent acts is stationary. In other words, the transition and reward functions do not change over time. In real scenarios, this constraint is very limiting and rarely met, leading to poor performance if not properly addressed.
For example, humans can solve very complex sequences of tasks while maintaining a very large degree of generalization[7]. This is due to many motivations. A human can recall previous experiences, to avoid learning from zero a task already encountered. Furthermore, the speed at which one can adapt is key to ensuring success in complex tasks sequences. For example, if a person suffers an accident where is momentarily unable to use a leg, they can quickly learn a new walking gait. If after some time their leg is available again, they will already know how to walk normally by recollecting how it is done, without the need of learning from zero how to walk again. Readiness to learn also translates into resistance to gradual changes. Again taking walking as an example, a person recovering from an accident will have a body dynamic that changes over time. Despite this, they will be able to complete their task by modifying their approach to the problem over time by adapting to the changes.
When an agent must act in a non-stationary environment, it must be trained with all of these considerations in mind.
Most of the current non-stationary RL algorithms do not consider scenarios like the one of the example, where both abrupt changes (sudden inability to use one leg) and gradual changes (recovery from an accident) can be present in the environment. The objective of this work is to expand the current state-of-the-art non-stationary RL by proposing a new algorithm called Game-MBCD, capable of handling both abrupt and gradual changes in either the transition or reward function. In doing so, Game-MBCD tries to cover as many non-stationary scenarios as possible. Furthermore, Game-MBCD aims to improve the performance recovery speed when new, unseen environments are encountered, in order to be resilient to unseen changes in the environment.
To achieve the goals explained before, this work proposes a new model structure composed of two parallel ensembles of probabilistic neural networks. One specialized in the detection of changes in the environment, and the other for policy optimization. Furthermore, Game-MBCD introduces an adaptation of game theory applied to hybrid model-based/model-free algorithms to enhance the sample efficiency of Dyna-style policy optimization algorithms. The improved sample efficiency gives the algorithm more resistance to gradual changes.
A key component of non-stationary RL works is the type of non-stationarity considered, and so the assumption and simplification done to make the different algorithms feasible from both a theoretical and computational point of view. In particular, the key components of a non-stationary environment, from the point of view of RL, can be recognized in the following components.

- Types of non-stationarity: there are two main sources of non-stationarity. Changes over the dynamic or changes over the goal. In the former case, the transition function of our agent is affected by an internal change of the agent (failing sensor, actuator, ...) or by an external change (different terrain, wind, ...).
  In the latter case, the reward function is not stationary. This type of change comes always from some external factor regarding the environment, because the goal, at least in the formulation of this work, is not imposed by the agent itself but it is provided by the environment.
- Type of changes: the changes affecting the agent or the environment can be abrupt or gradual. In the first case, a sudden change imposes a very quick modification of the transition function (complete breaking of an actuator) or reward function (jump in target velocity).
  In the second case, a slow and continuous change affects either the dynamics (wear of an actuator)

or reward function (slow change in target velocity).

- Task distribution: the tasks upon which an agent acts can be known or unknown. In the first case, the distribution can be used to make some assumptions regarding which tasks the agent will encounter after and during the training phase.
  In the second case, the problem becomes more challenging, because no task sequence structure is known a priori.
- Type of sequences: the exact number of tasks and task types can be known a priori. The number of tasks is useful to define the horizon of the environment. Knowing the number of different typologies is extremely useful and simplify greatly the tractability of the problem. Handling a possibly infinite number of different tasks imposes important limitations on the design of the algorithm.
  Another possible assumption involves the smoothness between tasks. In particular, at the start of the drift, the starting condition can be the same as the end of the previous task or differ.

The non-stationary environment considered in this work is characterized by a random sequence of Markov Decision Processes(MDPs) called contexts, which are drawn from an unknown distribution. Each context in our sequence can be stationary or non-stationary. The non-stationary contexts will be affected by a drift in dynamics or goal.
The agent has to learn how to optimize its behavior in a changing environment, which imposes changes to the transition function or reward function, according to some latent variable unknown to the agent. In other words, the agent should be capable of handling non-stationary environments, where the non-stationarity arises from both abrupt changes and drifts, possibly in the dynamics or goal. Furthermore, to handle realistic scenarios, the number of contexts is not know a priori. Limiting the algorithm to a finite number of typologies of contexts is a very strong assumption. It is very unusual to know a priori what the agent can encounter in a situation never explored. Furthermore, this type of constraint would require total knowledge of the environment to cover all the possible nuances of all the different contexts.
These objectives impose some great challenges

1. Quickly react to changes in the environment.
2. Be sample efficient
3. Handle both abrupt changes and drifts.
4. Reuse previously acquired skills, when already seen contexts are encountered again.
5. Avoid assumption on the types and number of contexts encountered

All the steps to fulfill the previously listed requirements are explained in this document as follows.
At the beginning, in Section 2, there is an analysis of the current state-of-the-art regarding non-stationary RL. After that, Section 3 provides a formal definition of the problem addressed. Subsequently, Section 4 gives a more detailed explanation of algorithms used as a baseline for Game-MBCD. Then, in Section 5 every component of Game-MBCD is described. The section starts from the explanation of how the non-stationary environment is modeled, then it passes to the discussion about how Game-MBCD uses a change point detection algorithm to manage the various contexts. After that, there is an analysis of the advantages of splitting the model networks based on the objective, and lastly, the section is concluded with the introduction of the game-theoretic framework for hybrid RL algorithms. In the end, in Section 6 a series of experiments regarding sample efficiency and drift management is conducted, to show the improvements of Game-MBCD over the state-of-the-art.

2. Related works

In RL literature many approaches have been developed to handle non-stationary environments.

Active approaches This family of methods tries to go beyond the unimodal task distribution typical of stationary RL, using multimodal distributions to model big environmental changes. Practically, the objective is to construct a library of tasks and solutions (policies), to construct a methodology that can recollect past experiences and deploy at will already learned policies. To achieve this goal, these methodologies are often coupled with active change point detection algorithms, to understand when to switch between the modes of the task distribution. Usually, these methods do not need a pre-trained model or a pre-training phase to work properly. This fact is a very good aspect of the active approaches.
In [21] an active model-free adaptation of Q-Learning, called Context Q-Learning, has been developed to deal with non-stationary environments. Unfortunately, the work poses some important assumptions. For example, model-change patterns are known and so a finite set of context is considered.
Another interesting approach is Trajectory-wise Multiple Choice Learning(T-MCL)[27]. This method is a model-based algorithm able to model a multimodal distribution of tasks through the specialization of a multi-headed network. It does so in an unsupervised manner, utilizing a novel loss function called trajectory-wise oracle. To specialize each head, given a trajectory, the head which maximizes the return is updated, doing so each head specializes in a specific type of environment. With this approach, there is no need to define a priori the tasks because they are discovered in an unsupervised manner. Even though there is no limit on the number of different environments that the agent can encounter, the number of heads is pre-defined, imposing a limit on the number of contexts types. Furthermore, the reward function is assumed to be known.
Model-Based RL Context detection(MBCD)[1] is a model-based method that constructs a mixture model composed of a (possibly infinite) ensemble of probabilistic dynamics predictors, which model the different modes of the distribution over underlying latent MDPs. If a context change is detected, the algorithm can recall a previously model context from the mixture model to act optimally with zero delay from the change point detection. MBCD does not need assumptions on task distribution and does not need a pre-training phase. This algorithm makes use of the CUSUM[22] statics to actively detect changes both in dynamics and goal. Unfortunately, the algorithm is not capable of handling drifting environments. It starts from the assumption that the non-stationary environment is a piece-wise stationary environment.

	Num Tasks	**Reward**	**Dynamic**	**Abrupt**	**Drift**	**Distr.**
Context Q-learning[21]	Finite	√	√	√		√
T-MCL[27]	Finite		√	√		
MBCD[1]	Infinite	√	√	√		
Prognosticator[4]	Finite	√	√		√	
GrBAL/ReBAL[20]	Infinite		√	√		√
GMPS[19]	Infinite	√	√	√		√
PEARL[26]	Infinite	√	√	√		√
CCM[13]	Infinite	√	√	√		√

Table 1: Table summarizing characteristics of some algorithms. Num. tasks: indicate if algorithm can handle an infinite amount of tasks. Reward: ticked if algorithm handles changes on the reward function. Dynamic: ticked if algorithm handles changes on the transition function. Dynamic: ticked if algorithm handles abrupt changes. Drift: ticked if algorithm handles drifting environments. Distr.: Ticked if algorithm needs to make some assumption on task distribution

Another interesting work is the Prognosticator algorithm[4]. It is a model-free algorithm that optimizes the current policy to perform better in the future. It starts from the assumption that the environment experiences slow changes caused by an exogenous signal. It assumes that these slow changes are predictable, so it is possible to anticipate the changes reducing the adaptation lag. This work sheds new light on the management of predictable drifts but completely ignores abrupt changes.

Meta-RL approaches Another class of algorithms is based on meta-learning techniques. As stated by Weng: "A good meta-learning model is expected to generalize to new tasks or new environments that have never been encountered during training. The adaptation process, essentially a mini learning session, happens at test time with limited exposure to the new configurations"[30]. In other words, the objective is to find a set of meta-parameters that constitute a good initialization of parameters, to encourage fast adaptation to new environments with few updates. This methodology makes the important assumption that a given context is drawn from a unimodal distribution. In particular, each context can be considered as a new Markov Decision Process(MDP), where both the transition function and the reward function can change.
A large number of recent meta-RL works are based on Model Agnostic Meta Learning(MAML)[12]. It is a model-free on-policy gradient-based Meta-RL algorithm. The main idea is to train a base model on some training tasks, to make the model easily trainable for new, unseen test tasks. In other words, MAML finds a good initialization of the policy parameters(meta-parameters), to train the model in an unseen task with just a few gradient steps.
GrBAL and ReBAL[20] are model-based on-policy Meta-RL learning algorithms heavily based on MAML. The paper proposes two algorithms. One is gradient based(GrBAL) and the other is recurrence based(ReBAL). These two algorithms were a big step forward compared to MAML in terms of sample efficiency. On the other hand, these algorithms consider only changes in the dynamic.
Another algorithm based on MAML is Guided Meta-Policy Search(GMPS)[19]. Its development sparks from the idea that the acquisition of efficient RL procedures does not need to be performed with RL. This is in opposition to many meta-RL algorithms, which rely on inefficient on-policy RL algorithms to meta-learn. GMPS uses supervised imitation learning to be more sample efficient during the meta-training phase. In particular, it uses expert actions to optimize the algorithm's ability to adapt to new tasks via RL. GMPS tries to find a set of parameters such that with a small number of gradient steps the algorithm obtains a policy that matches the expert's action. A downside of this approach is that it needs a large number of expert examples during the meta-training phase.
Another type of meta-RL algorithm is the context based approach. This type of algorithm tries to extract the latent context from the environment using only a small number of interactions. This compressed representation should contain useful information about the task distribution. Context-based Meta-RL methods can train a policy conditioned on the latent context to improve generalization.
Probabilistic Embeddings for Actor-critic RL(PEARL)[26] is a model-free off-policy meta-RL algorithm. The main focus of PEARL is to construct a sample efficient Meta-RL algorithm by leveraging off-policy methods(SAC)[14]. The off-policy approach makes it possible to use previously collected experiences.
Contrastive learning augmented Context-based Meta-RL(CCM)[13] is a context based off-policy Meta-RL algorithm. The paper proposes two main contributions: an unsupervised learning framework for context encoding leveraging contrastive learning and an information gain-based exploration strategy. Contrastive learning tries to learn a representation that agglomerates near to each other similar tasks and far apart different tasks.
Considering the objectives discussed previously, Meta-RL, despite having a lot of potentials, imposes important limitations in the form of assumptions. First of all, it requires a meta-training phase, which can be quite costly if we want to be sample efficient. Second, it makes assumptions about the task distribution. Furthermore, in the meta-RL literature drifting environments are rarely considered.

Starting from these considerations, the focus of this work is to develop an algorithm based on the active approaches. In particular, the focus is on integrating the current literature with the ability to handle drift as well as abrupt changes. A new algorithm called Game-MBCD is proposed. This algorithm uses MBCD[1] as a baseline, but enhances it with resistance to drifting environments and greater sample efficiency, to make the agent adapt faster in both abrupt and gradual changes scenarios. The game-theory framework discussed in [25] has been used to improve the performance during drifts.

3. Problem formulation

The non-stationary environment considered can be formulated as a concatenation of MDPs $\{\mathcal{M}_z(t_z)\}$ with $z \in \mathbb{N}^+$ defining the context and $t_z \in \mathbb{N}^+$ indicating the timestep inside the context z. Each MDPs is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}_z(t_z), \mathcal{R}_z(t_z), \mu_0, \gamma)$, where $\mathcal{S}$ is a continuous state space, $\mathcal{A}$ is a continuous action space, $\mathcal{T}_z(t_z) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{N}^+ \to [0, 1]$ is the transition function describing the probability of transitioning in state s', given a starting state s and performing action a at timestep t_z in context z. $\mathcal{R}_z(t_z) : \mathcal{S} \times \mathcal{A} \times \mathbb{N}^+ \to \mathbb{R}$ describes the reward function of performing an action a in state s at timestep t_z in context z. μ_0 is the starting state distribution and γ is the discount factor. The transition and reward function are modeled as a parametrized joint conditional probability distribution over next-state and reward defined as $p_{\theta_z}(s', r|s, a, t_z)$, where θ_z is a latent parameters set unknown to the agent.
The sequence S_M of MDPs is randomly sampled from an unknown distribution. Doing so, the agent cannot rely on the knowledge of the exact sequence of MDPs $S_M = (\mathcal{M}_1(t), \mathcal{M}_2(t), \dots)$, making the handling of the environment more challenging. The only assumption made, is that before and after a drifting context there are only stationary environments.
Another random distribution generates the random sequence of changing points timesteps $S_C = (C_1, C_2, \dots)$ indicating the timestep at which the environment switches from one context to another. Furthermore, the exact number of possible contexts is unknown at the start of the training.
As stated before, the environment is characterized by both abrupt changes and drift. Abrupt changes are modeled as a transition from a stationary context z_i to a stationary context z_j with a different dynamic or goal. Instead, drifts are modeled as slowly changing contexts.
The drifting contexts have a starting transition and reward function, which are respectively equal to the transition and reward function at the end of the previous context. The time index t_z is necessary, in the case of drifting environments, to define the trend of the drift. In particular, the drift is applied to variables that can be either used to modify the transition or reward function. For example, in a control scenario, the action of the agent can be rescaled by a factor $f_{action}(t_z)$, to simulate some gradual malfunctioning of some actuator, and so slowly change the transition function of the agent.
Another case could be a gradual change in the goal of the agent. For example, the target velocity of a moving agent can change linearly over time. In this case, the reward function could assume the following shape: $\mathcal{R}_z(t_z) = |(v_{target}(t_z) - v_{actual})|$, where $v_{target}(t_z) = \frac{t_z}{T} v_{end} + (1 - \frac{t_z}{T}) v_{start}$ and T is the number of timesteps for which the context z will be in place.
It is worth noticing that, in the scenarios involving drifts of either the dynamic or the goal, the change to the transition function or the reward function is applied online. This means that, in the case of episodic environments, the single episode can be in turn non-stationary. This fact stresses the agent to adapt in real-time while the changes are happening.

4. Background

The objective of the work is to extend the state of the art regarding the handling of non-stationary environments to a setting that covers the main characteristics of real environments, and to enhance the speed at which the agent is capable to adapt when new unseen contexts are encountered. To develop the proposal of this work (Game-MBCD), two main algorithms have been taken as a starting point: MBCD[1] and the game-theoretic framework developed by Rajeswaran et al. [25].
The MBCD algorithm constitutes a good starting point for the objectives described in Section 1. The major strengths are:

- the capability of handling a virtually infinite amount of contexts. The main idea is to use a mixture model which tries to describe the multimodal distribution of contexts. When a new context is recognized, a new entry in the mixture model is added to construct a library of contexts.
- the capability to recognize environment change through a sequential statistical test based on CUSUM[22] statistic. In particular, the change detection algorithm imposes an upper bound on the FAR(False Alarm Rate), giving theoretical guarantees on the robustness of the change detection algorithm.
- the capability to avoid catastrophic forgetting through context recollection. In this case, when an environment change is detected, the algorithm is capable of understanding if the current context has been already encountered or not. If it has been already seen, the context-specific model and policy are reloaded from the mixture model, and the agent has a zero-delay adaptation from the change detection point.

On the other hand, MBCD does not cover all the requirements that this work tries to fulfill. The main issues are:

- the lack of resistance to environments subject to concept drifts. The algorithm was intended to operate in environments characterized by a piece-wise stationary structure. This means, that the algorithm can only handle abrupt changes. This is a great limitation, because, in a real scenario, the environment changes can assume different shapes and characteristics, and are not limited to abrupt changes.
- the performance drops when a new environment is encountered. Speed in performance recovery after a context change is a major issue afflicting this algorithm.

To solve these issues concerning MBCD, the game-theoretic framework developed by Rajeswaran et al. [25] represents a good starting point. The main focus of the game-theoretic framework is to cast MBRL(Model Based Reinforcement Learning) as a game based on game theory. This game is composed of two players: policy and model player. The policy player maximizes the reward in the model learned by the model player. The model player minimizes the prediction error of data collected by the policy player. This setup sheds a very important light on the cross-dependency of model optimization and policy optimization in MBRL. Blindly training the two components without addressing the influence of one on the other, can penalize the final performance, both from an asymptotic and sample efficiency point of view. The framework has also theoretical guarantees regarding the optimality of the policy. This guarantee is described by a bound influenced by: a) the sub-optimality of the policy and model and b) by the presence of multiple Nash equilibria with different qualities. In [25] a brief analysis of the effect of non-stationarities on the performance has been done. The analysis shows how this framework can help in the case of changing environments. Game-MBCD merges MBCD with the game-theoretic framework introduced before, to equip MBCD with the ability to cope with drifts and to allow a faster recovery in case of abrupt changes.
To do so, some adjustments must be made. First of all, MBCD can be considered as a hybrid model-based/model-free algorithm, instead, the game-theory framework is purely model-based. MBCD can

be considered as a hybrid approach because it is based on MBPO[15]. The base idea is to generate simulated rollouts used as training data for an off-policy model-free optimizer based on SAC[14]. Considering the policy gradient theorem[9], the gradient of the policy π_θ parameters can be written as

$$\nabla_\theta J(\theta) = \int d^\pi(s,a)\nabla_\theta log\pi_\theta(a|s)Q^\pi(s,a)dsda,$$

where $d^\pi(s,a)$ is the stationary distribution of states and actions, and $Q^\pi(s,a)$ is the expected cumulative discounted return obtained by performing action a in state s. What makes an approach model-based or model-free is how $d^\pi(s,a)$ and $Q^\pi(s,a)$ are handled. In particular, this two terms depend on the dynamics $p(s'|s,a)$. If p is extracted directly from the real interaction between the agent and the environment, the gradient can be considered model-free. Otherwise, if p is extracted from a constructed model, the gradient can be considered model-based. MBPO uses the constructed model $\hat{p}$ for the term Q^π but uses the real interactions with the environment to construct the stationary distribution of states and actions $d^\pi(s,a)$. MBCD can be considered as a hybrid approach when the simulated rollout length is equal to 1 because simulated rollouts starting points are extracted from the real trajectory set.

5. Game-MBCD

Game-MBCD is a hybrid RL algorithm, which aims to construct a library of learned models $p(s',r|s,a)$, representing each context encountered during training. This library is implemented through a mixture model. To understand when to add a new $p(s',r|s,a)$ inside the mixture model, and which model to use in the current context, Game-MBCD uses a change point detection algorithm based on CUSUM[22]. The selected $p(s',r|s,a)$ is used to learn a context specific policy via a Dyna-style approach. The single model $p(s',r|s,a)$ is trained following a theoretical analysis based on game theory and an off-policy model-free optimizer based on SAC[14].
In this section, the main structure of Game-MBCD will be discussed. Starting from how the non-stationary environment is modeled. Then, the change detection algorithm enabling a fast response to changes in the environments will be explained. After that, the focus will be on the training of the single context models, and on the optimizations used to enhance the performance of the algorithm.

5.1. Non-stationary environment modeling

As introduced in Section 3, every context $\mathcal{M}_z$ is characterized by a distribution over the next state and reward $p(s',r|s,a)$. To handle more than one context, each model $p(s',r|s,a)$ is added to a mixture model M. This mixture model works as a library of encountered environments. The objective is to recollect a model $p(\cdot|\cdot)$ and the relative policy π_{ψ_z} when the corresponding context is recognized. Otherwise, if a new environment is encountered, a new model will be added to the mixture model M, to avoid the phenomenon of catastrophic forgetting. The mechanism enabling the recognition of the various contexts will be explained in Section 5.2.
Game-MBCD starts from the assumption that every single model $p(\cdot|\cdot)$ represents a multivariate Gaussian distribution modeling the true dynamic and reward function of context $\mathcal{M}_z$. This approach is used in various RL related works [1, 5, 15]. Furthermore, in the work of Chua et al. [5], an analysis on an ensemble of bootstrapped probabilistic neural networks is performed. From the analysis emerges that this methodology is capable of handling two phenomena afflicting the extraction of useful models: aleatoric and epistemic uncertainty.
Aleatoric uncertainty arises from every stochastic model, which is inherently afflicted by noise. This

type of uncertainty is modeled through the usage of $p(\cdot|\cdot)$ as a Gaussian distribution. The distribution models the dynamic and reward through its means, but also the uncertainty on the prediction through its variance. This gives the algorithm a measure of the uncertainty relative to the model.
Epistemic uncertainty arises from the lack of sufficient data, and consequently the inability to describe unambiguously a system. In the utopistic setting where an infinite amount of data is available, this type of uncertainty would be zero, but obviously, this requirement cannot be met. Handling this type of uncertainty is crucial to make a good prediction in regions of the state space which are less explored. To capture the epistemic uncertainty it is useful to use an ensemble of models.
As in MBCD [1], each model $p_{\theta_z}(s', r|s, a)$ is represented by a bootstrap ensemble of probabilistic neural networks parametrized by θ_z, whose outputs describe a multivariate Gaussian distribution with diagonal covariance matrix. The n^{th} network of the ensemble can be represented as

$$p^n_{\theta_z}(s', r|s, a) = \mathcal{N}(\mu^n_{\theta_z}(s, a), \Sigma^n_{\theta_z}(s, a)). \tag{1}$$

To get the output distribution of the ensemble, the distribution of the elements in the ensemble should be combined in a single distribution. Following the work done by Lakshminarayanan et al. [17], networks outputs can be combined as follow:

$$p_{\theta_z}(s', r|s, a) = \mathcal{N}(\mu^*_{\theta_z}(s, a), \Sigma^*_{\theta_z}(s, a)), \tag{2}$$

$$\mu^*_{\theta_z}(s, a) = N^{-1}\sum_{n=1}^{N}(\mu^n_{\theta_z}(s, a)), \tag{3}$$

$$\Sigma^*_{\theta_z}(s, a) = N^{-1}\sum_{n=1}^{N}(diag(\Sigma^n_{\theta_z}(s, a)) + \mu^2_{\theta^n_z}(s, a)) - \mu^2_*(s, a). \tag{4}$$

To train each model $p^n_{\theta_z}(s', r|s, a)$ the loss selected is the negative log-likelihood.

$$\ell_p(\theta^n_z, D) = \mathbb{E}_{s',r,s,a\sim D}[-log\, p_{\theta^n_z}(s', r|s, a)], \tag{5}$$

where D represents the memory of the agent. This set contains the transition acquired by an agent in a given context.
In Game-MBCD, the dynamics model is used for two purposes: a) to generate simulated rollouts used to train the policy in a Dyna-style approach and b) to create the mixture model M for context recollection. These two objectives have contrasting requirements. For this reason, it is useful to use two different networks. One for rollout generation (p_{θ_z}) and the other for context recognition ($p^C_{\theta_z}$) trained in different ways. In particular, they differ on the definition of set D. For p_{θ_z} only the last $\mathcal{G}$ most recent samples are used. For $p^C_{\theta_z}$ all the data acquired in a given context are used. A more detailed explanation of the differences will be provided in Section 5.3.

5.2. Change detection algorithm

To fully use the mixture model, Game-MBCD needs an online change point detection algorithm(CPD)[2]. CPDs have been applied with success to many fields like financial markets[6], aerospace[8] and biomedical applications[28]. In general, CPDs are designed to detect when the underlying parameters of a stochastic process change.
In the case of Game-MBCD, the selected CPD must be able to detect changes, as well as recognize in which environment the agent is operating. More formally, the online CPD algorithm task is to output a precise estimate Ω of the true change point $\mathcal{C}$. Ω, being a random variable, will have some

inherent noise caused by the missing information about the prior of the change point sequence and by the stochasticity of the MDPs where the agent acts.
Following the work in [1], it is considered as an example an abrupt change between two contexts $\mathcal{M}_0$ and $\mathcal{M}_1$. This example consider the case where both contexts have already been encountered. The change point between $\mathcal{M}_0$ and $\mathcal{M}_1$ is identified by $\mathcal{C}_1$. After the change point, the transition and reward functions parameters go from θ_0 to θ_1. Effectively after the change point $\mathcal{C}_1$ the model is described by $p_{\theta_1}(s', r|s, a)$. The objective of the CPD algorithm is to detect a change in the parameters θ, which parametrize the transition and reward function.
As studied in the work of Pollak[24], the CPD can follow a minimax formalization. The algorithm tries to minimize the worst-case expected detection delay $\Delta_{worst}(\Omega)$ of the random variable Ω with respect to the true change point $\mathcal{C}$, and tries to restrict the maximum FAR(False Alarm Rate). The worst-case expected delay and the FAR can be defined as:[1]

$$\Delta_{worst}(\Omega) = sup_{c\geq 1}\mathbb{E}[\Omega - \mathcal{C}|\Omega \geq \mathcal{C}, \mathcal{C} = c], \tag{6}$$

$$FAR(\Omega) = \frac{1}{\mathbb{E}[\Omega|\mathcal{C} = \infty]}. \tag{7}$$

From these definition the objective of the CPD algorithm can be defined as:

$$\underset{\Omega}{inf}\ \Delta_{worst}(\Omega)\ subject\,to\ FAR(\Omega) \leq \alpha, \tag{8}$$

where α represents the upper bound of FAR. Continuing with the example, the estimates of θ_0 and θ_1 can be used to calculate recursively the CUSUM statistics W_t. In particular, W_t can be written as:

$$W_t = max(0, W_{t-1} + L_t),\ W_0 = 0, \tag{9}$$

where L_t is the Log-Likelihood Ratio(LLR) and in the case of multivariate Gaussian is defined as:

$$L_t = log\frac{p_{\theta_1}(s', r|s, a)}{p_{\theta_0}(s', r|s, a)} \tag{10}$$

$$= log\frac{(2\pi)^{-\frac{d}{2}}|\Sigma_1|^{-\frac{1}{2}}exp(-\frac{1}{2}(Y_t - \mu_1)\Sigma_1^{-1}(Y_t - \mu_1))}{(2\pi)^{-\frac{d}{2}}|\Sigma_0|^{-\frac{1}{2}}exp(-\frac{1}{2}(Y_t - \mu_0)\Sigma_0^{-1}(Y_t - \mu_0))}, \tag{11}$$

where $Y_t = (s', r)$, $X_t = (s, a)$ and $\mu_i = \mu_{\theta_i}(s, a)$.
In practice, L_t is a measure on how likely the transitions and rewards obtained by the agent are coming from $p_{\theta_1}(s', r|s, a)$ instead of $p_{\theta_0}(s', r|s, a)$. In fact, if $p_{\theta_1}(s', r|s, a) > p_{\theta_0}(s', r|s, a)$ then $L_t > 0$, otherwise $L_t < 0$. If $L_t > 0$, then W_t will increase recursively, otherwise it will remain zero.
To summarize, the value of W_t is proportional to the likelihood of $p_{\theta_1}(s', r|s, a)$ being the current context. The problem now is to find an appropriate threshold for W_t. When W_t goes over a fixed threshold the CPD algorithm notifies a change in the environment.

$$\Omega = min(t \mid W_t > h). \tag{12}$$

In the work of Lorden[18], an analysis on the threshold h is conducted. In particular, to have a $FAR < \alpha$ the threshold should be $h = |log\alpha|$. With this consideration, the first requirement on FAR is respected. Concerning $\Delta_{worst}(\Omega)$ an interesting analysis has been carried by Lai[16]. In this work,

[1]The FAR equation is derived as follow. $FAR(\Omega) = \frac{FP}{FP+TN} = \frac{1}{1+\mathbb{E}[\Omega-1|\mathcal{C}=\infty]}$ where FP is the number of False Positives and TN is the number of True Negatives.

Lai finds an approximation of the delay under the assumption that $h = |log\alpha|$. In particular:

$$\Delta_{worst}(\Omega) \approx \frac{|log\ \alpha|}{\mathcal{D}_{KL}(p_{\theta_1}||p_{\theta_0})}\ as\ \alpha \to 0. \tag{13}$$

For now, all the consideration were made on an example with two contexts. Now lets consider the case were the mixture models contains K entries.
Following [1], at each timestep t, the current context model is compared with every entry in the mixture model plus an additional entry representing a new, unseen context. In particular, a CUSUM statistic will be calculated for every model k

$$L_{k,t} = log\frac{p_{\theta_k}(s', r|s, a)}{p_{\theta_{z_t}}(s', r|s, a)} \tag{14}$$

$$W_{k,t} = max\Big(0, W_{k,t-1} + L_{k,t}\Big),\ \forall k \in [1, K] \cup [new], \tag{15}$$

where z_t represents the index of the model used in the mixture model. $W_{new,t}$ can be interpreted as a measure on the likelihood that a completely new environment is encountered. This measure is based on whether the likelihood of all known contexts k is smaller than $p_{\theta_{new}}$. The likelihood of the new model can be written as:

$$\hat{Y}_t = Y_t + \delta\ diag(\Sigma)_{\theta_{z_t}}(X_t)) \tag{16}$$

$$p_{\theta_{new}}(Y_t, X_t) = \mathcal{N}(\hat{Y}_t, \Sigma_{\theta_{z_t}}(X_t)). \tag{17}$$

In particular, δ indicates the smallest variation in the mean of p that is worth detecting.
Finally, after all CUSUM statistics $W_{k,t}$ have been calculated, the most likely context is selected as follow:

$$z_t = \begin{cases} argmax_k\ W_{k,t}, & if\ \exists k \in [1, K] \cup [new]\ s.t.\ W_{k,t} > h, \\ z_{t-1}, & otherwise. \end{cases} \tag{18}$$

When a new unseen context is detected ($z_t \neq z_{t+1}$), the new context is initialized as follow into the mixture model M. The rollout network $p_{\theta_{z_{t+1}}}$ is initialized as $p_{\theta_{z_t}}$ in order to have continuity in case of drifts. The same happens with the policy $\pi_{\psi_{z_{t+1}}} = \pi_{\psi_{z_t}}$. The context recognition network $p^C_{\theta_{z_{t+1}}}$ is initialized randomly.

5.3. Environment model splitting

The two models $p^C_{\theta_z}$ and p_{θ_z} are both a representation of context z, but they are trained in two different ways. Both networks are trained with the same frequency, in particular every F time-steps both networks are updated. As anticipated in Section 5.1, the training differs on the subset of data used for training.
In the case of the context recognition network $p^C_{\theta_z}$, the objective is to create the most exhaustive representation of the current context, covering as much state space as possible. For this reason, it is advisable to use all available samples[1, 15] originated from context z, to cover all the states where the agent has been to.
In the case of the rollout generator network p_{θ_z}, the requirements are almost complementary. The network has to adapt as fast as possible to changes in the environment and should create a good representation of the states that are visited the most during the normal behavior of the agent. Avoiding to model states that are rarely encountered or unreachable. These considerations are even more important in the case of drifting contexts.
When the environment switches from a stationary context M_i, to a drifting one M_k, the algorithm will

Algorithm 1: Change point detection algorithm

```
Input      : Non-stationary environment $E$, threshold $h$
Initialize: $z_0 \leftarrow 1$; $K \leftarrow 1$; $W_{z_0,0} \leftarrow 0$; $W_{new,0} \leftarrow 0$; $N$
            Model CUSUM $p_{\theta^C_{z_0}}$; Policy $\pi_{\Psi_{z_0}}$; Datasets $D_{z_0}$
$M \leftarrow \{(p_{\theta_{z_0}}, p_{\theta^C_{z_0}})\}$
for $t = 0, \dots, \infty$ do
    Execute $a_t \sim \pi_{\Psi_{z_t}}$, observe $s_{t+1}$, $r_t$
    $W_{k,t} \leftarrow max\left(0, W_{k,t-1} + log\frac{p_{\theta^C_k}(Y_t|X_t)}{p_{\theta^C_{z_t}}(Y_t|X_t)}\right), \forall k \in [1,K] \cup [new]$
    $z_t \leftarrow \begin{cases} argmax_k(W_{k,t}) & \text{if } \exists k \in [1,K] \cup [new] \text{ s.t. } W_{k,t} > h \\ z_{t-1} & \text{otherwise} \end{cases}$
    // Context change
    if $z_t \neq z_{t-1}$ then
        $W_k \leftarrow 0, \forall k \in [1,K]$
        $D_{model} \leftarrow \{\}$
        // New context
        if $z_t = new$ then
            $K \leftarrow K + 1$; $z_t \leftarrow K$
            Initialize $D_{z_t}$
            $p_{\theta_{z_t}} \leftarrow p_{\theta_{z_{t-1}}}$
            $\pi_{\Psi_{z_t}} \leftarrow \pi_{\Psi_{z_{t-1}}}$
            $M \leftarrow M \cup \{p_{\theta_{z_t}}, p_{\theta^C_{z_t}}\}$
        end
    end
    $D_{z_t} \leftarrow D_{z_t} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
end
```

have a delay in the detection of the context switch. As shown by Equation 13, the worst detection delay is proportional to the similarity between the context M_i and M_k. This fact highlights the importance of managing adequately this delay period in presence of drifts because it will generally be longer compared to delays originated from abrupt changes. This is due to the smoothness assumption on the transition from a stationary context to a drifting context.

Between the real change point time-step $\mathcal{C}$ and the detection time-step Ω, the transitions and rewards acquired by the agent will come from the drifting environment. This implies that before the time-step Ω, the set D_i will contain transition and rewards coming from both M_i and M_k. Using all the samples in D_z during a drift could lead to some undesired effect:

1. using samples from the past stationary environment could hurt the performance of the model because they are sampled from a context that is different from the current one.
2. the more time it passes between the start of the stationary environment, and the start of the drift, the bigger is the number of samples relative to the stationary part compared to the one originating from the drift.

   In other words, the sample obtained from the drifting environment would contribute far less compared to the data obtained from the previous stationary context. This implies that the learning is not invariant to the duration of the various contexts, because a different ratio between the samples acquired from the stationary and drifting environment, would lead to different training results.

The network p_{θ_z} is trained using only the $\mathcal{G}$ most recent samples in D_z. In particular, the dataset used by p_{θ_z} is called D_z^{Sub} and it is defined as:

$$D_z^{Sub} \leftarrow \{D_{z,i}\}, \quad i \in \{|D_z| - S, \dots, |D_z|\}. \tag{19}$$

By doing so, the network will get a set of training data more representative of the current context, giving the model more resistance to changes. Furthermore, this training data is not dependent on the duration of the various contexts, because the size of the training set is fixed a priori.

This approach has also some advantages during stationary contexts. The most recent samples acquired

will come from an agent that was trained for a longer time. Starting from the assumption that the performance of the agent increases over time, the transition and rewards generated more recently are more meaningful for the training. Especially in the early stage of training when the performance rises faster, the subspace of the state-space explored by the most recent transitions corresponds to states leading to higher performance.
In hindsight, it is very important to use all samples for the context recognition network $p^C_{\theta_z}$. To maintain a good parametrization when the context change to M_k, it is very important to continue training on the data originated from the stationary context M_i.

5.4. Game theoretic policy Dyna-style optimization

Game-MBCD context-specific policies are learned through a Dyna-style approach[11]. This approach consists of using the model of the environment to generate imaginary rollouts, used as training data for some model-free algorithm. One of the advantages of Dyna-style approaches is the sample efficiency compared to their model-free counterpart, still maintaining comparable asymptotic performance. The sample efficiency derives from the ability to generate a virtually infinite amount of training data for policy learning. The quality of the training data is heavily dependent on the performance of the environment modelization.
Dyna-style approaches rely on the cooperation of two components: model learning and policy learning. This cross-dependency has to be addressed properly to construct algorithms capable of extracting the most from the usage of environment models. Blindly training the two components without addressing the influence of one on the other, can penalize the final performance.
To address this co-dependency, Rajeswaran[10] developed a methodology to cast MBRL algorithms into a game-theory framework. In particular, the main idea is to consider the interaction between the model and policy optimizer as a cooperative game. The class of games studied to construct this framework is Stackelberg games.
Stackelberg games are asymmetric sequential games where players make decisions in a pre-specified order. In particular, there is a leader player which takes a decision first, then a follower player which takes a decision, called "response", based on the leader player's move. More formally, if A is the leader player, and B is the follower player the game formulation can be written as follow:

$$\min_{\theta_A} \mathcal{L}_A(\theta_A, \theta^*_B(\theta_A))$$
$$s.t.\ \theta^*_B(\theta_A) = \underset{\tilde{\theta}}{argmin} \mathcal{L}_B(\theta_A, \tilde{\theta}),$$

where $\mathcal{L}_A$ and $\mathcal{L}_B$ are respectively the loss of player A and B. θ_A and θ_B are players' parameterizations. In the case of Game-MBCD the two players are:

1. Policy player: This player is in charge of finding a policy for a given context. It maximizes the reward in the model learned by the model player.
2. Model player: This player tries to find a good representation of the environment. It minimizes the prediction error of data collected by the policy player

Due to the asymmetric nature of Stackelberg games, the game can take two forms, depending on the choice of the leader player. In this work, Game-MBCD considers the variant Policy player As a Leader(PAL). This choice comes from the fact that this version is more easily adaptable to the baseline used, and from the non-stationary analysis done in the work of Rajeswaran[10]. The PAL formulation for the single context z can be described as:

$$\max_{\psi} \big\{ J(\psi, D^{sim})\ s.t.\ D^{sim} \sim p_\theta(\cdot|\cdot) = \underset{\theta}{argmin}\ \ell(\theta, D^{sub}) \big\}, \tag{20}$$

where J is the loss of policy π parameterized by ψ. D^{sim} is the set of simulated rollouts sampled from the learned environment model $p_\theta(\cdot|\cdot)$.
To avoid making the notation too complicated, all the z subscript are omitted from now on, because the discussion now considers only a single context.
In practice, Game-MBCD uses the following procedure:

1. **Policy player move: Policy optimization**. The policy player (leader player) optimizes the policy using the simulated rollouts generated from the environment model.
2. **Policy player move: True data gathering**. The agent will act in the true environment following the newly updated policy π_ψ. By doing so, it will gather a fixed number of new transitions and rewards.
3. **Model player move: Environment model optimization**. This newly acquired data will be used by the model player to update the environment model.
4. **Model player: Simulated rollouts generation**. The newly updated model is then used to generate the simulated rollouts.

Now every point in the enumeration will be discussed to explain how every component works.

Policy optimization One of the positives of using a Dyna-style approach is its modularity in policy learning. This approach can be adapted to virtually every model-free policy learner because it is capable of generating virtual datasets, which can be used by the policy learner. For this work, the Soft Actor Critic(SAC)[14] algorithm has been used as model-free policy learner.
SAC is an off-policy actor-critic RL algorithm based on the maximum entropy reinforcement learning framework. The objective of the framework is to find a good policy that maximizes the obtained reward while maintaining a good level of exploration. Exploration is ensured by entropy maximization.
SAC merges off-policies updates with a stochastic actor-critic formulation which encourages to explore more while dropping off clearly unsuitable policies. Furthermore, the policy can consider multiple actions in situations where those actions are equally performing.
More formally, SAC alternates between a policy evaluation step, which estimates Q-function q_{π_ψ} using the Bellman backup operator, and a policy improvement step. This step optimizes the policy π_ψ by minimizing the expected KL-divergence between the current policy and the exponential of a soft Q-function [14]. The loss used to optimize the policy can be written as:

$$J(\psi, D^{sim}) = \mathbb{E}_{s\sim D^{sim}}\left[\mathbb{E}_{a\sim\pi_\psi}(\beta log(\pi_\psi(a|s))) - q_{\pi_\psi}(s,a)\right]. \tag{21}$$

Following the Dyna-style approach, the data used by SAC are entirely simulated.

Environment model optimization After the agent has gathered new data from the real environment, the D^{sub} set is updated. The model player optimizes the environment model $p_\theta(\cdot|\cdot)$ with D^{sub}. Because the model player is the follower player, the update of the network consists of a response to the policy player. In particular, following the formulation of the Stackelberg game provided by Equation 20, the model player should update the parameter θ till convergence. This type of response is called *best response*, because it minimizes the loss relative to the parameters.
To achieve this, Game-MBCD makes use of an approach similar to [15]. The true data D^{sub} acquired by π_ψ are split in a training set and a validation set sampled uniformly from D^{sub}. The validation set is used to perform early stopping. In particular, the training procedure continues until the validation score decreases. During the training procedure, the algorithm keeps track of the best validation score. If after E epochs the best validation score has not been improved, then the training stops, avoiding overfitting to training data. This early stopping procedure becomes more and more important the less data we have in the training set.

Game-MBCD manages a trade-off on the number of samples $|D^{sub}|$ used for training. The smaller is $|D^{sub}|$ the more the training procedure will be able to follow changes in the environment. This is because the elements of D^{sub} would be very recent, and so they are more representative of the current dynamic. On the other hand, a small training set could be very sensitive to overfitting and to noise in the data gathered. Because of this, the early stopping procedure described before is crucial for a good learning procedure, because it allows Game-MBCD to use smaller training sets and so be more resistant to changes in the environment, still maintaining a good generalization performance.

Simulated rollouts generation As said before, the generation of the simulated rollouts is a crucial part of the algorithm. In [1] the rollouts are generated as follow:

1. From the true transitions set D, the algorithm samples a set of starting states.
2. Using policy π_ψ an action is selected for every starting state.
3. For every action the next state and reward are estimated using the approximated model of the environment $p(\cdot|\cdot)$.
4. The newly sampled transitions are used for policy training

One of the main limitations of this procedure is the inability to generate simulated rollouts longer than one step. To generate longer rollouts, the samples generated in point 3) could be used as the starting states of point 1) in an iterative way. This procedure suffers from the compounding error deriving from the concatenation of the dynamic predictions. This fact limits the planning ability of the agent because the dataset provided does not take into account sequences of transitions and their effects on the future.

To solve this issue, Game-MBCD makes use of the approach developed by Pan[23] called Masked Model-based Actor-Critic(M2AC). The objective of M2AC is to construct a score u which measures the quality of each simulated rollout. The rollouts are ranked based on this score, and only the top-performing rollouts are used for training.

More in detail, the score u is measuring the uncertainty on the prediction of the next state and reward. To do so, M2AC uses how the model $p_\theta(s', r|s, a)$ is constructed. In particular u is a measure of the disagreement of the elements of the ensemble modeling $p_\theta(s', r|s, a)$. Following a One versus the Rest approach, at each rollout simulation iteration, a random network n is chosen from the ensemble. Then, using the Kullback-Leibler divergence, the estimated Gaussian distribution of network n is compared with the distribution estimated by all the other networks in the ensemble. More formally:

$$u_n(s, a) = D_{KL}\big(p^n_\theta(s', r|s, a)||p^{-n}_\theta(s', r|s, a)\big), \tag{22}$$

where $p^{-n}_\theta(s', r|s, a) = \mathcal{N}(\mu^{-n}_\theta(s, a), \Sigma^{-n}_\theta(s, a))$ is the ensemble prediction calculated combining all the element in the ensemble except model n. To combine these elements, an approach similar to equation 3 and 4 is used.

$$\mu^{-n}_\theta(s, a) = \frac{1}{N-1}\sum_{i\neq n}^{N}(\mu^i_\theta(s, a)), \tag{23}$$

$$\Sigma^{-n}_\theta(s, a) = \frac{1}{N-1}\sum_{i\neq n}^{N}(diag(\Sigma^i_\theta(s, a)) + \mu^2_{\theta^i}(s, a)) - \mu^2_{\theta^{-n}}(s, a). \tag{24}$$

To recap the complete rollout generation procedure is:

1. From the true transitions set D, the algorithm samples a set of starting states S.
2. Using policy π_ψ an action is selected for every starting state s_i.
3. Randomly choose a network n in the ensemble constructing $p_\theta(s', r|s, a)$.
4. Sample s'_i and r_i using $p^n_\theta(s', r|s, a)$.

Algorithm 2: Game-theoretic Dyna-style policy optimization

```
Input      : Non-stationary environment $E$, Number of network in ensemble $N$
Initialize: Model $p_\theta$; Model CUSUM $p_{\theta C}$; Policy $\pi_\Psi$; Datasets $D$, $D^{sim}$ and $D^{sub}$
for $t = 0, \dots, \infty$ do
    if $t\ mod\ F = 0$ then
        $D^{Sub} \leftarrow \{D_i\}, \quad i \in \{|D| - M, \dots, |D|\}$
        // Update model till convergence using $\mathcal{D}^{Sub}$. The set contains the $M$ most recent elements of $D$
        $\theta \leftarrow \underset{\theta}{argmin}(\mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D^{Sub}}[-log(p_\theta(s_{t+1}, r_t|s_t, a_t))])$
        // Update model till convergence using all samples in $\mathcal{D}$
        $\theta^C \leftarrow \underset{\theta^C}{argmin}(\mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}[-log(p_\theta(s_{s_t+1}, r_t|s_t, a_t))])$
        $D^{sim} \leftarrow \{\}$
        Randomly sample $B$ states from $D$ with replacement $S \leftarrow \{s_i\}_i^B$
        for $h = 0, \dots, H_{max} - 1$ do
            for $i = 1, \dots, B$ do
                $a_i \leftarrow \pi_\Psi(s_i)$, $s_i \in S$
                Randomly choose $n$ in $[1, N]$
                $(s_i', \tilde{r_i}) \leftarrow p_{\theta^n}(r, s^{'}|s_i, a_i)$
                $u_i \leftarrow D_{KL}[\mathcal{N}(\mu_\theta^n(s_i, a_i), \Sigma_\theta^n(s_i, a_i)) \parallel \mathcal{N}(\mu_\theta^{-n}(s_i, a_i), \Sigma_\theta^{-n}(s_i, a_i))]$
            end
            Rank samples by $u_i$
            Get first $\lfloor wB \rfloor$ samples' indexes, $\{i_j\}_j^{wB}$
            $D^{sim} \leftarrow D^{sim} \cup \{(s_{i_j}, a_{i_j}, \tilde{r_{i_j}} - \alpha u_{i_j}, s_{i_j}^{'})\}$
            $S \leftarrow \{s_i^{'}\}_i^B$
        end
        for $s = 1, \dots, \eta F$ do
            $\Psi \leftarrow \Psi - \lambda_\pi \nabla \mathbb{E}_{s_t \sim D^{sim}}\left[\mathbb{E}_{a_t \sim \pi_\Psi}\left(\beta log(\pi_\Psi(a_t|s_t)) - q_\Psi(s_t, a_t)\right)\right]$
        end
    end
end
```

Algorithm 3: Game-MBCD

```
Input      : Non-stationary environment $E$, Number of network in ensemble $N$
Initialize: Model $p_\theta$; Model CUSUM $p_{\theta C}$; Policy $\pi_\Psi$; Datasets $D$, $D^{sim}$ and $D^{sub}$
for $t = 0, \dots, \infty$ do
    Check for context changes using algorithm 1
    if $t\ mod\ F = 0$ then
        Update policy $\pi$ using the game-theoretic Dyna-style approach of algorithm 2
    end
end
```

5. Calculate score u_i with equation 22.
6. Rank samples acquired using the scores u
7. Get the top $m\%$ samples and add them to the simulated rollout set D^{sim}.
8. Refresh the starting states set with next states s_i', which have been added to D^{sim}
9. Return to point 2. if the desired rollout length has not been reached yet.

The entire policy learning procedure is summarized in the pseudo-code of Algorithm 2. The complete Game-MBCD algorithm can be found in Algorithm 3. Furthermore a complete formulation of the pseudo-code can be found in Appendix A.

6. Experiments

Game-MBCD has been evaluated on various non-stationary environments to test its resistance to various types of non-stationarity. In particular, the algorithm was tested on the continuous-state, continuous-action half-cheetah environment taken from OpenAI Gym [3] using physics engine MuJoCo[29]. Half-Cheetah agent is made up of 7 rigid links (1 for torso, 3 for front leg, and 3 for the back leg), connected by 6 joints positioned in the legs. Each joint is also an actuator. To summarize the agent characteristics:

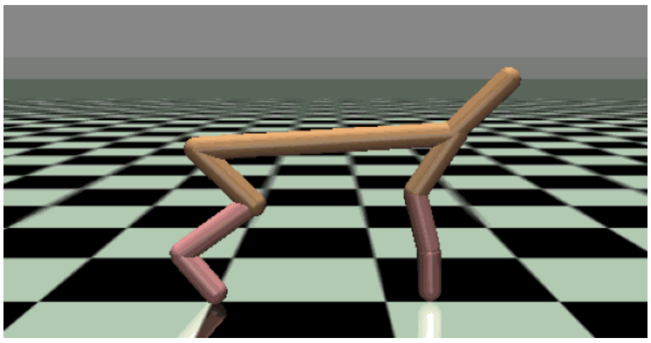


Figure 1: Half-cheetah agent structure in MuJoCo[29] physics simulator

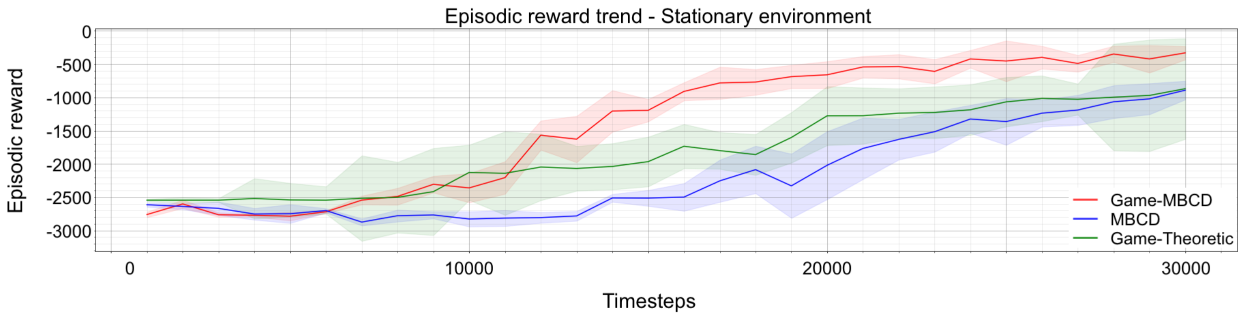


Figure 2: Half-cheetah episodic reward in a stationary environment

- Actions: $a \in [-1, 1]^6$. The action vector a, represents the torque applied to each joint actuator in percentage in a range from -100% to $+100\%$.
- Observation: $s \in \mathbb{R}^{17}$. The observed state vector s includes the position and speed of the agent, and the position and angular velocity of each joint
- Reward: $r = -|v_x - v_{target}| - 0.1 \parallel a \parallel^2$. The objective of this agent is to run at a target velocity, minimizing the energy consumed to activate the actuators.

The representation of the agent can be seen if Figure 1.
Game-MBCD was designed to extend the capabilities of the baseline used, in the scenario of unseen contexts. The experiments were designed to test how quickly the agent can adapt to these situations. The non-stationarity considered in the experiments involves both changes in the dynamics and changes in the goals, to study the effects of changes on the transition function and the reward function.
In particular:

- Dynamics changes: two actuators of the back leg can malfunction. In particular, the agent actions relative to these two joints are rescaled based on a factor.
- Goal changes: the target velocity can change during the execution.

These two types of changes can be applied to the environments in two ways:

- Abrupt changes: either the back leg actuators are completely disabled or the target velocity changes with a jump
- Drifts: either the back leg actuators go linearly from completely functional to completely disabled in the span of a context or the velocity changes linearly from v_{start} to v_{end}

Stationary context experiments The first experiment aims to study the sample efficiency of Game-MBCD compared to model-based and hybrid model-based/model-free state-of-the-art algorithms.
In the experiment, a single stationary context is considered. In particular, the task of the half-cheetah agent is to run to a pre-specified velocity. From Figure 2, it is evident that Game-MBCD outperforms

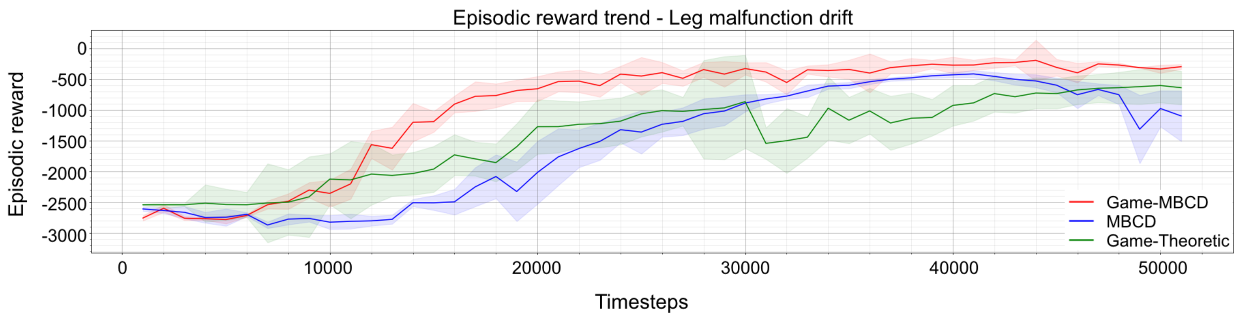


Figure 3: Half-Cheetah episode reward in an environment composed of two contexts. From 0 to 30K timesteps the context is stationary with no anomalies. From 30K to 50K the context changes to a dynamics drift applied to the back leg of the half-cheetah agent

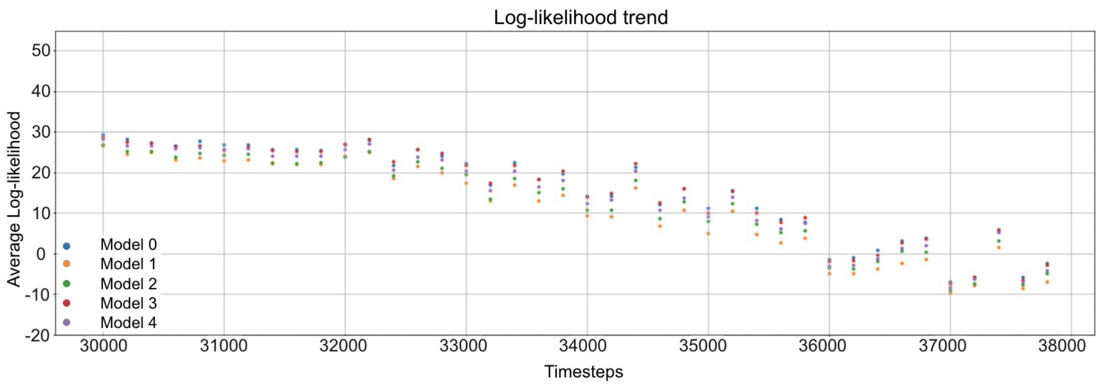
MBCD[1] and the game-theoretic[10] approach both from a sample efficiency and asymptotic performance point of view. Furthermore, it can be seen that Game-MBCD has an overall variance that is lower over the execution period.

Drifting contexts experiments The second set of experiments studies the behavior of Game-MBCD in case drifts in the environment. In the first case, the environment consists of two contexts. The first consists of a stationary contest which lasts 30K timesteps. After that, the environment switches to a drifting context, where the back leg of the half-cheetah agent is linearly disabled. In the span of 20K timesteps, the actuators go from fully functional to completely disabled. In Figure 3, it can be seen how MBCD is not able to follow the changes in the environment. The performance drops as the drift unfolds, and the back leg is more and more inactive. On the other hand, the two methods based on game theory can follow the changes without having any drop in performance. Furthermore, Game-MBCD has overall a higher performance compared to the two other methodologies. In the stationary part, Game-MBCD has a higher sample efficiency, and during the drift, it has a higher asymptotic performance.
As explained in Section 5.4, Dyna-style approaches rely on a good environment model to generate simulated rollouts. An experiment has been conducted to analyze how well the environment model fits the real model, and which are the differences between Game-MBCD and MBCD in the case of drifts. To measure the fit, the experiment relies on the measure of the log-likelihood. In particular, all the samples acquired from the real environment are saved in a buffer. After that, the data are divided into batches of equal size. In this case, every batch consists of 200 samples. The set B of batches is defined as:

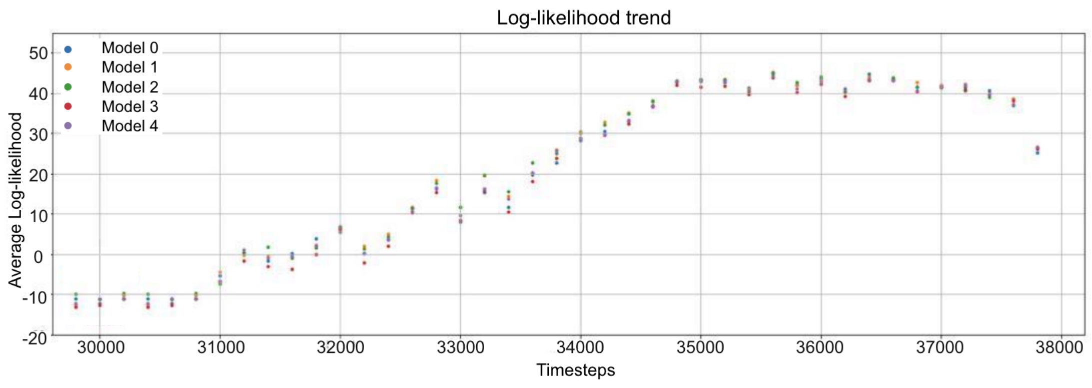
$$B = \{B_0, \dots, B_N\}$$

$$B_i = \{(s_{i+1}, r_i) \mid i \in [t - 200i, t - 200(i+1)]\}$$

where t is the current timestep.
Each sample consists of a transition and a reward at a given timestep. The context parametrization at the current timestep t defines a $p_{\theta_t}(s', r|s, a)$ which approximate the real environment. The Gaussian distribution defined by $p_{\theta_t}(s', r|s, a)$ is used to calculate the mean log-likelihood over every data batch in B. By doing so, the fit of the current parameterization of the environment can be tested on the past transitions and rewards. This is helpful in the case of drifts because it can estimate the fit of the current model on the past, and so on different stages of the drift. In this specific case, the experiment considers the same drift of Figure 3, where at the $30K^{th}$ timestep the leg actuator drift starts.
In Figure 4, it can be seen the trends of the likelihood over 8K timesteps, from 30K to 38K timestep. Doing so, it can be seen the evolution of the log-likelihood during the drift. Each dot corresponds to the mean log-likelihood of a data batch evaluated on a model of the ensemble describing the current



(a) MBCD



(b) Game-MBCD

Figure 4: Log-likelihood trend comparison between (a)MBCD and (b)Game-MBCD. Each dot represents the mean likelihood of a given data batch, evaluated on the current parameterization of a single model in the ensemble

context.
From the trends in Figure 4, it can be appreciated the difference between MBCD and Game-MBCD. For MBCD, the log-likelihood follows a downward trend, which indicates that the model is better at describing the environment in the past, more precisely at the start of the drift. Instead, the likelihood corresponding to the most recent transitions and rewards has a poor performance. This is very bad behavior because it is exactly the opposite of what the algorithm should do in this situation.
During a drift, the agent should adapt and follow the changes in the environment, to avoid any performance drop. A not well performing context model will hinder the policy learner performance, due to the decrease in quality of the generation of simulated rollouts.
The opposite situation can be seen in the log-likelihood plot corresponding to Game-MBCD. In this case, the trend follows a shape similar to a sigmoid.
At the beginning of the plot, due to the drift, the batches are sampled from an environment that is different from the current one. Traversing the plot from left to right, the batches will be sampled from an environment more and more similar to the current one.
It can be seen how the log-likelihood increases as the batches are sampled from more recent timesteps. This means that the current parameterization can fit very well the current context, and can produce a good rollout simulation for policy learning. As older data are considered the log-likelihood decreases due to the augmenting differences between the present and past dynamics.
The last experiment involving drifts has been performed by changing the target velocity gradually. In particular, the environment is composed of a starting stationary context lasting 30K timesteps, followed by a drifting context lasting 30K timesteps, where the target velocity is linearly increased.

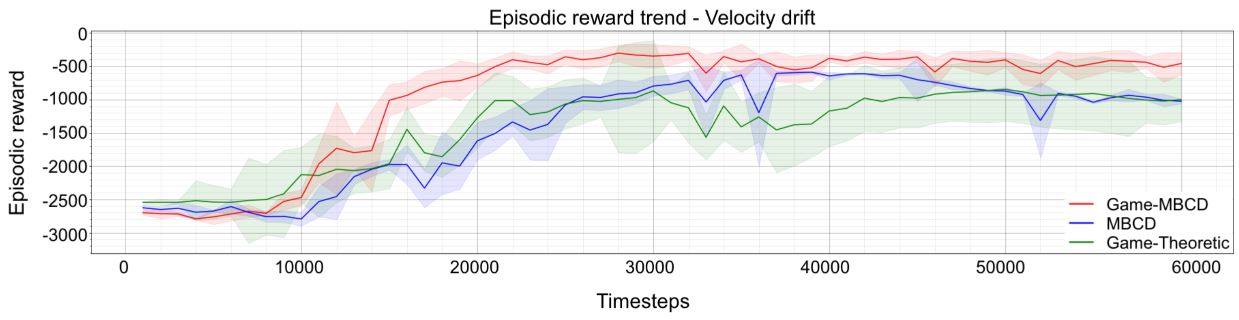


Figure 5: Half-Cheetah episode reward in an environment composed of two contexts. From 0 to 30K timesteps the context is stationary with no anomalies. From 30K to 60K the environment switches to a drifting context with non-stationarity applied to the reward function. The goal drift is applied to the target velocity of the half-cheetah agent

Once more from Figure 5, it is clear how Game-MBCD is capable of maintaining a constant high episodic reward performance, despite the non-stationarity of the environment. Instead, MBCD and the game-theoretic framework have a performance drop in the presence of velocity drifts. MBCD presents a drop late in the drifting context because of its structure in model training. The game-theoretic framework following the Policy As a Leader(PAL) structure has a lower performance when used in environments with drift in the reward function. This is a result comparable with the analysis done in [10].

Abrupt changes experiments The third set of experiments studies the behavior of Game-MBCD in an environment characterized by abrupt changes.
In the first case, the non-stationarity consists in disabling actuators of the foot and knee of the back leg of the half-cheetah agent. The environment is composed as follows: the first context is a stationary environment with no anomalies. This context lasts 30K timesteps. After that, the current context changes to a context with the dynamic anomaly described before. This last context lasts for 30K timesteps.
From the plot in Figure 6, it can be seen that at the contexts change point, all algorithms have a performance drop due to the context change. The drop in performance of Game-MBCD is lower compared to the other algorithm studied. Game-MBCD is also better at recovering, it can be seen that, after the performance drop, the algorithm outperforms both MBCD and the game-theoretic framework in the number of timesteps needed to regain a near asymptotic performance. It can be seen how the game-theoretic framework is afflicted by some sort of negative transfer because the asymptotic performance post-change is lower than the one before the change. This effect is not seen in both Game-MBCD and MBCD. Furthermore, the variance of the game-theoretic approach is larger compared to the variance of Game-MBCD and MBCD.
The last experiment analyses the performance of the agent in the presence of abrupt changes in the reward function. In particular, the target velocity is changed abruptly.
A similar behavior to the one examined in the previous experiments can be seen. Game-MBCD is capable of handling the abrupt change also in this case. Similarly, the game-theoretic approach suffers once more of negative transfer. These results can be seen in Figure 7.

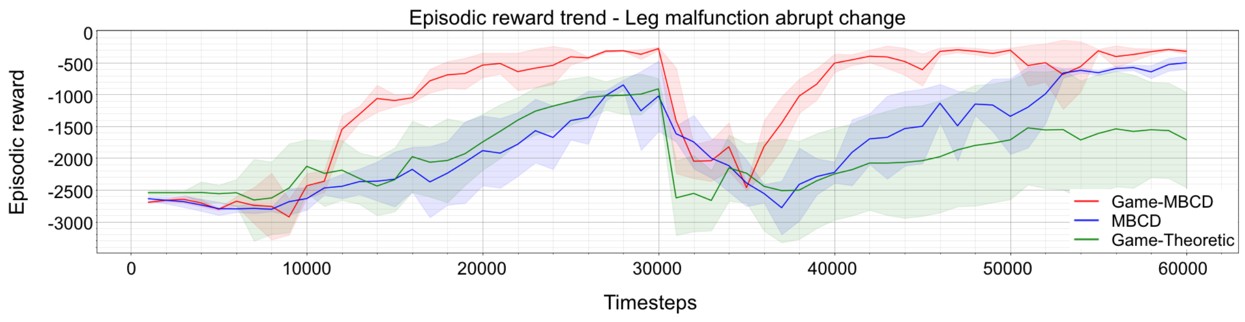


Figure 6: Half-Cheetah episodic reward in an environment composed of two contexts. From 0 to 30K timesteps the context is stationary with no anomalies. From 30K to 60K the context changes and the back leg actuator of foot and knee are completely disabled

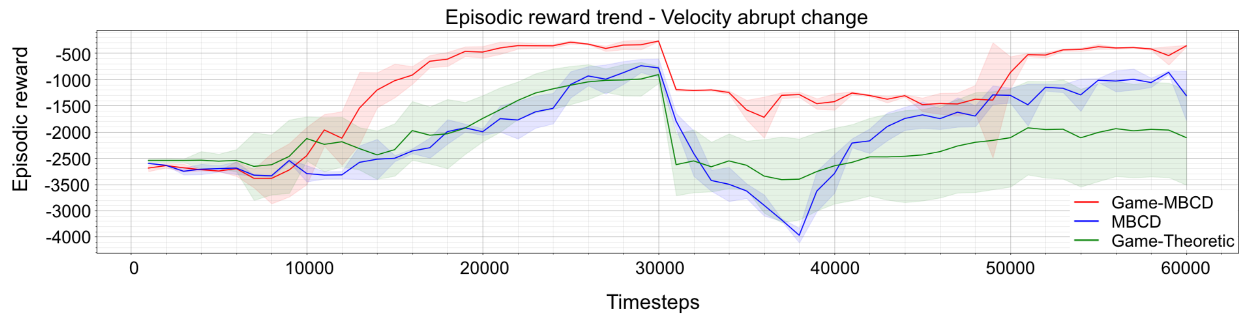


Figure 7: Half-Cheetah episodic reward in an environment composed of two contexts. From 0 to 30K timesteps the context is stationary with no anomalies. From 30K to 60K the context changes and the target velocity increases abruptly

7. Conclusions

This work introduces Game-MBCD, a hybrid model-based/model-free approach based on game theory, capable of handling non-stationary environments affected by both abrupt changes and drifts.
This work explores the effects of applying game theory to hybrid RL algorithms. In particular, its usage enhances the algorithm performance in presence of drift in the transition function or reward function, and its sample efficiency when new unseen contexts are encountered.
Given the extreme importance of the environment model to construct a dataset of simulated data for policy learning, this work analyses some approaches to improve this methodology. In particular, the approach based on KL-divergence developed by [23] has been used to create a score to evaluate and filter the best simulated rollouts.
From the experiments conducted, Game-MBCD is capable of handling both abrupt changes and drift in the environment associated with the transition and reward functions. Furthermore, its sample efficiency surpasses recent MBRL algorithms like [10, 15].
This work can be expanded by implementing a mechanism to transfer the knowledge between contexts, to further reduce the adaptation type in unseen contexts. Some interesting research directions include meta-learning and context encoding approaches to reuse the knowledge acquired in the past to make learning faster.
Another possible extension of this work could explore the integration of a change detection algorithm capable of classifying the ongoing changes. In particular, with a classifier, it would be possible to implement custom behaviors in case of drift or abrupt changes.

References

[1] Lucas N Alegre and Ana L C Bazzan. Minimum-delay adaptation in non-stationary reinforcement learning via online high-confidence change-point detection. 2021.

[2] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection, 2017.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Yash Chandak, Georgios Theocharous, Shiv Shankar, Martha White, Sridhar Mahadevan, and Philip S. Thomas. Optimizing for the future in non-stationary MDPs. 2020.

[5] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

[6] Banerjee et al. Change-point analysis in financial networks, 2019.

[7] Brenden M. Lake et. al. Human-level concept learning through probabilistic program induction. *Science 350, 6266*, page 1332–1338, 2015.

[8] Liao et al. Structural damage detection and localization with unknown postdamage feature distribution using sequential change-point detection method, 2018.

[9] R. S. Sutton et al. Policy gradient methods for reinforcement learning with function approximation. page 1057–1063, 1999.

[10] Rajeswaran et al. A game theoretic framework for model based reinforcement learning. 2020.

[11] Sutton et al. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. page 216–224, 1990.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. 2017.

[13] Haotian Fu, Hongyao Tang, Jianye Hao, Chen Chen, Xidong Feng, Dong Li, and Wulong Liu. Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. 2020.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[15] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2021.

[16] Tze Leung Lai. Information bounds and quick detection of parameter changes in stochastic systems. pages 2917–2929, 1998.

[17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017.

[18] Gary Lorden. Procedures for reacting to a change in distribution. page 1897–1908, 1971.

[19] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. 2020.

[20] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. 2019.

[21] S et. al. Padakandla. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence 50*, page 3590–3606, 2020.

[22] E. S. Page. Continuous inspection schemes. page 100–115, 1954.

[23] Feiyang Pan, Jia He, Dandan Tu, and Qing He. Trust the model when it is confident: Masked model-based actor-critic, 2020.

[24] Moshe Pollak. Optimal detection of a change in distribution. page 206–227, 1985.

[25] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning, 2021.

[26] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. 2019.

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

[28] Thabani Sibanda and Nokuthaba Sibanda. The cusum chart method as a tool for continuous monitoring of clinical outcomes using routinely collected data. 2007.

[29] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control, 2012.

[30] Lilian Weng. Meta reinforcement learning, https://lilianweng.github.io/2019/06/23/meta-reinforcement-learning.html, 2019.

A. Complete pseudo-code

In this section a more detailed and complete pseudo-code is presented to better understand Game-MBCD algorithm.

Algorithm 4: Game-MBCD

```
Input      : Non-stationary environment $E$, Number of network in ensemble $N$
Initialize: Model $p_\theta$; Model CUSUM $p_{\theta^C}$; Policy $\pi_\Psi$; Datasets $D$, $D^{sim}$ and $D^{sub}$
$M \leftarrow \{(p_{\theta_{z_0}}, p_{\theta^C_{z_0}})\}$
for $t = 0, \dots, \infty$ do
    Execute $a_t \sim \pi_{\Psi_{z_t}}$, observe $s_{t+1}$, $r_t$
    $W_{k,t} \leftarrow max\left(0, W_{k,t-1} + log\frac{p_{\theta^C_k}(Y_t|X_t)}{p_{\theta^C_{z_t}}(Y_t|X_t)}\right), \forall k \in [1, K] \cup [new]$
    $z_t \leftarrow \begin{cases} argmax_k(W_{k,t}) & \text{if } \exists k \in [1,K] \cup [new] \text{ s.t. } W_{k,t} > h \\ z_{t-1} & \text{otherwise} \end{cases}$
    // Context change
    if $z_t \neq z_{t-1}$ then
        $W_k \leftarrow 0, \forall k \in [1, K]$
        $D_{model} \leftarrow \{\}$
        // New context
        if $z_t = new$ then
            $K \leftarrow K + 1$; $z_t \leftarrow K$
            Initialize $D_{z_t}$
            $p_{\theta_{z_t}} \leftarrow p_{\theta_{z_t - 1}}$
            $\pi_{\Psi_{z_t}} \leftarrow \pi_{\Psi_{z_t - 1}}$
            $M \leftarrow M \cup \{p_{\theta_{z_t}}, p_{\theta^C_{z_t}}\}$
        end
    end
    $D_{z_t} \leftarrow D_{z_t} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    if $t \, mod \, F = 0$ then
        $D^{Sub}_{z_t} \leftarrow \{D_{z_t,i}\}, \quad i \in \{|D_{z_t}| - M, \dots, |D_{z_t}|\}$
        // Update model till convergence using $\mathcal{D}^{Sub}_{z_t}$. The set contains the $\mathcal{G}$ most recent elements of $D_{z_t}$
        $\theta_{z_t} \leftarrow \underset{\theta_{z_t}}{argmin}\big(\mathbb{E}_{(s_t,a_t,r_t,s_{t+1}) \sim D^{Sub}}[-log(p_{\theta_{z_t}}(s_{t+1}, r_t|s_t, a_t))]\big)$
        // Update model till convergence using all samples in $\mathcal{D}_{\ddagger_\sqcup}$
        $\theta^C_{z_t} \leftarrow \underset{\theta^C_{z_t}}{argmin}\big(\mathbb{E}_{(s_t,a_t,r_t,s_{t+1}) \sim D}[-log(p_{\theta_{z_t}}(s_{s_t+1}, r_t|s_t, a_t))]\big)$
        $D^{sim} \leftarrow \{\}$
        Randomly sample $B$ states from $D_{z_t}$ with replacement $S \leftarrow \{s_i\}^B_i$
        for $h = 0, \dots, H_{max} - 1$ do
            for $i = 1, \dots, B$ do
                $a_i \leftarrow \pi_{\Psi_{z_t}}(s_i)$, $s_i \in S$
                Randomly choose $n$ in $[1, N]$
                $(s'_i, \tilde{r_i}) \leftarrow p_{\theta^n_{z_t}}(r, s'|s_i, a_i)$
                $u_i \leftarrow D_{KL}[\mathcal{N}(\mu^n_{\theta_{z_t}}(s_i, a_i), \Sigma^n_{\theta_{z_t}}(s_i, a_i)) \parallel \mathcal{N}(\mu^{-n}_{\theta_{z_t}}(s_i, a_i), \Sigma^{-n}_{\theta_{z_t}}(s_i, a_i))]$
            end
            Rank samples by $u_i$
            Get first $\lfloor wB \rfloor$ samples' indexes, $\{i_j\}^{wB}_j$
            $D^{sim} \leftarrow D^{sim} \cup \{(s_{i_j}, a_{i_j}, \tilde{r_{i_j}} - \alpha u_{i_j}, s'_{i_j})\}$
            $S \leftarrow \{s'_i\}^B_i$
        end
        for $s = 1, \dots, \eta F$ do
            $\Psi_{z_t} \leftarrow \Psi_{z_t} - \lambda_\pi \nabla \mathbb{E}_{s_t \sim D^{sim}}\left[\mathbb{E}_{a_t \sim \pi_{\Psi_{z_t}}}\left(\beta log(\pi_{\Psi_{z_t}}(a_t|s_t)) - q_{\Psi_{z_t}}(s_t, a_t)\right)\right]$
        end
    end
end
```

B. Learning hyper-parameters

In this section the major hyper-parameters used for training are reported in the following table.

Symbol	Value	Meaning
N	5	Number of networks composing the environment model ensemble
Dynamics network architecture	4 layers 200 neurons	Structure of the network which generates simulated rollouts
Dynamics learning rate	0.001	
B	100000	Number of simulate rollouts generated
H_{max}	10	Max length of a simulated rollout
w	0.5	Percentage of best performing rollouts used for learning
$\|D^{sim}\|$	100000	Size of the buffer containing all simulated rollouts
α	0.001	Reward rescaling factor for simulated rollouts
M	4096	Size of the real transitions dataset used for model training
Policy network architecture	2 layers 256 neurons	Structure of the network used for policy learning
λ_π	0.0003	Policy learning rate
γ	0.99	Discount factor
h	100	CUSUM threshold
F	$10 \rightarrow 500$	Policy training period
η	$20 \rightarrow 2$	Number of gradient step multiplier

$A \rightarrow B$ means that the particular value transition from A to B during training.

C. Algorithm scheme

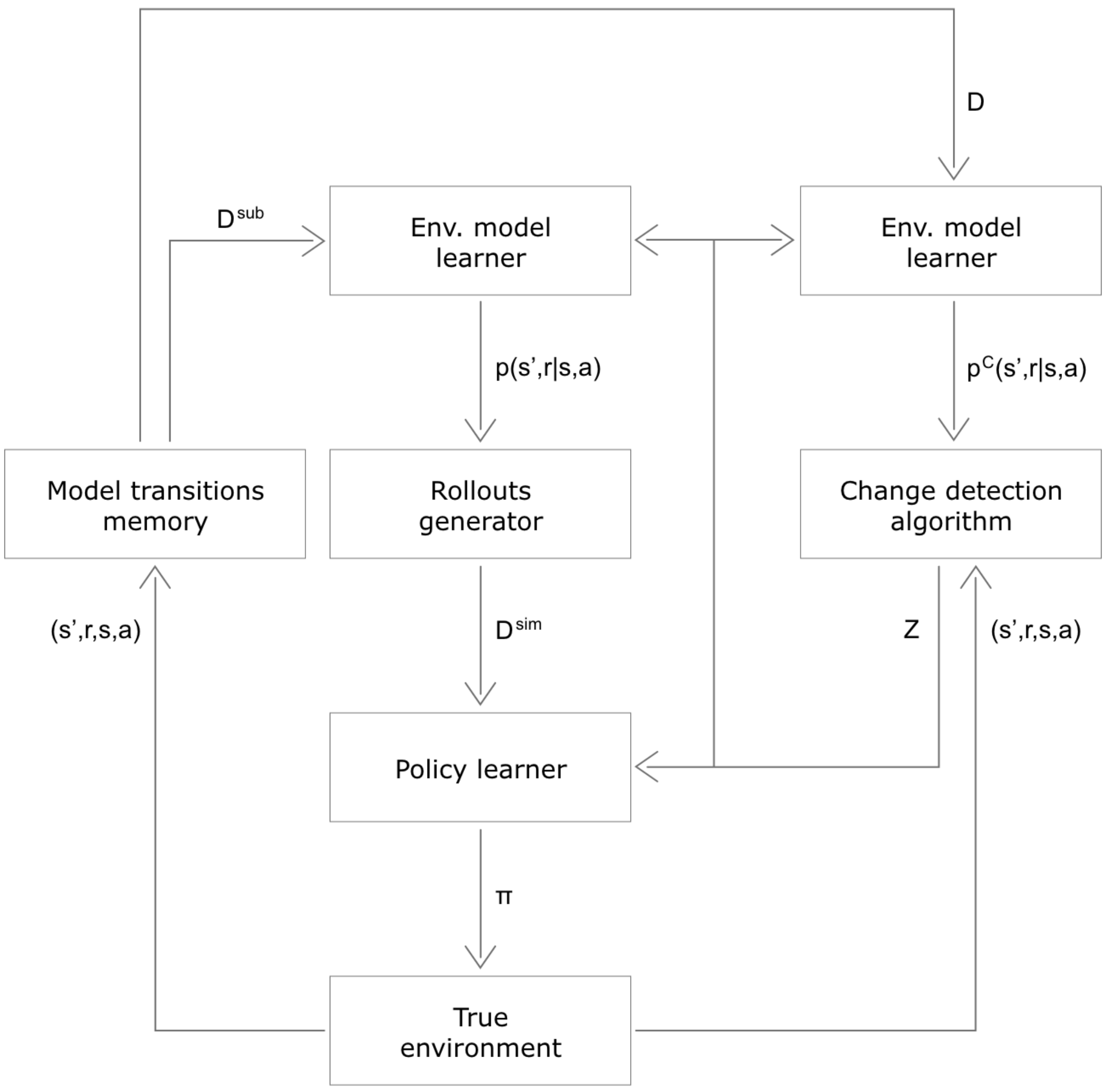


Figure 8: Scheme representing Game-MBCD structure

Abstract in lingua italiana

Gli ambienti non stazionari sono scenari impegnativi per gli algoritmi di Reinforcement Learning, a causa della natura mutevole delle funzioni di transizione e di ricompensa. Lo scenario analizzato da questo lavoro, considera una infinita sequenza casuale di Markov Decision Processes (MDP), ognuno dei quali è campionato da una distribuzione non conosciuta. Per considerare l'impostazione più realistica possibile, l'algoritmo non fa ipotesi sull'esistenza di una fase di pre-addestramento o sulla conoscenza a priori del numero, o dei confini tra i contesti. Questo lavoro introduce Game-MBCD, un approccio ibrido model-based/model-free basato sulla teoria dei giochi, capace di gestire ambienti non stazionari affetti sia da cambiamenti bruschi che da derive. Game-MBCD non richiede una fase di pre-addestramento. In particolare, uno degli obiettivi di questo lavoro è migliorare le prestazioni dello stato dell'arte quando si incontra un nuovo contesto mai visto. La politica per ogni contesto viene estratta con una procedura basata sulla teoria dei giochi, che tiene conto della codipendenza tra la modellazione dell'ambiente e l'ottimizzazione della politica negli algoritmi RL in stile Dyna. Inoltre, il punto di partenza utilizzato per lo sviluppo dell'algoritmo è stato arricchito con un approccio basato sulla divergenza di KL, per migliorare la qualità delle sequenze di dati simulati, utilizzati per l'addestramento delle politiche. Gli esperimenti condotti mostrano che Game-MBCD è più resistente alle varie classi di ambienti non stazionari, rispetto agli algoritmi model-based e agli algoritmi di RL non stazionari dello stato dell'arte.

Parole chiave: Apprendimento per rinforzo, Ambienti non-stazionari, rilevamento dei cambiamenti, Derive, Cambiamenti improvvisi

Acknowledgements

I want to express my gratitude to prof. Roveri, prof. Restelli and PhD Canonaco for all the help they have given me to complete this work. I would like to deeply thank everyone who has been there for me throughout my academic journey. My parents, who were always supportive, and gave me the possibility to study what I love. My girlfriend, for being my reference point during all these years and for giving me the strength to complete this journey. I also want to thank all the amazing colleagues I've met for making my college journey unforgettable.