Sorbonne Université
Master ANDROIDE

TD-MPC

UE de projet M1

Erisa Kohansal – Sama Satariyan – Kim Saïdi – Yanis Djermouni

2024

Table des matières

Chapitre 1

Introduction

This project is part of an IT project unit of the ANDROIDE master's program. Its purpose is to train us in carrying out a team project from start to finish, from analyzing the subject to software development, as well as experimenting with the work done. Our project topic consists in implementing and evaluating a reinforcement learning algorithm, TD-MPC, within a specific library, BBRL. This project was supervised by Mr. Olivier Sigaud.
The Temporal Difference Model Predictive Control (TD-MPC) algorithm presents notable advancements over the current state-of-the-art in reinforcement learning. As a model-based algorithm, it can learn with a limited number of training samples. It effectively integrates temporal difference methods from reinforcement learning with optimal action sequence search methods, making it highly efficient.
The project's goal is to develop a TD-MPC code using a reinforcement learning library created by the supervisor, and then to test the implementation in various environments commonly referenced in the literature. The github link of the project.

Chapitre 2

État de l'art

In this section, we explain some of the different concepts discussed in the article.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of artificial intelligence where an autonomous agent (it's a being that can take actions, it can be the algorithm itself or a robot, conversational agent, etc.) learns to take actions based on experiences in order to optimize a quantitative reward over time. Simply put, RL involves learning through trial and error to achieve complex goals. The agent is immersed within an environment, which is the world around it and includes everything external to the agent [1].
Reinforcement learning is unique because it uses rewards as a feedback signal. RL agent receive rewards based on their actions within the environment. These rewards indicate how well the agent's actions are aligned with its goals, influencing its future behavior.
The agent makes decisions based on its current state, defined as the concrete and immediate situation that the agent finds itself in at a particular moment. In return, the environment provides the agent with a reward, which can be either positive or negative. Through multiple experiences, the agent seeks to develop a behavior that maximizes the sum of rewards (cumulative reward) over time. In a deterministic case, this behavior is specified through a *policy*, which is a function that associates the current state with the action to be executed. In a stochastic case, it determines the probability of the agent taking each action when in a particular state. The agent will then select an action by sampling randomly from this distribution.

2.1.1 Model-Based VS Model-Free

RL algorithms can be mainly divided into two categories – model-based and model-free.

Model-Based use interactions with the environment to build a model of the world : Initially, the agent interacts with the environment to gather data and observations. Instead of directly learning from these interactions, it constructs an internal model of how the environment behaves in response to its actions. This model represents the dynamics of the world, including how states change based on actions taken by the agent. Once the agent has built a model, it can leverage this understanding to predict the next action

and learn an effective policy (the strategy behind its actions). By simulating possible future scenarios using its model, the agent can anticipate the outcomes of different actions and make decisions that maximize its long-term rewards. This planning process allows the agent to learn more efficiently by considering potential consequences before taking actions in the real environment.
Model-Free algorithms learn directly by interacting with the environment and estimate the value of different states and actions. Through trial and error, they estimate the value of being in different situations (states) and taking different actions. This value reflects the expected future rewards the agent can obtain based on its choices. Algorithms like Q-learning and Policy Gradients are popular examples of this approach. These methods involve the agent experimenting with various actions and observing the consequences. Based on the rewards received, the agent constantly updates its internal estimates of value for different states and actions. This continuous learning process allows the model-free agent to develop a successful strategy (policy) that maximizes its long-term reward accumulation.

2.2 Temporal Difference

In reinforcement learning, the concept of the value function plays a central role. It represents the estimated long-term desirability of a state s, denoted as $V^{\pi}(s)$ for a specific policy π. This value reflects the expected cumulative reward an agent can anticipate from a given state onwards, considering the actions it will take according to its policy.
Temporal-Difference (TD) learning provides a powerful method for agents to continuously refine their value function estimates. Unlike waiting for the final outcome, TD updates happen at each step. The agent compares the rewards received with its predictions, represented by the current value function. This comparison generates a temporal difference (TD error), highlighting the discrepancy between the expected and actual reward. By leveraging this TD error, the agent adjusts its value estimates for both the current state (s_t) and the next state (s_{t+1}). This iterative process, expressed by the update rule

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

(where r_{t+1} is the observed reward, α is the learning rate, and γ is the discount factor) allows the agent to progressively improve its predictions of future rewards. As the agent interacts with the environment and accumulates experience, the value function becomes increasingly accurate, guiding the exploration and optimization of policies for long-term reward maximization.

2.3 Model Predictive Control

Model Predictive Control (MPC) represents a model-based approach to decision-making. In MPC, the agent maintains a predictive model of the environment, typically in the form of a transition model that describes how the state of the environment evolves in response to actions. MPC focuses on optimizing a sequence of actions over a finite time horizon. At each time step, the controller uses a model of the system to predict the future state based on a set of candidate control actions. It then selects the control sequence that optimizes a specific objective function.

In MPC, the terminal value function plays a pivotal role where it helps optimize future actions by considering both short-term rewards and long-term returns. By leveraging the terminal value function, agents can navigate complex environments more effectively, balancing immediate gains with future objectives to achieve optimal performance over extended time horizons.

2.4 TD-MPC

2.4.1 Introduction

Data-driven Model Predictive Control (MPC) has emerged as a powerful approach for Model-Based Reinforcement Learning (MBRL). It leverages historical data to learn a model of the system dynamics, enabling efficient exploration and control policy optimization. Compared to purely model-free methods, data-driven MPC offers several advantages. Firstly, by learning a model, it can utilize data more efficiently, potentially reducing the need for extensive trial-and-error interaction with the environment. Secondly, as computational resources become more abundant, data-driven MPC can leverage probabilistic ensembles for trajectory sampling to plan over longer horizons, leading to potentially superior performance. However, challenges remain. Planning over long horizons remains computationally demanding, requiring careful consideration of the vast number of future possibilities. Additionally, accurately modeling complex environments can be difficult, potentially limiting the effectiveness of data-driven MPC.

Looking back at Section 2.2, TD-Learning is a pivotal method in Reinforcement Learning (RL) where the learning process is propelled by the difference between estimated values of states across successive time steps. This capability aids in predicting future rewards and optimizing the expected return. The question then arises : Can TD leverage MPC to mitigate its challenges ? The answer is yes. In this novel method, MPC yields temporally local optimal solutions (instead of global optimal solutions which is the case for traditional MPC) and TD offers terminal value function that approximates globally optimal solutions. It back-propagates gradient from the reward and TD-objective through multiple steps of the model [2], improving value predictions over long horizons.

Temporal Difference Learning for Model Predictive Control (TD-MPC) is a specific framework that embodies this data-driven approach for MBRL. It integrates a learned model with a terminal value function estimated using TD learning. This method, introduced by Hansen et al. (2022) [2], bridges the gap between model-free and model-based approaches. TD-MPC stands out for its efficient data utilization and demonstrably superior performance compared to previous methods. This superiority has been observed across various continuous control tasks, including those using state or image-based inputs. Experiments conducted on tasks from DMControl and Meta-World consistently showcased TD-MPC's effectiveness. Moreover, when considering asymptotic performance, TD-MPC maintains its advantage, demonstrating not only its ability to achieve good results quickly but also its capacity to continuously improve with more data and experience.

2.4.2 Task-Oriented Latent Dynamics Model

TD-MPC handle complex environment by learning from state or image observations. This is particularly advantageous when the relationships between various control variables are intricate and not directly observable. These hidden or indirectly observable aspects are referred to as latent dynamics.
To address these latent dynamics, TD-MPC employs a specialized model called the Task-Oriented Latent Dynamics Model (TOLD). In contrast to traditional methods that attempt to model the entire environment, TOLD focuses specifically on learning elements critical for reward prediction. This targeted approach significantly simplifies the learning task for TD-MPC.
Once image observations are collected from the environment, the TOLD model is trained using TD-learning principles. This training process, along with the influence of rewards received by the agent, allows TOLD to capture the essential elements of the environment that are predictive of reward. The terminal value function, trained separately, estimates the long-term value of reaching different states. Both TOLD and the value function leverage the agent's experience (observations, rewards) to inform their predictions within TD-MPC. During inference, the TD-MPC framework leverages the learned TOLD model for trajectory optimization. TOLD estimates short-term rewards through model rollouts based on the image inputs, while the terminal value function provides estimates of long-term returns. This combined approach enables TD-MPC to make informed decisions in complex environments rich with visual information.

2.4.3 Planning

In TD-MPC, planning future actions is essential for effective decision-making in complex and dynamic environments. Planning allows the agent to anticipate the consequences of its actions and select trajectories that lead to desired outcomes. Unlike model-free algorithms that rely solely on trial-and-error learning, planning with a model enables the agent to proactively explore and exploit the model's predictions of the environment. By performing trajectory optimization at each decision step, using short-term reward estimates from the learned model, and incorporating long-term return estimates from the learned value function, TD-MPC can make informed decisions that balance immediate rewards with long-term objectives.

Cross Entropy Method
The Cross-Entropy Method (CEM) is an iterative optimization technique designed to find the best sequence of actions in a given scenario. This method encompasses several key steps : initially, a set of potential action samples is generated, followed by the evaluation of each sample's performance. Subsequently, the distribution of these samples is adjusted to prioritize the generation of actions that yield higher rewards. Notably, CEM introduces Gaussian noise to newly generated action samples at each iteration, enabling more comprehensive exploration of the action space. By incorporating such stochasticity, the algorithm mitigates the risk of becoming entrenched in suboptimal regions of the search space.
In the context of TD-MPC, the interplay between the learned model (TOLD) and the value function is crucial for planning future actions.

Brief summary
The world model, TOLD, continuously learns the environment's hidden dynamics from the agent's experience (observations and rewards). This allows TD-MPC to predict future outcomes of actions. The reward predictor, the value function, estimates the long-term value of reaching different states. By considering both predictions, TD-MPC makes informed decisions, balancing exploration (trying new actions) and exploitation (focusing on valuable actions) to thrive in dynamic environments.
TD-MPC is built around the interaction between a learned model and a value function. The Task-Oriented Latent Dynamics Model (TOLD), continuously updated through TD learning, captures the essential latent dynamics of the environment based on image observations and rewards received by the agent. This updated TOLD model then serves as the foundation for planning future actions.

Planning with TOLD and the Terminal Value Function
During planning, the agent leverages the TOLD model to envision multiple potential trajectories based on its intended actions. These simulated trajectories represent various sequences of states and corresponding actions the agent might encounter under different policy choices.
The terminal value function estimates the long-term reward an agent can expect from a given state and a given action. By combining the short-term rewards predicted by TOLD model rollouts with the long-term estimates from the terminal value function, the agent can evaluate the overall expected reward for each simulated trajectory.

Decision Making via Trajectory Evaluation
The agent analyzes the expected rewards associated with each trajectory. By prioritizing trajectories with the highest expected rewards, the agent effectively identifies the most promising course of action. This integration of the TOLD model and the terminal value function allows the agent to generate and assess potential action sequences, ultimately optimizing its decision-making in the environment.

2.5 SAC

Soft-Actor-Critic (SAC) is a model-free reinforcement learning algorithm that improves stochastic policy optimization through off-policy methods, linking it to approaches similar to DDPG. SAC integrates the clipped double-Q trick. The stochastic nature of SAC's policy additionally gains advantages from mechanisms similar to target policy smoothing. SAC uses entropy regularization. It trains the policy to balance the expected return with the entropy, which is essentially the level of randomness in the policy decisions. This balance is crucial for managing the exploration-exploitation dilemma : higher entropy fosters more exploration, which not only speeds up learning but also helps avoid early convergence to suboptimal solutions.
The authors chose SAC as their main point of comparison with TD-MPC due to its popularity and strong performance on both DMControl and Meta-World [2].

Chapitre 3

Contribution

In this section, we explain step by step of the implementation of this project and the decisions we made during implementation.

3.1 BBRL

BBRL library, which stands for "BlackBoard Reinforcement Learning", is a Python library that derives from SaLinA library [3]. SaLinA's user-friendly interface is what sets it apart when it comes to developing complex sequential decision models, and its modular agent-based architecture allows it to be highly adaptable. By focusing on a blackboard-like workspace, BBRL enhances this foundation by enabling agents, which are extensions of PyTorch modules, to interact with one another by sequentially reading and writing temporal data. This structure makes it possible to integrate multiple agents simultaneously, which can significantly improve the efficiency and scalability of reinforcement learning models.

Beginners in Reinforcement Learning can gain a more comprehensive understanding of the subject and its mechanics through the BBRL library's learning environment. By decomposing each part of the RL algorithm into an agent that performs specific tasks within the environment, BBRL demonstrates the mechanics of the algorithm and how its different components work together to produce the desired outcome [4].

An important question that arises is : What constitutes an Agent ? According to the SaLinA article [3], "In Reinforcement Learning, an environment is considered an Agent that reads actions (at time $t-1$) and writes observations, rewards, etc." In essence, anything that interacts with the Workspace qualifies as an Agent.

3.2 Code Structure

3.2.1 Preprocessing

We start by applying a preprocessing layer to the environment we intend to work with. Reducing the dimensionality is crucial in TD-MPC because high-dimensional data can lead to significant computational complexity. Our approach was inspired by a project conducted by Mathis Koroglu on "Robust Architecture for Reinforcement Learning in

BBRL".
We apply the following wrappers in the specified order to the Gym environment :

PixelOnlyObservation : This is a custom wrapper designed to configure the environment to return observations solely as pixels. It is derived from `PixelObservationWrapper` of Gym.

ResizeObservation : Found in `gym.wrappers`, this wrapper reduces the size of the images to a predefined dimension, standardizing the input size for subsequent processing.

GrayScaleObservation : Converts color images to grayscale, effectively reducing the complexity of the input data by minimizing the number of color channels.

BinarizeObservation : A custom wrapper that inherits from `gym.ObservationWrapper`. Its purpose is to binarize the observation images using a specified threshold, simplifying the visual input into binary (black and white) images.

FrameStack : This wrapper stacks multiple consecutive frames together, providing temporal context that is crucial for understanding the dynamics within the environment.

3.2.2 Initialization of agents

For this part, our approach was inspired by the implementation of the TD3 algorithm in BBRL [5], given that TD-MPC represents an "enhanced" version of TD3. We start by creating the environment agents, `train_agent` and `eval_agent`. Subsequently, we initialize each component of the TOLD system as an agent, because these components are the ones that interact with the Workspace and manage the reading and writing of necessary information. So we define an agent for each of the 5 parts of the TOLD : Encoder, Dynamics, Reward, Policy, Double Q-values.

3.2.3 Planning

Planning is used twice in the code of the authors of TD-MPC : once during training and once after it during evaluation.

We calculate how many trajectories should be sampled using the policy network (π), based on a specified proportion of the total samples. The latent state z is initialized by encoding the current observation. If a previous mean exists from a prior call, it may be used to initialize the mean for trajectory optimization. The Cross-Entropy Method (CEM) loop involves sampling actions around the current mean using the standard deviation, evaluating them using the estimate value function, and updating the mean and standard deviation based on elite samples. After the optimization, the best action is selected, adding noise during non-evaluation modes to ensure continued exploration.

3.2.4 TOLD

We fetch a batch of experiences from the replay buffer of Workspace in BBRL, consisting of observations, actions, rewards, and next observations, which are transferred to the appropriate computational device. We set TOLD components (encoder, dynamics, reward,

policy (π), and the value (QValues) PyTorch modules) to training mode to enable gradient computation. The latent representation (z) of the observations is computed using the encoder network, and a list (zs) is initialized to store these representations. Iterating over the time horizon, we compute consistency, reward, and value losses, combine them into a total loss, and optimize the parameters using backpropagation. The policy network is updated using the sequence of latent states, calculating the policy loss. Target networks are updated by applying an exponential moving average to their parameters, smoothing the learning updates. Finally, we set all network components to evaluation mode, disabling gradient computation, and return various loss metrics and the gradient norm.

3.2.5 TD-MPC Main Loop

We initialize the logger and define a variable to track the highest reward achieved during training. The computational device for tensor operations is determined. Using ParallelGymAgent from BBRL, we set up training and evaluation environment agents. After creating instances of encoder, dynamics, reward, policy (π), and the value (QValues) components, we configure the optimizers for these networks based on the settings (here, Adam). The training loop involves collecting transitions from the environment, storing them in the replay buffer, performing training updates, and periodically evaluating the trained model using the evaluation agent. Metrics such as episode rewards, losses, and training times are recorded and logged to monitor the training progress.

3.3 Results

3.3.1 TD-MPC and SAC

Although SAC and TD-MPC share several fundamental similarities, they are different in many ways.
SAC is a model-free algorithm, but TD-MPC combines elements of model-based and model-free approaches.
SAC uses a policy network and a value network to derive the policy that dictates action choices. It does not involve explicit planning over future states but makes decisions based on current policy evaluations.
TD-MPC on the other hand utilizes MPC where it plans over a short finite horizon using a learned model to simulate future states and rewards.
Both SAC and TD-MPC make use of value functions in their learning. SAC uses a soft Q-value function to guide both policy improvement and behavior policy, while TD-MPC uses a terminal value function to estimate long-term returns at the end of a planned short trajectories.
According to the [2] article, "For the TD-objective in TD-MPC instead of estimating the quantity $max_{a_t} Q_{\theta^-}(z_t, a_t)$, it learns a policy π that maximizes Q_θ by minimizing a new objective, which is a temporally weighted adaptation of the policy objective commonly used in model-free actor-critic methods such as DDPG and SAC.
While LOOP (a hybrid algorithm that extends SAC with planning and a learned model [6]) relies on the maximum entropy objective of SAC for exploration, TD-MPC learns a deterministic policy".

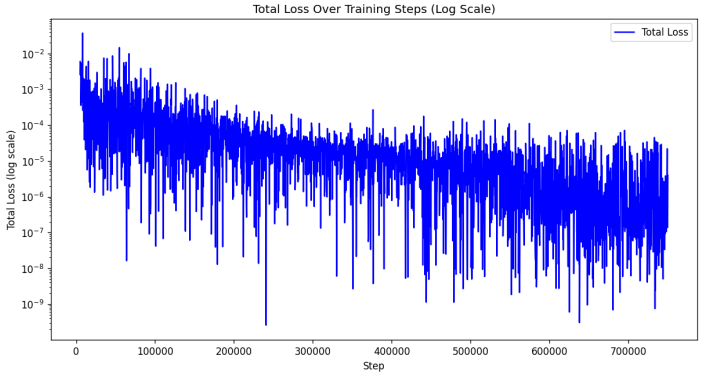
Although both SAC and TD-MPC are suitable for environments with continuous action

spaces, the study that the authors conducted indicated that in most of the DMControl tasks, TD-MPC displayed large performance gains on tasks with complex dynamics and performed better than SAC.

In [2], TD-MPC demonstrates significant improvements in sample efficiency and performance in continuous control tasks. Evaluated on DMControl and Meta-World benchmarks, TD-MPC excels by solving complex tasks such as Humanoid and Dog locomotion, achieving up to 38-dimensional continuous action spaces with just 1 million environment steps. The method's effectiveness is highlighted by its ability to solve the Walker Walk task 16 times faster than Learning Options through Observation Planning (LOOP) while using 3.3 times less compute per 500,000 steps. Additionally, TD-MPC matches the time-to-solve of Soft Actor-Critic (SAC) on both Walker Walk and Humanoid Stand tasks, significantly closing the performance gap between model-free and model-based approaches.

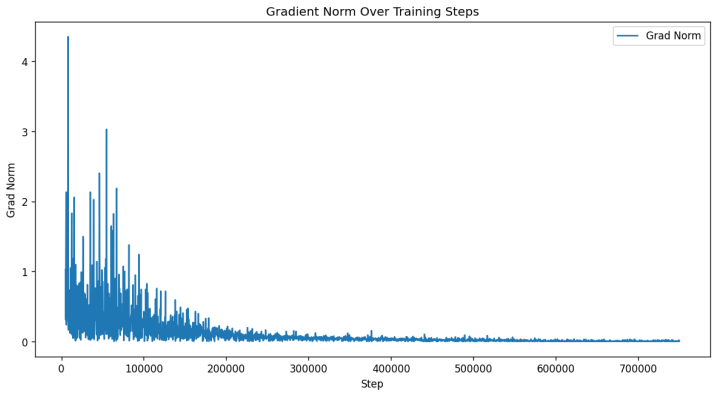
3.3.2 Our objectives

In our approach, achieving satisfactory results proved challenging. One significant limitation was the high computational cost of our code, making debugging and obtaining results difficult. This constraint prevented us from fully realizing our goals. Despite these challenges, we managed to execute our code and make some valuable observations. Notably, the total loss showed a decreasing trend, which is a positive indicator. In machine learning, the loss function quantifies the difference between the predicted outputs and the actual values. A decreasing loss indicates that the model is improving its predictions over time, learning more accurately from the data. Here is a diagram of this trend below :



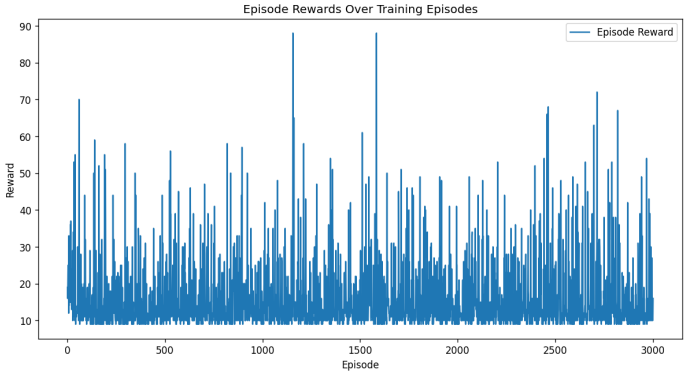
(a) Total loss over training steps (logarithmic scale)

Moreover, in TD-MPC, the gradient norm indicates the magnitude of change applied to the model's internal parameters during each training iteration. A decreasing gradient norm over time suggest that our model is converging towards an optimal solution. With increasingly accurate predictions, the model's updates become less significant, contributing to a more stable and efficient learning trajectory. Here's a diagram showing the gradient norm is decreasing over training steps :



(a) Gradient norm over training steps

However, the policy loss did not decrease, and the return over episodes did not increase. This means we are unable to make a comprehensive comparison between our code and the code from the article. Here is a diagram showing that episode rewards stagnate over training episodes :



(a) Episode rewards over training episodes

Chapitre 4

Conclusion

Our project provided an introduction to research in Reinforcement Learning by focusing on implementing the TD-MPC algorithm with the BBRL library. Our initial step was to thoroughly research resources on reinforcement learning, Temporal Difference algorithms, and Model Predictive Control, while also familiarizing ourselves with the capabilities of the BBRL library. Subsequently, we transitioned into the coding phase, where our primary objective was to refactor and tailor the code provided by Nicklas Hansen, Xiaolong Wang, and Hao Su to seamlessly integrate with the BBRL framework. The aim of this process was to ensure compatibility and functionality within the BBRL environment through careful adaptation and optimization.
This project was our first experience in research in a new field, and it was really enriching. We learned a lot, both in theory and in practice. The challenges we faced, such as dealing with complex computational requirements and time constraints, provided valuable lessons and insights. Overall, this experience has greatly expanded our understanding and capabilities in this area.

In conclusion, we were unable to achieve the desired results due to several factors, including the high computational cost of our code and a lack of time towards the end, which we find quite unfortunate. Nevertheless, our observations indicate that our code is on the right track, as the model consistently improves its predictions over time, except for the policy aspect and stagnant rewards.

Bibliographie

[1] Richard S. SUTTON et Andrew G. BARTO. *Reinforcement Learning : An Introduction*. Second. The MIT Press, 2018. URL : http://incompleteideas.net/book/the-book-2nd.html (page 2).

[2] Nicklas HANSEN, Xiaolong WANG et Hao SU. *Temporal Difference Learning for Model Predictive Control*. 2022. arXiv : 2203.04955 [cs.LG] (pages 4, 6, 9, 10).

[3] Ludovic DENOYER, Alfredo de la FUENTE, Song DUONG, Jean-Baptiste GAYA, Pierre-Alexandre KAMIENNY et Daniel H. THOMPSON. *SaLinA : Sequential Learning of Agents*. 2021. arXiv : 2110.07910 [cs.LG] (page 7).

[4] Olivier SIGAUD. *BBRL foundations*. https://www.isir.upmc.fr/personnel/sigaud/. ISIR (page 7).

[5] Olivier SIGAUD. *Implementation of TD3 in BBRL*. https://github.com/osigaud/bbrl_examples/blob/master/bbrl_examples/algos/td3/td3.py (page 8).

[6] Harshit SIKCHI, Wenxuan ZHOU et David HELD. « Learning Off-Policy with Online Planning ». In : *CoRR* abs/2008.10066 (2020). arXiv : 2008.10066. URL : https://arxiv.org/abs/2008.10066 (page 9).

Annexe A

Cahier des charges

A.1 Introduction

A.1.1 Context

This project is part of an IT project unit of the ANDROIDE master's program. Its purpose is to train us in carrying out a team project from start to finish, from analyzing the subject to software development, as well as experimenting with the work done. Our project topic consists of implementing and evaluating a reinforcement learning algorithm, TD-MPC, within a specific library, BBRL.

A.1.2 Subject

Our project topic is as follows : Implementation of TD-MPC in BBRL.

TD-MPC is a combination of Temporal Difference (TD) learning and Model Predictive Control (MPC). This approach utilizes *Temporal Difference* predictions to enhance planning and action execution in *Model Predictive Control*, enabling better anticipation and adaptation to environmental changes for decision optimization. BBRL, which stands for « BlackBoard Reinforcement Learning », is a simple and flexible library for reinforcement learning, derived from *SaLinA*.

A.1.3 Aim of the project

The objective of the project is to develop TD-MPC code using the reinforcement learning library BBRL, and then to test the implementation in various environments commonly used in the literature.

A.2 Description of TD-MPC

After briefly describing the reinforcement learning classes TD and MPC, we detail the process of the TD-MPC algorithm.

A.2.1 Introduction to Reinforcement Learning

Let's start with a brief presentation of reinforcement learning, which is a fundamental method in the field of artificial intelligence. The main idea is that an agent is placed in a simulated environment where it interacts by taking actions based on a defined policy, representing its action strategy in various situations. Each action results in an observation of the environment's state and the assignment of a corresponding reward for that transition. This dynamic interaction between the agent and its environment allows it to gradually learn to optimize its actions to maximize the rewards it receives, through mechanisms of exploration and exploitation.

A.2.2 Modeling Environmental Dynamics in TD-MPC

In a complex environment, there are often relationships between different variables that are not immediately obvious. We call these relationships latent dynamics, referring to the hidden or indirectly observable aspects of a system.

In the context of TD-MPC, Task-Oriented Latent Dynamics (TOLD) is a model designed to capture and represent these latent dynamics, thereby helping the agent make better decisions. Once observations are collected, the goal is to learn from this data to capture the relationships between states, actions, and rewards in the environment.

A.2.3 Planning Future Actions

Cross-Entropy Method

The Cross-Entropy Method (CEM) is an iterative method that involves generating a set of possible action samples, evaluating the performance of each sample, and then updating the distribution of samples to favor better sample generation. When generating new action samples at each iteration of CEM, Gaussian noise is added to the previous samples, allowing the algorithm to better explore the action space and avoid getting stuck in a suboptimal region of the search space.

In the context of TD-MPC, once the TOLD model is updated, the agent plans future actions using CEM to maximize its long-term rewards in the environment.

Update of TOLD Model

The agent uses the updated TOLD model to simulate multiple possible trajectories of the environment based on the actions it intends to take. These simulated trajectories represent different sequences of states and possible actions that the agent might encounter by executing different policies in the environment. Once simulated, the agent uses CEM to select the best actions to take. The selection criterion is the expected reward at the end of the generated action sequence.

In summary, the agent plans future actions using CEM, allowing it to generate and evaluate sequences of actions to take in its environment.

A.2.4 Temporal Difference

The agent compares the rewards it has observed during its interactions with the environment to the future rewards predicted by the TOLD model. This comparison allows it to compute the Temporal Difference (TD) error, which measures the difference between the actual rewards and the predicted rewards. Using this TD error, the agent updates its value estimates for the states and actions involved in the observed transitions, gradually adjusting its value estimates to minimize this error over time. These estimates thus become increasingly accurate as the agent interacts with the environment and learns to correctly predict future rewards.

A.2.5 An Iterative Algorithm

After completing the previous steps, the agent repeats the process by continuing to interact with the environment, collect new data, update its models, and estimate the values of states and actions. This iteration allows the agent to learn and adapt to its environment over time, improving its estimates, predictions, and decision-making policies.

A.3 Technical Description

The entire code is written in Python.

A.3.1 BBRL Library

The BBRL (BlackBoard RL) library is derived from SaLinA, inheriting all its properties. This library is based on a workspace, a blackboard where all agents read and write temporary data. All other elements are represented as agents.

A.3.2 Other Libraries Used

In our implementation, we use the *Gymnasium* library, an extension of the *Gym* library developed by OpenAI. The goal of this library is to provide a stable, extensible, and maintained platform for the development and comparison of reinforcement learning algorithms, while remaining true to the original spirit of OpenAI Gym.

Annexe B

User Manual

The project is available on GitHub.
Before continuing, you need to install Python3.
Then, follow these steps in a terminal to install the project :

```
git clone https://github.com/ElDjeee/Codage_TD-MPC.git
cd Codage_TD-MPC
python3 -m pip install -r requirements.txt
```

Once this is done, you can run the project with this command :

```
python3 src/main.py
```