Article

Not peer-reviewed version

Hierarchical Episodic Control

Rong Zhou , Zhisheng Zhang [*] , Yuan Wang [*]

Posted Date: 31 August 2023

doi: 10.20944/preprints202308.2135.v1

Keywords: episodic memory; deep reinforcement learning; hierarchical reinforcement learning

Article

Hierarchical Episodic Control

Rong Zhou [1,†], Zhisheng Zhang [1,*] and Yuan Wang [2,*]

[1] Mechanical Engineering School, Southeast University ; rl.quant.rl@gmail.com
[2] Institute of Aeronautics Engineering, Air Force Engineering University; wangy_af@163.com
* Correspondence: oldbc2713@gmail.com, wangy_af@163.com
† These authors contributed equally to this work.

Abstract: Deep reinforcement learning is one of the research hotspots in artificial intelligence and has been successfully applied in many research areas, however, the low training efficiency and high demand for samples are problems that limit the application To address these problems, a hierarchical episodic control model extending episodic memory to the domain of hierarchical reinforcement learning is proposed in this paper. The model is theoretically justified and employs a hierarchical implicit memory planning approach for counterfactual trajectory value estimation. Starting from the final step and recursively moving back along the trajectory, a hidden plan is formed within the episodic memory. Experience is aggregated both along trajectories and across trajectories, and the model is updated using a multi-headed backpropagation similar to bootstrapped neural networks. This model extends the parameterized episodic memory framework to the realm of hierarchical reinforcement learning and is theoretically analyzed to demonstrate its convergence and effectiveness. Experiments conducted in Four Room, Mujoco, and UE4-based active tracking , highlight that the hierarchical episodic control model effectively enhances training efficiency. It demonstrates notable improvements in both low-dimensional and high-dimensional environments, even in cases of sparse rewards.

Keywords: episodic memory; deep reinforcement learning; hierarchical reinforcement learning

1. Introduction

Episodic reinforcement Learning (ERL)[1] introduces a non-parametric memory mechanism in reinforcement learning, which relies on stored memory data For value function learning or decision-making. This approach to some extent addresses the issue of large sample requirements that were challenging For the first generation of deep reinforcement learning models.

Hierarchical reinforcement Learning (HRL) applies an "abstraction" mechanism to traditional reinforcement learning[2]. It decomposes the overall task into subtasks at different levels, allowing each subtask to be solved in a smaller problem space. The policies learned For these subtasks can be reused, thus accelerating the problem-solving process. In this article , we introduce the Hierarchical Episodic Control which combines the advantages of episodic memory and the option-critic architecture to further enhance the efficiency of reinforcement learning. The model utilizes a hierarchical implicit memory planning approach to estimate the value of counterfactual trajectories. It recursively traverses from the last step to the first step along trajectories, forming a hidden planning scheme within episodic memory. Experiences are aggregated both along trajectories and across trajectories, and updates are propagated through reverse propagation For model improvement. This model extends the episodic memory framework to the field of hierarchical reinforcement learning and provides theoretical analysis, demonstrating the model's convergence. Finally, the effectiveness of the algorithm is verified through experiments.

In cognitive science and neuroscience research, episodic memory is a form of "autobiographical" subjective memory[3]. Episodic memory is a type of long-term memory, and in neuroscience, memories lasting For more than two weeks are considered long-term. Episodic memory involves the recollection of personal experiences, events that occurred at a specific time and place. For instance, we often remember the content of a presentation from a previous meeting, which constitutes an episodic memory tied to a particular time, place, and individual's personal experience. Another type of long-term memory is semantic memory, which pertains to organized facts and is independent of time and space. For example, the memory that "Nanjing is the capital of Jiangsu" is a semantic memory. Semantic memories are stored in neural networks formed within the hippocampus, medial temporal lobe, and the thalamus.

The medial temporal lobe, including the hippocampus and the anterior temporal cortex, is involved in forming new episodic memories. Patients with damage to the medial temporal lobe struggle to remember past events. Patients with bilateral hippocampal damage exhibit significant impairment in forming new memories of both their experiences and new events. Damage to the anterior temporal cortex can lead to a lack of time or location information in memories. Some researchers believe that episodic memories are always stored in the hippocampus, while others suggest that the hippocampus stores them briefly before they are consolidated into the neocortex For long-term storage. The latter perspective is supported by recent evidence, suggesting that neurogenesis in the hippocampal region of adults might contribute to removing old memories and enhancing the efficiency of forming new ones.

Recent research indicates that the hippocampus plays a significant role in value-based decisions[1]. It supports adaptive functioning and serves value-based decisions, reflecting how memories are encoded and utilized. The principles of episodic memory offer a framework for understanding and predicting the factors influencing decision-making. The hippocampus's influence on decision-making can occur unconsciously, allowing for automatic and internal impacts on behavior. In value-based decision-making, interactions between the hippocampus and the striatum, as well as other decision-related brain regions like the orbitofrontal and prefrontal cortices, are crucial[3].

2. Related Work

2.1. Episodic Control

Blundell and colleagues introduced the Model Free Episodic Control (MFEC) algorithm [1] as one of the earliest episodic reinforcement learning algorithms. Compared to traditional parameter-based deep reinforcement learning methods, MFEC employs non-parametric episodic memory For value function estimation, which results in higher sample efficiency compared to DQN algorithms. Neural Episodic Control (NEC) [4] introduced a differentiable neural dictionary to store episodic memories, allowing For the estimation of state-action value functions based on the similarity between stored neighboring states.

Savinov et al. [5] utilized episodic memory to devise a curiosity-driven exploration strategy. Episodic Memory DQN (EMDQN) [6] combined parameterized neural networks with non-parametric episodic memory, enhancing the generalization capabilities of episodic memory. Generalizable Episodic Memory (GEM) [7] parameterized the memory module using neural networks, further bolstering the generalization capabilities of episodic memory algorithms. Additionally, GEM extended the applicability of episodic memory to continuous action spaces.

These algorithms represent significant advancements in the field of episodic reinforcement learning, offering improved memory and learning strategies that contribute to more effective and efficient training processes.

2.2. Hierarchical reinforcement Learning

Reinforcement learning improves policies through trial and error interaction with the environment. Its characteristics of self-learning and online learning make it an essential branch of machine learning

[1] The term "value-based decisions" as referred to in the cited literature corresponds to model-free reinforcement learning.

research. Typical reinforcement learning algorithms represent behavior policies using state-action pairs, leading to the "curse of dimensionality" phenomenon where the number of learning parameters exponentially grows with the dimensionality of the state variables. Traditional methods to tackle the curse of dimensionality include state clustering, finite policy space search, value function approximation, and hierarchical reinforcement learning.

HRL introduces an "abstraction" mechanism to traditional reinforcement learning by decomposing the overall task into subtasks at different levels[2]. Each subtask is solved in a smaller problem space, and the policies learned For subtasks can be reused, accelerating the problem-solving process.

The Option is an HRL approach proposed by Sutton[8], which abstracts learning tasks into options. Each option can be understood as a sequence of actions executed with a certain policy defined on a specific substate space to complete a subtask. Each action can be either a basic primitive action or another option. Hierarchical control structures are formed by invoking lower-level options or primitive actions through higher-level options. In the hierarchical reinforcement learning system, these options are added as a special kind of "action" to the original action set. Options can be predetermined by designers based on expert knowledge or generated automatically.

Consider the call-and-return option execution model, where the agent first selects an option ω based on the option policy π_Ω, then selects an action policy based on the option's intra-policy π_ω, and when the termination function value β_ω equals 1, the option terminates, and the control returns to the higher-level policy to select a new option. The parameters of the intra-policy and termination function of option ω are parameterized as $\pi_{\omega,\theta}$ and $\beta_{\omega,\eta}$, where θ and η are trainable model parameters. The update of the policy is performed using policy gradients, and the return value is denoted as: $\rho\left(\Omega, \theta, \eta, s_0, \omega_0\right) = \mathbb{E}_{\Omega,\theta,\omega}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0, \omega_0\right]$. Compared to the original policy gradient, here (s_0, ω_0) is used instead of s_0, where (s_0, ω_0) is an augmented state corresponding to s_0.

Based on this, various value functions are defined. The option-state value function is defined as corresponding to $V(s)$: the expected return when selecting option ω in state s:

$$Q_\Omega(s, \omega) = \sum_a \pi_{\omega,\theta}(a \mid s) Q_U(s, \omega, a) \tag{1}$$

(s, ω) can be understood as an augmented state space, $Q_U : \mathcal{S} \times \Omega \times \mathcal{A} \rightarrow \mathbb{R}$ represents the action-value function within the option.

The intra-policy action-value function (corresponding to $Q(s, a)$) when choosing action a in state s and option ω:

$$Q_U(s, \omega, a) = r(s, a) + \gamma \sum_{s'} \mathrm{P}\left(s' \mid s, a\right) U\left(\omega, s'\right) \tag{2}$$

Here, $U : \Omega \times \mathcal{S} \rightarrow \mathbb{R}$ represents the option value function upon arrival, analogous to $V(s')$ in standard reinforcement learning. The value function upon arrival (corresponding to $V(s')$): represents the expected return when entering state s' after executing option ω:

$$U\left(\omega, s'\right) = \left(1 - \beta_{\omega,\eta}\left(s'\right)\right) Q_\Omega\left(s', \omega\right) + \beta_{\omega,\eta}\left(s'\right) V_\Omega\left(s'\right) \tag{3}$$

Here, $V_\Omega : \mathcal{S} \rightarrow \mathbb{R}$ represents the value function for state s, and $V_\Omega\left(s'\right) = \sum_{\omega'} \pi_\Omega\left(\omega' \mid s'\right) Q_\Omega\left(s', \omega'\right)$.

For the intra-policy model parameter θ, the gradient of the expected return at the initial condition (s_0, ω_0) with respect to θ is:

$$\frac{\partial Q_\Omega\left(s_0, \omega_0\right)}{\partial \theta} = \sum_{s,\omega} \mu_\Omega\left(s, \omega \mid s_0, \omega_0\right) \sum_a \frac{\partial \pi_{\omega,\theta}(a \mid s)}{\partial \theta} Q_U(s, \omega, a) \tag{4}$$

Here, $\mu_\Omega\left(s, \omega \mid s_0, \omega_0\right)$ is the discounted factor of state-option pairs starting from (s_0, ω_0), and it represents the weighting of each augmented state when calculating the derivative. This means that the effect of changing θ has different impacts on different (s, ω) combinations, and when calculating the gradient, the probabilities of occurrence For these augmented states are used as weights, similar to the role of $\mu(s)$ in the original policy gradient. This gradient describes how small changes in the policy affect the overall discounted return.

For the termination function (termination function) of options, with model parameters η, the gradient of the value function at initial condition (s_1, ω_0) with respect to η is:

$$\frac{\partial U\left(\omega_0, s_1\right)}{\partial \eta}=-\sum_{s', \omega} \mu_{\Omega}\left(s', \omega \mid s_1, \omega_0\right) \frac{\partial \beta_{\omega, \eta}\left(s'\right)}{\partial \eta} A_{\Omega}\left(s', \omega\right) \tag{5}$$

Here, $\mu_{\Omega}\left(s', \omega \mid s_1, \omega_0\right)$ represents the discounted factor of state-option pairs starting from (s_1, ω_0), and it is used to calculate the weighting For each augmented state. $A_{\Omega}\left(s', \omega\right)=Q_{\Omega}\left(s', \omega\right)-V_{\Omega}\left(s'\right)$ represents the advantage function. In non-hierarchical policy gradient algorithms, the advantage function is used to reduce the variance of the gradient estimate. In hierarchical policy gradient algorithms, the advantage function determines the result of the policy gradient. If the value of the current option is lower than the average, the advantage function is negative, which increases the probability of termination, and if the value is higher than the average, the advantage function is positive, which tends to continue the option.

During training, the model uses two different time scales for updates: learning the value function on a faster time scale and learning the intra-policy and termination function on a slower time scale.

3. Hierarchical Episodic Control

3.1. Hierarchical Implicit Memory Planning

The update of hierarchical episodic memory is based on implicit memory planning to estimate the optimal rollout value For each state-action pair. At each step, the best cumulative reward along the trajectory up to that point is compared with the value obtained from the memory module, and the maximum of the two is taken. The memory module associated with an option includes an option value memory module and an option-internal memory module, and the value to choose between them is determined by the termination equation. M_θ and $M_{\Omega,\alpha}$ are induced from similar experiences, representing value estimations For counterfactual trajectories related to options. This process recursively forms an implicit planning scheme in the episodic memory, aggregating experience along and across trajectories. The entire backpropagation process can be expressed in the form of Equation 6.

$$R_t=\begin{cases} r_t+\gamma \max \left(R_{t+1}, M_\theta\left(s_{t+1}, \omega_{t+1}, a_{t+1}\right)\right) & \text{if } t<T, \beta_{\omega, \eta}\left(s_{t+1}\right)=0 \\ r_t+\gamma \max \left(R_{t+1}, M_{\Omega, \alpha}\left(s_{t+1}, \omega\right)\right) & \text{if } t<T, \beta_{\omega, \eta}\left(s_{t+1}\right)=1 \\ r_t & \text{if } t=T \end{cases} \tag{6}$$

Where t denotes the step along the trajectory, and T represents the episode length. The backpropagation process in Equation 6 can be expanded and rewritten as Equation 7.

$$\begin{gathered} V_{t,h}=\begin{cases} r_t+\gamma V_{t+1,h-1} & \text{if } h>0 \\ M_{\Omega,\alpha}\left(s_{t+1}, \omega\right) & \text{if } h=0, \beta=1 \\ M_\theta\left(s_t, \omega_t, a_t\right) & \text{if } h=0, \beta=0 \end{cases} \\ R_t=V_{t,h^*} \end{gathered} \tag{7}$$

Where $h^*=\underset{h>0}{\arg\max} V_{t,h}$.

Algorithm 1 Update Memory

```
for stored trajectories $\tau$ do
  for one of the trajectory $\tau$ $t = T \to 1$ do
    according current chosen Option $\omega$,executing $\tilde{a}_{t+1} \sim \pi_{\omega,\theta}(a \mid s)$
    if $\beta = 0$ then
      computing $M_{\theta}^{(1,2)}(s_{t+1}, \omega, a_{t+1})$
    else
      computing $M_{\Omega,\alpha}^{(1,2)}(s_{t+1}, \omega)$
    end if
    For $h = 0 : T - t$ computing $V_{t,h,\omega}^{(1,2)}$ according (6)
    computing $R_{t,h,\omega}^{(1,2)}$ according (7)
    saved into the memory
  end for
end for
```

3.2. *Option Episodic Memory Model*

In the Option Episodic Memory (OptionEM) model, a parameterized neural network M_θ is used to represent the parameterized option-internal memory, and a parameterized neural network M_α is used to represent the parameterized option memory, both learned from a tabular memory M. To utilize the generalization capability of M_θ and M_α, an enhanced reward is propagated along the trajectories using the value estimates from M_θ and M_α, as well as the true rewards from M, to obtain the best possible value over all possible rollouts. The enhanced target is regressed during training to train the generalizable memories M_θ and M_α, with the value chosen based on the termination equation. This enhanced target is then used to guide policy learning and establish new objectives for learning OptionEM.

One crucial issue with this learning approach is the overestimation caused by obtaining the best value along the trajectory. During backpropagation along the trajectory, overestimated values tend to persist and hinder learning efficiency. To mitigate this issue, a Twin Network similar to the Double Q-learning idea is employed for the backpropagation of value estimation. Vanilla reinforcement learning algorithms with function approximation are known to exhibit a tendency to overestimate values, making reducing overestimation critical. To address this problem, the Twin Network structure is used to make value estimates from M_θ more conservative. The training uses three different timescales to update the memory network, the termination function, and the option policy.

3.3. *Theoretical Analysis*

In this section, a theoretical analysis of the OptionEM algorithm is presented. The focus is on the convergence and non-overestimation properties of the algorithm.

3.3.1. Non-Overestimation Property

The algorithm's attribute of not overestimating value estimation errors during the bidirectional propagation process is investigated as a fundamental concept in value-based methods. Theorem 1 demonstrates that the OptionEM method does not overestimate the true maximum in expectations.

Theorem 1. *Given an independent unbiased estimation*

$$\tilde{M}_{\theta}^{(1,2)}\left(s_{t+h}, \omega_{t+h}, a_{t+h}\right) = M^{\pi}\left(s_{t+h}, \omega_{t+h}, a_{t+h}\right) + \epsilon_h^{(1,2)} \tag{8}$$

and

$$\tilde{M}_{\Omega,\alpha}^{(1,2)}\left(s_{t+h}, \omega_{t+h})\right) = M_{\Omega}^{\pi}\left(s_{t+h}, \omega_{t+h}\right) + \epsilon_h^{(1,2)} \tag{9}$$

Algorithm 2 Option Episodic Memory(OptionEM)

```
Initializes the episodic memory network and option network
Initializes the target network parameters$\theta'_{(1)} \leftarrow \theta_{(1)}, \theta'_{(2)} \leftarrow \theta_{(2)}, \alpha'_{(1)} \leftarrow \alpha_{(1)}, \alpha'_{(2)} \leftarrow \alpha_{(2)}, \phi' \leftarrow \phi, \zeta' \leftarrow \zeta, \eta' \leftarrow \eta$
Initializes the episodic memory $\mathbb{M}$
for $t = 1, ..., T$ do
    Choose Option $\omega$,execute action $a$
    receive reward $r$ and next state $s'$
    store tuple $(s, \omega, a, r, s', \omega', \beta)$ in memory $\mathbb{M}$
    for $i\ in \{1, 2\}$ do
        sample N tuples $(s, a, r, s', \beta, R_t^i)$ from memory $\mathbb{M}$
        if $\beta = 0$ then
            update $\theta_{(i)} \leftarrow \min_{\theta_{(i)}} \sum \left(R_t^{(i)} - M_\theta^i(s, \omega, a)\right)^2$
        else
            update $\alpha_{(i)} \leftarrow \min_{\alpha_{(i)}} \sum \left(R_t^{(i)} - M_\alpha^i(s, \omega)\right)^2$
        end if
    end for
    if $t\ mod\ u = 0$ then
        $\theta'_{(i)} \leftarrow \tau\theta_{(i)} + (1-\tau)\theta'_{(j)}$
        $\alpha'_{(i)} \leftarrow \tau\alpha_{(i)} + (1-\tau)\alpha'_{(i)}$
        $\eta'_{(i)} \leftarrow \tau\eta_{(i)} + (1-\tau)\eta'_{(j)}$
        $\phi'_{(i)} \leftarrow \tau\phi_{(i)} + (1-\tau)\phi'_{(i)}$
        update Episodic Memory according Algorithm 1
    end if
    if $t\ mod\ p = 0$ then
        update $\phi$ ,$\nabla_\phi J(\phi) = \nabla_a M_{\theta_1}(s, \omega, a)\big|_{\omega=\pi_\omega, a=\pi_\phi(s)} \nabla_\phi \pi_{\phi(s)}$ according policy gradient
        update $\zeta$ ,$\nabla_\zeta J(\zeta) = \nabla_\omega M_{\alpha_1}(s, \omega)\big|_{\omega=\pi_\omega} \nabla_\zeta \pi_{\zeta(s)}$ according policy gradient
    end if
    if $t\ mod\ q = 0$ then
        update $\eta$ ,$\frac{\partial M_\alpha(\omega_0, s_1)}{\partial \eta} = -\sum_{s', \omega} \mu_\Omega(s', \omega \mid s_1, \omega_0) \frac{\partial \beta_{\omega,\eta}(s')}{\partial \eta} A_\Omega(s', \omega)$ according policy gradient
    end if
end for
```

$$\mathbb{E}_{\tau,\epsilon}\left[R_t^{(1,2)}(s_t)\right] \leq \mathbb{E}_{\tau}\left[\max_{0\leq h<T-t} Q_{t,h}^{\pi}(s_t,,\omega_t,a_t)\right] \tag{10}$$

$$M_{t,h}^{\pi}(s,\omega,a) = \sum_{i=0}^{h}\gamma^i r_{t+i} + \begin{cases} \gamma^{h+1}M^{\pi}(s_{t+h+1},\omega_{t+h+1},a_{t+h+1}), & if\, h<T-t, \beta_{t+h+1}=0 \\ \gamma^{h+1}M_{\Omega}^{\pi}(s_{t+h+1},\omega_{t+h+1}), & if\, h<T-t, \beta_{t+h+1}=1 \\ 0, & if\, h=T-t \end{cases} \tag{11}$$

$\tau = \{(s_t,\omega_t,a_t,r_t,s_{t+1},\omega_{t+1})\}$*is a trajectory,*

The Appendix A gives the proof of theorem 1 .The bidirectional propagation process maintains the non-overestimation property of the double-DQN, ensuring the reliability of the proposed value propagation mechanism. The subsequent analysis further demonstrates the convergence property of the OptionEM algorithm.

3.3.2. Convergence Analysis

In addition to analyzing the statistical properties of value estimation, the convergence of the algorithm is also examined. Consistent with the environmental assumptions of van et al.[9] and hu et al.[7] in their respective studies, we first derive convergence guarantees for the algorithm under deterministic scenarios.

Theorem 2. *In a finite Markov Decision Process with a discount factor less than 1, the parameterized memory of Algorithm 2 converges to Q^* under the following conditions:*
(1) $\sum_t \alpha_t(s,\omega,a)=\infty, \sum_t \alpha_t^2(s,\omega,a)<\infty$
(2) Given that the environment's state transition function is completely deterministic.

while $\alpha_t \in (0,1)$ is learning rate.

The proof of Theorem 2 is an extension of the work by van Hasselt et al.[9], and can be seen in Appendix B. This theorem is applicable only to deterministic scenarios, which is a common assumption in episodic memory-based algorithms[10]. To establish a more precise characterization, we consider a broader class of MDPs, known as approximately deterministic MDPs.

Definition 1. *Define $M_{max}(s_0,\omega_0,a_0)$ as the maximum value obtain from the trajectory which start from the point (s_0,ω_0,a_0):*

$$M_{\max}(s_0,\omega_0,a_0) := \max_{\substack{(s_1,\cdots,s_T),(\omega_1,\cdots,\omega_T),(a_1,\cdots,a_T) \\ s_{i+1}\in \operatorname{supp}(P(\cdot|s_i,\omega_i,a_i))}} \sum_{t=0}^{T}\gamma^t r(s_t,\omega_t,a_t) \tag{12}$$

μ_1,μ_2 are parameters of this nearly deterministic Markov Decision Process (MDP)

$$M_{\max}(s,\omega,a) \leq M^*(s,\omega,a)+\mu_1 \tag{13}$$

$$M_{\Omega,\max}(s,\omega) \leq M_{\Omega}^*(s,\omega)+\mu_2 \tag{14}$$

Where μ is a threshold that depends on the constraint of environmental randomness. Based on the definition of nearly deterministic MDPs, the performance guarantee of the method is formulated as follows:

Lemma 1. *The value functions $M(s,\omega,a)$ and $M_\Omega(s,\omega)$ computed from Algorithm 2 satisfy the following inequalities:*

$$\forall s \in \mathcal{S}, \omega \in \Omega, a \in \mathcal{A}, M^*(s,\omega,a) \leq M(s,\omega,a) \leq M_{\max}(s,\omega,a) \tag{15}$$

$$\forall s \in \mathcal{S}, \omega \in \Omega, M^*_\Omega(s,\omega,a) \leq M_\Omega(s,\omega) \leq M_{\Omega,\max}(s,\omega) \tag{16}$$

Satisfy Theorem 2.

Theorem 3. *In the nearly deterministic environment, the value function of OptionEM satisfies:*

$$V^{\tilde{\pi}}(s) \geq V^*(s) - \frac{2\mu}{1-\gamma}, \forall s \in \mathcal{S} \tag{17}$$

Theorem 3 ensures the applicability of the OptionEM method in nearly deterministic environments, which closely resembles real-world scenarios.The proof can be seen in Appendix C.

4. Experimental Results and Analysis

4.1. Four Room Game

In the classic Four-Room reinforcement learning environment , agent navigates a maze of four rooms connected by four gaps in the walls. In order to receive a reward, the agent must reach the green target square. Both the agent and the goal square are randomly placed in any of the four rooms. The environment is contained in the minigrid [?] library contains a collection of 2D grid world environments that contain goal-oriented tasks. The agents in these environments are red triangular agents with discrete action spaces. These tasks include solving different maze maps and interacting with different objects (e.g., doors, keys, or boxes). The experimental parameter settings for the Four Room game are presented in Table 3.

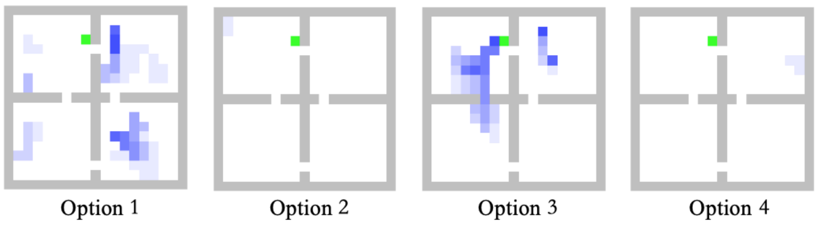
Table 1. Experimental Parameter Settings For Four Room Game

Parameter	Value
AC Learning Rate	2×10^{-1}
OC Learning Rate	8×10^{-1}
OptionEM Learning Rate	8×10^{-1}
η	0.3
Number of Episodes	500
Max Steps per Episode	1000

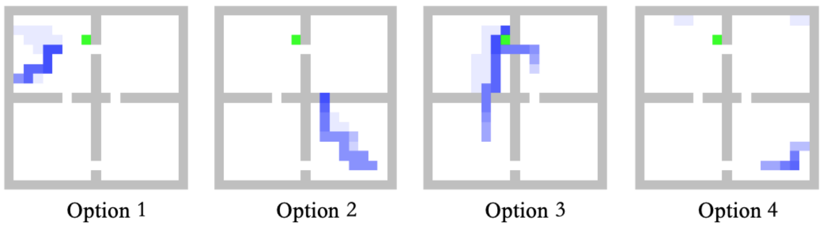
In the Four-Room game environment, a single-layer fully connected neural network is employed to estimate the value function. The input dimension is 3, the hidden layer has 100 units, and the output dimension is 4. The neural network is trained using the error backpropagation method. Based on Martin Klissarov's work [11], five different values of $\eta = 0.2, 0.3, 0.5, 0.7, 1.0$ were tested. The experimental results indicated that the best performance was achieved with $\eta = 0.3$. ThereFore, this parameter value is used directly in this study.

In Figure 2, assuming that options have been provided and are kept fixed throughout the learning process, the only element being updated is the option value function $Q_\Omega(s,\omega)$. The option policy uses this function for selecting among options. The multi-option update rule from Equation 3 is compared to the case of updating only the sampled option in the state. Four options designed by Sutton et al. are used in this scenario. In this case, the OptionEM method achieves significantly better sample efficiency, demonstrating how the proposed method accelerates learning the utility of a suitable set of options.

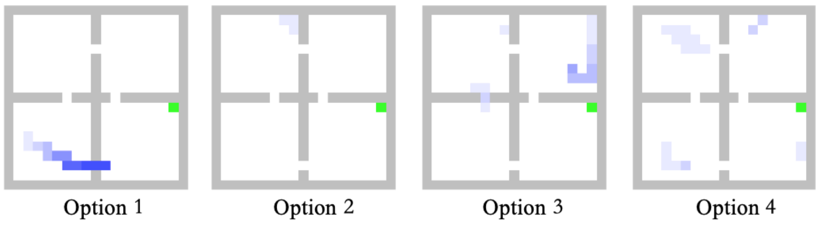
For the case of learning all option parameters, the learning process is conducted using Equation 4. In this setting, the OptionEM method is compared against the Option-Critic (OC) algorithm and the Actor-Critic (AC) algorithm. Both OC and AC outperform the baseline algorithm in terms of hierarchical agent performance. However, the agent is able to achieve similar performance to OC in approximately half the number of episodes. Additionally, it's noteworthy that OptionEM exhibits lower variance across different runs, which is consistent with the anticipated reduction in variance effect of the expected updates from prior research. Figure 2 displays five paths taken by the agent in the game, with fixed start and goal points. Training and exploration processes showcase the comparison experiments between OptionEM and the Option-Critic algorithm. The Option-Critic algorithm exhibits some imbalance among options, where a single option is consistently preferred by the policy. This imbalance is more pronounced when using tabular value functions, leading to the degradation of the Option-Critic algorithm into a regular Actor-Critic algorithm. However, this imbalance is mitigated when using neural networks for value function approximation. Due to the shared information among options, the robustness of learning strategies on both the environment's stochasticity and option policy learning process is improved, allowing the OptionEM algorithm to achieve a balanced performance based on state space separation.



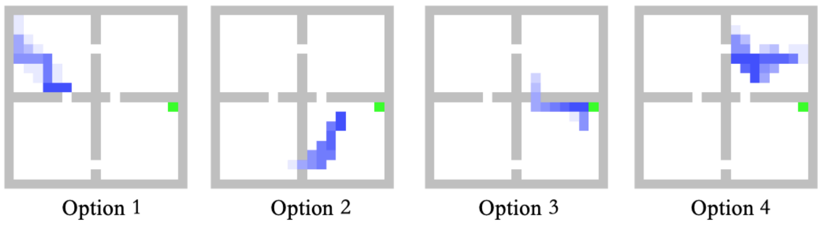
(a) Goal A Option-Critic Option Sample



(b) Goal B Option-Critic Option Sample



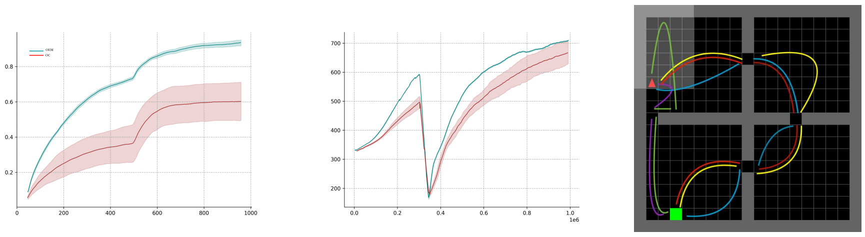
(c) Goal A OptionEM Option Sample



(d) Goal B OptionEM Option Sample

Figure 1. Samples of the Option in Fourroom Game

The Figure 1 illustrates the option intentions of the OptionEM algorithm and the Option-Critic algorithm during the testing phase. In order to effectively showcase the learning process of different options, we have chosen to set the game's background to a white color.In the figure, the green color represents the target location, and the blue portion indicates the current option's position and distribution. It can be observed that the Option-Critic algorithm exhibits a noticeable imbalance in the use of options, which might lead to the degradation of the Option-Critic into an Actor-Critic algorithm. The options learned by the OptionEM, using the twin network mechanism, are more balanced compared to the Option-Critic. However, there still exists some degradation in option 0 within OptionEM.



(a) rewards on training (b) the option updata (c) the sample path

Figure 2. Four-room Game Experiments

4.2. Mujoco

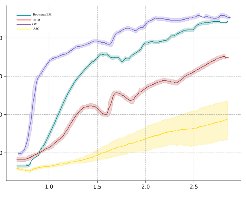
MuJoCo[12] (Multi-Joint dynamics with Contact) is a physics simulator for research and development in robotics, biomechanics, graphics and animation, machine learning, and other fields that require fast and accurate simulation of the interaction of articulated structures with their environment . The default Mujoco games do not effectively demonstrate the sub-goal or option characteristics of hierarchical reinforcement learning. ThereFore, in this section, the Ant-Maze and Ant-Wall environments are used for training and testing. The Ant-Maze game environment utilizes open-source code from GitHub[2], while the Ant-Wall game environment is custom-built, featuring randomly initialized walls. In addition to training actions for the Ant agent itself, the agent also needs to learn path planning strategies to navigate through mazes or around obstacles/walls. This represents a typical scenario For hierarchical reinforcement learning.

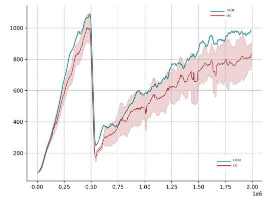
Table 2 presents the parameter settings for the Mujoco games in this section. When the number of options is set to 2, it is evident that for both the Ant-Maze and Ant-Wall environments, the agent employs one option when it is far from obstacles/walls, and another option when it approaches obstacles/walls. Learning methods based on options provide valuable insights for the agent's long-term learning/planning in such scenarios.

Table 2. MuJoCo Game Experiments Parameters

	Ant-Maze	Ant-Wall
learning rate	$2e-4$	$2e-4$
γ	0.99	0.99
λ	0.95	0.95
η	0.95	0.95
batch size	32	32
Entropy coefficient	0.0	0.0
max length	1000	1000

[2] Environment source code: https://github.com/kngwyu/mujoco-maze.git


(a) Rewards in training processes


(b) Option Update

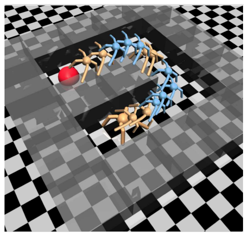
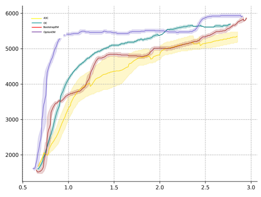
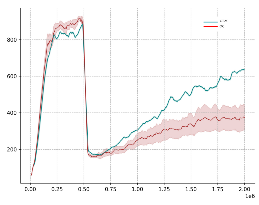

(c) Ant-Maze Option

Figure 3. Mujoco Ant-Maze game


(a) Rewards in training processes


(b) Option Update

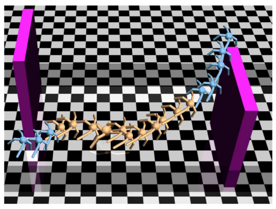

(c) Ant-Wall Option

Figure 4. Mujoco Ant-Wall game

Table 3. different numbers of options in Mujoco Ant-Maze game

Algrithms	2 Options	4 Options	8 Options
OC	343(72)	291(82)	207(61)
Flexible OC	601(93)	582(73)	539(86)
OptionEM	612(110)	571(102)	558(97)

Observing the similarity in path planning between the Ant-Wall and Ant-Corridor games, an attempt was made to transfer the model trained in the Ant-Wall environment to the Ant-Corridor environment. Figure 3 presents the qualitative results of the agent's learned option categories. In this experiment, the performance from six independent runs was averaged. Additionally, the OptionEM algorithm was compared with two option-based algorithm frameworks, and the results for different numbers of options were compared. It is noteworthy that when using 8 options, the performance eventually drops significantly. This could be attributed to the lower-dimensional environment of the Mujoco experiments and the simplicity of the environment, making such a high number of options unnecessary and decreasing sample efficiency. One way to validate this is to apply the same algorithm in a continuous learning environment. Furthermore, due to the distinct update rules of the OptionEM algorithm compared to OC baseline and Flexible OC algorithms, it can effectively utilize limited sample data, resulting in better performance.

4.3. UE4-based Active Tracking

The UE4-based target tracking experiment is mainly designed to verify the generalization of the episodic memory module. A highly realistic virtual environment based on Unreal Engine 4 (UE4) is used for independent learning. The virtual environment has a three-layer structure. The bottom layer consists of simulation scenes based on Unreal 4, which contains a rich variety of scene instances, and provides a general communication interaction interface based on UnrealCV[13], which realizes the communication between the external program and the simulation scene. The agent-environment interaction interface is defined by the specific task, along with the relevant environment elements, such as: reward function, action state space, etc. The interaction interface design specification is compatible with the OpenAI Gym environment.

The agent actively controls the movement of the camera based on visual observations in order to follow the target and ensure that it appears in the center of the frame at the appropriate size. A successful tracking is recorded only if the camera continues to follow the target for more than 500 steps. During the tracking process, a collision or disappearance of the target from the frame is recognized as a failed tracking. Accurate camera control requires recognition and localization of the target and reasonable prediction of its trajectory. Each time the environment is reset, the camera will be placed anywhere in the environment and the target counterpart will be placed 3 meters directly in front of the camera.

The specific tracker reward is:

$$r = A - \left(\frac{\sqrt{x^2 + (y-d)^2}}{c} + \lambda|\omega| \right) \tag{18}$$

For $A > 0, c > 0, d > 0, \lambda > 0$, c is set as a normalized distance. The environment defines the maximum reward to a position where the target is directly in front of the tracker (the tracker is parallel to the target character's shoulders) at a distance of d. With a constant distance, if the target rotates sideways, the tracker needs to turn behind the target to get the maximum reward.

In order to enhance the generalization ability of the model , it is necessary to increase the diversity of the surrounding environment and the target itself as much as possible during the training process. In response to the diversity of the surrounding environment, the environment enhancement in UE4 can be used to easily modify the texture, lighting conditions and other elements of the surrounding environment, specifically, randomly selecting images from a common image dataset deployed on the surface of the environment and objects to modify the texture, and randomly deploy the position, intensity, color, and direction of the light source in the environment to change the scene lighting conditions. Randomization of texture and illumination prevents the tracker from overfitting the specific appearance of the target and background. The diversity requirements of the target itself can be realized by varying the target's trajectory and speed of movement. The start and end position coordinates of the target can be randomly generated, and the corresponding trajectories are generated using the UE4 engine's built-in navigation module (the trajectories generated by the built-in navigation module automatically avoid obstacles, and do not crash into walls). The motion speed of the target is randomly sampled within the range of $(0.1m/s, 1.5m/s)$. The randomization of the target motion trajectory and motion velocity allows the bootstrapped sequence encoder to learn the target motion mode and implicitly encode the motion features, avoiding the situation of a single motion pattern. For the training process, at the beginning of each episode, the target character walks along the planned trajectory from a randomly set start position toward the end position, and the tracker starts walking from a position 3 meters directly behind the target character, and the tracker needs to adjust the position and camera parameters to make the target always in the center of the tracker screen. For the testing process, the target character's movement speed is randomly sampled in the range of $(0.5m/s, 1.0m/s)$ to test the generalization ability of the model.

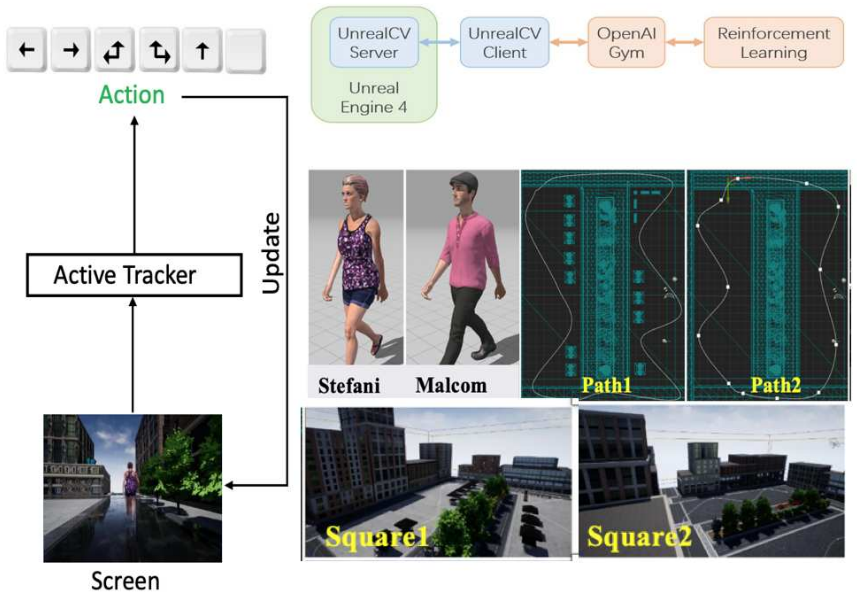
Train the agent and compared with the A3C algorithm adopted by Zhong, Fangwei et al. in the paper [14].The hyper-parameter settings in the experiments are kept the same as in that paper in order to facilitate the comparison. Each game is trained in parallel for 6 episodes. randomly set the seed (seed) of the environment. The agents are trained from zero, and during the training process, the validation is unfolded in parallel. the validation environment is set up in the same way as the training, and the game process with the highest score in the validation environment is selected to report the results for experimental comparison.

An end-to-end conv - lstm network structure is adopted to the tracker agent, which is consistent with the network structure adopted by Luo, Wenhan , Zhong, Fangwei et al. in their paper [14]

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 August 2023 doi:10.20944/preprints202308.2135.v1

[15]. The convolutional layer and the temporal sequencing layer (LSTM) are not connected by a fully connected layer. The convolutional layer portion uses four layers of convolution and the temporal layer portion uses a single LSTM layer, each containing a ReLU activation layer. The network parameters were updated using the shared Adam optimizer. The observed frame is adjusted to an RGB image of $84 \times 84 \times 3$ dimensions, which is input to the conv-lstm network, where the convolutional layer extracts features from the input image, and the fully-connected layer transforms the feature representation into a 256-dimensional feature vector. Each layer contains a ReLU activation layer. The sequence encoder is a 256-unit single-layer LSTM that encodes the image features temporally. The output of the previous time step is used as part of the input of the next time step, so that the current time step contains the feature information from all previous time steps.

The environment of the tracking game is directly adopted from the environment setting of Zhong, F et al. in their paper [14] shown in Figure 5. The game environment consists of a random combination of three elements: character, path, and square.



* source image: End-to-end Active Object Tracking and Its Real-world Deployment via Reinforcement Learning [14]

Figure 5. UE4 Environment and Settings

The effectiveness of the algorithm was tested in four different environments with the following combinations S1SP1: Square1StefaniPath1 ; S1MP1: Square1MalcomPath1 ; S1SP2: Square1StefaniPath2 ; S2MP2: Square2MalcomPath2. and compared with the Luo, Wenhan Zhong, Fangwei et al.'s tracker used in their paper [14] (in their paper, the A3C algorithm was used to train the agents).

The comparison experiments were conducted using bootstrappedEM, OptionEM with A3C method. The options in OptionEM were set to 4 and 8, denoted as OptionEM(4), OptionEM(8), respectively.

Based on the results in Table 4, the following conclusions can be drawn:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 August 2023 doi:10.20944/preprints202308.2135.v1

Table 4. Rewards in eperiments on UE4 tracking

Environment	A3C	bootstrappedEM	OptionEM(4)	OptionEM(8)
S1SP1	2495.712.4	2653.12.7	2791.420.1	2488.13.0
S1MP1	2106.029.3	2345.821.6	2539.423.1	2142.471.1
S1SP2	2162.548.5	2401.033.8	2519.554.0	2097.345.8
S2MP2	740.0577.4	1624.911.3	1741.221.4	1672.442.6

Compared with S1SP1, S1MP1 changes the target role, and all three algorithms perform well on the generalization results of changing the target appearance. Compared with S1SP1, S1SP2 changes the path, and all three algorithms perform well on the generalization result of changing the path. Compared with S1SP1, S2MP2 changes the map, target, and path at the same time, and the generalization results of all three algorithms for this are slightly insufficient compared to the previous two, but still can track the target relatively stably, and the model has some generalization potential, which may need to be improved by migration learning or more environmental enhancements. In most cases, the trackers trained by the bootstrappedEM algorithm and the OptionEM algorithm outperform those trained by A3C. The experimental results of the OptionEM(4) algorithm outperformed OptionEM(8), and the performance of the model was instead reduced with 8 options, which also indicates that 4 options is a more appropriate setting in tracking scenarios.

Figure 6 shows the distribution of options learned by the OptionEM algorithm when the number of options is set to 4. Observation reveals that the distribution of options can be roughly described as: When the target has its back to the tracker and is at a distance, the option with purple marking is selected; when the target has its back to the tracker and is at a closer distance, the option with yellow marking is selected; when the target is not in the screen or is facing the tracker, there are two possible options, the option with orange marking bootstrapped the tracker to do self-rotating motion, while the option with blue marking appears more randomly and the action has no obvious law to be summarized. The model options trained by OptionEM are able to distinguish more clearly between the situation where the target is in the field of view and the situation where the target is lost, when the tracker loses the target in its field of view, the trained options can control the tracker to perform a rotational movement in order to find the lost target as soon as possible. Overall, the distribution of these four options is somewhat consistent with the logic of humans doing target tracking in real scenes.

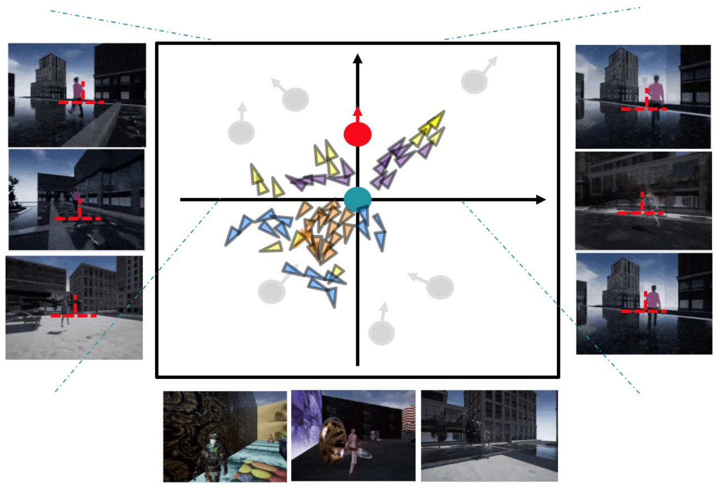


Figure 6. Option Samples in UE4 tracking

5. Summary

In this article, we introduce a hierarchical episodic control model , which employs a hierarchical implicit memory planning approach for value estimation of counterfactual trajectories. The planning process starts from the last step and recursively moves backward along the trajectory, forming an implicit plan in the episodic memory. Experiences are aggregated both along trajectories and across trajectories, and the model is updated using a multi-headed backpropagation mechanism similar to bootstrapped neural networks. This model extends the episodic memory framework to the field of hierarchical reinforcement learning and is theoretically analyzed to demonstrate its convergence and effectiveness.

The results from various experiments, including Four Room, Mujoco, UE4 target tracking, indicate that the hierarchical episodic control model effectively enhances training efficiency. It demonstrates significant improvements in both low-dimensional and high-dimensional environments, as well as performing well in sparse environments. Additionally, a summary of the average training times for episodic memory models and hierarchical episodic control models is provided across 57 Atari games, the UE4 tracking game. This summary underscores how the proposed algorithms in this paper can effectively improve training efficiency and reduce the required training time across diverse games and practical applications.

Author Contributions: Conceptualization, methodology, software, validation, writing, Rong Zhou; resources, data curation, project administration, Yuan Wang; supervision, Zhisheng Zhang and Yuan Wang. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Natural Science Foundation of Shaanxi Province under Grant 2022JQ-584.

Data Availability Statement: The source code of environments is contained within the article.

Acknowledgments: This work was supported in part by the Natural Science Foundation of Shaanxi Province under Grant 2022JQ-584.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

Proof: Consider independent unbiased estimate

$$M_\theta^{(1,2)}\left(s_{t+h},\omega_{t+h},a_{t+h}\right)=M^\pi\left(s_{t+h},\omega_{t+h},a_{t+h}\right)+\epsilon_h^{(1,2)} \tag{A1}$$

the equation satisfies:

$$\mathbb{E}_{\tau,\epsilon}\left[R_t^{(1,2)}\left(s_t,\omega_t\right)\right]\leq\mathbb{E}_\tau\left[\max_{0\leq h\leq T-t-1}M_{t,h}^\pi\left(s_t,\omega_t\right)\right] \tag{A2}$$

while,

$$M_{t,h}^\pi(s,\omega,a)=\begin{cases}\sum_{i=0}^h\gamma^i r_{t+i}+\gamma^{h+1}M^\pi\left(s_{t+h+1},\omega_{t+h+1},a_{t+h+1}\right) & \text{if } h<T-t,\beta_{t+h+1}=0\\ \gamma^{h+1}M_\Omega^\pi\left(s_{t+h+1},\omega_{t+h+1}\right), & \text{if } h<T-t,\beta_{t+h+1}=1\\ \sum_{i=0}^h\gamma^i r_{t+i} & \text{if } h=T-t\end{cases} \tag{A3}$$

$$R_t^{(1,2)}=V_{t,h^*}=\begin{cases}\sum_{i=0}^{h^*}\gamma^i r_{t+i}+\gamma^{h^*+1}M_\theta^{(2,1)}\left(s_{t+h^*+1},\omega_{t+h^*+1},a_{t+h^*+1}\right), & \text{if } \beta_{t+h+1}=1\\ \sum_{i=0}^{h^*}\gamma^i r_{t+i}+\gamma^{h^*+1}M_{\Omega,\alpha}^{(2,1)}\left(s_{t+h^*+1},\omega_{t+h^*+1}\right) & \text{if } \beta_{t+h+1}=0\end{cases} \tag{A4}$$

$$\begin{aligned}\mathbb{E}_{\epsilon}\left[R_t^{(1,2)} - M^{\pi}_{t,h^*_{(1,2)}}\left(s_t, \omega_t\right)\right] &= \mathbb{E}\left[V_{t,h^*_{(1,2)}} - M^{\pi}_{t,h^*_{(1,2)}}\left(s_t, \omega_t\right)\right] \\ &= \mathbb{E}\left[\gamma^{h^*+1}\left(M^{(2,1)}_{\Omega,\theta}\left(s_{t+h^*+1}, \omega_{t+h^*+1}\right) - M^{\pi}_{\Omega}\left(s_{t+h^*+1}, \omega_{t+h^*+1}\right)\right)\right] \\ &= 0\end{aligned} \tag{A5}$$

so,

$$\mathbb{E}_{\tau,\epsilon}\left[R_t^{(1,2)}\right] = \begin{cases}\mathbb{E}_{\tau}\left[M^{\pi}_{t,h^*_{(1,2)}}\left(s_t, \omega_t, a_t\right)\right] \leq \mathbb{E}_{\tau}\left[\max_{0\leq h\leq T-t} M^{\pi}_{t,h}\left(s_t, \omega_t, a_t\right)\right], & \text{if } \beta_{t+h+1} = 0 \\ \mathbb{E}_{\tau}\left[U^{\pi}_{t,h^*_{(1,2)}}\left(s_t, \omega_t\right)\right] \leq \mathbb{E}_{\tau}\left[\max_{0\leq h\leq T-t} U^{\pi}_{t,h}\left(s_t, \omega_t\right)\right], & \text{if } \beta_{t+h+1} = 1\end{cases} \tag{A6}$$

This concludes the proof of Theorem 1.

The subsequent analysis further demonstrates the convergence property of the OptionEM algorithm.

Appendix B

Proof:

$\Delta_t = M_t^{(1)} - M^*$ when $\beta_t = 0$,$F_t\left(s_t, \omega_t, a_t\right) = R_t^{(1)} - M^*\left(s_t, \omega_t, a_t\right)$, when $\beta_t = 1$, $F_t\left(s_t, \omega_t, a_t\right) = R_t^{(1)} - M^*_{\Omega}\left(s_t, \omega_t\right)$

so,

$$\Delta_{t+1} = \left(1 - \alpha_t\right)\Delta_t + \alpha_t F_t \tag{A7}$$

define:

$$G_t = F_t + \left(\tilde{R}_t^{(1)} - R_t^{(1)}\right) \tag{A8}$$

when $\beta_t = 0$:

$$\begin{aligned}\tilde{R}_t^{(1)} - M^*\left(s_t, \omega_t, a_t\right) &\geq r_t + \gamma\sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, \tilde{a}^*\right)\tilde{M_{\Omega}}^{(1)}\left(\omega, s_{t+1}\right) - M^*\left(s_t, \omega_t, a_t\right) \\ &= r_t + \gamma\sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, \tilde{a}^*\right)\tilde{M_{\Omega}}^{(1)}\left(\omega, s_{t+1}\right) - r_t - \gamma\sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, a^*\right)M^*_{\Omega}\left(\omega, s_{t+1}\right) \\ &= \gamma\left(\sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, a_t\right)\tilde{M_{\Omega}}^{(1)}\left(\omega, s_{t+1}\right) - \sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, a_t\right)M^*_{\Omega}\left(\omega, s_{t+1}\right)\right) \\ &= \gamma\sum_{s_{t+1}} \mathrm{P}\left(s_{t+1} \mid s_t, a_t\right)\left(\tilde{M}^{(1)}_{\Omega}\left(\omega, s_{t+1}\right) - M^*_{\Omega}\left(\omega, s_{t+1}\right)\right) \\ &\geq -\gamma\left\|\Delta_t\right\|\end{aligned} \tag{A9}$$

$$\begin{aligned}
\tilde{R}_t^{(1)} - M^*\left(s_t, \omega_t, a_t\right) &= \sum_{i=0}^{h^*_{(1)}} \gamma^i r_{t+i} + \gamma^{h^*_{(1)}+1} \tilde{M}^{(1)}\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, \tilde{a}^*\right) - M^*\left(s_t, \omega_t, a_t\right) \\
&\leq \sum_{i=0}^{h^*_{(1)}} \gamma^i r_{t+i} + \gamma^{h^*_{(1)}+1} \tilde{M}_\pi^{(1)}\left(s_{t+h^*_{(1)}+1}\right) \\
&\quad - \left(\sum_{i=0}^{h^*_{(1)}} \gamma^i r_{t+i} + \gamma^{h^*_{(1)}+1} M^*\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, a^*\right)\right) \\
&= \gamma^{h^*_{(1)}+1}\left(\tilde{M}^{(1)}\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, \tilde{a}^*\right) - M^*\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, a^*\right)\right) \\
&\leq \gamma\left(\tilde{M}^{(1)}\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, \tilde{a}^*\right) - M^*\left(s_{t+h^*_{(1)}+1}, \omega_{t+h^*_{(1)}+1}, a^*\right)\right) \\
&\leq \gamma \left\|\Delta_t\right\|
\end{aligned} \tag{A10}$$

The proof of Theorem 2 is an extension of the work by van Hasselt et al.[9]. This theorem is applicable only to deterministic scenarios, which is a common assumption in episodic memory-based algorithms[10]. To establish a more precise characterization, we consider a broader class of MDPs, known as approximately deterministic MDPs.

Appendix C

Proof:

$$\begin{aligned}
&M_\Omega^*\left(\omega, s'\right) - M_\Omega^{\tilde{\pi}}\left(\omega, s'\right) \\
&= M_\Omega^*\left(s, \omega^*\right) - M_\Omega^{\tilde{\pi}}(s, \omega) \\
&= U^*\left(s, \omega^*\right) - \tilde{M}_\Omega\left(s, \omega^*\right) + \tilde{M}_\Omega\left(s, \omega^*\right) - M_\Omega^{\tilde{\pi}}(s, \tilde{a}) \\
&\leq \epsilon + \tilde{M}_\Omega(s, \tilde{a}) - M_\Omega^{\tilde{\pi}}(s, \omega) \\
&= \epsilon + \left(\tilde{M}_\Omega(s, \omega) - M_\Omega^*(s, \omega)\right) + \left(M_\Omega^*(s, \omega) - M_\Omega^{\tilde{\pi}}(s, \omega)\right) \\
&\leq 2\epsilon + \gamma\left(M_\Omega^*\left(\omega, s'\right) - M_\Omega^{\tilde{\pi}}\left(\omega, s'\right)\right)
\end{aligned} \tag{A11}$$

$$\begin{aligned}
&V^*(s) - V^{\tilde{\pi}}(s) \\
&= M^*\left(s, \omega, a^*\right) - M^{\tilde{\pi}}(s, \omega, \tilde{a}) \\
&= M^*\left(s, \omega, a^*\right) - \tilde{M}\left(s, \omega, a^*\right) + \tilde{M}\left(s, \omega, a^*\right) - M^{\tilde{\pi}}(s, \tilde{a}) \\
&\leq \epsilon + \tilde{M}(s, \tilde{a}) - M^{\tilde{\pi}}(s, \tilde{a}) \\
&= \epsilon + \left(\tilde{M}(s, \omega, \tilde{a}) - M^*(s, \omega, \tilde{a})\right) + \left(M^*(s, \omega, \tilde{a}) - M^{\tilde{\pi}}(s, \omega, \tilde{a})\right) \\
&\leq 2\epsilon + \gamma\left(V^*(s) - V^{\tilde{\pi}}(s)\right)
\end{aligned} \tag{A12}$$

References

1. Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J.Z.; Rae, J.; Wierstra, D.; Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460* **2016**.
2. Jing, S. Research on hierarchical reinforcement learning. PhD thesis, Harbin engining University, 2006.
3. Tulving, E. Episodic memory: From mind to brain. *Annual review of psychology* **2002**, *53*, 1–25.
4. Pritzel, A.; Uria, B.; Srinivasan, S.; Badia, A.P.; Vinyals, O.; Hassabis, D.; Wierstra, D.; Blundell, C. Neural episodic control. In Proceedings of the International Conference on Machine Learning. PMLR, 2017, pp. 2827–2836.
5. Savinov, N.; Raichuk, A.; Marinier, R.; Vincent, D.; Pollefeys, M.; Lillicrap, T.; Gelly, S. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274* **2018**.
6. Lin, Z.; Zhao, T.; Yang, G.; Zhang, L. Episodic memory deep q-networks. *arXiv preprint arXiv:1805.07603* **2018**.

7. Hu, H.; Ye, J.; Zhu, G.; Ren, Z.; Zhang, C. Generalizable episodic memory for deep reinforcement learning. *arXiv preprint arXiv:2103.06469* **2021**.
8. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* **1999**, *112*, 181–211.
9. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the Thirtieth AAAI conference on artificial intelligence, 2016.
10. Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J.Z.; Rae, J.; Wierstra, D.; Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460* **2016**.
11. Klissarov, M.; Precup, D. Flexible Option Learning. *Advances in Neural Information Processing Systems* **2021**, *34*, 4632–4646.
12. Todorov, E.; Erez, T.; Tassa, Y. Mujoco: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ international conference on intelligent robots and systems. IEEE, 2012, pp. 5026–5033.
13. Qiu, W.; Yuille, A. Unrealcv: Connecting computer vision to unreal engine. In Proceedings of the European Conference on Computer Vision. Springer, 2016, pp. 909–916.
14. Luo, W.; Sun, P.; Zhong, F.; Liu, W.; Zhang, T.; Wang, Y. End-to-end active object tracking via reinforcement learning. In Proceedings of the International conference on machine learning. PMLR, 2018, pp. 3286–3295.
15. Zhong, F.; Sun, P.; Luo, W.; Yan, T.; Wang, Y. Ad-vat+: An asymmetric dueling mechanism for learning and understanding visual active tracking. *IEEE transactions on pattern analysis and machine intelligence* **2019**, *43*, 1467–1482.