Research Article

Integrating Functionalities To A System Via Autoencoder Hippocampus Network

Siwei Luo[1]

1. Software School, Jiangxi Normal University, China

Integrating multiple functionalities into a system poses a fascinating challenge to the field of deep learning. While the precise mechanisms by which the brain encodes and decodes information, and learns diverse skills, remain elusive, memorization undoubtedly plays a pivotal role in this process. In this article, we delve into the implementation and application of an autoencoder-inspired hippocampus network in a multi-functional system. We propose an autoencoder-based memorization method for policy function's parameters. Specifically, the encoder of the autoencoder maps policy function's parameters to a skill vector, while the decoder retrieves the parameters via this skill vector. The policy function is dynamically adjusted tailored to corresponding tasks. Henceforth, a skill vector graph neural network is employed to represent the homeomorphic topological structure of subtasks and manage subtasks execution.

Corresponding author: Siwei Luo, luosiwei@jxnu.edu.cn

1. Introduction

Recently, deep learning models, designed and optimized to excel at a particular problem, are task-specific. Multi-functionality capability is a fascinating topic and memorization plays an important role in this integration and incorporation process.

The crucial structure hippocampus locates in the medial temporal lobe of the brain, specifically within the limbic system[1]. Named for its resemblance to the shape of a seahorse, the hippocampus plays a vital role in various cognitive functions, particularly memory. The hippocampus is composed of several distinct regions, including the dentate gyrus, the cornu ammonis fields, and the subiculum. These regions work together to facilitate the encoding, consolidation, and retrieval of memories. The hippocampus is integral to this process, helping to encode and store these memories for later retrieval. In

 doi.org/10.32388/9C2UE7.2

addition to its role in declarative memory, the hippocampus also plays a significant role in spatial navigation and orientation. It is involved in the formation of cognitive maps, which help us navigate our environment and understand spatial relationships. The hippocampus is highly interconnected with other brain regions, including the cortex, amygdala, and thalamus. These connections allow it to integrate information from various sources and coordinate cognitive processes. The hippocampus is a vital brain structure that plays a crucial role in memory, spatial navigation, and orientation.

The neuroimaging technique Functional magnetic resonance imaging (fMRI) measures changes in blood flow and oxygen consumption in the brain associated with neuronal activity[2]. This technology utilizes magnetic fields and radio waves to produce images of the brain that highlight regions of activation during various cognitive tasks or in response to external stimuli. When neurons are active, they consume more oxygen and nutrients, leading to increased blood flow in the local region to meet the metabolic demands. Specifically, fMRI relies on the blood oxygen level-dependent (BOLD) contrast[3].

Dynamical Hierarchical Reinforcement Learning (DHRL) is an active and advanced field within reinforcement learning that incorporates hierarchical structures to tackle complex and dynamic environments[4][5]. The essential idea of DHRL is decompose task into subtasks, solving the task in divide-and-conquer manner, aiming to reduce the complexity of task and enhance learning efficiency that subject to long-term dependencies and sparse reward signal in reinforcement learning. Hierarchical task graph decompose task into subtasks, every vertice in graph represents a subtask and corresponding edge between subtasks indicates calling relationship. The dependencies and priorities among subtasks are concerned by DHRL and guarantee the accomplishment of a task. Maxq method decomposes the value function of a complex task into the sum of value functions of several subtasks. By doing so, it significantly reduces the number of state-action pairs to be considered, thereby improving learning efficiency[6]. FeUdal Networks[7] adopt a manager-worker architecture, where high-level managers set goals and low-level workers execute the specific actions to achieve these goals. This architecture enables the model to effectively handle long-term dependencies.

Meta-learning is an advanced paradigm in deep learning where models acquire the ability to generalize across different tasks by learning from past experiences. Instead of training a model from scratch for each new task, meta-learning enables rapid adaptation to new tasks with minimal data and computational resources[8].

The primary function of memory is to trade space for time while coordinating modules involved in cognitive processes and behavior. Although DHRL and meta-learning demonstrate advantages and potential in the self-adaptive learning process, the practice suggests that we shall better separate memorization from task management and decision-making as a middleware bridging them. The very observation is: first, a specific region in the brain is in charge of memory and plays a crucial role in this procedure of coordination and execution; second, the brain doesn't change the structure during the procedure but consume different amount of resources. To mimic this process, this paper introduces a design that integrate functionalities to a single system architecture via autoencoder memorization mechanism and tasks graph management. The main contributions of this article are summarized as below: 1) memorization mechanism is implemented via Autoencoder Hippocampus network; 2) motivation mechanism is implemented and embedded into the system via graph neural network.

2. Supervised learning based on classical control theory

Reinforcement learning techniques enables agent to learn optimal behavior strategies through interactions with environment[9] without the need for explicit human supervision or labeling observation-action pair datasets. This reduces the amount of human effort required for training and makes the process more scalable. Significant progress has been made in terms of algorithmic advancements, simulation environments, and computing power. Advantage Actor-Critic(A2C)[10], Asynchronous Advantage Actor-Critic(A3C)[11], and Proximal Policy Optimization(PPO)[12] are reinforcement learning algorithms that utilize different techniques to learn optimal policies for sequential decision-making problems. A2C and A3C utilize the Actor-Critic framework with an advantage function, while PPO focuses on achieving stable and efficient policy updates through a clipped surrogate objective function and an adaptive KL penalty term. The availability of high-quality simulation environments, such as OpenAI Gym[13], Isaac Gym[14], etc., has greatly accelerated reinforcement learning research. These environments provide a testbed for reinforcement learning algorithms, allowing researchers to quickly iterate and evaluate novel ideas. The increasing availability of powerful computing hardware, including GPUs and distributed computing clusters[15], has enabled researchers to train large-scale reinforcement learning models efficiently. In reinforcement learning, a policy function $\pi(s|\mathbf{W}) : s \rightarrow a$ is a mapping from states to actions, where s is a state, a is an action and $\mathbf{W}$ is parameters of policy neural network. It specifies the behavior of an agent within an environment. Given a state, the policy function determines which action the agent should take to maximize the cumulative

reward over time. The policy function serves as the agent's decision-making mechanism. The essence idea of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies[16].

Reinforcement learning often faces difficulties in grasping long-term dependencies and managing sparse reward signals. One effective strategy to boost learning efficiency, when feasible, is to incorporate knowledge of classical control. Many OpenAI classical control environments can be solved by classical control theory. Take the Lunar Lander environment, a classic environment that simulates the task of landing a spacecraft on the lunar surface, as an example. This problem can be solved by Proportional-Integral-Differential(PID) control[17]. Classical control theory provides a framework for designing and analyzing systems to achieve desired performance objectives through the use of feedback and advanced functional. Classical controller, such as PID controller, is a mapping $P(s) : s \rightarrow a$, calculates action vector from the observation or current state of the agent, which provides a set of ground truth for controlling the agent accomplish its task. Then, deep learning neural network can learn PID control through the objective function measuring the Euclidean distance between the output of policy function and PID controller, optimization algorithms attain the optimal policy function parameters $\mathbf{W}$:

$$\mathbf{W} = argmin(MSE(\pi(s|\mathbf{W}), P(s)))$$

The policy function approximates the PID controller after training:

$$\pi(s|\mathbf{W}) \approx P(s)$$

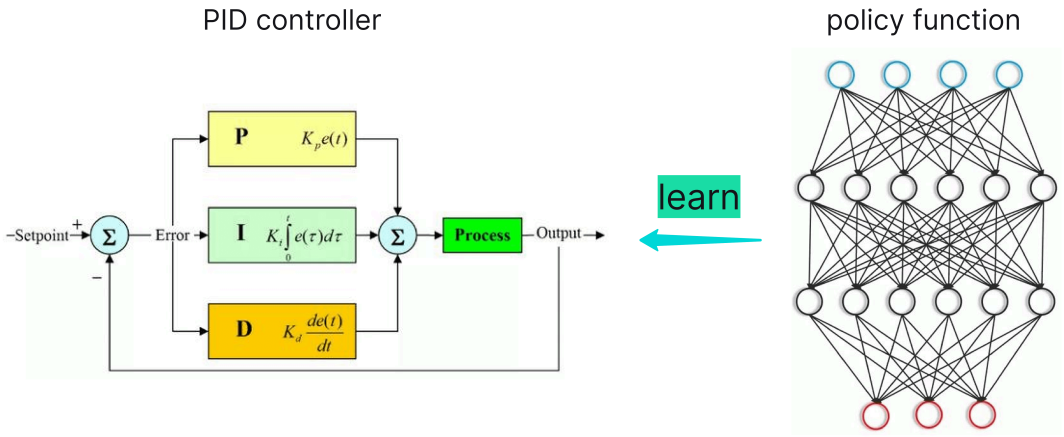

Figure 1. Available classical control can serve as the ground truth for actions and assist in training the policy function through supervised learning.

doi.org/10.32388/9C2UE7.2

Moreover, the nonlinearity of neural network can represent mapping from state to action obtained from classical control approach such as Bellman optimality equation[18][19][20] or Linear Quadratic Regulator. In summary, classical control if available can be provided as ground truth of actions and help train policy function in supervised learning manner.

3. Autoencoder Hippocampus Network

For different tasks, a same structured policy function but with different sets of parameters values can be trained. Saving and loading corresponding parameters values according to different tasks is a feasible but trivial solution. If we learn a wealth of skills, such as reading, cooking, riding, etc., then save isolated countable files in the brain will make the file system hardly manageable, which is very unlikely the case. Thus, abstract memorization, effective and efficient encoding and decoding procedure, plays a very essential role in learning, inference and execution.

Many neural networks are capable of memorization, including recurrent neural network, Long Short-Term Memory[21], gate recurrent unit[22], Hopfield network[23], autoencoder[24] and so forth. Memorization module concerns two key questions what information in what format shall be restored in memory and how to retrieve information from memory. An autoencoder is a type of neural network trained to learn efficient data encoding and decoding[25], consisting of an encoder that compresses the input data into a latent layer vector and a decoder that decompresses the vector back into a representation similar to the original input. The autoencoder is more suitable and preferable for the memorization module compared to other neural networks, as its encoder-decoder architecture makes retrieving stored information more convenient and straightforward.

In this design, parameters of policy function $\mathbf{W}$ are memorized by autoencoder hippocampus network. The encoder maps learned parameters value to a reduced dimensional latent layer vector called skill vector(or task vector) and the decoder maps latent layer vector, i.e. skill vector, to original parameters tensor. The latent layer of autoencoder provides an interface for retrieving restored information.

The memorization autoencoder is a mapping consisting of an encoder $E : R^m \to R^n$ and a decoder $D : R^n \to R^m$. Policy function's parameters $\mathbf{W}$ memorized by the autoencoder is

$$\mathbf{W}' = D(E(\mathbf{W}))$$

For a well-trained autoencoder, $\mathbf{W}'$ should closely resemble $\mathbf{W}$, and the degree of similarity between them can be quantified by the Euclidean distance metric. The essence of this design lies in utilizing the

autoencoder hippocampus network to derive a fixed-structure policy function, parameterized by a set $\{\mathbf{W}_i\}$, that is capable of executing actions tailored to various corresponding tasks.

The autoencoder hippocampus network memorizes the learned parameters tensor. Importantly, memorization process discussed here doesn't remember anything related to state, action, reward, etc, but the policy function's optimal parameters only. Given a skill vector, decoder of autoencoder recall (or generate) parameters tensor and assigned them to policy function. The policy function plays a role of capacitor, what parameters tensors are filled in is task oriented.

The skill vector $\mathbf{S}$ is encoded policy function parameters via the encoder:

$$\mathbf{S} = E(\mathbf{W})$$

The process of decoder recalling parameters $\mathbf{W}$ from the skill vector $\mathbf{S}$ can be written as:

$$\mathbf{W} = D(\mathbf{S})$$

Then, the action of policy function upon state s reads:

$$a = \pi(s|\mathbf{W}) = \pi(s|D(S)) = \pi(s|D(E(\mathbf{W})))$$

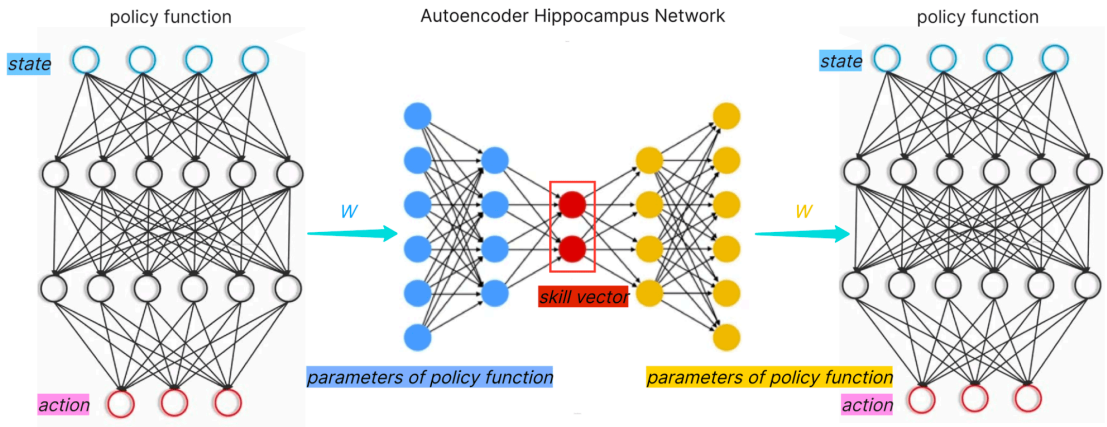

Figure 2. The parameters of the policy function can be encoded, stored, and retrieved via an autoencoder hippocampus network.

It is impossible in real world to enumerate endless tasks or skills, that is, the learning and execution process shall be extensible and inferable. A person who has learned how to speak does not necessarily know how to ride a bicycle. Conversely, someone who knows how to ride a bicycle is quite likely to

possess the skills to ride a motorcycle as well. The skill vectors within the latent layer of an autoencoder embody the concept of Euclidean distance, where a shorter distance signifies a higher degree of skill relevance and, consequently, a greater similarity in parameter values. The memorization module of an autoencoder is capable of inference, including zero-knowledge inference. Provided a skill vector, no matter whether it learned before, as an input for decoder of autoencoder, it will map to or generate a set of corresponding parameters value. Then, for execution phase of a task, the policy function loads corresponding parameters value and behaves accordingly.

The design of the memory-embedded policy function involves maintaining a fixed architecture for the policy function while utilizing a memory module to store the parameter values of the policy function during the learning process. During the exploitation procedure, the memory module is used to recall and assign the appropriate parameter values to the policy function. This approach allows the policy function neural network to maintain a consistent architecture but also to possess dynamic parameters that are tailored to specific tasks.

4. Execution via traversing skill vector graph

The brain has the capability to divide a task to a set of subtasks based on common sense, knowledge, inference and so forth. Some subtasks may have higher priority than the others and form a graph structure in nature. The homeomorphic relationship between subtasks and skill vectors exhibits a one-to-one, bijective mapping. Equivalently, breaking down a task into a graph of subtasks is akin to constructing a graph for skill vectors. Let $G = (\mathbf{S}, e)$ be a task graph with subtask vertices $\mathbf{S}$ and edges e. Graph Neural Networks(GNNs)[26][27] capture the dependencies and structures within skill vector graph[28]. With the autoencoder, encoded skill vector help recall and deploy parameters tensor to policy function, providing an interface for tasks graph management module. When a complicated task can be divided to a topological structure of vector skills set $\{\mathbf{S}_i\}$, the accomplishment of the task can be executed according to skill vector graph traversal:

$$a_i = \pi(s|W_i) = \pi(s|D(\mathbf{S}_i))$$

Putting an elephant into a refrigerator requires three steps: 1) Open the refrigerator; 2) Put the elephant inside; 3) Close the refrigerator. Nodes in skill vector graph are embedded in a Euclidean space, and edges can be represented by vector between connected skill vectors as well. Skill vector graph occupies both Euclidean properties and graph connections typically combines the characteristics of geometric space

and topological representations. The spectrum of skill vector graph on Euclidean data refers to the analysis of graph structures and their corresponding eigenvalues derived from key matrices such as the Laplacian matrix when working with data embedded in Euclidean space[29].

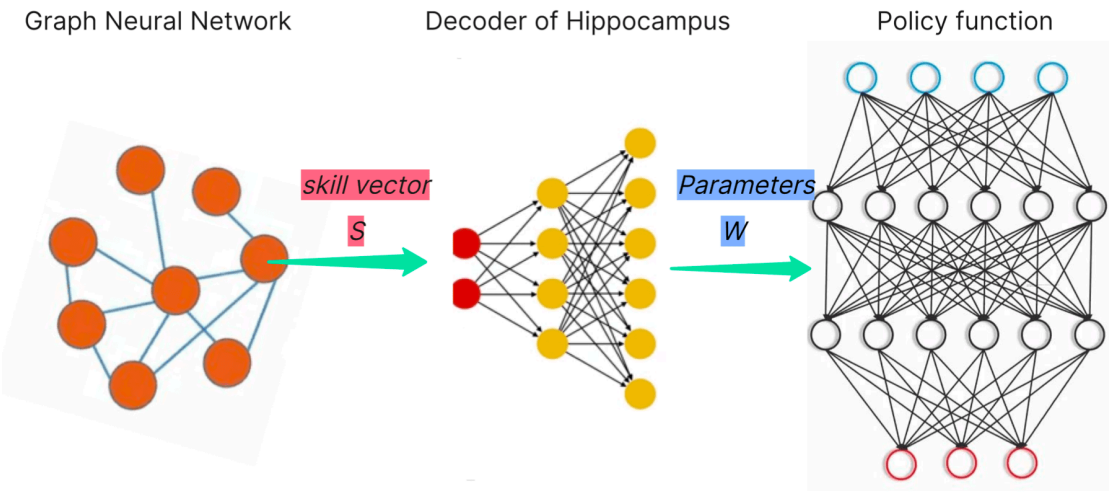

Figure 3. The latent layer of an autoencoder hippocampus network provides an interface for retrieving the parameters of the policy function. By traversing the graph of skill vectors, a complex task can be completed, and a corresponding sequence of behaviors can be generated.

The dynamics of traversing skill vector graph requires to know subtask at hand and determine the phase of task. The cognitive function neural network, a mapping $C : s \to S_i$, can be trained to determine subtask confronted, select the very skill vector and drive the dynamics of traversing skill vector graph:

$$S_i = C(s)$$

Cognitive function recognizes if the refrigerator is open or not and whether elephant is inside refrigerator to decide the skill vector. The state, the input of policy function, undergoes a bifurcation of computing procedure and ultimately determines the parameters of policy function as well. Variable parameterized policy function assigned by decoder of autoencoder hippocampus network D and managed by cognitive function C, enhance the capability of control procedure.

$$a_i = \pi(s|W_i) = \pi(s|D(\mathbf{S}_i)) = \pi(s|D(C(s)))$$

doi.org/10.32388/9C2UE7.2

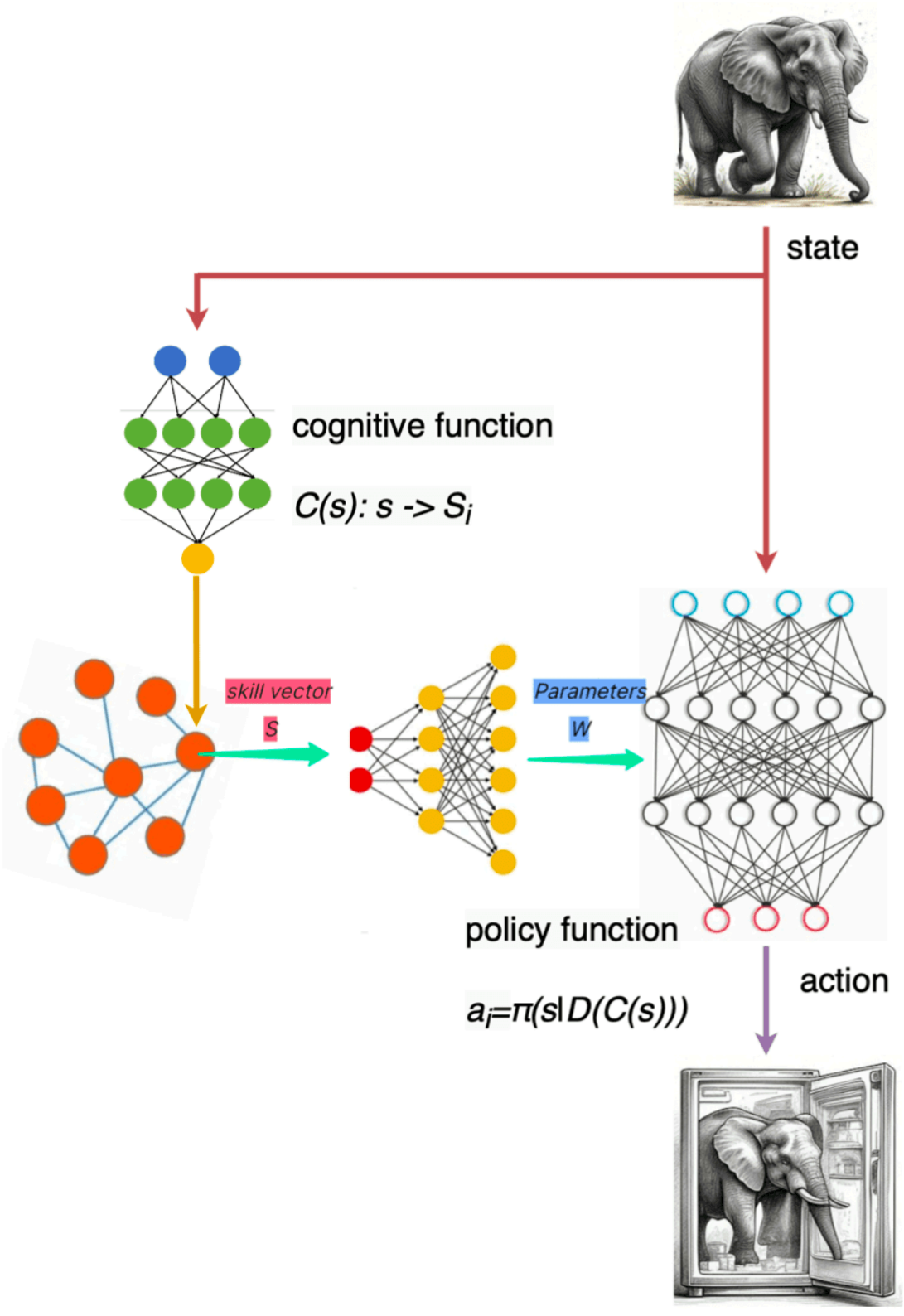

Figure 4. How to put an elephant into a refrigerator

doi.org/10.32388/9C2UE7.2

In summary, the minimal prerequisites for generating a sequence of control encompass: 1) an autoencoder hippocampus mechanism capable of memorizing and recalling parameters of the policy function; 2) construct skill vector topological graph for the task; and 3) dynamics related to traversing the skill vector graph.

5. Conclusion

Dynamically integrating an autoencoder hippocampus network and tasks graph neural network management into the system, parameters tensors can be assigned to the policy function through the autoencoder hippocampus network. This separation of memorization and task execution from decision-making fosters a more interpretable system tailored for diverse tasks. With the incorporation of the autoencoder hippocampus network, the system can demonstrate rich and diverse dynamic behaviors.

Statements and Declarations

Ethics and Morality

THIS TECHNOLOGY CAN BE HAZARDOUS! ITS IMPLEMENTATION MUST ADHERE TO STRICT SAFETY PROTOCOLS!

References

1. ^*Note: Eichenbaum, H., Dudchenko, P., Wood, E., Shapiro, M. and Tanila, H., 1999. The hippocampus, memory, and place cells: is it spatial memory or a memory space?. Neuron, 23(2), pp.209-226. Cited by: §1.*
2. ^*Note: Heeger, D.J. and Ress, D., 2002. What does fMRI tell us about neuronal activity?. Nature reviews neuroscience, 3(2), pp.142-151. Cited by: §1.*
3. ^*Note: Logothetis, N.K., Pauls, J., Augath, M., Trinath, T. and Oeltermann, A., 2001. Neurophysiological investigation of the basis of the fMRI signal. nature, 412(6843), pp.150-157. Cited by: §1.*
4. ^*Note: Haarnoja, Tuomas, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In International Conference on Machine Learning, pp. 1851-1860. PMLR, 2018. Cited by: §1.*
5. ^*Note: Stulp, Freek, and Stefan Schaal. Hierarchical reinforcement learning with movement primitives. In 2011 11th IEEE-RAS International Conference on Humanoid Robots, pp. 231-238. IEEE, 2011. Cited by: §1.*

6. ^ *T. G. Dietterich (2000) Hierarchical reinforcement learning with the maxq value function decomposition. J ournal of artificial intelligence research 13, pp. 227–303. Cited by: §1.*
7. ^ *P. Dayan and G. E. Hinton (1992) Feudal reinforcement learning. Advances in neural information processin g systems 5. Cited by: §1.*
8. ^ *R. Vilalta and Y. Drissi (2002) A perspective view and survey of meta-learning. Artificial intelligence revie w 18, pp. 77–95. Cited by: §1.*
9. ^ *Note: Kaelbling, L.P., Littman, M.L. and Moore, A.W., 1996. Reinforcement learning: A survey. Journal of arti ficial intelligence research, 4, pp.237-285. Cited by: §2.*
10. ^ *Note: Chu, T., Wang, J., Codecà, L. and Li, Z., 2019. Multi-agent deep reinforcement learning for large-scale t raffic signal control. IEEE Transactions on Intelligent Transportation Systems, 21(3), pp.1086-1095. Cited by: §2.*
11. ^ *Note: Du, J., Cheng, W., Lu, G., Cao, H., Chu, X., Zhang, Z. and Wang, J., 2021. Resource pricing and allocation in MEC enabled blockchain systems: An A3C deep reinforcement learning approach. IEEE Transactions on Network Science and Engineering, 9(1), pp.33-44. Cited by: §2.*
12. ^ *Note: Mazyavkina, N., Sviridov, S., Ivanov, S. and Burnaev, E., 2021. Reinforcement learning for combinatori al optimization: A survey. Computers & Operations Research, 134, p.105400. Cited by: §2.*
13. ^ *Note: Palanisamy, P., 2018. Hands-On Intelligent Agents with OpenAI Gym: Your guide to developing AI ag ents using deep reinforcement learning. Packt Publishing Ltd. Cited by: §2.*
14. ^ *Note: Serrano-Munoz, A., Chrysostomou, D., Bøgh, S. and Arana-Arexolaleiba, N., 2023. skrl: Modular and f lexible library for reinforcement learning. Journal of Machine Learning Research, 24(254), pp.1-9. Cited by: § 2.*
15. ^ *Note: Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M. and Fox, D., 2018, October. Gpu-accel erated robotic simulation for distributed reinforcement learning. In Conference on Robot Learning (pp. 270 -282). PMLR. Cited by: §2.*
16. ^ *R. S. Sutton and A. G. Barto (1999) Reinforcement learning: an introduction. Robotica 17 (2), pp. 229–235. Ci ted by: §2.*
17. ^ *Note: Knospe, Carl. PID control. IEEE Control Systems Magazine 26, no. 1 (2006): 30-31. Cited by: §2.*
18. ^ *Note: Bellman, Richard, and E. Stanley Lee. Functional equations in dynamic programming. Aequationes mathematicae, 17, no. 1 (1978): 1-18. Cited by: §2.*
19. ^ *Note: Peng, Shige. A generalized dynamic programming principle and Hamilton-Jacobi-Bellman equation. Stochastics: An International Journal of Probability and Stochastic Processes, 38, no. 2 (1992): 119-134. Cited*

by: §2.

20. ^*Note: Dreyfus, Stuart. Richard Bellman on the birth of dynamic programming. Operations Research, 50, n o. 1 (2002): 48-51. Cited by: §2.*
21. ^*Note: Sherstinsky, Alex. Fundamentals of recurrent neural network (RNN) and long short-term memory (L STM) network. Physica D: Nonlinear Phenomena 404 (2020): 132306. Cited by: §3.*
22. ^*Note: Gao, Yuan, and Dorota Glowacka. Deep gate recurrent neural network. In Asian conference on machi ne learning, pp. 350-365. PMLR, 2016. Cited by: §3.*
23. ^*Note: Hopfield, John J. Neural networks and physical systems with emergent collective computational abili ties. Proceedings of the national academy of sciences 79, no. 8 (1982): 2554-2558. Cited by: §3.*
24. ^*Note: Han, Kuan, Haiguang Wen, Junxing Shi, Kun-Han Lu, Yizhen Zhang, Di Fu, and Zhongming Liu. Vari ational autoencoder: An unsupervised model for encoding and decoding fMRI activity in visual cortex. Neu roImage 198 (2019): 125-136. Cited by: §3.*
25. ^*Note: Zhai, J., Zhang, S., Chen, J. and He, Q., 2018, October. Autoencoder and its various variants. In 2018 IEE E international conference on systems, man, and cybernetics (SMC) (pp. 415-419). IEEE. Cited by: §3.*
26. ^*Note: Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. and Sun, M., 2020. Graph neural net works: A review of methods and applications. AI open, 1, pp.57-81. Cited by: §4.*
27. ^*Note: Zhang, Zhen, Chen Xu, Kun Liu, Shaohua Xu, and Long Huang. A resource optimization scheduling model and algorithm for heterogeneous computing clusters based on GNN and RL. The Journal of Superco mputing 80, no. 16 (2024): 24138-24172. Cited by: §4.*
28. ^*Note: Yao, Zonggui, Jun Yu, Jian Zhang, and Wei He. Graph and dynamics interpretation in robotic reinforc ement learning task. Information Sciences 611 (2022): 317-334. Cited by: §4.*
29. ^*B. Mohar, Y. Alavi, G. Chartrand, and O. Oellermann (1991) The laplacian spectrum of graphs. Graph theor y, combinatorics, and applications 2 (871-898), pp. 12. Cited by: §4.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.