Task-Invariant Agent using Model Dynamics

Gene Chung[*], Alfred Cueva[†], Taehong Kim[‡], Sumin Ye[†]
[*]Department of Mechanical Engineering, Seoul National University, Seoul, Korea Email: piggene00, alfred11@snu.ac.kr, kimtaehong07, ysm3868@snu.ac.kr

***Abstract*—In this study, we introduce the Task Invariant Agent (TIA) network that aims to enable agents to efficiently adapt and perform across a diverse set of tasks. Our proposed model incorporates a unique task abstraction process, allowing the network to quickly generate meaningful task representations that serve as essential inputs for decision-making. We plan to assess the TIA network in different task variations of the CartPole environment, where the agent's objective is to perform well on broad of tasks. We expect that the TIA network will not only perform well on tasks it has been trained on, but also rapidly adapt to new, previously unseen tasks.**

I. Introduction

Over the years, artificial intelligence has made significant advancements, with some algorithms demonstrating practical performance that rivals or surpasses human intelligence, especially in domains like reinforcement learning [1]. Also recent research has even proposed a promising approach of developing modular policies for multi-task and multi-robot systems [2] [3]. This approach combines reinforcement learning and neural networks to train a low-level control policy that can be reused across different tasks and robots. The high-level task-specific policy is then learned through reinforcement learning, which selects and combines the appropriate low-level control policies.

Although such method provides possibility and performance for learning variety of tasks simultaneously, this model still lacks in efficient probability of need to update weights or even make a new network every time new tasks are added to situation. In this research, we propose Task Invariant Agent (TIA) network, which shares weight through variety of tasks and could also perform well on new untrained tasks. The objective of this network is to develop an agent that can perform well across a range of tasks, similar to how humans can perform diverse tasks with ease. By using model abstraction that will be discussed at related works and considering it as an input of policy, the proposed task invariant policy network aims to create a more robust and adaptable agent that can quickly learn new tasks and generalize to different environments.

In this study, we propose the (TIA) network, an innovative approach to multi-task reinforcement learning. We plan to test this network in the cart-pole balancing environment, where the agent's tasks will range from maintaining balance at a desired position. We anticipate that the TIA network will exhibit robust performance across known tasks and swiftly adapt to new ones.

II. Related Works

The idea of network which could be applied in to variant tasks isn't really new area of research. This section provides quick summary of each research's main ideas with achievements and limitations. Also, representing a model feature through latent vector, and using it as a input of policy network is also covered through variety of researches [4] [5] [6].

A. HDQN

Hierarchical Deep Reinforcement Learning (HDQN) [7] is a variant of the popular reinforcement learning technique that enables agents to learn and solve complex tasks by breaking them down into smaller subtasks, or subgoals. HDQN involves the use of multiple interconnected deep neural networks, with each network responsible for learning and solving a specific subgoal. This article gave the idea of learning multiple tasks, but still which was sub-task for ultimate goal task.

B. World Models

The"World Models" article by David Ha and Jürgen Schmidhuber [4] proposes a novel approach to reinforcement learning by using a combination of three different components: a vision module, a memory module, and a controller module.

Vision module extracts feature vector from given observation(image data) using VAE(Variational Auto Encoder) Network, and combining this with memory module which combined RNN and MDN (Mixture Density Network) [8] to successfully predict future state from given action and state estimation. By using hidden state vector of RNN, control module also possess information of model, enabling policy network to consider such information.

This research gives intuition of applying model's information as an input of policy network, but still suffers from its complex structure, and also not quite applicable for variant kinds of tasks.

C. Rapid Motor Adaptation

"Rapid Motor Adaptation" [6] enables legged robot to quickly adapt and perform well on unknown terrain and physics. Only by experiencing several minutes of new environment with unknown physics, this paper succeed for legged robot to quickly perform walking. This research shares fundamental idea of optimizing latent vector to rapidly adapt for unknown environment rather than updating weights of network, which is way more efficient in the matter of time.

Method used in this research is to quickly estimate latent vector representing of model's dynamics such as joint-friction, rigid body mass etc, from sequence of state and action pairs received from short experience. By using this latent representation of model's dynamics as an input to a policy network, robot could perform well through variety of terrains and could also perform well on untrained new environment.

D. DeepMPC

”DeepMPC: Learning Deep Latent Features for Model Predictive Control” [9] is a research paper that introduces a deep learning-based approach for enhancing the performance of model predictive control (MPC) by learning deep latent features.

The paper proposes a novel framework that combines deep neural networks and MPC to learn latent representations of system dynamics directly from data. By leveraging the expressive power of deep neural networks, DeepMPC aims to capture complex and nonlinear relationships in the system dynamics, allowing for more accurate predictions and control actions.

The key idea behind DeepMPC is to learn a latent space representation that captures the essential features of the system dynamics. This is achieved by training an autoencoder, which is a type of neural network architecture that learns to encode the input data into a lower-dimensional latent space and decode it back to the original space. The encoder part of the autoencoder learns to extract meaningful features from the input data, while the decoder part reconstructs the input from the learned latent representation.

E. Sequence to Sequence Learning with Neural Networks

Seq2Seq (Sequence-to-Sequence) [10] models are a type of neural network architecture widely used for various natural language processing (NLP) tasks, such as machine translation, text summarization, and conversational agents. This article provides a summary of the key concepts and applications of Seq2Seq models.

Seq2Seq models consist of two main components: an encoder and a decoder. The encoder processes an input sequence, such as a sentence, and transforms it into a fixed-length vector called the ”context vector” or ”thought vector.” This vector is designed to capture the semantic meaning of the input sequence. The decoder takes this context vector and generates an output sequence, often of a different length or in a different language, based on the learned representation.

III. Task Invariant Policy Network

We propose TIA network, providing policy Network that could perform well through variant of tasks and also for unknown tasks with quick adaptation as [6] with experiencing several minutes with new environment. Algorithm of our research is composed with 3 main networks of Model Predictor, Encoder, And main policy network, which is modification of DQN [11]. Our algorithm uses latent vector representation of task z, which will be used as an input of policy's network. Training stage of TIA network needs to make good policy network but also need to learn how to represent the task as a vector.

A. Structure of Networks

- **Policy Network** is modification of DQN network. Input of this network will be state vector at given time s_t and latent vector of model estimation z. So action of given time will be received as $a_t = \pi(s_t, z)$. Actual Output of this network will be estimated Q values of given state s_t, task z and for possible actions which corresponds to each node of output layer.
- **Encoder** network's purpose is to generate model representation from given sequences of experiences. Sequence of experiences τ_t contains recent m sequences $t-1$ to $t-m$ of state, action and received action pairs $\{(s_{t-1}, a_{t-1}, r_{t-1}), ..., (s_{t-m}, a_{t-m}, r_{t-m})\}$. Encoder will generate latent estimation of task as $z_t = \mu(\tau_t)$. This encoder gets sequential data as an input like seq2seq auto-encoder. [10] Since this latent estimation z_t would have high variation through time, using this value directly as an model estimation z would be inappropriate. Thus we will accumulatively update latent vector $z = (1-\alpha)z + \alpha z_t$.
- **Model Predictor** Model Predictor is used to give valuable meaning to latent vector z. This model predictor is also a robust Model Predictor that could be applied to a variant of tasks. From past state and action pair s_{t-1}, a_{t-1} and latent estimation of task z, Model Predictor ψ estimates next state, reward, and termination info $s_t, r_{t-1}, t_t = \psi(s_{t-1}, a_{t-1}, z)$.

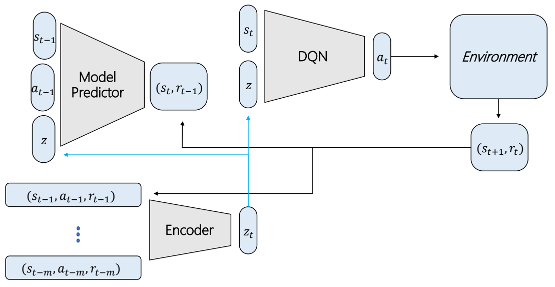

Fig. 1. Structure of TIA Network at training stage

B. Overall Algorithm

Main algorithm uses replay buffer conceived from DQN [11]. Every step, choose random task agent expects latent representation z calculated from Encoder from given history sequence tau. With using this vector z and using it as an input of DQN network, agent guesses optimal-action. With epsilon-greedy action selecting process, between the optimal-action and random action, agent choose action a. After applying this action a, we observe next state reward and termination info s', r, t. Update this sequence of history tau and append $\{s, a, r, s', t, z\}$ to replay buffer. Then every update frequency of each networks, selecting sequences from replay-buffer, we update network's weights following loss below. Structure of this algorithm is shown in Fig. 1.

- **Policy Loss** Policy Loss uses Q value loss with n-step TD Learning. Loss follows average of total m, n-step TD losses sampled from replay buffer. With given $\{s, a, r, s', z, t\}$ sample, we estimate future trajectories

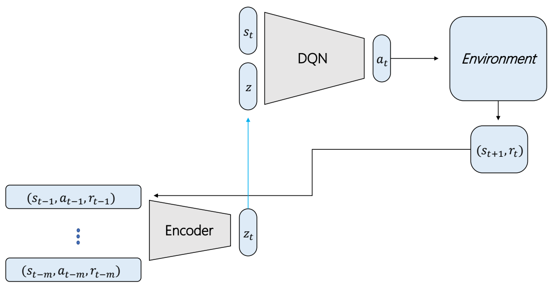

Fig. 2. TIA Network quickly adjusting for unknown task

$s'', a'', r'', t'', s''', a''', r''', t''', \ldots$ obtained by using model predictor and using behavior policy as epsilon-greedy from DQN network. By using this estimated future trajectories inside n-step TD learning, we could obtain the TD-error loss.

$$TD\ error = \sum_{i=1}^{n} \gamma^{i-1} r^{(i)} + \gamma^{i} Q(s^{(n)}, a^{(n)}) - Q(s, a) \tag{1}$$

$$\mathcal{L}_{policy} = \frac{1}{number\ of\ sample} \sum_{sample} TD\ error^2 \tag{2}$$

- **Predictor Loss** For predictor's loss, from sampled $\{s, a, r, s', z, t\}$, we use input of state, action and latent representation s, a, z to estimate reward, next state and termination info r, s', t. Loss function is simply calculated as
$$\mathcal{L}_{predictor} = MSE(r, s', t | \mu(s, a, z)) \tag{3}$$
- **Encoder Loss** For encoder/decorder's loss, we use to regenerate history sequence tau in to form of vertor z, and reconstruct it back to original sequence tau. SO the loss would be simple as below
$$\mathcal{L}_{seq2seq} = MSE(tau | \psi(tau)) \tag{4}$$

The overall architecture's algorithm is shown in Pseudo Algorithm 1 is in Appendix A.

C. Real-World Deployment

Once the TIA network has been sufficiently trained, it can be deployed in real-world scenarios. For new tasks, the agent rapidly adapts to the environment by using the encoder network to generate an appropriate latent task representation z. The policy network then uses this task representation to make decisions, while the Model Predictor helps the agent anticipate the consequences of its actions, thus enabling efficient and robust performance across a wide variety of tasks. Figure representation of deployment is shown at Fig 2.

IV. Experimental Setup

The environment used for experimentation is a variation of CartPole-v0 from openAI gym [12]. The task chosen was for the CartPole to stay at a desired position (at the peak for convenience) in some range. We deploy it for six simultaneous CartPoles. The main reward for the agent was based on the desired position. We deployed two reward distributions, (5) and (6), for Experiment 1 and 2 respectively.

$$r = 1 - a \cdot int(|b(x - x_d)|) \tag{5}$$

$$y = e^{-\frac{(x - x_d)^2}{2\sigma^2}} \tag{6}$$

Where each reward gives maximum 1 at desired cart position. Different ranges for the displacement were tried. The wider the range was, there was more variance in the episodic rewards which was due to the random initialization of the desired positions within said range. The respective reward function used in training (tuned hyper-parameters) is shown in Fig 3

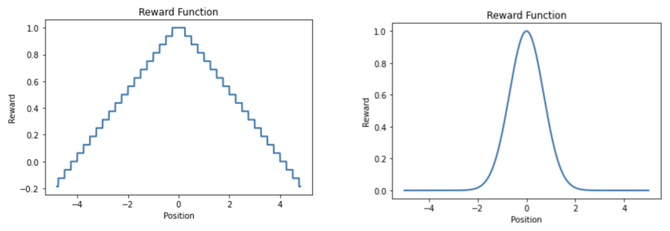

Fig. 3. 2 kind of reward used each for experiment 1 and 2

V. Result

A. Experiment 1

The reward function used is Fig 3 using equation (5). The distribution of rewards itself has a large variance, specially at the limits, making the tracking poor. The gamma was 1 and there wasn't termination when trying to stay at a desired position. This set-up wasn't learning properly and didn't meet the expected results. When deployed, the tests with negative initial desired positions produced more promising maximum rewards while the ones starting at positive initial desired positions weren't as good.

In Fig 4 we see the tracking of the actual position against the desired position of the Cart Pole. Initially the position oscillates but around 600 it converges to the desired position (-1) but eventually overshoots. Then, we conclude Experiment 1 didn't achieve the desired task efficiently.

B. Experiment 2

The initial reward distribution (5) didn't allow the agent to train properly when the initial desired positions where positive. To alleviate that, we used a Gaussian Distribution since it can model most random distributions at infinite (6). We found the most optimal performance at $x_d = 0$ and $\sigma = 0.7$ as hyper-parameters reflected in Fig 3

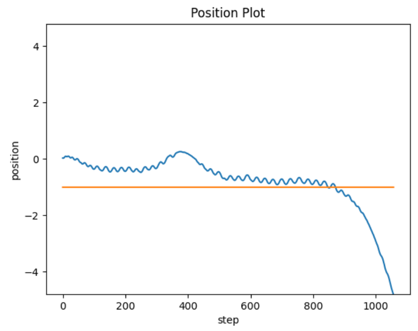

Fig. 4. Position Tracking performance used in Experiment 1. Orange line indicates desired position where blue line was real cartpole position

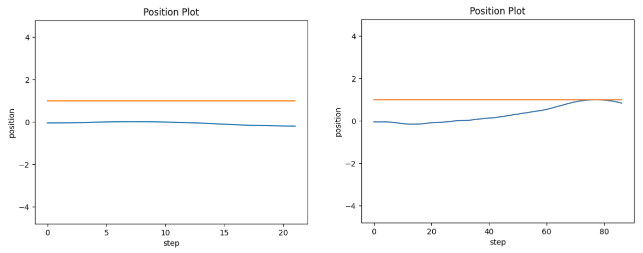

Fig. 5. PPosition Tracking performance used in Experiment 2. Orange line indicates desired position where blue line was real cartpole position

In the first iterations the Cart Pole doesn't converge completely to the desired position. Eventually, after about 100 episodes, the agent performs better tracking shown in Fig 5.

VI. Conclusion

We propose the Task Invariant Agent (TIA) network, a multi-task reinforcement learning approach. The TIA network consists of three main components: a Policy Network, an Encoder, and a Model Predictor. The Policy Network is a modified version of DQN that takes both the state and a latent task representation as inputs to select actions. The Encoder generates the latent representation of the task from sequences of experiences. The Model Predictor estimates the next state, reward, and termination information based on the previous state, action, and task representation. The experimental setup was for a Cart-Pole to stay at a desired position.

As found in results of experiments above, we could successfully conclude that TIA network was able to adapt in newly seen task when performed in similar tasks before. Experiment we had done wasn't broad enough to cover the diversity we wanted from the first sight, which leaves several potential improvements. Nevertheless, we could still observe that sequence of state, action and reward could represent the overall reward and dynamics of given environment and also could be used to estimate the reward and dynamics of given environment.

VII. Appendix

A. Appendix A : Pseudo Algorithm of TIA Network

Algorithm 1 TIA Network

```
1: Initialize $\tau_i \leftarrow$ empty list for $1 \leq i \leq N$
2: while $step \leq total\_step$ do
3:     select $i$ randomly from 1 to $N$
4:     $\hat{z}_i \leftarrow \mu(\tau_i)$
5:     $z_i \leftarrow (1-\alpha)\cdot z_i + \alpha \cdot \hat{z}_i$
6:     $a \leftarrow \pi(s, z_i)$
7:     observe $s', r, t$
8:     store $\{s, a, r, s', t, z\}$ to Replay Buffer
9:     append $(s, a, r, t)$ to $\tau_i$
10:    $t_i \leftarrow t_i + 1$
11:    $step \leftarrow step + 1$
12:    if $step \equiv 0 (\mathrm{mod}\ update\_epoch\_DQN)$ then
13:        sample $\{s, a, r, s', t, z\}$ from Replay Buffer
14:        update weight of $\pi$
15:    end if
16:    if $step \equiv 0 (\mathrm{mod}\ update\_epoch\_Predictor)$ then
17:        sample $\{s, a, r, s', t, z\}$ from Replay Buffer
18:        update weight of $\mu$
19:    end if
20:    if $step \equiv 0 (\mathrm{mod}\ update\_epoch\_Encoder)$ then
21:        update weight of $\psi$
22:    end if
23: end while
```

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[2] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2169–2176.

[3] R. Yang, H. Xu, Y. WU, and X. Wang, "Multi-task reinforcement learning with soft modularization," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4767–4777.

[4] D. Ha and J. Schmidhuber, "World models," 2018.

[5] L. Zhang, G. Yang, and B. C. Stadie, "World model as a graph: Learning latent landmarks for planning," 2021.

[6] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," 2021.

[7] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," 2016.

[8] C. M. Bishop, "Mixture density networks," 1994. [Online]. Available: https://publications.aston.ac.uk/id/eprint/373/

[9] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control." in *Robotics: Science and Systems*, L. E. Kavraki, D. Hsu, and J. Buchli, Eds., 2015. [Online]. Available: http://dblp.uni-trier.de/db/conf/rss/rss2015.htmlLenzKS15

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.