Boosting Few-Shot Meta Q-Learning with Reptile Optimization Algorithm

Spandan Rout, Milinda Basavaraju, Nithish Balu

spandanrout@icloud.com, miluvarsha@gmail.com, nithishmoritz@gmail.com

I. INTRODUCTION

Highly complex computational models excel when ample training data is available anddespite their computational intensity, the training process can be notably protracted. In contrast, human cognition exhibits rapid learning and the ability to make extrapolations based on only a scant amount of data. The challenge of effectivelearning under data scarcity remains a critical issue in the realm of computational intelligence. In this study, we seek to empirically validate the efficacy of few-shot learning within the domain of decision-making optimization. Through experimentationwithin a simulated control environment, we aim to showcase the potential of this approach.

II. BACKGROUND

Meta-learning is the machine learning field pertaining to the problem of "learning howto learn." Few-shot learning is the sub-field that addresses how a machine can learn when only given a few samples. Traditionally, a computational system undergoes extensive familiarization on a suite of closely related computational challenges and subsequently encounters a distinct yet related challenge, with access to only a limited number of novel data points. Essentially, the objective of few-shot learning revolves around enabling robust generalization when constrained to take merely a few k gradient descent steps. There have been three major approaches for few-shot learning:

1. Using prior knowledge about similarity: Whereas ML models cannot discriminate between classes not present in the training data, few-shot learning enables models to separate classes that are not present, allowing the models to separate unseen classes. Examples of this include Siamese Networks (discriminating two unseen classes) [2]and Matching Networks (discriminating multiple unseen classes) [7].
2. Exploiting prior knowledge: By using the structure and variability of the data, modelscan generalize recurring patterns. This is especially prevalent in computer vision tasks like object detection [3].
3. Constraining the learning algorithm: By choosing specific hyperparameters or choos- ing learning update rules that converge quickly, few-shot learning enables models to generalize well on small amounts of training data. Examples of these include Reptile [5] (an application of the Shortest Descent algorithm) and SNAIL [4] (whichuses temporal convolutions to aggregate contextual information).

The third approach is the focus of our study. Typically, such few-shot learning algorithms are general; they only specify the way in which gradient updates are made. In the case of the Algorithm-M for Rapid Adaptation (AMRA), the researchers were able to generalize the algorithm to work on regression, classification, and reinforcement learning problems, and showed it working on MiniImageNet and Mujoco's HalfCheetah and Ant environments [1]. MAML presented a problem, however, in that it required Hessian computations, rendering it an intensive algorithm for machines with limited memory and power. Subsequent studies such as the first- order MAML (FOMAML) and Reptile tried to alleviate this problem by taking an initial linear estimation of the Hessian matrix and computing the average stochastic gradient across all tasks. While Reptile showed promising results for classification using the Omniglot and MiniImageNet datasets and CNN models, the researchers showed "negative results" on Reinforcement Learning [5]. This presents an opportunity to apply Reptile on reinforcement learning tasks. Recent developments in this field have also studied the effect of weight sharing based on the ordering of the tasks across time [6].

III. APPROACH

The goal of our project is to understand and explore limited-data adaptability withinthe Decision Optimization domain. At first, we tried to replicate the MAML paper by running Mujoco's HalfCheetah and Ant environments. However, we found the task too computationally complex and time-intensive, since the task was in continuous space. Thus, to implement few-shot learning, we used OpenAI Gym's CartPole environment.

[8] We then built and trained a Deep Q-Network on a set of related tasks and asked themodel to generalize its learning on a hold-out set of tasks based on a few trainingsteps.

IV. SIMULATION ENVIRONMENTS

We modified OpenAI Gym's CartPole environment to create unique but related simu- lations. The Pendulum-Cart System, consists of a pole connected to a mobile cart via apassive linkage mechanism. This system operates on a smooth, low-friction track. Initially, the system is in an unstable state, necessitating training to enable the control mechanism to manage the pole's balance. Control is achieved by applying a directional force of +1 or -1 to the cart, with the objective being to maintain the pole's vertical posi- tion. Performance is evaluated through a continuous reward mechanism, and episodes terminate under specific conditions. By varying the gravity and pole mass, different but related tasks were created. Attached is a summary of the simulationenvironments:

Environment Name	Max Episode Steps	Reward Threshold	Standard Gravity (g)	Mass of Pole (kg)
CartPole-v0	200	195	9.8	0.1
CartPole-v2	200	195	20	0.1
CartPole-v3	200	195	30	0.1
CartPole-v4	200	195	40	0.1
CartPole-v5	200	195	50	0.1
CartPole-v6	200	195	60	0.1
CartPole-v7	200	195	10	0.2
CartPole-v8	200	195	10	0.4
CartPole-v9	200	195	10	0.8
CartPole-v10	200	195	10	1.0

Our Deep Q-Network model was trained randomly on a subset of the above tasks,with the exception of CartPole-v6 and CartPole-v10, which were held out as a validation set to test our few-shot learning implementation.

V. NEURAL NETWORK ARCHITECTURE

Deep Q Networks Advanced Q-Optimization is a conventional model-independent technique within the domain of Decision Optimization. Given a Q-function: S ⇥ A ! R, we can construct a policy that maximizes our long-term rewards:

$$⇡ = \text{argmax}q(s, a)$$

For large state or action spaces, a non-linear function approximator like a neural network can be useful. Then, we establish an update rule based on the Bellman equation:

$$q(s, a) = Rt+1(s, a)+ ,\text{max}aq(st+1, at+1)$$

However, the use of non-linear function approximators to represent the Q-function introduces instabilities in the model. These instabilities stem from correlations between the target value and the actual Q-function estimates. To achieve an ideal policy, our objective is the minimization of the L divergence between the q-function estimation and the target Q-value yˆ(s, a):

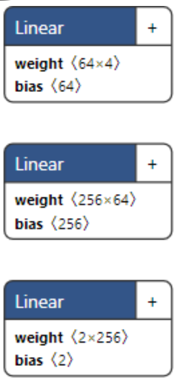
$$\text{loss} = Q(s, a) - \hat{y}(s, a)$$

To reduce these instabilities, methods like experience replay introduce a replay buffer.

With a replay buffer, it is then possible to collect prior transitions {s, a ,s , R(s, a)}

And sample from prior transitions before any updates are made.

Parameters We use the following DQN with 3 fully connected layers to play overthe various CartPole environments:



#____________________DQN PARAMS______ #GAMMA = 0.99

BATCH_SIZE = 16

memory_replay = Replay_Buffer(10000)episodes = 500

target_updates = 10

state_dim = 4

action_dim = 2

h_dim1 = 64

h_dim2 = 256

#____________________REPTILE HYERPARAMETERS____#

meta_step_size_final = .1

meta_step_size = .1 k_iters = 200 # Tunable

Fig. 1: Deep Q-Network Architecture

VI. Reptile Algorithm

Reinforcement Learning tasks are often complex and require significant computationalresources. Because Reptile does not require us to compute Hessian vector products, it should be suitable for quickly learning in Reinforcement Learning. Reptile is meta- learning algorithm that work by first sampling a task, running SGD or Adam for k steps, and then updating the weights of the meta-learner. Like many meta-learning algorithms, the objective of Reptile its expected loss over the distribution of given

tasks, where {Uk}

denotes the k steps of the descent algorithm:

minE $L_{⊠}$ (U_k(¢)) (1)cp ⊠

In the context of q-learning, we attempt to find a set of parameters ¢ that maximizesthe q-function over the distribution of tasks. Then, we can write the loss as:

$L_{⊠}$ (U_k(¢)) = E $q_k(s_t, a_t)$ (2)

where the q-function is the sum of discounted rewards .

Note that ⇡⊠ denotes the policy of task ⊠ and a_t represents the decision executed attimestamp t.

Algorithm's pseudocode is as follows:

Boosting Few-Shot Meta Q-Learning with Reptile Optimization Algorithm

Algorithm 1 Sequential Iterative Adaptation (SIA)

1: Commence by setting c, as the starting parameter vector2: **for** At the outset of each cycl i = 1, 2, 3,. .. **do**

3: Randomly select a task ⊠ , with its corresponding loss

function $L_{⊠}$ on mass $\tilde{k}$ directions c

```
4:      Compute c  = U (c), by applying k rounds of stochastic
        gradient descent (SGD) or Adam optimization within the environment
5:      Execute an adjustment c  - c + $\epsilon$·(c
6:  end for
```

For Reptile, ϵ is stepsize of our update to the parameters of the meta-learner. We compute by taking a convex combination of the final epsilon value and an initial epsilonvalue. The convexity parameter is the fraction of meta iterations that have been completed.

VII. PROCEDURE

1. Task Generation

The training environment was imported from OpenAI's gym package. To alter the gravity and masspole parameters of the environment, we altered our local machine's gym/gym/envs/classic_control/cartpole.py to take in gravity and masspole as parameters. We also registered different environments in gym/gym/envs/ init .py.

2. Training

We employed the squared error loss function and a stochastic optimization method for our neural network model [9]. This technique enables the iterative refinement of neural network weights by leveraging first-order gradient information. Upon obtaining the gradients of the objective function, this method dynamically estimates both the first and second moments of these gradients, subsequently determining adaptive learning rates. It represents a leading-edge stochastic optimization methodology within the context of neural network training.

First, we randomly sampled the tasks from set of training tasks. For each task ☒ , we trained the DQN by taking k gradient steps in the environment for 200 episodes. We then updated the parameters of the meta-learner per iteration of the outer loop withthe difference between the parameters for ☒ and current parameters of meta-learner.After one task was trained, subsequent tasks took fewer iterations to reach themaximum reward over an episode.

In an effort to understand this problem in the context of Lifelong Learning, we also set the tasks sequentially, from lowest to high gravity (9.8 to 60 m/s) andlowest to highest mass of the pole (.1 to 1.0). We did not find any significant differences for rate of convergence of algorithm when randomly sampling the tasks orin the Lifelong Learning setup.

3. Validation

After extensively training the model on the sampled CartPole tasks, we used two held- out validation tasks to validate the implementation. These two tasks had unique gravityand mass for the poles, respectively. We trained the model on these tasks for five episodes each, but for each episode, we limited the number of optimization steps k thatthe model was able to perform. We ran this experiment a number of times, each time varying k. This parameter k relates to the generalizability of the model and how quickly it can learn based on the previous related tasks.

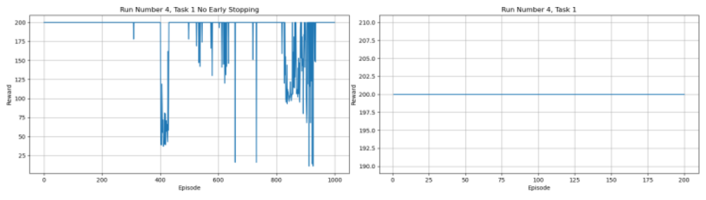
VIII. EXPERIMENTAL RESULTS AND DISCUSSION

1. Few-Shot Learning Training Metrics

See **Appendix** for training results on our implementation of Reptile.

To reduce wasted computation power, we implemented early stopping of thetraining loop if the loss had already converged to the global optima. To ensure we did not stop early prior to convergence, we maintained a running counter c for number of conver- gences, and exited if c > ⇤ NUM EPISODES.

From an initial comparison of the training curves of our model with and without early stopping, we observe that for some tasks like Task 1 (training curve below), evenafter the reward converges to 200, there is a potential that the model temporarily moves away from the point of convergence, leading to multiple gradient descent optimizations within the training loop.



(a) No Early Stopping for Task 1 (1000 (b) Early Stopping for Task 1 (exited afterepisodes) 200 episodes)

2. Few-Shot Learning Validation Metrics

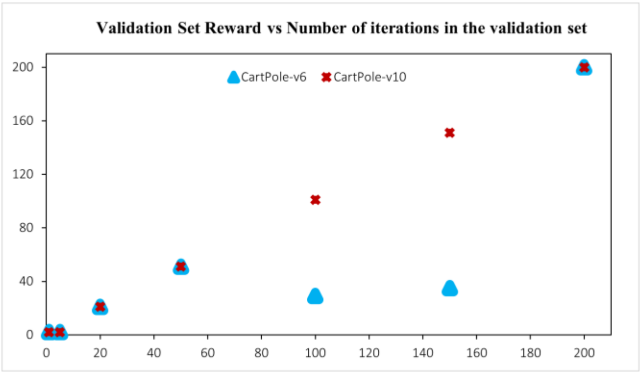


Fig. 3: Data table and plot of validation set rewards

		1	5	20	50	100	150	200
Validation	CartPole-v6	2	2	21	51	29	35	200
Environment	CartPole-v10	2	2	21	51	101	151	200

Number of Gradient Steps (k)

A key observation is that the best possible reward of 200 on the validation tasks isonly achieved when we let the validation task train for the full 200 gradient steps, comparable to the number of steps we ran the training set tasks. Thus, the number of gradient steps k still is very important to generalization, even with few training examples.

We believe this under-performance for few-shot learning could be caused either by:

(1) our neural network not being complex enough to capture the underlying dynamics which are constant across the training tasks, such as how gravity impacts the pole, or how the mass of the pole impacts dynamics, or (2) the tasks that we chose not having enough similarities. The former is more likely than the latter, as the tasks only had oneparameter change.

In cases where the real-world situation we are deploying this model in does not re- quire 100% accuracy, and where speed of deployment is prioritized, this implementation of Reptile would be a good use-case, granted k is chosen to be greater than 20.

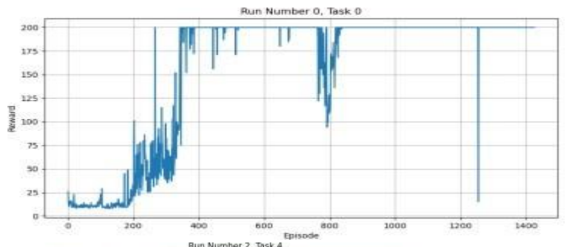
IX. Conclusion and Next Steps

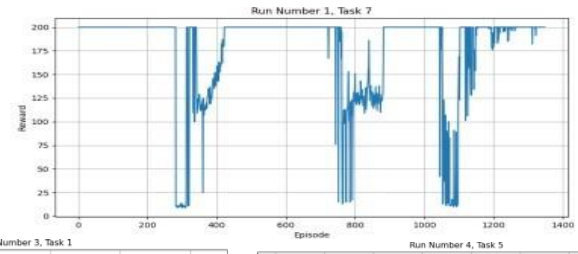
We have evaluated a method for learning a simple RL task which requires 200gradient steps to learn optimally.This suggests that your RL problem is relatively straightforward, and it takes a certain number of iterations or updates (gradient steps) for a learning algorithm to reach optimal performance on this task. Our observations demonstrate that validation task performance scales scale approximately linearly with respect to the number of gradient steps.It means as we increase the number of gradient steps, the performance on your validation task also improves in a roughly linear fashion. This is a positive result as it indicates that more training leads to better performance. Our experiments demonstrate

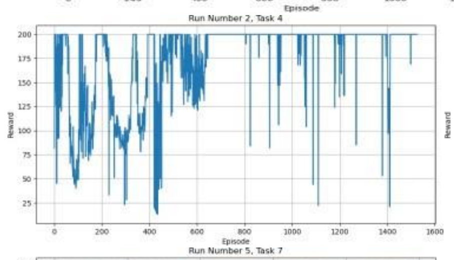
that a vanilla implementation of Reptile with k-shot learning model on a DQN model is able to train quite well in training-time, but does not generalize well. Even when thetasks were aligned in sequential order, the validation results stayed roughly the same. This means that even when you apply the model to similar tasks, its performance doesn't improve or stays roughly the same.Thus, Reptile does not work optimally for very similar CartPole tasks in the Reinforcement Learning domain. This suggests that the Reptile algorithm, in this setup, isn't able to leverage the knowledge gained from one task to substantially improve performance on the next task, even if they areclosely related.

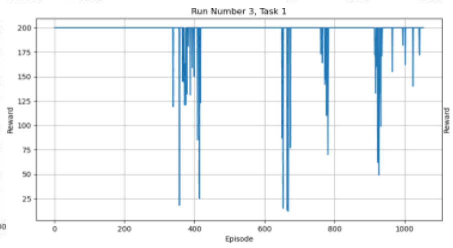
X. Appendix

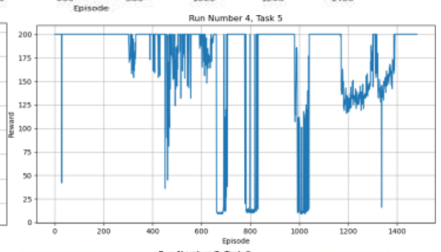
1. Training with 5000 episodes and early stopping

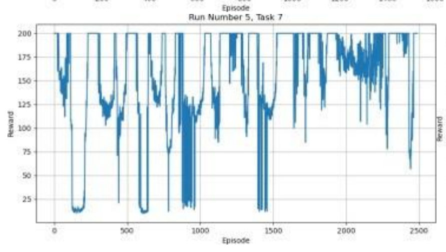


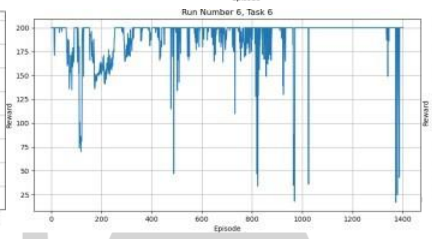


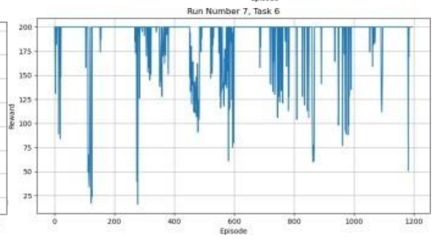




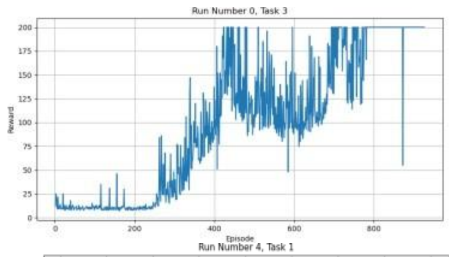


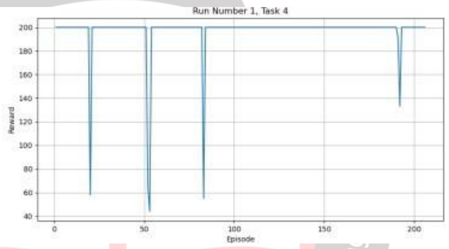


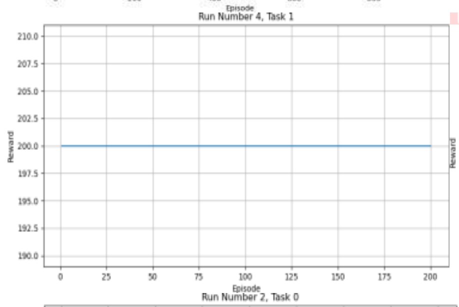


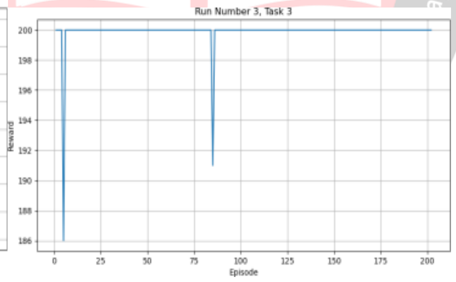


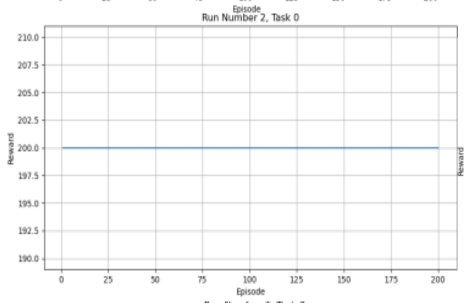
2. Training with 1000 episodes and early stopping

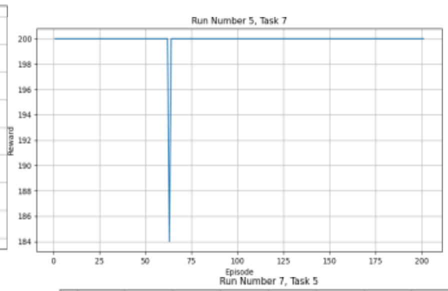


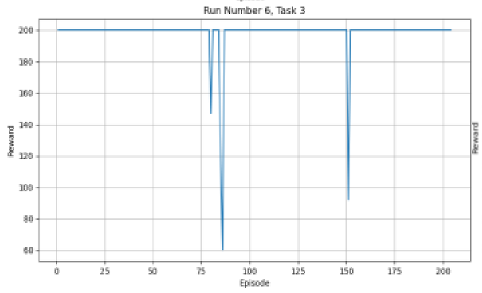


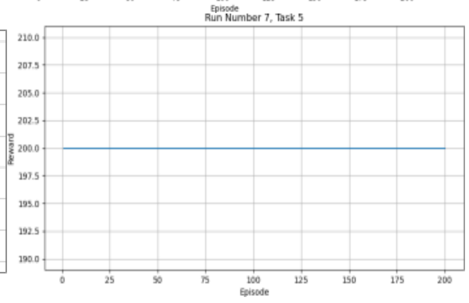






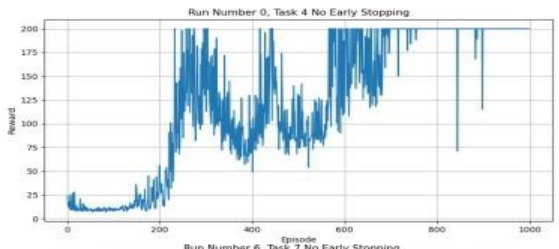


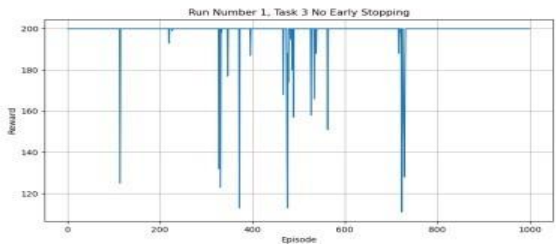


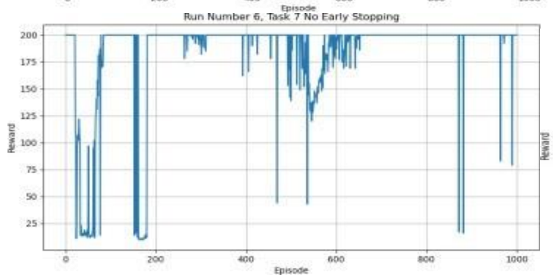


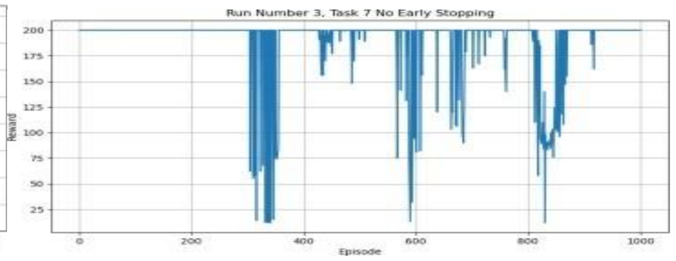
ISSN : 2454-9150 Vol-09, Issue-06, Sep 2023

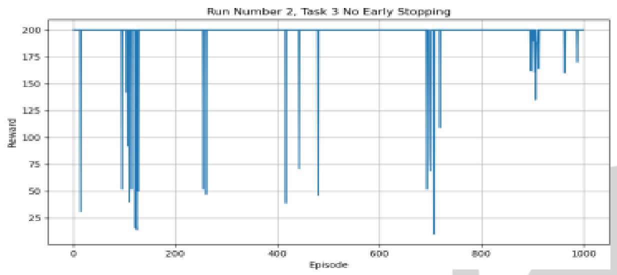
3. Training with 1000 episodes without early stopping

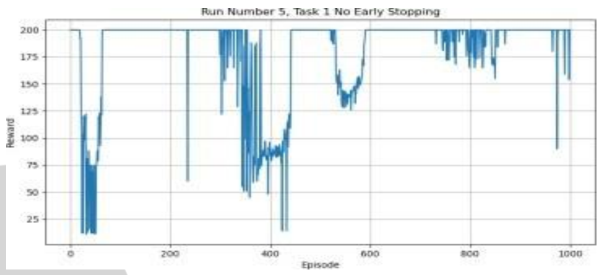


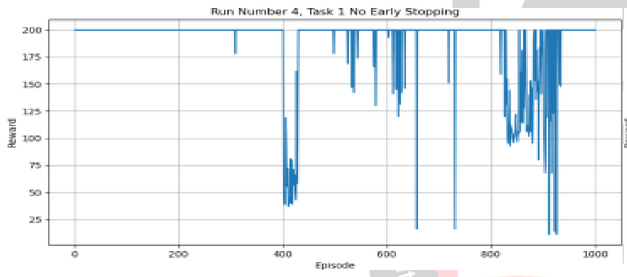


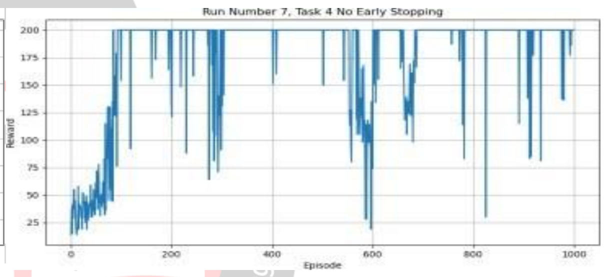












XI. REFERENCES

[1] Finn, C., Abbeel, P., Levine, S.: Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. International Conference on Machine Learning (ICML) (2017), http://arxiv.

[2] Lake, B.M., Salakhutdinov, R., Tenenbaum, J.B.: Human-level concept learning through probabilistic program induction. Science 350(6266), 1332–1338 (2015)

[3] Mishra, N., Rohaninejad, M., Chen, X., Abbeel, P.: A simple neural attentive meta-learner. arXiv preprint arXiv:1707.03141 (2017)

[4] Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. arXiv preprint arXiv:1803.02999 (2018)

[5] Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., Tesauro, G.: Learning to learn without forgetting by maximizing transfer and minimizing interference (2019)

[6] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shotlearning. In: Advances in neural information processing systems. pp. 3630–3638 (2016)

[7] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. Openai gym. arXiv preprint arXiv:160601540. (2016)

[8] Spall, J.C.. Stochastic Optimization. In: Gentle, J., Härdle, W., Mori, Y et al. Handbook of Computational Statistics. Springer Handbooks of Computational Statistics. Springer, Berlin, Heidelberg. (2012)