Few-Shot Meta Q-Learning with Reptile

Rahul Maganti {rmag@seas}, Sheil Sarda {sheils@seas}, Jason Xian {xjason@seas}

December 15, 2020

1 Introduction

Deep neural networks work well only when there is a lot of training data present. Moreover, training these networks is often slow. Human intelligence, on the other hand, learns quickly and extrapolates well only from only a small number of data points. We believe learning in a low-data regime remains a key challenge for Deep Learning. In this paper, we wish to quantitatively validate the effectiveness of few-shot learning in the context of Reinforcement Learning. By testing the Reptile algorithm in OpenAI's CartPole simulation environment using a Deep Q-Network model, we believe that the results obtained from this context will enhance the studies shown on few-shot learning approaches. The implementation for our project can be found here: https://github.com/sheilsarda/ESE546_Final_Project

2 Background

Meta-learning is the machine learning field pertaining to the problem of "learning how to learn." Few-shot learning is the sub-field that addresses how a machine can learn when only given a few samples. Typically, an agent or model is trained extensively on a set of related tasks and then given a different but related task for which it can only see a small amount of new data. In other words, the goal for few-shot learning is to have a model that can generalize well when only allowed to take a few k gradient 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) [Koch et al., 2015] and Matching Networks (discriminating multiple unseen classes) [Vinyals et al., 2016].

2. Exploiting prior knowledge: By using the structure and variability of the data, models can generalize recurring patterns. This is especially prevalent in computer vision tasks like object detection [Lake et al., 2015].

3. Constraining the learning algorithm: By choosing specific hyperparameters or choosing 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 [Nichol et al., 2018] (an application of the Shortest Descent algorithm) and SNAIL [Mishra et al., 2017] (which uses 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 Model-Agnostic Meta-Learning (MAML) algorithm, 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 [Finn et al., 2017]. 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 a first-order Taylor approximation to the Hessian 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 [Nichol et al., 2018]. 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 [Riemer et al., 2019].

3 Approach

The goal of our project is to understand and analyze few-shot learning in the context of Reinforcement Learning. 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. We then built and trained a Deep Q-Network on a set of related tasks and asked the model to generalize its learning on a hold-out set of tasks based on a few training steps.

3.1 Simulation Environments

We modified OpenAI Gym's CartPole environment to create unique but related simulations. CartPole, also known as an Inverted Pendulum, is a pendulum with a center of gravity above its pivot point. A pole is attached to a moving cart on an un-actuated joint. The cart moves along a frictionless track. The system is initially unstable, but it can be trained by teaching an agent how to adjust the pivot point. The system is controlled by applying a force of +1 or -1 to the cart. Thus, the goal in this environment is to keep the pole balanced and upright by applying the appropriate forces to the pivot point. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. By varying the gravity and pole mass, different but related tasks were created. Attached is a summary of the simulation environments:

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.

3.2 Neural Network Architecture

3.2.1 Deep Q Networks

Deep Q-Learning is a standard model-free Reinforcement Learning algorithm. Given a Q-function: $S \times A \rightarrow \mathbb{R}$, we can construct a policy that maximizes our long-term rewards:

$$\pi = \underset{a}{argmax} q(s, a)$$

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:

$$q(s, a) = R_{t+1}(s, a) + \gamma max_a q(s_{t+1}, a_{t+1})$$

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 arrive at an optimal policy, we aim to minimize the L^1 loss between the q-function estimate and the target Q value $\hat{y}(s, a)$:

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

To reduce these instabilities, methods like experience replay introduce a replay buffer. With a replay buffer, it is then possible to collect prior transitions $\{s, a_i, s^{'}, R(s, a)\}$ and sample from prior transitions before any updates are made.

3.2.2 Parameters

We use the following DQN with 3 fully connected layers to play over the 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
```

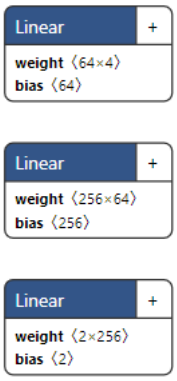


Figure 1: Deep Q-Network Architecture

3.3 Reptile Algorithm

Reinforcement Learning tasks are often complex and require significant computational resources. 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 $\{U_k\}$ denotes the k steps of the descent algorithm:

$$\min_{\phi} \mathop{\mathbb{E}}_{\tau}\Big[L_\tau(U_k(\phi))\Big] \tag{1}$$

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

$$L_\tau(U_k(\phi)) = \mathop{\mathbb{E}}_{\tau}\Big[q_k(s_t, a_t)\Big] \tag{2}$$

where the q-function is the sum of discounted rewards til time T for a finite horizon task:

$$q_k(s_t, a_t) = \mathop{\mathbb{E}}_{\pi_\tau}\Big[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s^t = s, a_t = a\Big] \tag{3}$$

Note that π_τ denotes the policy of task τ and a_t is the action taken at time t.

The pseudocode for the algorithm is as follows:

Algorithm 1 Reptile (Serial Version)

1: Initialize ϕ, the vector of initial parameters
2: **for** each iteration $i = 1, 2, 3, \ldots$ **do**
3: Sample task τ, corresponding to loss L_τ on weight vectors $\tilde{\phi}$
4: Compute $\tilde{\phi} = U_\tau^k(\phi)$, denoting k steps of SGD or Adam through the environment
5: Update $\phi \longleftarrow \phi + \epsilon(\tilde{\phi} - \phi)$
6: **end for**

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

4 Procedure

4.1 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`.

4.2 Training

We used the Adam Optimizer and MSE loss for our DQN model. Adam is an efficient stochastic optimization method to update network weights iteratively based on first-order gradients information. Once we have gradients of objective function, Adam can adaptively estimate the first and second moments of the gradients, and further compute adaptive learning rate. Adam is a state-of-the-art stochastic optimization algorithm in Deep Q-Learning.

First, we randomly sampled the tasks from set of training tasks. For each task τ, 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 with the difference between the parameters for τ and current parameters of meta-learner. After one task was trained, subsequent tasks took fewer iterations to reach the maximum 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^2) and lowest 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 or in the Lifelong Learning setup.

4.3 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 gravity and 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 gradient steps k that the model was able to take. 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.

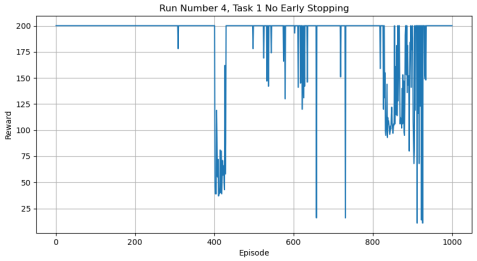
5 Experimental Results and Discussion

5.1 Few-Shot Learning Training Metrics

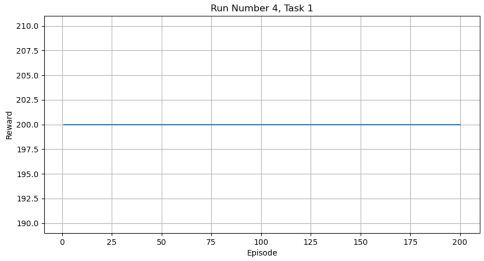
See **Appendix** for training results on our implementation of Reptile.

To reduce wasted computation power, we implemented early stopping of the training 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 convergences, and exited if $c > \frac{1}{5} *$ 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), even after 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 episodes)



(b) Early Stopping for Task 1 (exited after 200 episodes)

5.2 Few-Shot Learning Validation Metrics

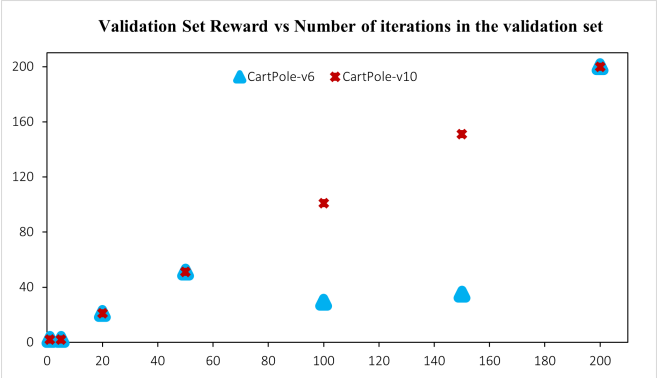


Figure 3: Data table and plot of validation set rewards

		Number of Gradient Steps (k)						
		1	5	20	50	100	150	200
Validation Environment	CartPole-v6	2	2	21	51	29	35	200
	CartPole-v10	2	2	21	51	101	151	200

A key observation is that the best possible reward of 200 on the validation tasks is only 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 one parameter change.

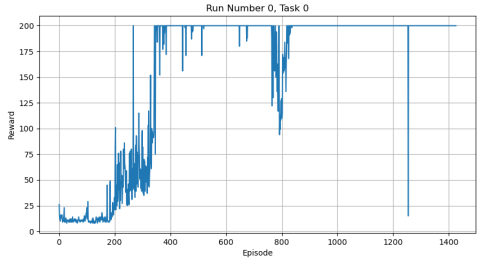
In cases where the real-world situation we are deploying this model in does not require 100% accuracy, and where speed of deployment is prioritized, this implementation of Reptile would be a good use-case, granted k is chosen to be greater than 20.

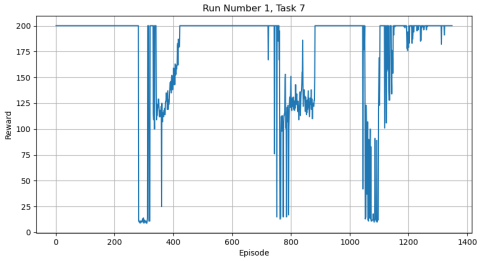
6 Conclusion and Next Steps

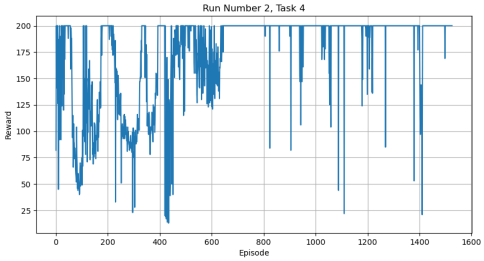
We have evaluated a method for learning a simple RL task which requires 200 gradient steps to learn optimally. Our observations demonstrate that validation task performance scales scale approximately linearly with respect to the number of gradient steps. Our experiments demonstrate that a vanilla implementation 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 the tasks were aligned in sequential order, the validation results stayed roughly the same. Thus, Reptile does not work optimally for very similar CartPole tasks in the Reinforcement Learning domain.

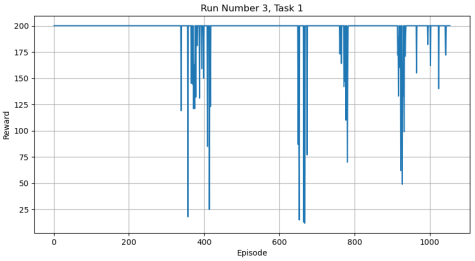
7 Appendix

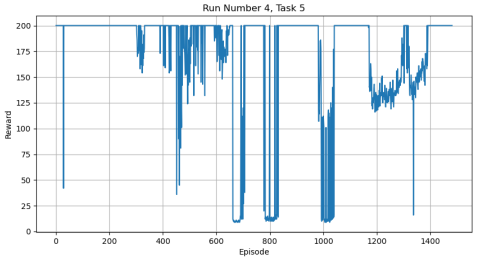
7.1 Training with 5000 episodes and early stopping

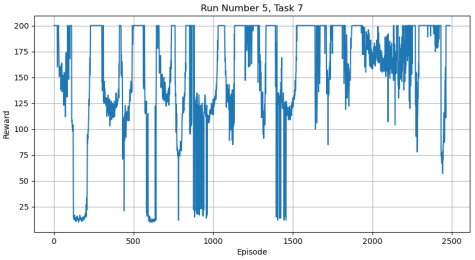


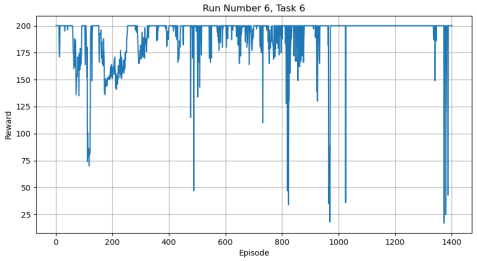


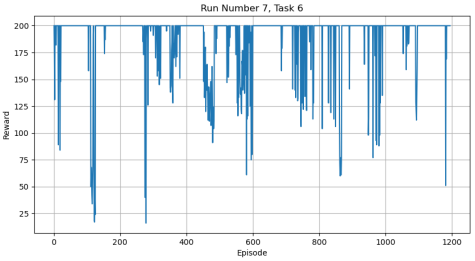




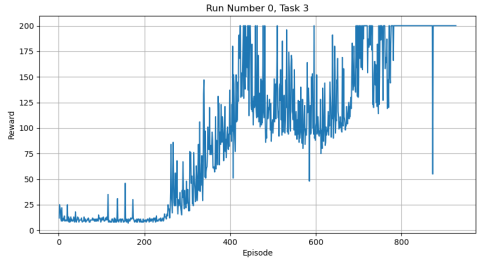


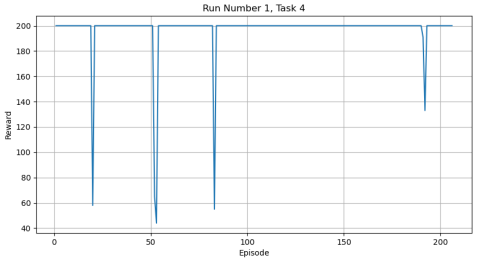


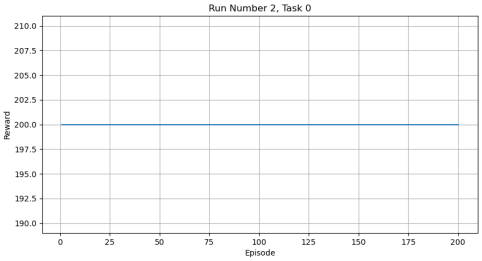


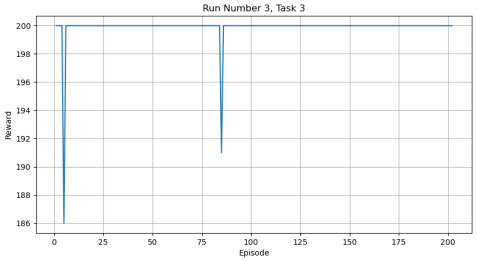


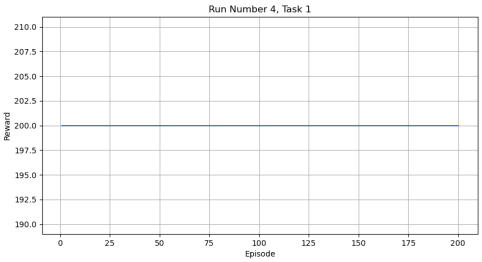
7.2 Training with 1000 episodes and early stopping

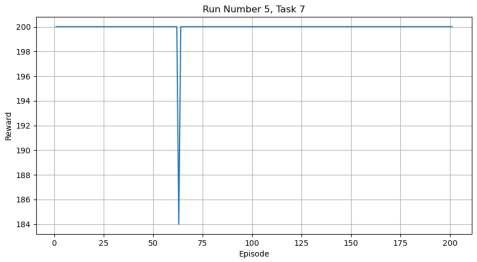


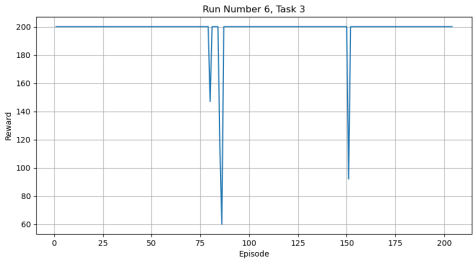


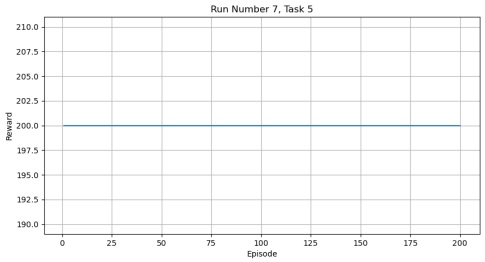




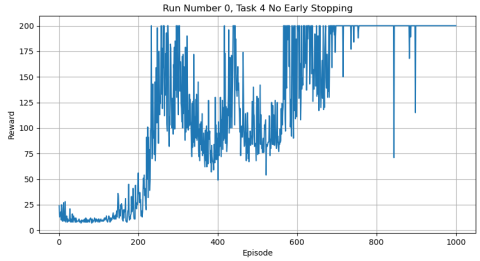


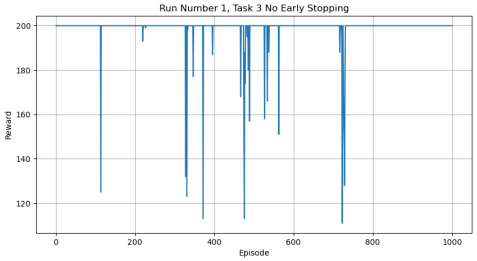


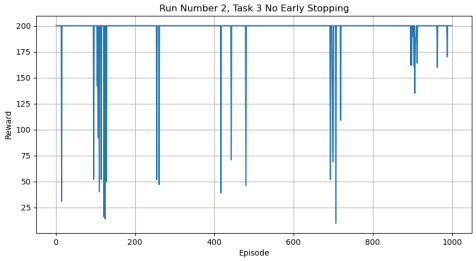


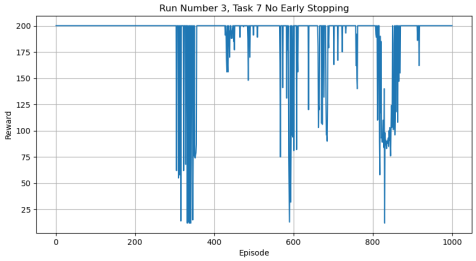


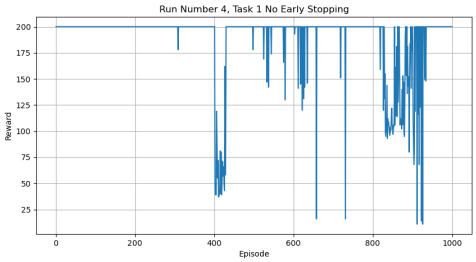
7.3 Training with 1000 episodes without early stopping

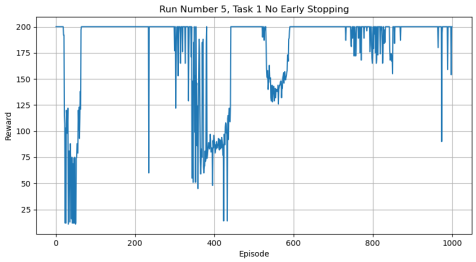


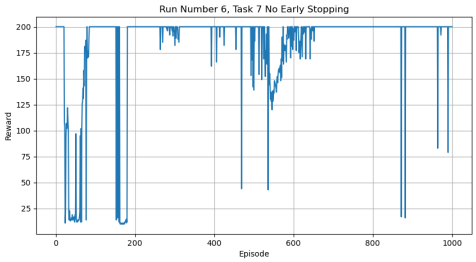


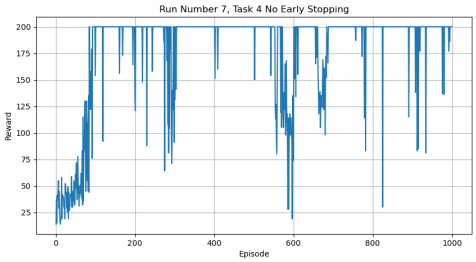












References

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning (ICML)*.

[Koch et al., 2015] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.

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

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

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

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

[Vinyals et al., 2016] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.