Multi-Task Reinforcement Meta-Learning in Neural Networks

Ghazi Shakah
Software Engineering Department, Faculty of Information and Technology
Ajloun National University, P.O. Box 43, Ajloun 26810, Jordan

Abstract*—Artificial Neural Networks (ANN) is one of the main and widespread tools for creating intelligent systems. And, they are actively used for data analysis in many areas such as robotics, computer vision, natural language processing, etc. The learning process of ANN is one of the most labor-intensive stages in ANN. There are many different modifications of ANNs and methods for their training. Currently, deep neural networks are becoming one of the most popular methods of machine learning due to their effectiveness in areas such as speech recognition, medical informatics, computer vision, etc. It is known that ANN training depends on the type of input data. In this paper, reinforcement learning is considered, as popular method used in cases where information is reinforced by signals from the external environment with which the model interacts. The purpose of this paper is to develop a reinforcement meta-learning algorithm that would be efficient in terms of quality and speed of learning. However, despite the significant scientific progress in deep learning, existing algorithms are not efficient enough to solve problems in the real world. In addition, such algorithms require a significant amount of learning time, which complicates the development process. To solve these problems, the use of meta-learning or "learning to learn" algorithms has recently been especially relevant. The paper proposes an approach to reinforcement meta-learning using a multitasking weight optimizer. experimentally shown that the proposed approach is more efficient than the known MAML (Model-Agnostic Meta-Learning) algorithm. The proposed MAML SPSA-Track method shows an improvement in efficiency by an average of 4%, and MAML SPSA-Delta by 8%, respectively. Moreover, the last algorithm spends on average 2 times less time on push-v2 and pick-place-v2 tasks.

Keywords—Multitasking; meta-learning; reinforcement learning; neural networks; optimization

I. Introduction

Humans have an innate ability to learn new skills quickly and easily. For example, we can look at one instance of a knife and distinguish all knives from other cutlery such as spoons and forks. Our ability to learn new skills and quickly adapt to a new environment based on a small number of examples is not just limited to identifying new objects, learning a new language, or figuring out how to use a new tool; our possibilities are much more diverse. [1], [2]. In contrast, machines—specifically, deep reinforcement learning algorithms—generally learn quite differently [3]. They require very large amounts of data and computational resources to achieve acceptable performance. The reason why people can learn quickly and adapt to a new environment is that they use the knowledge gained from previous experience to solve new problems. Similarly, meta-learning uses little experience gained from data to solve new problems quickly and efficiently. Through this method, it is possible to significantly speed up the training of neural networks with reinforcement significantly. Neural networks with reinforcement require quite large amounts of training data and computational resources. Creating such datasets is costly, especially when you need to involve a domain expert. While pre-training is useful, these approaches become less efficient for domain-specific problems, which it still requires large amounts of task-specific labeled data to achieve good performance. In addition, some existing problems are characterized by a wide and unbalanced distribution of data, which can make it difficult to collect training examples [4]. On the other hand, it is possible to use a pre-trained network from another task and then finish training it on the current small training set. However, depending on the specifics of the problem, this is not always possible, especially if the task on which the neural network was trained is significantly different. It is important to note that the ability to quickly learn new tasks during model inference is something traditional machine learning approaches do not attempt. This is what makes meta-learning especially attractive. Meta-learning is particularly interesting and can be used for the following reasons [5].

- The ability to learn from just a few examples.
- Quick adaptation to new tasks.
- The ability to create more versatile systems.

Meta-learning is especially successful in situations where a large amount of data is required; for example, robots are tasked with learning new skills in the real world and often encounter new environments [6].

Finally, the task was formally set to develop a meta-learning algorithm with reinforcement of a machine learning model that would be efficient in terms of learning.

II. Literature Review

Meta-learning tries to gain general knowledge about the target area by learning the set of tasks belonging to it [7]. The idea of meta-learning is to train the model by showing it only a few examples for each class and [8] then test it on new examples from the same classes that were taken from the original dataset.

The author in [9] proposed a formal description of the few-shot learning task as meta-learning. The data set of each class is randomly divided into a support set and a query set. The support set consists of labeled examples that are used to predict classes of untagged examples from the query set. The set of classes at the training stage does not intersect with the set of classes at the testing stage.

The author in [10] introduced a multitasking loss function based on maximizing the Gaussian likelihood with a task-dependent uncertainty. The proposed single model for all tasks outperformed separate models for each task.

Subsequently, a multitasking approach was applied in [11] to solve the problem of character recognition from a small number of examples, which led to an improvement in the overall recognition efficiency compared to the base model.

The author in [12] established a close relationship between the optimization problems of multitask learning and optimization-based meta-learning. Different from existing works, this paper focuses on improving the meta-learning stage. Only inductive methods that use a meta-learning process without prior training are considered. To do this, MAML was chosen as an example of the use of optimization-based learning, since the works describing this method are among the most cited in this field.

III. Material and Method

A. The Task of Reinforcement Meta-Learning

Reinforcement learning (RL) is one of the methods of machine learning, the purpose of which is to find an optimal strategy for the behavior of the model in the environment and maximize the reward received from the environment throughout the entire time the model interacts with the environment. The main concepts in RL are the agent and the environment: The environment represents the world in which the agent lives and interacts. In Fig. 1, at each interaction step, the agent observes (perhaps only partially) the state of the environment [3]. While the agent then decides what action to take. The environment changes when the agent acts on it, but it can also change by itself. The agent also receives a reward from the environment, a number that tells him how good or bad the current state of the world is. Accordingly, the agent's goal is to maximize his total reward, called profitability. Reinforcement learning methods are approaches by which an agent can learn the desired behavior to achieve a goal [13].

Reinforcement meta-learning is meta-learning in the field of reinforcement learning. Usually, training and test problems are different, but they are taken from the same family of problems.

Let's we have a distribution of tasks, each of which is The Markov Decision Process (MDP).

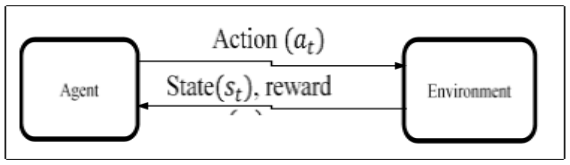


Fig. 1. The Cycle of Interaction between the Agent and the Environment [3].

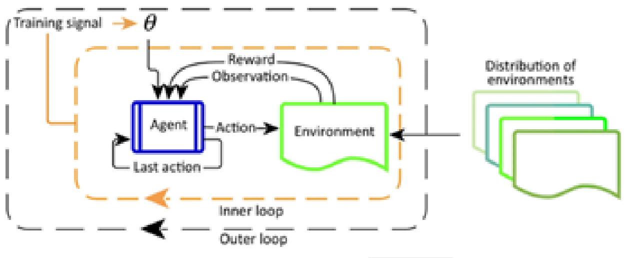


Fig. 2. A Meta-RL representation Containing Two Optimization Loops [4].

$M_i \in M$, where M_i is defined by the set $\langle S, A, P_i, R_i \rangle$. In Fig. 2, at each iteration of the external cycle, a new environment is selected and the parameters that determine the behavior of the agent are adjusted using the metal earning algorithm. In the inner loop, the agent interacts with the environment and maximizes the reward using a reinforcement learning algorithm [4], [14]. Note that the general state S and action space A, so the stochastic policy is:

$\pi_\theta: S \times A \rightarrow R_+$

Will receive input data that is compatible with different tasks. Test items are selected from the same or slightly modified distribution M. In general, reinforcement meta learning is very similar to regular reinforcement learning, except that the last reward r_{t-1} and the last action $a_{t-1}$are also included in the observation in addition to the current state. s_t:

- In reinforcement learning $\pi_\theta(s_t) \rightarrow a$
- In Reinforcement Meta-learning $\pi_\theta(a_{t-1}, r_{t-1}, s_t) \rightarrow a$

This is done so that the policy will learn the changes between states, rewards and actions in the current MDP and can adjust its strategy accordingly. This is done so that the policy can assimilate the changes between states, rewards and actions in the current MDP and can adjust its strategy accordingly.

IV. Development of the Training Method

A. MAML Meta-Learning Algorithm

Reinforcement meta-learning uses two optimization loops: external and internal. During the outer loop, a meta-learning algorithm is applied. It is important to note that MAML is compatible with any model that can be trained using gradient descent, which is its main advantage. There aren’t any restrictions on the loss function. The algorithm is applicable to such a wide range of problems as regression, classification, and reinforcement learning. MAML does not change the structure of the learning model, but only changes the network parameters in such a way that a small number of gradient descent steps are required on a small training dataset of a new problem to obtain a good generalization ability on this problem [15]. However, this algorithm requires taking second-order derivatives, which is the main disadvantage of this algorithm.

B. Method Formulation of the Problem

Despite the fact that there are a number of algorithms that do a good job of this kind of task, the speed of learning these algorithms, even on the most productive equipment, takes an extremely long time. Moreover, due to the dynamism and variety of tasks in the real world, the quality of the solution on new test problems can be much worse than the quality of training ones. So, the goal of this thesis is to develop a reinforcement meta-learning algorithm that would be effective in terms of quality and speed of learning. Also, during the execution of the work, the following tasks were set:

- Modify the basic MAML (Model-Agnostic Meta-Learning). An algorithm based on a multitasking approach.
- The tasks of deep learning, reinforcement learning, meta-learning, multitasking learning, and reinforcement meta-learning have been described.
- The main approaches to meta-learning were analyzed - based on models, metrics and optimization, and modern meta-learning algorithms for each approach.
- Compare the efficiency of the modified and unmodified algorithms.
- The common problem of low efficiency for these methods was analyzed in terms of both data and time resources spent on training the model.
- Formally, the task was to develop a meta-learning algorithm with reinforcement of a machine learning model that would be effective in terms of quality and speed of learning.

V. Software Implementation of the Console Application

A. Architecture and Composition

The console application was developed in the popular Python programming language for further experiments. Fig. 3 describes the standard process for developing a strategy model, along with the important modules associated with solving the problem.

Meta-World's ML1 environment was used as an environment for the agent. To implement the neural network of the agent, the torch.nn module of the well-known PyTorch framework was used. [16].

To realize the basic MAML algorithm and the proposed SPSA-Delta and SPSA-Track algorithms, the modules were optim and torch. Autograd was used, which presents various standard optimization algorithms for training neural networks. The training results were written to the hard disk using the torch.utils.tensorboard module. PyTorch is one of the most popular open-source machine learning frameworks in the Python programming language.

The main PyTorch modules that are used in the software implementation are: torch.nn, torch.optim, torch.autograd, torch.utils.tensorboard. The torch.nn module defines computational graphs and works with gradients, which makes it easy to build neural networks. The following module torch.optim introduces various optimization algorithms for training neural networks. The torch.autograd module implements the automatic differentiation method. The torch.utils.tensorboard module helps to save and visualize the results.

The first step in any deep learning project involves loading and processing training data. Reinforcement learning of a model consists of its interaction with the simulated environment. Meta-World allows you to design environments according to the Env interface of the Gym framework. First, we need to create the desired test, and then an instance of the environment. A task is assigned to the environment using the set_task() method from the corresponding already defined training and test tasks of the created test. In the current project, a function was described that returns an instance of the environment given the benchmark test and the task name task name.

The process of creating the environment of the Meta-World framework for the subsequent training of the model is shown in Fig. 4. The agent's interaction with the environment is implemented through the environment's step() method. For the convenience of interacting with the environment, the Runner class was described, an instance of which receives from the model the action to be performed, passes it to the simulated environment and receives from it information about the current state, the value of the reward, success rate and other metadata. Then, an instance of the ReplayMemory class collects all the received information about states, actions, etc. into the corresponding tensors in order to further transfer it to the main MAML algorithm for processing. The strategy is presented as a neural network. To create it, PyTorch uses the corresponding torch.nn module. It provides the implementation of all commonly used neural network components such as fully connected and convolutional layers, activation layers and associated loss functions.

The neural network representing the main strategy consists of one hidden layer with a size of 128 neurons and an activation function nn.Tanh() between the layers Fig. 5.

The input of the neural network is a vector of length 39 about the state of the ML1 Meta-World test environment, at the output the neural network gives a vector of length 4 about the next action by the agent in the environment.

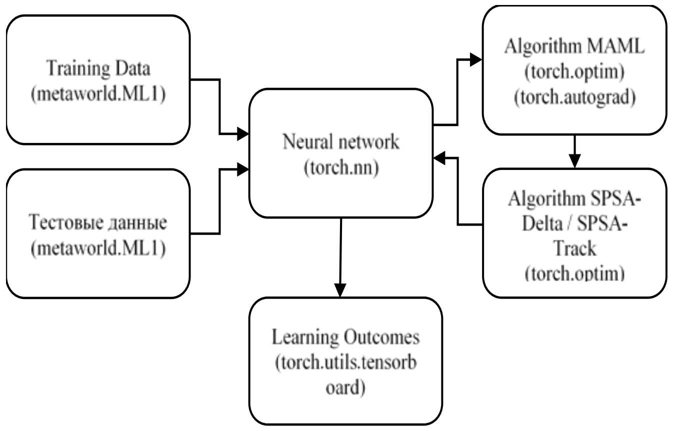
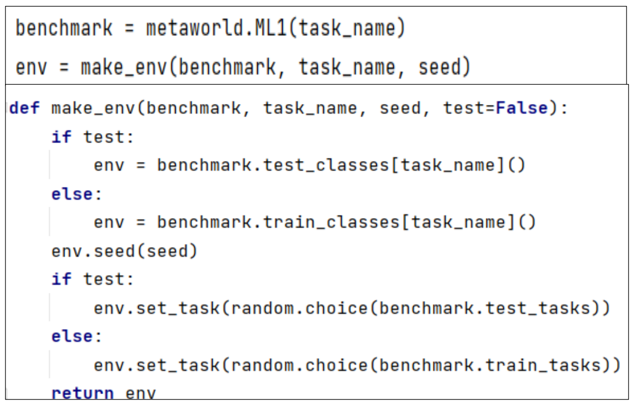

Training Data (metaworld.ML1)
Тестовые данные (metaworld.ML1)
Neural network (torch.nn)
Algorithm MAML (torch.optim) (torch.autograd)
Algorithm SPSA-Delta / SPSA-Track (torch.optim)
Learning Outcomes (torch.utils.tensorb oard)

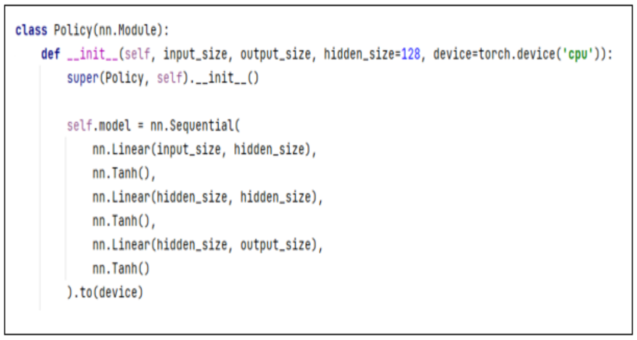
Fig. 3. Diagram of the Development Process of a Neural Network Model.



```
benchmark = metaworld.ML1(task_name)
env = make_env(benchmark, task_name, seed)
```

```
def make_env(benchmark, task_name, seed, test=False):
    if test:
        env = benchmark.test_classes[task_name]()
    else:
        env = benchmark.train_classes[task_name]()
    env.seed(seed)
    if test:
        env.set_task(random.choice(benchmark.test_tasks))
    else:
        env.set_task(random.choice(benchmark.train_tasks))
    return env
```

Fig. 4. The Initialization of the Meta-World Environment.



```
class Policy(nn.Module):
    def __init__(self, input_size, output_size, hidden_size=128, device=torch.device('cpu')):
        super(Policy, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, output_size),
            nn.Tanh()
        ).to(device)
```

Fig. 5. The Strategy Neural Network Initialization.

For each iteration of the meta-learning stage 10 training tasks were selected to obtain the needed data. The number of adaptation steps of the MAML algorithm was equal to 1, i.e. during testing, adaptation to a new problem occurred in one step of the gradient descent algorithm. The maximum length for a single task in the environment was 500. The maximum length for a single task in the environment was 500. The reward discount factor was 0.99.

During the adaptation phase, the weights of the neural network were changed with a learning rate factor of 1×10^{-4}, during the meta-learning phase 1×10^{-3}. The optimization method TRPO was used as a meta-optimizer with a maximum number of steps for linear search of 15 and a step of 5×10^{-3}.

The neural network was trained for 600 epochs for environment with reach-v2, pick-place-v2 tasks and 1200 epochs for push-v2. Also, the multitasking weight optimizer only started optimization after 50 epochs so that the model had time to adjust the initially randomized weights according to the task and environment.

B. *General Description of the MetaWorld Environment*

Meta World is an open-source simulated test for reinforcement meta-learning and multitasking learning, consisting of 50 different environments with robotic manipulations [17]. Task T in Meta World is defined as a set consisting of a reward function, an object's starting position, and its target position. Metal-earning makes two important assumptions:

1) Meta-training and meta-testing tasks have a common distribution $p(T)$.

2) The task distribution $p(T)$, has a general structure that can be used to effectively adapt to new tasks.

If $p(T)$, is defined as a family of variations within a specific problem, as in previous works [6], [10], then it is unreasonable to hope for a generalization to completely new problems. For example, an agent has little chance of being able to quickly learn to open a door without ever hitting a door before if it has only been trained on a set of uniform and narrow tasks during meta-learning. Thus, in order to allow reinforcement meta-learning methods to adapt to completely new tasks, a sufficiently large set of tasks is needed, where continuous changes in parameters cannot be used to describe the differences between tasks. In Meta World, all tasks are performed by a robotic arm. The action space is a set of two elements, consisting of changing the 3D space of the gripper.

Actions in this space range from -1 to 1. The observation space is represented as a set of 6 3D Cartesian positions of the gripper, a normalized measurement of how open the grip is, the 3D position of the first object, the quaternion of the first object, the 3D position of the second object, the quaternion of the second object, all previous measurements in the environment, and finally 3D position of the target. If there is no second object or the target is not supposed to be included in the observation, then the values corresponding to them are set to zero. The state space is always 39-dimensional. The reward functions for all tasks have the same value, which is in the range from 0 to 10, where 10 always corresponds to the fact that the task was solved. It should be noted that all tasks were implemented using the MuJoCo physics engine [18], [19], which allows to simulate the physical laws of the real world quickly and efficiently. The Multi-world interface [20] and interfaces of the popular OpenAI Gym environment [21] were taken as the basis, which makes this framework quite easy to use.

VI. Experimental Methodology

As an applied task for solving and evaluating the efficiency of the developed algorithms, we consider the ML1 test in the Meta-World environment. ML1 is aimed at evaluating the adaptation of the algorithm in several steps to change the goal within the same task. ML1 uses separate Meta-World tasks, where training tasks correspond to 50 random initial positions of objects and targets, and testing tasks correspond to 50 held positions. Algorithms are evaluated on three tasks from Meta-World:

- reach-v2.
- push-v2.
- pick-place-v2.

where either the position to be reached or the target position of the object varies. Target positions are not specified in world states, which forces reinforcement meta-learning algorithms to adapt to the target through trial and error.

In the reach-v2 task, the robotic arm needs to reach a target position, which is given randomly. The next push-v2 challenge is to push the puck towards the net. In the pick-place-v2 problem, it's important to take and place the puck in the goal. The positions of the puck and the goal in the tasks are set randomly.

Since the reward values do not directly indicate the success of the chosen policy, Meta-World defines an interpretable success metric for each task, which is used as the main criterion. Since all tasks involve the manipulation of one or more objects in the target configuration, this success metric is usually based on the distance between the task-relevant object and its final target position, i.e.,$||o - j|| < \varepsilon$, where ε is the threshold, for example, 5 cm. The software implementation was carried out in the Python programming language version 3.8. The environment was modeled using the Meta-World framework version 2.0 together with the OpenAI Gym framework version 0.19. As noted earlier, MetaWorld contains ready-made implementations of various environments for meta-reinforcement learning and agent testing. To build models of neural networks, the PyTorch framework version 1.10 was used, which contains implementations of various layers and algorithms for optimizing neural networks.

VII. Analysis of Results

Fig. 6 shows the maximum success rate, averaged over 5 runs, in the ML1 Meta-World test environment. Based on the results obtained, all 3 algorithms do an excellent job of solving the reach-v2 problem both at the training and testing stages. On more complex push-v2 and pick-place-v2 tasks, the MAML SPSA-Delta algorithm is the most efficient among all those considered. The improvement relative to the basic algorithm was 17% at the training stage and 21% at the testing stage on the push-v2 task, 8% and 3% on the pick-place-v2 task, respectively. However, on the pick-place-v2 problem, the ability of the MAML SPSA-Delta method to generalize is not much higher than the MAML SPSA-Track algorithm (60% for MAML SPSA-Delta and 59% for MAML SPSA-Track). The following figures show examples of the moving average success rate for the reach-v2 task over 6×10^7steps in a test environment (Fig. 6 is the training phase, Fig. 7 is the testing phase). For all constructed charts, the moving average coefficient is 0.8. As can be seen from the graph, the modified MAML algorithms solve the problem no worse than the original MAML algorithm environment (Fig. 7 is the training phase; Fig. 8 is the testing phase). For all constructed charts, the moving average coefficient is 0.8.

As can be seen from the graph, the modified MAML algorithms solve the problem better than the original MAML algorithm. Now let's look at the moving average success rate plots for the following push-v2 task over 2×10^8steps in a test environment (Fig. 9 is the training phase, Fig. 10 is the testing phase). Compared to other tasks, the SPSA-Delta MAML algorithm took significantly longer to adapt to the environment and overtake the basic MAML method. The SPSA-Track algorithm also shows good results, but they do not differ significantly from the results of the unmodified MAML algorithm. Finally, let's analyze the constructed plots of the moving average success rate for the last pick-place-v2 task over 6×10^7 steps in the test environment (Fig. 11 is the training phase, Fig. 12 is the testing phase). It follows from the graph that both MAML algorithms with modifications are significantly more efficient at the testing stage than the original MAML algorithm.

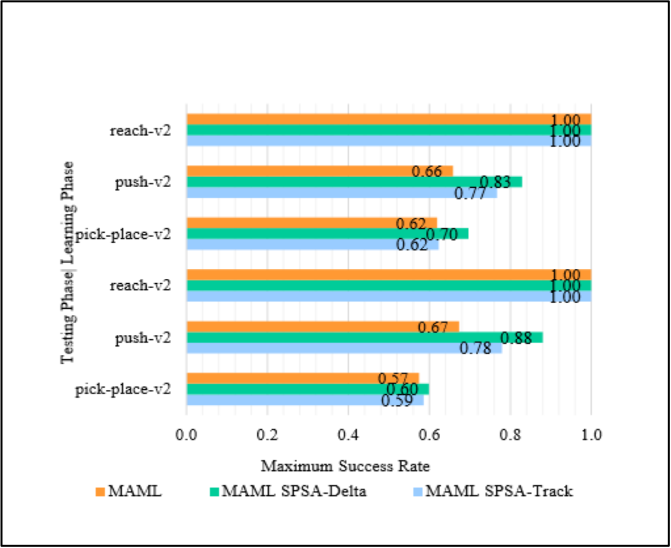


Fig. 6. The Maximum Success rate of Algorithms in the ML1 Meta-World Test Environment.

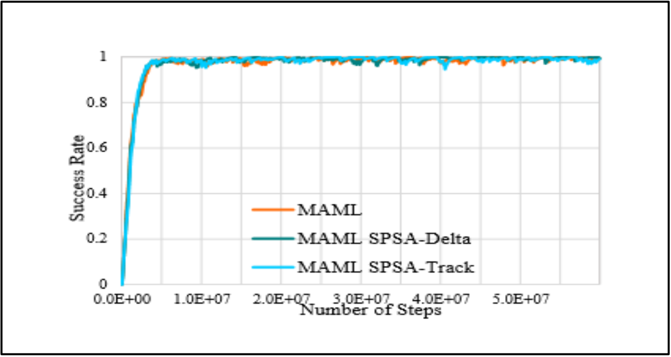


Fig. 7. Average Success Rate of Algorithms on the Reach-v2 Problem at the Training Stage.

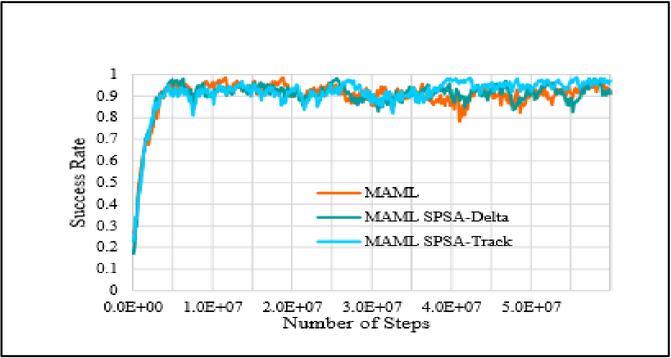


Fig. 8. Success Rate of Algorithms on the Reach-v2 Problem at the Testing Stage.

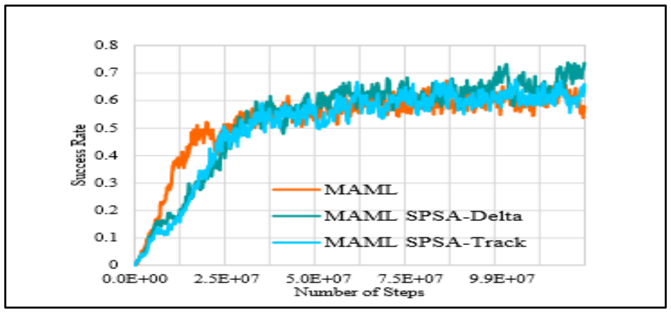


Fig. 9. Success Rate of Algorithms on the Push-v2 Problem at the Training Stage.

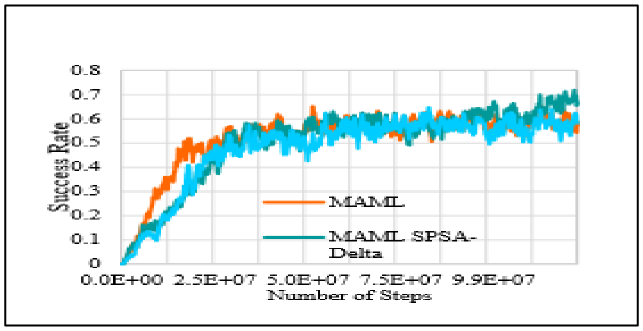


Fig. 10. Success Rate of Algorithms on the Push-v2 Task at the Testing Stage.

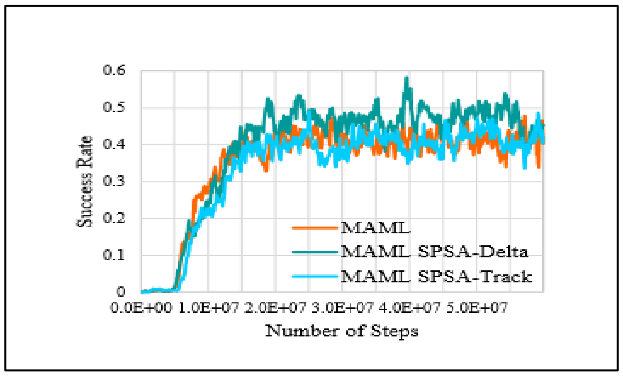


Fig. 11. Success Rate of Algorithms on the Pick-place- v2 Problem at the Training Stage.

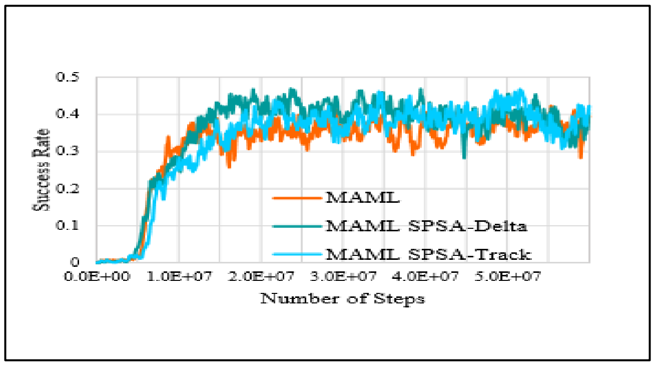


Fig. 12. Success Rate of Algorithms on the Pick-place-v2 Task at the Testing Stage.

Moreover, the MAML SPSA-Delta and MAML with SPSA-Track algorithms achieve the same high success rate, but the first algorithm learns 2 times faster than the second.

Consider the average maximum success rate achieved by each algorithm in the ML1 Meta-World test environment, information about which is shown in Table I.

Based on the results of experiments in the ML1 MetaWorld test environment with three different methods of deep reinforcement meta-learning: the original MAML algorithm, MAML SPSA-Delta, MAML SPSA-Track, the proposed MAML SPSA-Track method shows an average efficiency improvement of 4%, and MAML SPSA-Track Delta by 8%, respectively. Moreover, the latter spends on average 2 times less time for training on push-v2 and pick-place-v2 tasks. According to the obtained results, it is safe to say that the use of a multitasking loss function and its stochastic approximation with simultaneous perturbation can significantly improve the efficiency of deep reinforcement learning algorithms.

TABLE I. MAXIMUM SUCCESS RATE AVERAGED OVER ALL TASKS IN THE ML1 META-WORLD TEST ENVIRONMENT.

Algorithms	**Learning Phase**	**Testing Phase**
MAML	**76%**	75%
MAML SPSA-Delta	84%	83%
MAML SPSA-Track	80%	79%

VIII. CONCLUSION

Based on the tasks of deep learning, reinforcement learning, meta-learning, meta-learning with reinforcement and multitasking learning and their relevance are described.

After making comparison of the basic MAML algorithm with the proposed MAML SPSA-Delta and MAML SPSA-Track by conducting computational experiments to train the agent on reach-v2, push-v2, pick-place-v2 tasks in the ML1 Meta World test environment, it was concluded that the MAML SPS-Track algorithm is on average 4% more efficient compared to the original MAML method, and the MAML SPSA-Delta algorithm is 8% more efficient. Moreover, the last algorithm spends on average 2 times less time on push-v2 and pick-place-v2 tasks.

REFERENCES

[1] Wang, Y. X. Ramanan, D. and Hebert M. “ Learning to Model the Tail”. Advances in Neural Information Processing Systems 30, 2017, pp.58-69.

[2] Zidong Zhang, Dongxia Zhang, and Robert C. Qiu,” Deep reinforcement learning for power system applications: an overview”.,csee journal of power and energy systems, vol. 6, no. 1, march 2020,pp.213-225.

[3] Martín H., J. A., de Lope, J., and Maravall, D. “The kNN-TD Reinforcement LearningAlgorithm”, Lecture Notes in Computer Science,2009, pp.305-314.

[4] Ezzeldin, T. and Kassis, A.” Beyond Explanations: Recourse via Actionable Interpretability – Extended”,Research gate, 2020, pp.1-17.

[5] L. David, A. Michael, and B. Richard. “ Dynamic core competences through meta-learning and strategic context “,Elsevier, Journal of Management, Volume 22, Issue 4, 1996, PP. 549-569.

[6] G. Shakah. “ The Devices of the Internet of Things Based on the Recognition of Handwriting Words with Mobile Assisted “, International Journal of Interactive Mobile Technologies (iJIM),Vol.14, No. 4,2020, pp.74-85.

[7] O. Aissam, E,Hamza, and J, Philippe Leroy. " Dynamic Access Control Policy based on Blockchain and Machine Learning for the Internet of Things", (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 8, No.7, 2017, pp.417-424.

[8] R. Vilalta, and D.Youssef ." A perspective view and survey of meta-learning. Artificial Intelligence Review", Vol.18, No2, 2002,pp.77–95.

[9] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D.Wierstra. " Matching networks for one shot learning. Advances in neural information", processing systems, Vol,29, 2016, pp.3630–3638.

[10] A. Kendall, Y. Gal, and R. Cipolla." Multi-task learning using uncertainty to weigh losses for scene geometry and semantics", In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7482–7491.

[11] A. Boiarov, O. Granichin, and O Granichina. " Simultaneous perturbation stochastic approximation for few-shot learning " , IEEE, European Control Conference (ECC), 2020, pp. 350–355.

[12] H. Wang, H. Zhao, and B. Li. "Bridging multi-task learning and meta-learning: Towards efficient training and effective adaptation". Vol,139. In International Conference on Machine Learning. PMLR, 2021, pp. 10991-11002.

[13] H. Kono, Y. Murata G, A.Kamimura, K. Tomita, and T,Suzuki. "Transfer Learning Method Using Ontology for Heterogeneous Multi-agent Reinforcement Learning", (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 5, No. 10, 2014,pp.156-164.

[14] M. Botvinick M, S. Ritter, X.Wang, Kurth-Nelson Z. Blundell.,and D. Hassabis . "Reinforcement Learning, Fast and Slow" , Trends in Cognitive Sciences, Vol. 23, No, 5 , 2019, pp. 26-41.

[15] A. Nichol, J. Achiam, and J. Schulman. "On First-Order Meta-Learning Algorithms", NIPS, 2018, pp.55-69.

[16] S. Jadon, and A. Garg. "Hands-On One-shot Learning with Python" Packet Publishing, 2020.

[17] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning", Proceedings of the Conference on Robot Learning, PMLR 100, 2020, pp.1094-1100.

[18] E. Todorov, T. Erez, Y. Tassa. And A. Mujoco." physics engine for model-based control", In International Conference on Intelligent Robots and Systems, IEEE, October 2012.

[19] A. Al-Oqaily, and G .Shakah, "solving Non-linear Optimization Problems Using Parallel Genetic Algorithm", International Conference on Computer Science and Information Technology (CSIT), IEEE, 2018.pp.103-106.

[20] A. V. Nair, V. Ong. M. Dalal. S. Bahl, S. Lin, and S. Levine. "Visual reinforcement learning with imagined goals", Advances in Neural Information Processing Systems 31, 2018.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. arXiv:1606.01540, 2016.