Manipulator Meta-Imitation Learning Algorithm with Memory Weight Integration

Mingjun Yin, Qingshan Zeng
School of Electrical Engineering, Zhengzhou University, Zhengzhou 450001, China
whitman1314@163.com

Abstract. Versatility is one of the key characteristics of general agent. In order to enable the manipulator to quickly and effectively acquire the ability to perform multiple tasks in an unknown environment, a large capacity model is essential. In this paper, the memory weight integration term adapted to meta-learning algorithm is proposed. By adjusting the plasticity of neurons, the manipulator can learn to learn more effectively in the process of learning multi-task and improve the forgetting problem of multi-task learning. Then, this paper combines the memory weight integration with meta-imitation learning, so that the manipulator can acquire new skills from a single demonstration task. Finally, a 7-DoF manipulator in PusherEnv experiment is used to explore the influence of different integration coefficients on the algorithm. The results show that the memory weight integration can effectively improve the success rate of tasks.

1. Introduction

With the upsurge of artificial intelligence, the proposal of imitation learning has infused new blood into the manipulator. Imitation learning enables the manipulator to achieve fast learning based on the behavior of imitating demonstrations with higher efficiency and intelligence. Many imitation learning strategies have emerged as the times require. Lots of machine learning techniques have been applied in imitation learning strategies and some good results have been achieved[1-2]. However, with the third wave of development of neural networks, deep neural networks have been superior to competing AI systems based on other machine learning technologies[3]. Due to the inherent shortcomings of deep learning itself, a huge number of training sets and trial error are needed, which makes it difficult to apply deep learning to the real manipulator. Therefore, in the framework of deep learning, the manipulator should have the ability to learn, which is meta-learning.

In the most popular meta-learning process, the model focuses on learning different tasks in training. There are usually two optimization processes: learners learning new tasks and meta-learners training learners[4-5]. The existence of meta-learners enables the strategy to adapt quickly to new tasks and achieve the goal of fast learning. The MAML (Model-Agnostic Meta-Learning) algorithm combined with deep neural network greatly improves the generality of meta-learning strategy[6]. The core idea of the algorithm is to train the initialization parameters of the model by using the information shared between multi-tasks, so that the model can be updated by one or a few gradients after learning a small amount of new task data, which shows better performance on new tasks. In further research, Chelsea[7] combined imitation learning with meta-learning and used demonstration data sets of many other tasks to carry out meta-learning. This model learned the new task from the example of a single new task, enabling the robot to learn the new task end-to-end with high efficiency without the introduction of additional mechanisms and other parameters of the task. However, due to the difficulty of the task,
there is room for further improvement in the success rate of the task.

In the training process of the neural network, because the neural network learns new tasks by fast overwriting, it will lose the necessary parameters to perform the tasks. This problem is called catastrophic forgetting. Similarly, in the process of meta-learning and training, there are also catastrophic forgetting problems. Marcus[8] found that continuous learning in the cerebral cortex depends on synaptic consolidation of specific tasks, in which knowledge can acquire persistent learning by reducing plasticity in some synapses. This means that the plasticity of individual neurons can be changed according to the importance of each neuron. Subsequently, the elastic weight consolidation method effectively overcomes the catastrophic forgetting in continuous learning and improves the learning ability of the neural network[9].

In this paper, the memory weight integration is added in the meta-updating stage of the algorithm to explore the effect of adjusting the plasticity of neurons on the learning ability of the model. Then, a model is constructed based on meta-imitation learning algorithm and CNN network. Finally, OpenAI Gym PusherEnv in Mujoco physical engine is used as a validation experiment to verify the learning performance of the algorithm after adding memory weight integration.

2. Meta-imitation learning algorithm with memory weight integration
Catastrophic forgetting is a common problem in deep neural network learning multiple tasks. Although meta-imitation learning algorithm (MIL) has the ability to learn many kinds of tasks, catastrophic forgetting is still difficult to overcome when the number of tasks increases and the complexity of tasks is high. Moreover, due to the small number of samples used in one-shot learning, forgetting in one-shot learning will lead to a significant reduction in the efficiency of new tasks in model learning. Therefore, this paper expects to find a way to strengthen multi-task learning ability and reduce the impact of catastrophic forgetting.

2.1. Memory weight integration
Meta-imitation learning aims to optimize model parameters so as to maximize effective behavior in one or a few gradient steps when learning new tasks. That is to say, this method strives to find a model parameter that maximizes the sensitivity of the loss function of learning new tasks. When the algorithm trains multiple tasks without forgetting the old ones, the model parameters will get an excellent initialization parameter, which improves the sensitivity of learning the loss function of new tasks. In addition, the plasticity of neuron parameters is also crucial. Therefore, for this reason, a penalty term is added to the loss function of the meta-gradient updating process to restrict each weight to ensure that the algorithm still has the ability to perform the old tasks when learning new tasks.

Learning a new task is to optimize the model performance by adjusting the weights and deviation parameter sets $\theta$ of each layer. Sometimes the same model performance can be obtained by different configurations. It is assumed that the model parameters $\theta$ are obtained after the model is initialized. Then, the learning parameters of each task $\theta'_{i}$ are obtained through the internal gradient updating stage. Taking learning two tasks as an example, two learning parameters are obtained: $\theta'_{a}$, $\theta'_{b}$. In the gradient updating stage of meta-learners, the model expects to get a model parameter $\theta$ with good performance on both tasks. Generally, the meta-update process will make the model parameters decrease toward the low error area of the new task. This is the main reason for the decline of performance on old tasks. Therefore, the memory weight integration protects the performance of the two tasks by restricting the updated model parameters to the intersection of low error regions centered on $\theta'_{a}$ and $\theta'_{b}$ or to a better performance region, respectively, as shown in Figure 1.

For all parameters, the low error region of each task is different. In order to restrain the effect of memory weight integration on loss function and achieve better performance of multi-task learning, the integration coefficient $\lambda$ is introduced as a constraint, $\lambda \in (0,1]$. 
In order to prove that the constraint is reasonable, the training process of the model is effective from the perspective of probability. From this point of view, optimizing model parameters is no different from finding the most probable estimates of model parameters under given data set $D$. Generally, Bayesian rules can be used to calculate the prior probability of parameters $p(\theta)$ and the probability of data sets $p(D|\theta)$.

$$
\log p(\theta|D) = \log p(D|\theta) + \log p(\theta) - \log p(D)
$$

(1)

Assume that the data is divided into two separate parts, one for task A ($D_A$) and the other for task B ($D_B$). Then, equation (1) is replaced by:

$$
\log p(\theta|D) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B)
$$

(2)

In the equation (2), the left side is still a posteriori probability given the parameters of the whole data set, while the right side $\log p(D_B|\theta)$ depends on the loss function of task B. As can be seen from the equation (2), when new tasks need to be learned, network parameters are adjusted by prior and this prior is the posterior distribution of previous tasks on given data parameters. Therefore, the key to implementing the memory weight integration is that the information about the old tasks must be absorbed into the prior distribution $p(\theta)$, so as to reduce the forgetting of the old tasks. Finally, the memory weight integration is listed as follows:

$$
\Omega(\theta_i') = \frac{\lambda}{2} (\theta - \theta_i')^2
$$

(3)

$\lambda$ is the integration coefficient and $i$ is the number of tasks.

When learning multiple tasks, the regularization item tries to keep the model parameters close to the learning parameters of each task. This can be achieved by multiple individual regularization items, but it is not difficult to find that the sum of multiple quadratic regularization items is itself a quadratic regularization item.

2.2. Manipulator meta-imitation learning algorithm

In meta-learning, assuming that the task obeys distribution $p(T)$, a strategy $\pi$ can map the observation $o$ to the predicted action $\hat{a}$. Each task is $T_i = \{t_i, a_{i1}, \ldots, a_{it_i}, o_{i1}, \ldots, o_{it_i}\}$. Among them, demonstration $T$ is generated by expert strategy $\pi_t^*$, and $L(a_{it}, \hat{a}_{it})$ is the loss function of the imitation task. The aim of the algorithm is to learn the weight $\theta$ of model $f_\theta$, so that the gradient descent can be carried out quickly on the new task $T$ obtained by $p(T)$ when a few demonstrations are adapted. Then, the meta-objective becomes:
\[
\min_{\theta} \sum_{T \sim m} L_{T}(f_{\theta}) = \sum_{T \sim m} L_{T}(f_{\theta}^C) 
\]

The updated parameter \( \theta' \) is calculated using one or task \( T_i \). In order to simplify the symbols, only one gradient updating is considered in this paper. The input of the model is \( o_t \), which is the observation value of time \( t \), for example, a picture, while the output \( a_t \) is the action taken in time \( t \), for example, the moment applied to the joint of the robot. Using the mean square error as the loss function of the strategy parameter \( \phi \), the following results are obtained:

\[
L_{T}(f_{\phi}) = \sum_{T \sim t} \| f_{\phi}(o_t^{(j)}) - a_t^{(j)} \|^2_2 
\]

The algorithm assumes that there are at least two examples for each task in the example data set. Then, using one of the examples of each task, gradient descent is used to calculate \( \theta' \) for each task according to equation (5). Next, the second example of each task calculates the gradient descent of the meta-objective based on equation (4) and equation (5). Finally, the memory weight integration is added in the meta-update phase, and \( \theta \) is updated according to the gradient of the meta-objective. The specific algorithm is as follows:

Algorithm 1. Meta-Imitation Learning with Memory Weight Integration

```
Require: \( p(T) \), \( \alpha \), \( \beta \), \( \lambda \)
1: randomly initialize \( \theta \)
2: while meta-train do
3: \hspace{1em} Sample batch of tasks \( T_i \sim p(T) \)
4: \hspace{1em} for all \( T_i \) do
5: \hspace{2em} Sample demonstration \( T = \{o_1, a_1, ..., o_t, a_t\} \) from \( T_i \)
6: \hspace{2em} Evaluate \( \nabla_{\phi} L_{T}(f_{\phi}) \) using \( T \) and \( L_{T} \) in equation (5)
7: \hspace{2em} Compute adapted parameters with gradient descent: \( \theta' = \theta - \alpha \nabla_{\phi} L_{T}(f_{\phi}) \)
8: \hspace{2em} Sample demonstration \( T' = \{o'_1, a'_1, ..., o'_t, a'_t\} \) from \( T_i \) for the meta-update
9: \hspace{1em} end for
10: \hspace{1em} Update \( \theta \leftarrow \theta - \beta [L_{T}(f_{\phi}) + \frac{\lambda}{2}(\theta - \theta')^2] \) using each \( T' \) and \( L_{T} \) in equation (5)
11: end while
12: return
```

3. Experiment

In order to evaluate the performance of meta-imitation learning algorithm with memory weight integration, the OpenAI gym Pusher experiment in Mujoco physics engine is used to verify the superiority of the algorithm. The experiment is to control a 7-DoF manipulator to push the target object at random starting position on the plane to the red target, where another object is placed as interference. The experiment involves challenges in many fields, including torque control of a 7-DoF manipulator, multi-task learning and visual diversity. Therefore, the experiment can verify the representation ability of the algorithm very well. The experimental model runs under the Tensorflow deep learning framework and uses NVIDIA 1070 graphics card to complete the operation.

3.1. Model structure

This model uses the same model structure in meta-imitation learning[7]. The model has four strided convolution layers with 16 5x5 filters, one spatial softmax layer with hidden dimension 200 and three fully-connected layers, as shown in the following Figure 2. The input consists of 125x125 RGB image,
joint velocities, joint angles and end-effector pose. The output is manipulator action. ReLU is used as the activation function. The normalization strategy is layer normalization.

3.2. Experiment setup
After the training of the model, 74 tasks will be tested to calculate the success rate and evaluate the performance of the model. Each task will be tested six times. In the 100 time steps of each test, when the center of the target object is in the red target circle in at least 10 time steps, the task is judged to be successful. There are 116 target objects in the task, 105 of which are used for training and 11 for evaluation. In addition, 5000 images will be randomly extracted for object surface texture. Specific demonstrations and test tasks are shown in the Figure 3. Each pair of pictures shows the initial scene and the end scene. Each demonstration is generated by expert policy. The initial position of the object for each test task is different. For optimization each method, 15 tasks are used for a meta-batch size. The model uses 1 inner gradient descent step with step size $\alpha = 0.01$. The iteration step of model training is 30000.

3.3. Experimental result
In order to explore the effect of memory weight integration on the model, different integration coefficients $\lambda$ will be selected to train the model. The task success rate will be verified and compared through experiments. According to Table 1, when the integration coefficient is 0.6, the task
success rate is the highest, compared with 0, the success rate is increased by 4.12%. Before $\lambda = 0.6$, task success rate increased with the increase of integration coefficient. When the integration coefficient increases further, the success rate will not increase but decrease. This is due to the fact that the memory weight integration consolidates the prior as well as some priori parameters that are useless for the old tasks, resulting in poor learning effect for the new tasks. However, how to select the important parameters in the prior is not the focus of this study and will be placed in the future research. But the experiment shows that the memory weight integration has a good performance. By remodeling the plasticity of neurons, it has a good effect on learning new tasks and reducing the forgetting of old tasks.

| $\lambda$ | success rate |
|----------|--------------|
| 0.0      | 85.43%       |
| 0.1      | 86.20%       |
| 0.2      | 86.92%       |
| 0.3      | 87.41%       |
| 0.4      | 88.02%       |
| 0.5      | 88.97%       |
| 0.6      | 89.55%       |
| 0.7      | 89.49%       |
| 0.8      | 89.51%       |
| 0.9      | 89.02%       |
| 1.0      | 88.87%       |

In addition, the training and validation errors can be found by observing the meta-imitation learning algorithm and memory weight integration (MWI). Although the loss errors fluctuate in a larger range at $\lambda = 0.6$, their errors remain in a lower range, which makes the model have better expression performance, as shown in the Figure 4. In the training process, the training error of MWI converges to 41.75 after 30,000 iterations, which is 2.66% lower than that of MIL. In the validation process, the validation error of MWI is 5.29% lower than that of MIL. The experimental analysis shows that the memory weight integrator makes the final model parameters in a better state of expression learning.
4. Conclusion

In this paper, a memory weight integration term suitable for meta-learning algorithm is proposed. Combining it with meta-imitation learning algorithm, a deep meta-learning model based on convolutional neural network is built. Through probability angle analysis, the validity of memory weight integration term in meta-learning algorithm is verified. The performance of the model is validated by the simulation experiment of a 7-DoF manipulator. The selection of the integration coefficient is compared and discussed. Experiments show that the performance of the model is the best when the integration coefficient is 0.6. Compared with the meta-imitation learning algorithm, the task success rate is increased by 4.12%. Therefore, this item can improve the learning performance of the algorithm by changing the plasticity of neurons and reducing the forgetting of old tasks by using prior while learning multiple tasks at one time.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61473265).

References

[1] Li C. (2017) Research on object recognition and crawl location based on machine vision and deep learning. North University of China.
[2] Du X., Cai Y., Lu T., Wang S., Yan Z. (2017) A robotic grasping method based on deep learning. Robot, 39(6): 820-837.
[3] Goodfellow I., Bengio Y., Courville A. (2016) Deep learning. MIT Press, Cambridge.
[4] Bengio S., Bengio Y., Cloutier J. (1992) On the optimization of a synaptic learning rule. Optimality in Artificial and Biological Neural Networks, 3: 6-8.
[5] Schmidhuber J. (1992) Learning to control fast-weight memories: An alternative to dynamic recurrent net works. Neural Computation, 4(4): 131-139.
[6] Finn C., Abbeel P., Levine S. (2017) Model-agnostic meta-learning for fast adaptation of deep networks. International Conference on Machine Learning. Sydney, pp. 1126-1135.
[7] Finn C., Yu T., Zhang T., Abbeel P., Levine S. (2017) One-shot visual imitation learning via meta-learning. Conference on Robot Learning. Mountain View. pp. 357-368.
[8] Benna M K., Fusi S. (2016) Computational principles of synaptic memory consolidation. Nature neuroscience, 19(12): 1697.
[9] Kirkpatrick J., Pascanu R., Rabinowitz N. (2017) Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences, 114(13): 3521-3526.