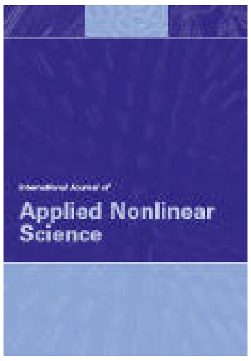


International Journal of Applied Nonlinear Science

ISSN online: 1752-2870 - ISSN print: 1752-2862
https://www.inderscience.com/ijans

Meta Q-network: a combination of reinforcement learning and meta learning

Min Lu, Yi Wang, Wenfeng Wang

DOI: 10.1504/IJANS.2022.10050329

Article History:

Received:	31 May 2021
Last revised:	22 August 2021
Accepted:	26 January 2022
Published online:	06 September 2022

Meta Q-network: a combination of reinforcement learning and meta learning

Min Lu and Yi Wang

School of Science,
Shanghai Institute of Technology,
Shanghai, 201418, China
Email: mlanran@163.com
Email: 2072949477@qq.com

Wenfeng Wang*

Sino-Indian Joint Research Center
of Artificial Intelligence and Robotics,
Interscience Institute of Management and Robotics,
Bhubaneswar, 752054, India
and
School of Electrical and Electronics Engineering,
Shanghai Institute of Technology,
Shanghai, 201418, China
Email: wangwenfeng@nimte.ac.cn
*Corresponding author

Abstract: Deep reinforcement learning develops rapidly by using neural network to approximate the learning data of reinforcement learning, which makes the sequential decision in continuous space making achieve preliminary results. However, deep reinforcement learning is over-dependent on huge amount of training and requires accurate reward. For many problems in the real world, such as robot learning, there is generally no good reward and no unlimited training, which requires the ability to learn quickly. In this paper, we propose a deep reinforcement learning model with meta-learning, which we call meta Q-network (MQN). The model uses a LSTM-based meta-learner to update the Q-network. This optimises a series of problems such as the difficulty in stability of Q-network in deep reinforcement learning model, and we have proved this improved performance through experiments. It is not optimal, though, still the combination of meta-learning and reinforcement learning is very desirable.

Keywords: machine learning; reinforcement learning; meta-learning; artificial intelligence.

Reference to this paper should be made as follows: Lu, M., Wang, Y. and Wang, W. (2022) 'Meta Q-network: a combination of reinforcement learning and meta learning', *Int. J. Applied Nonlinear Science*, Vol. 3, No. 3, pp.179–188.

Biographical notes: Min Lu is a postgraduate student of Shanghai Institute of Technology, School of Science. His research interests include optical and machine-learning.

Yi Wang is a postgraduate student of Shanghai Institute of Technology, School of Science. His research interests include MEMs and machine-learning.

Wenfeng Wang is a Professor of Shanghai Institute of Technology, and a Tenured Professor of the Indian institute of IMT. He has long been mainly engaged in the cognitive computing research, focuses on interdisciplinary application mathematics, and has been involved in hydrology, ecology, geography, computer vision, robotics, security, medical and other fields.

1 Introduction

The concepts and terms of ‘reinforcement’ and ’reinforcement learning’ were first proposed by Minsky in 1954. In 1989, Q learning proposed by Watkins further expanded the application of reinforcement learning and perfected reinforcement learning. Q learning makes it possible to find the optimal action strategy in the absence of knowledge of the immediate reward function and the state transition function. The best example of the value of reinforcement learning is that DeepMind trained AI to play Atari 2600 games using the deep-Q-network method in 2013 (Mnih et al., 2013), which achieved a good transfer learning value. In other words, the AI trained on a certain game can quickly learn to play other games without special training. Then, in 2015, DeepMind came up with an improved version of the DQN algorithm (Mnih et al., 2015), which uses delayed updates to reduce dependencies between network parameters. In 2016, DeepMind proposed a reinforcement learning algorithm called deep deterministic policy gradient (DDPG) for solving the continuous motion space, and in 2017, OpenAI proposed proximal policy optimisation (PPO) algorithm, which is better than the previous deep reinforcement learning algorithm in terms of convergence and stability, and is also the most versatile algorithm at present. Structurally, both DDPG and PPO belong to actor-critic algorithm.

Deep learning has achieved remarkable development in recent years, and has achieved a great deal of achievements in many fields, such as computer vision (He et al., 2016; Wei et al., 2019b), speech recognition (Oord et al., 2016), translation (Wu et al., 2016), and go (Silver et al., 2016). Deep learning, however, still only solves one-to-one problems and relies on a lot of training. Meta-learning, on the other hand, has recently shown outstanding performance in small sample learning tasks (Naik and Mammone, 1992; Thrun and Pratt, 1998; Schmidhuber, 1987). Different from deep learning, meta-learning is used to solve one-to-many tasks. At present, the main meta-learning algorithms include metric-based methods (Koch et al., 2015; Vinyals et al., 2016; Sung et al., 2018), model-based methods (Santoro et al., 2016; Munkhdalai and Yu, 2017), and optimisation-based methods (Ravi and Larochelle, 2016; Nichol et al., 2018; Finn et al., 2017). Although the methods are different, they share the same purpose, that is, they want a meta-learner to be able to learn quickly from a very limited samples and to be able to adapt to different tasks.

Here, we believe that deep reinforcement learning and meta-learning have certain similarities in the tasks they face. An important part of deep reinforcement learning is the value approximation function. In the actual training process, complex environment and a large number of samples make it difficult for the network to converge. Therefore, we hope to use the meta-learning method to improve the adaptability of deep reinforcement learning. So, we propose a model combining the optimisation-based meta-learning and DQN to improve the stability and performance of DQN. We try learn a set of parameter updating methods by training a LSTM and used them to replace gradient descent methods, although gradient descent method shows its advantages on many occasions (Wei et al., 2017). However, there are still limitations when the training sample is unstable, and for multiple separated tasks, the network usually performs random parameter initialisation, which seriously affects the ability of the network to complete the optimisation with fewer parameter updates. In addition, this method can also be applied to more complex deep reinforcement learning algorithms, such as DDPG (Lillicrap et al., 2015) and PPO (Schulman et al., 2017), as a method to replace gradient descent for parameter updating.

2 Model description

In this section, we will describe our model and how it trains and updates parameters in each network in detail. MQN consists of two parts, the first part is the base learner. In the concept of meta-learning, we usually refer to the principal network used to achieve the target function as the base learner, that is, the Q-network in a general DQN. The second part is the embedded model. Embedded models tend to play an auxiliary role, helping base learners perform better, where we use the meta-LSTM, which is a special LSTM proposed by Ravi and Larochelle (2016). We denote Q-network in DQN as Q, and meta-LSTM as M. Now, let's start with the first part.

For the base learner, we use DQN with experience replay and target Q-network. DQN is a typical value-based algorithm which only applies to discrete motion spaces, and different from actor-critic algorithm represented by DDPG and trust region policy optimisation (TRPO) there is only one action-value function network and no policy network. Which makes a relatively simple model to train, and this is one of the reasons we chose DQN. A simple model does not always mean a draw-back. On some simple tasks, it can get a better result with less cost, which makes it easier for us to carry out some experiments. On the other hand, the embedded model is also applicable to DDPG, PPO and other algorithms.

The core of DQN is action-value function network which is called Q-network, and a neural network is commonly used as nonlinear function approximator and trained to let $Q(s, a; \theta) \approx Q^*(s, a)$ and its parameters is updated by gradient decent. The loss function of Q-network at each epoch t can be written as (Mnih et al., 2015):

$$\mathcal{L}_t(\theta_t) = \mathbb{E}[(y_t - Q(s, a; \theta_t))^2]$$

with $y_t = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \theta_{t-1})|s, a]$ we can have the gradient by differentiating the loss function with respect to the parameters:

$$\nabla_{\theta_t} \mathcal{L}_t(\theta_t) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{t-1}) - Q(s, a; \theta_t)\right) \nabla_{\theta_t} Q(s, a; \theta_t)\right]$$

It is worth noting here that, unlike typical deep learning networks, Q-networks are usually difficult to converge in the process of training, especially in some complex tasks. In other words, an arbitrarily finite neural network does not have infinite representation capacity, we are just looking for a parameter among possible network space parameters that can minimise the loss. In the continuous state space, where there is an infinite number of states, the cumulative return can suddenly get better and then suddenly get worse at two states that are very close. In the process of training, Q-network is likely to show sensitivity to the input state space parameters, resulting in chaos. This leads to the very unstable nature of DQN. Therefore, we proposed an algorithm combining reinforcement learning and meta-learning. We used the idea of meta-learning to let a meta-learner learn a parameter update method to optimise the performance of DQN.

Referring to Ravi and Larochelle (2016), we hope to use a LSTM (meta-learner) to update its cell state form instead of the original gradient descent. The updating of cell state of LSTM can be written as following (Hochreiter and Schmidhuber, 1997):

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

and with $i_t = \alpha_t,\ c_t = \theta_t,\ \tilde{c}_t = -\bigtriangledown_{\theta_t} \mathcal{L}_t$ we can apply this to the gradient of Q-network:

$$\tilde{c}_t = -\mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{t-1}) - Q(s, a; \theta_t)\right) \nabla_{\theta_t} Q(s, a; \theta_t)\right]$$

$$\theta_t = f_t \theta_{t-1} + i_t \tilde{c}_t$$

also the i_t and f_t is defined in parametric forms so that meta-learner can determine optimal values through the course of the updates:

$$i_t = \sigma[W_I(\nabla_{\theta_t}\mathcal{L}_t(\theta_t), \mathcal{L}_t(\theta_t), \theta_{t-1}, i_{t-1}) + bI]$$

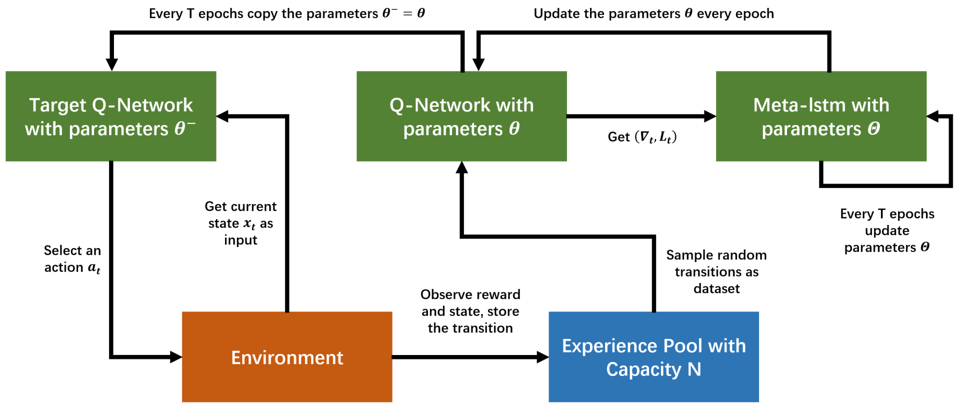
$$f_t = \sigma[W_F(\nabla_{\theta_t}\mathcal{L}_t(\theta_t), \mathcal{L}_t(\theta_t), \theta_{t-1}, f_{t-1}) + bF]$$

Therefore, we hope that through the meta-learner, learners can consider both short-term knowledge and long-term knowledge of a task, and converge to a good solution. We call this combined model ‘meta Q-network’ (MQN), that is, DQN under the optimisation-based meta-learning method.

3 Proposed algorithm

The training process is shown as Figure 1. It should be noted that meta-learner does not update parameters of Q-network synchronously with meta-learner’s own parameters. In order to make Q-network more stable, meta-learner should update its parameters independently only after Q-network is updated for several times. Similarly, in DQN, when the neural network is used to approach the Q value function, the update of Q value is prone to shock and presents unstable learning behaviour. Therefore, the target Q-network is introduced (Mnih et al., 2015). The parameters of target Q-network are updated at every T epochs independently of the Q-network. Here, an epoch means that the Q-network completes a parameter update. This allows the Q value to be fixed temporarily during the training process, making the learning process more stable. It can be seen that the update strategy of target Q-network is consistent with the purpose of update of meta-learner. Thus, we think it makes perfect sense to keep the frequency of them in line, which means, every T epochs meta-learner will update its parameters $\Theta_T = \Theta_{T+1}$ and at the same time, the parameters of Q-network θ_T will copied to target Q-network. We believe that this will increase the stability of the algorithm.

Figure 1 Graph of MQN's training process (see online version for colours)


Every T epochs copy the parameters $\theta^- = \theta$
Update the parameters θ every epoch
Target Q-Network with parameters θ^-
Q-Network with parameters θ
Meta-lstm with parameters Θ
Get (∇_t, L_t)
Get current state x_t as input
Select an action a_t
Every T epochs update parameters Θ
Sample random transitions as dataset
Observe reward and state, store the transition
Environment
Experience Pool with Capacity N

Algorithm 1 Meta Q-network

```
Initialise replay memory $D$ to capacity $N$
Initialise action-value function $Q$ with random weights $\theta$
Initialise target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
Initialise meta-LSTM $M$ with random weights $\Theta$
for episode = 1, M do
  Initialise sequence $s_1 = x_1$ and pre-processed sequence $\phi_1 = \phi(s_1)$
  for d = 1, C do
    for t = 1, T do
      select a random $a_t$ with probability $\varepsilon$ otherwise $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
      excute action $a_t$ and observer reward $r_t$ and state $x_t$
      set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      sample random minibatch of transition $(\phi_t, a_t, r_t, \phi_{j+1})$ from $D$
      set $y_i = r_j$ (if episode terminates) otherwise $y_i = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$
      update parameters $\theta$ with meta-LSTM $\theta_t = M(\nabla_{\theta_{t-1}} L_t, L_t; \Theta_{d-1})$
    End for
    update parameters $\Theta$ with $\nabla_{\Theta_T} L_T$ and reset $\hat{Q} = Q$
  End for
End for
```

4 Experiments and results

We chose two different tasks to test and evaluate our model, Cartpole and Lunarlander. They belong to OpenAI's Gym, an open source interface to reinforcement learning tasks.

And as a contrast, we have trained a DQN as baseline, DQN with policy gradient and actor-critic are also iconic benchmarks.

Figure 2 (a) CartPole (b) LunarLander in OpenAI's gym (see online version for colours)

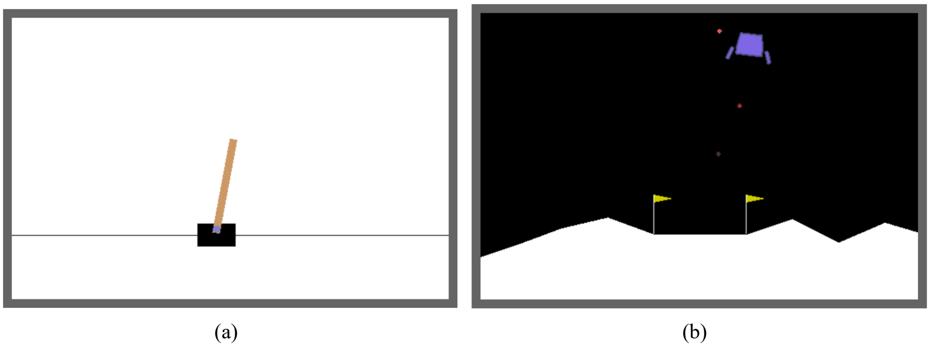
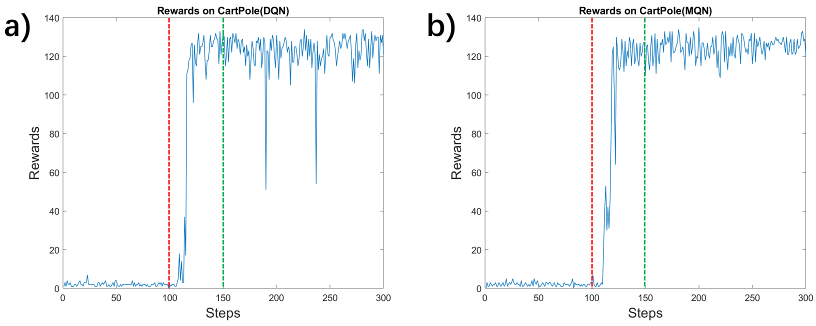


Figure 3 The plots show the rewards on CartPole, the red dash-line means that the memorise capacity is full and the model start training (see online version for colours)

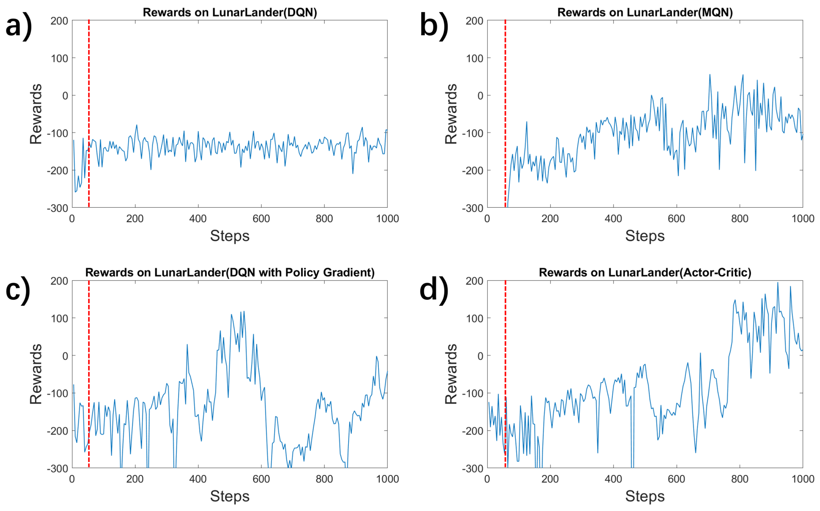


For Q-network in our MQN, we use a simple neural network containing two hidden layers with ReLU nonlinearity, each of which has 64 and 32 nodes. The loss function is MSEloss. For meta-learner, we use a two-layer LSTM, where the first layer is a normal LSTM and the second layer is meta-LSTM. The gradient and loss are preprocessed and input into the first layer LSTM to get the regular gradient coordinates and the second layer of LSTM is used to implement the state update.

For vanilla DQN (Mnih et al., 2015), DQN with policy gradient and actor-critic, we use the same Q-network (actor and critic network) as in MQN, and Adam optimiser (Kingma and Ba, 2014) is chosen to update the parameters.

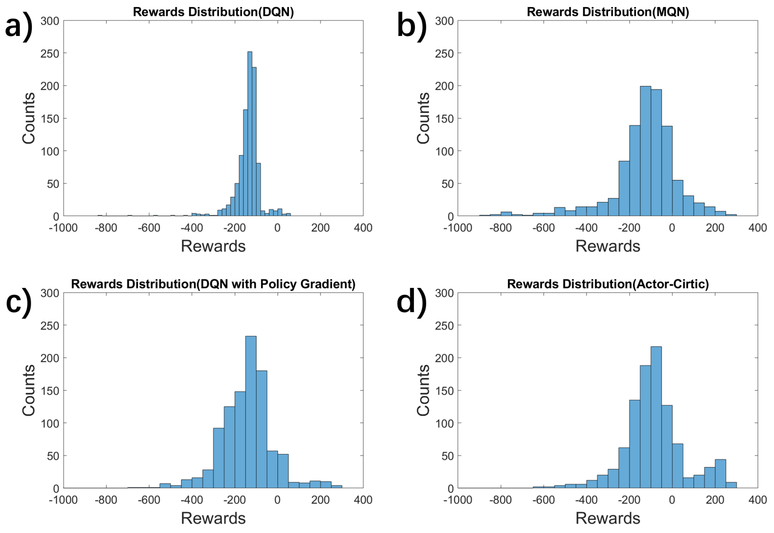
All of them share the same learning rate of 0.01, batch size of 256, reward decay of 0.9, memorise capacity of 1,024. In the first task (CartPole), we let exploration rate ε = 0.3 at the very first, and gradually decline to 0, also, we modified the reward for a quick start. For the second task (LunarLander), we keep the ε = 0.15. Each task is repeated by five times, and the best one is given as the result in Figures 3–5. In addition, since we are training the model from scratch, we need to ensure that we start training only when the experience pool is full, so as to reduce the correlation of samples over time. When the experience pool is not full, the model will act randomly. We distinguish the two stages by the red dash-line in the figure. A step means the completion of a game, and then we record a total reward.

Figure 4 The plots show the rewards on LunarLander, the red dash-line means that the memorise capacity is full and the model start training (see online version for colours)



Note: Each point is the average score of every five steps.

Figure 5 The histogram show the distribution of the rewards corresponding to each case in Figure 4 (see online version for colours)



5 Discussion

Let’s focus on the CartPole task first (Figure 4 and Table 1), and we can see from the results that vanilla DQN performs pretty well on CartPole, which is not surprising since CartPole is a relatively simple task. However, we can still find that the overall performance of MQN is better than that of vanilla DQN. It is worth noting that what we are discussing here is the convergence of rewards, rather than the convergence of Q value or loss in Q-network, because Q value or loss here do not reflect the performance of the model. Although we hope to have a small loss in the training process, we are more concerned about the maximisation and stability of rewards. At the same time, for nonlinear neural networks, it is also difficult to prove whether Q-network can converge in the end. So we want the total return to converge to a larger value, which means, a consistent, excellent performance. MQN achieved better average rewards and better stability, although the advantage in average reward was not very obvious, as can be seen from Figure 4, the convergence of MQN was better than that of vanilla DQN. In the meantime, we can confidently guess that when we reduce the memory capacity N, the advantage of MQN will become evident, because this is consistent with the characteristics of meta-learning.

DQN, on the other hand, performed poorly on Lunarlander, which can be seen in Figure 4 and Table 2. Compared with CartPole, Lunarlander is a more difficult and complex task. It is not hard to notice that it is difficult to achieve an ideal result in 1,000 steps with four different algorithms. This can be attributed to many reasons, such as the improper selection of architecture or hyperparameters, and also we know that deep reinforcement learning is difficult to converge on complex tasks. However, we can ensure that our experiments are fair to any of the algorithms, so this does not affect our evaluation. Let's take a close look at Figure 4, DQN with policy gradient is the first one to reach the an average positive rewards at about 360 steps. Then, around 600 steps, however, performance suddenly deteriorates. Actor-critic reaches an average positive rewards late at about 675 steps, but after that, it seems to converge to a better solution. MQN is kind of a combination of the policy gradient and actor-critic. It reaches an average positive rewards at about 520 steps, and then it tends to converge to a not so good reward. It is worth mentioning that MQN got off to a bad start and we think that had an impact on its performance for a while To facilitate qualitative evaluation, we defined rewards above 200 as 'excellent', which is very close to the optimal strategy. Rewards in the 150–200 range were considered 'good', while those below 0 were considered to be 'poor'. Rewards in the 0–150 range are difficult to evaluate, because sometimes the intelligence will deliberately perform 'suicidal actions' to obtain a less low reward, which we consider, is a poor local optimal solution. As can be seen from Figure 5(a), vanilla DQN never get more than 100 rewards at a time, and its overall performance is poor. Figures 5(b)–5(d) shows that the remaining three algorithms can all obtain the result of more than 200 awards, that is, an 'excellent'. However, as shown in Table 2, we can clearly see the obvious fluctuation, and the direct cause of this problem, we think, should be the unstable Q value. This may imply the chaotic behaviour of Q-network. However, due to the high dimensional input, in-depth analysis of this phenomenon is difficult. but we can see that this phenomenon is evident in DQN with policy gradient and actor-critic, though it has reached the highest rewards. While the solution to this problem is still unclear, although MQN shows better stability. At the same time, we think that applying a clustering algorithm might be a good approach (Wei et al., 2019a).

As can be seen from the experimental results, MQN has a more stable performance than DQN in simple tasks, and is superior to DQN in responsible tasks. At the same time, MQN shows some competitiveness with policy gradient and actor-critic. This is in line with our expectations, as we are still using the vanilla DQN in MQN. MQN is an optimisation-based combination of meta-learning and reinforcement learning. This is not in conflict with other model-based reinforcement learning methods. In other words, it means that we can apply the same optimisation methods to more mature reinforcement learning algorithms such as DDPG or PPO and expect them to have better performance.

Table 1 The average reward and standard deviation of each model on CartPole corresponding to Figure 3

Model	*Average reward*	*Standard deviation*
DQN	119.9789	32.9168
MQN	114.1650	21.6688

Note: Only part of the data to the right of the green dash-line was counted.

Table 2 The average reward and standard deviation of each model on LunarLander corresponding to Figure 4

Model	*Average reward*	*Standard deviation*
DQN	–136.2507	51.0999
MQN	–105.3460	118.1283
DQN with policy gradient	–142.5020	122.4410
Actor-critic	–80.7989	175.1368

Note: Only part of the data to the right of the red dash-line was counted.

6 Conclusions

In this paper, we propose a reinforcement learning model combined with meta-learning, and illustrate its advantages over deep reinforcement learning. Although the structure of this model is relatively simple, and there are still many deficiencies, more importantly, we believe that the method of introducing meta-learning into reinforcement learning is likely to become a branch of reinforcement learning and artificial intelligence development in the future, although there are few similar studies in this direction. Applying different meta-learning methods to more mature reinforcement learning methods will be our task in the future.

References

Finn, C., Abbeel, P. and Levine, S. (2017) 'Model-agnostic meta-learning for fast adaptation of deep networks', in *International Conference on Machine Learning*, PMLR, pp.1126–1135.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) 'Deep residual learning for image recognition', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.770–778.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long short-term memory', *Neural Computation*, Vol. 9, No. 8, pp.1735–1780.

Kingma, D. and Ba, J. (2014) *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs.LG].

Koch, G., Zemel, R. and Salakhutdinov, R. (2015) 'Siamese neural networks for one-shot image recognition', in *ICML Deep Learning Workshop*, Lille, Vol. 2.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2015) 'Continuous control with deep reinforcement learning', arXiv:1509.02971 [cs.LG].

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013) *Playing Atari with Deep Reinforcement Learning*, arXiv:1312.5602 [cs.LG].

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. et al. (2015) 'Human-level control through deep reinforcement learning', *Nature*, Vol. 518, No. 7540, pp.529–533.

Munkhdalai, T. and Yu, H. (2017) 'Meta networks', in *International Conference on Machine Learning*, PMLR, pp.2554–2563.

Naik, D.K. and Mammone, R.J. (1992) 'Meta-neural networks that learn by learning', in *Proceedings 1992 International Joint Conference on Neural Networks*, IEEE, Vol. 1, pp.437–442.

Nichol, A., Achiam, J. and Schulman, J. (2018) *On First-Order Meta-Learning Algorithms*, arXiv preprint arXiv:1803.02999.

Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. (2016) *WaveNet: A Generative Model for Raw Audio*, arXiv preprint arXiv:1609.03499.

Ravi, S. and Larochelle, H. (2016) 'Optimization as a model for few-shot learning', in *Proc. ICLR*, pp.1–11.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D. and Lillicrap, T. (2016) 'Meta-learning with memory-augmented neural networks', in *International Conference on Machine Learning*, PMLR, pp.1842–1850.

Schmidhuber, J. (1987) *Evolutionary Principles in Self-Referential Learning, or on Learning How to Learn: The Meta-Meta-... Hook*, PhD thesis, Technische Universität München.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017) *Proximal Policy Optimization Algorithms*, arXiv.1707.06347 [cs.LG].

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016) 'Mastering the game of go with deep neural networks and tree search', *Nature*, Vol. 529, No. 7587, pp.484–489.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H. and Hospedales, T.M. (2018) 'Learning to compare: relation network for few-shot learning', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1199–1208.

Thrun, S. and Pratt, L. (1998) 'A survey of connectionist network reuse through transfer', in *Learning to Learn*, pp.19–43, Springer Science & Business Media.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K. and Wierstra, D. (2016) *Matching Networks for One Shot Learning*, arXiv preprint arXiv:1606.04080.

Wei, W., Song, H., Wei, L., Shen, P. and Vasilakos, A. (2017) 'Gradient-driven parking navigation using a continuous information potential field based on wireless sensor network', *Information Sciences*, Vol. 408, No. 2, pp.100–114.

Wei, W., Xu, X., Marcin, W., Fan, X. and Ye, L. (2019a) 'Multi-sink distributed power control algorithm for cyber-physical-systems in coal mine tunnels', *Computer Networks*, Vol. 161, No. 9, pp.210–219.

Wei, W., Zhou, B., Połap, D. and Woźniak, M. (2019b) 'A regional adaptive variational PDE model for computed tomography image reconstruction', *Pattern Recognition*, Vol. 92, pp.64–81, DOI: 10.1016/j.patcog.2019.03.009.

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al. (2016) *Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation*, arXiv preprint arXiv:1609.08144.