Transfer Learning in Deep Reinforcement Learning

Tariqul Islam, Dm. Mehedi Hasan Abid, Tanvir Rahman, Zahura Zaman, Kausar Mia, and Ramim Hossain

Abstract Reinforcement learning has quickly risen in popularity because of its simple, intuitive nature, and its powerful results. In this paper, we study a number of reinforcement learning algorithms, ranging from asynchronous q-learning to deep reinforcement learning. We focus on the improvements they provide over standard reinforcement learning algorithms, as well as the impact of initial starting conditions on the performance of a reinforcement learning agent.

Keywords Deep learning · Transfer learning · Reinforcement learning · Convolutional neural networks · Q-networks

1 Introduction

Reinforcement learning is a class of machine learning algorithms that are designed to allow agents provided with only the knowledge of the states it visits and the actions available to the agent to learn how to maximize its reward function, quite similar to the trial-and-error approach. There are different techniques used for reinforcement learning, one of the most popular ones being Q-learning where an agent develops a policy that chooses the action that is estimated to lead to the greatest total

T. Islam (✉) · Dm. M. H. Abid · T. Rahman · Z. Zaman · K. Mia · R. Hossain
Daffodil International University, Dhaka, Bangladesh
e-mail: tariqul15-2250@diu.edu.bd

Dm. M. H. Abid
e-mail: mehedi15-226@diu.edu.bd

T. Rahman
e-mail: tanvir15-2245@diu.edu.bd

Z. Zaman
e-mail: zahura15-1381@diu.edu.bd

K. Mia
e-mail: kausar15-2248@diu.edu.bd

R. Hossain
e-mail: ramim15-2246@diu.edu.bd

X.-S. Yang et al. (eds.), *Proceedings of Seventh International Congress on Information and Communication Technology*, Lecture Notes in Networks and Systems 447,
https://doi.org/10.1007/978-981-19-1607-6_13

future rewards. Reinforcement learning has seen great recent success, particularly in Playing Atari with Deep Reinforcement Learning [1] and Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [2] as it is a relatively simple yet extremely powerful algorithm, making it an interesting class of learning algorithms to study. Furthermore, the training of reinforcement learning agents is extremely slow, since the information it is provided is minimal, which means that there is a lot of room for improvement with reinforcement learning algorithms. Transfer learning on the other hand is a class of machine learning algorithms that seek to transfer knowledge gained from solving one problem and applying it to another problem, so transfer learning can solve the problem of speed for reinforcement learning agents. In this paper, we discuss the impact of initial conditions with transfer learning on the convergence of reinforcement learning agents. In real life, we know that initial starting conditions matter. Consider a person who chooses to learn a sport: the athletic ability, age, equipment, training, and instructor will all influence the time it takes for the person's skill to peak. If it were at all possible, we would want to transfer the traits of high-performing athletes to the beginner to provide better chances at performing well. Based on this intuition, we want to experiment with transferring the models of trained reinforcement learning agents as the initial starting conditions of reinforcement learning algorithms and confirm that this hypothesis does indeed apply here too. That is, we want to show that given better initial conditions, an agent will likely achieve high performance faster than an agent with worse initial conditions. This is reasonable, and we can easily produce simple examples that illustrate the point. Consider an extreme example where an agent uses a neural network to model its policy, and all the weights in the network are initialized to zero. Then all the weights follow the same gradients, and the policy will likely perform poorly. Conversely, an agent with a policy model that has been trained for extremely long periods of time will likely be much closer to optimality: Hence, it will likely take much less time to converge. Intuitively, it makes sense that better initial conditions lead to optimal performance faster, and we wish to establish this for reinforcement learning, by means of a simple form of transfer learning.

2 Related Work

An interesting improvement to Q-learning is asynchronous Q-learning (AQL). This technique involves one central, shared neural network. Then each asynchronous agent copies the shared network as its own individual network, learns on its own, and periodically shares its accumulated updates with the shared neural network. Furthermore, each agent will periodically copy the shared neural network as its own individual neural network, making use of the learning that other agents have done. In effect, an AQL agent searches across multiple locations in the state space while sharing information with other agents, speeding up its learning process. Wang et al. [3] represent the ride dispatching problem and suggest suitable solutions which are based on deep Q-networks. Nowadays, the GPS authorization applications are

getting more popular and are also used in ride-sharing. To get the result, they use a window of 100 circumstances for counting the reward curve and the total number of training duration is 40,000 circumstances. Our work has suggested a procedure which has based on DQN for this dispatching platform. They are successful to show that CFPT is most successful and better than other methods. Victoria et al. [4] aim to establish a method for deep reinforcement learning that will refine the effectiveness and capacity of this advisable method by structural perceptivity and relational argument. They advisable relational model has gained favorable performance and solved more than 98% of levels. Lample et al. [5] focus on representing a structure to face 3D infrastructure in FPS games. In recent times, deep reinforcement education has shown much success to achieve human-level control. In this paper, they describe a procedure to increase the efficiency of the model to utilize the information of game features. They apply the DQRN model because of its good performance accuracy. This model is instructed and used to shorten Q-learning. Our advisable structure is trained to permit various models in various phases of FPS games. This paper [6] aims to establish an efficient model that will repetitively store the results of a chemical reaction and select new exploratory conditions to upgrade feedback outcomes. Here they take random 5000 functions, and the DRO takes 32 steps to arrive at the standard of regret, where some other algorithms such as CMA-ES takes 111 steps, SNOBFIT takes 187 steps, and Nelder–Mead fails. Our established DRO model has shown its remarkable performance to optimize chemical reactions. This model has already shown its ability to optimize and also increase the speed of reaction. Baldazo et al. [7] aim to suggest a new model for the mean embedding of distribution which is based on DRL. Nowadays, DRL has widely used for solving various multi-agent collaboration problems. In our advisable model, they use the agents as a sample and as input use mean embedding. Besides they describe various features of the mean embedding by using radial basis functions and training neural networks. The paper [8] aims to establish some effective methods to upgrade exploration conjunctional optimization based on DRL. In recent times, DRL has successfully shown an excellent improvement to solve different kinds of control problems. The paper [9] aims to explore mobile edge computing for smart (IoT) based on deep reinforcement learning. In incent times, there has been tremendous advancement in developing IoT. Basically, Kiran et al. [10] aim to show a classification of automated driving activity where can apply DRL methods. With the advancement of the DRL network, the autonomous driving system has gained high fidelity.

3 Background

3.1 Reinforcement Learning

The reinforcement learning task is often formulated as a Markov decision process, a modeling framework useful for partially random, partially controlled environments,

which is certainly the case in reinforcement learning where the environment may behave randomly, but the agent has control over its own actions. In the reinforcement learning task, a Markov decision process consists of the following elements:

1. S_{E}: The set of states that the environment (with the agent in it) E can be in.
2. A_{E}: The set of possible actions that the agent can take in the environment.
3. W_{E}: $S_{\mathrm{E}} \times A_{\mathrm{E}} \rightarrow S_{\mathrm{E}}$: The function that determines the resulting state given a starting state and an action.
4. R_{E}: $S_{\mathrm{E}} \times S_{\mathrm{E}} \rightarrow R$: The function that gives the immediate reward for a state transition.

The agent constructs a policy π_{E}: $S_{\mathrm{E}} \rightarrow A_{\mathrm{E}}$ that maps a state in the state space to an available action that leads to the highest total immediate and future rewards. So we formulate a utility function $U_{\pi E}$: $S_{\mathrm{E}} \rightarrow R$ that determines all rewards received by following the policy given a starting point s_0:

$$\mathrm{U}\pi_{\mathrm{E}}(\mathrm{s}_0) = \sum_{t=0}^{\infty} \gamma^t R_{\mathrm{E}}(s_{\mathrm{t}}, W_{\mathrm{E}}(s_{\mathrm{t}}, \pi_{\mathrm{E}}(s_{\mathrm{t}}))) \tag{1}$$

Then the policy for our reinforcement learning agent can be defined as follows:

$$\pi_{\mathrm{E}}(s_{\mathrm{t}}) = \mathrm{argmax}_{a \epsilon \mathrm{AE}} U_{\pi \mathrm{E}}(W_{\mathrm{E}}(st,\ a)) \tag{2}$$

3.2 Q-learning

Often times, an agent does not have access to W_{E}, and in such cases, the agent's policy is said to be model-free. The agent must, then, estimate the utility function by its internal Q-value function $Q_{\pi \mathrm{E}}$: S_{E} ! R. A simple way to represent $Q_{\pi \mathrm{E}}$ is to use a table in which each possible state and action pair is listed, and the estimated cumulative reward is the entry. To learn the optimal Q-value function which we denote by Q_{E}, we use the Q-learning algorithm on our Q-value function $Q_{\pi \mathrm{E}}$. In one-step Q-learning, the algorithm takes one step at every training iteration t from state s_{t}, observes the reward received $\mathrm{r_t}$ and the new state $s_{\mathrm{t}} + 1$, and updates the policy as follows:

$$Q_{\pi \mathrm{E}}(s_{\mathrm{t}}) = Q_{\pi \mathrm{E}}(s_{\mathrm{t}}) + \alpha(r + \gamma Q_{\pi \mathrm{E}}(s_{\mathrm{t}} + 1) - Q_{\pi \mathrm{E}}(s_{\mathrm{t}})) \tag{3}$$

with α as the learning rate, typically a real number between 0 and 1. This algorithm sets the target value to be the discounted sum of all the future rewards estimated by $\gamma Q_{\pi \mathrm{E}}(s_{\mathrm{t}} + 1)$ added to the observed immediate reward r. The difference between the target value and output value is then a weighted by the learning rate and used to update the Q-function. To avoid settling for a non-optimal policy (premature convergence of policy), an exploration factor is introduced: Is the probability that the agent will

ignore its policy and execute a random action, to diversify its experiences and to avoid local minima in its policy. As time progresses, the exploration rate is decayed, so that the agent relies more (but not completely) on its policy. However, it is often the case that the exploration rate is not allowed to decay to 0 and is instead held at some fixed minimum exploration rate, to discourage the policy from sinking into a local minimum.

3.3 Q-networks

It becomes hard to maintain such a Q-table when the size of the state space increases: for example, consider an agent playing a video game, using the screen's pixel values as its state space. If the state is a $84 \times 84 \times 3$ array of 8 bit pixels, and there are four actions available, the q-value table will hold $284 * 84 * 3 + 2 \approx 106{,}351$ entries! A popular solution to the problem of poorly scaling tables is the use of artificial neural networks in Q-learning termed Q-networks [11, 12]. Q-networks map states in the state space, represented by frames from the game, to q-values for each possible action. Q-networks learn to approximate Q in a way intuitively similar to the update formula for the Q-table, by computing gradients for the network based on the output of the network (determined without knowing the next states) and target Q-values (determined using the next states) for the network [1, 13]. Q-networks are far more powerful than Q-tables because they can also approximate Q-values for states it has not yet seen and scales much better in terms of size. However, they come with the downside of being harder and slower to train.

4 Approach and Experiments

We first describe the infrastructure available to us for our experiments. For high numbers of independent experiments, we use a distributed high-throughput computing resource through the Center for High Throughput Computing (CHTC) available at UW-Madison. For our guaranteed convergence experiment, we tested using a custom maze environment with a state size of 4 and an action space size of 4. For our initial conditions experiment, we describe how we done operational convergence criteria for our problem setting [14, 15]. We say that the agent's learning has stopped if the winning rate over the last 100 evaluations averages to a value greater than 78, the same stopping criteria for the environment FrozenLake.

4.1 Guaranteed Convergence Given Infinite Time

We know from Even-Dar et al. [16] that using the action elimination algorithm, our reinforcement learning agent will converge given infinite time. Our hypothesis was that the algorithm would work for a reinforcement learning agent in an environment with an extremely simple problem with an extremely small state space. We expected to see the algorithm converge given a couple month's time. Unfortunately, the algorithm's progress exponentially diminished, and we never saw the convergence (or anything even close) after 2 months of running the algorithm on a high-throughput computing cluster. As such, we affirm that "Infinite time" really does mean some enormous time quantity that is infeasible. We ran this experiment on a Google Cloud Compute Engine instance with 8 cores and 32 GB RAM.

4.2 Impact of Initial Conditions on Convergence

We hope to find that given better initial conditions, our DQN agent will converge faster. We provided these initial conditions as trained DQN models, saved after various periods of pre-training. We hypothesize that models that have had more pre-training will require less time to converge, while models that have had little pre-training. We first show the baseline performances of each initial condition in Fig. 3. Then we show training times until convergence starting from each initial condition in Fig. 4. Our hypothesis is affirmed through this experiment as we can see that indeed agents with more pre-training had faster times to convergence. The pre-training was done on a system with an Intel i7-7700 k overclocked to 4.9 GHz with 32 GB DDR4 3200 MHz SDRAM on a Samsung 960 EVO M.2 drive. When testing each initial condition, we used CHTC. Each job was run on a system with 8 CPUs and 10 GB memory (Figs. 1 and 2).

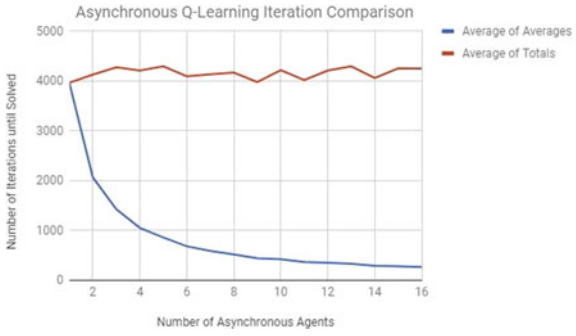

Fig. 1 Comparison of average total number of iterations over all agents until the task was solved and the average of the number of iterations of each agent

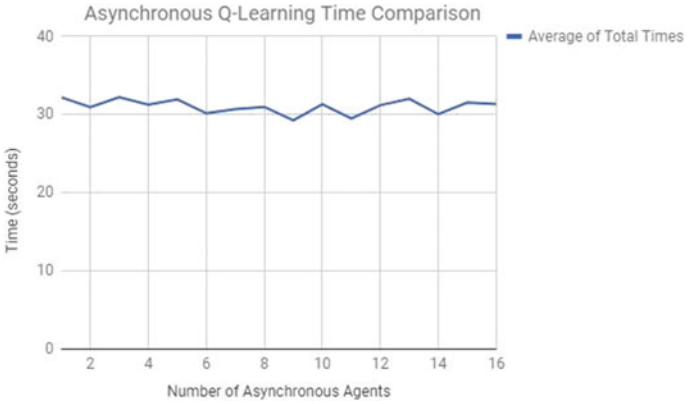

Fig. 2 Comparison of average total time over all agents until the task was solved

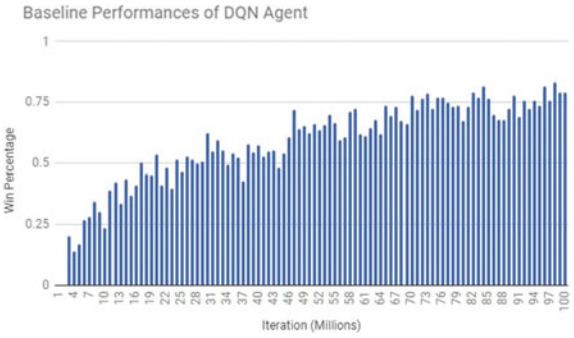

Fig. 3 Baseline performances of the DQN agent. The *x*-axis shows the number of iterations that had passed when the agent was saved while the *y*-axis shows the winning percentage of the agent over 1000 games

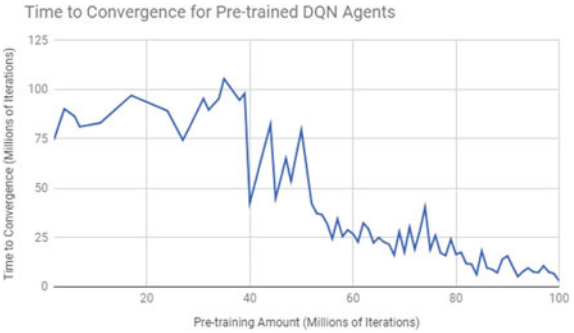

Fig. 4 Comparison of time to convergence for different amounts of pre-training for DQN agents

5 Conclusion and Future Work

We have shown that initial conditions greatly impact the rate of convergence for reinforcement learning. As a result, transfer learning shows great potential for accelerating the convergence rate of reinforcement learning agents. Transfer learning has already seen great success in deep reinforcement learning, and we hope that this research is now further motivated. In the future, we would want to study adversarial learning in the reinforcement learning setting [17]. Intuitively, presenting challenges allows humans to learn better, and we believe that this translates to reinforcement learning agents as well. In fact, it has already been shown that this adds robustness [18]. Furthermore, adversarial learning is perfectly suited for two-player games like many of the Atari games. Hence, our future work should include studies in adversarial learning in the reinforcement learning setting. We would also like to study learning models for multiple games and use transfer learning to apply these models to different reinforcement learning tasks.

References

1. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning
2. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lan-tot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K, Hassabis D (2017) Mastering chess and shogi by self-play with a general reinforcement learning algorithm
3. Wang Z et al (2018) Deep reinforcement learning with knowledge transfer for online rides order dispatching. In: 2018 IEEE International Conference on Data Mining (ICDM). IEEE
4. Manfredi V et al (2021) Relational deep reinforcement learning for routing in wireless networks. In: 2021 IEEE 22nd international symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). IEEE
5. Lample G, Chaplot DS (2021) Playing FPS games with deep rein-forcement learning. In: Thirty-first AAAI conference on artificial intelligence
6. Zhou Z, Li X, Zare RN (2017) Optimizing chemical reactions with deep reinforcement learning. ACS Cent Sci 3(12):1337–1344
7. Baldazo D, Parras J, Zazo S (2019) Decentralized multi-agent deep reinforce-ment learning in swarms of drones for flood monitoring. In: 2019 27th European Signal Processing Conference (EUSIPCO). IEEE
8. Landajuela M et al (2021) Discovering symbolic policies with deep reinforcement learning. In: International conference on machine learning, PMLR
9. Zhao R et al (2020) Deep reinforcement learning based mobile edge computing for intelligent Internet of Things. Phys Commun 43:101184
10. Kiran BR et al (2021) Deep reinforcement learning for autonomous driving: a survey. IEEE Trans Intelligent Transp Syst
11. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition
12. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Angue-lov D, Erhan D, Van-houcke V, Rabinovich A (2015) Going deeper with convolutions
13. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran

D, Wierstra D, Legg S, Hassabis D (2014) Human-level control through deep reinforcement learning
14. Yin H, Pan SJ (2017) Knowledge transfer for deep reinforcement learning with hierarchical experience replay
15. Mnih V, Badia AP, Mirza M, Graves A, Harley T, Lillicrap TP, Silver D, Kavukcuoglu K (2016) Asynchronous methods for reinforcement learning
16. Even-Dar E, Mannor S, Mansour Y (2006) Action elimination and stopping conditions for reinforcement learning
17. Lin Y-L, Hong Z-W, Liao Y-H, Shih M, Liu M, Sun M (2017) Tactics of adversarial attack on deep reinforcement learning agents
18. Pinto L, Davidson J, Sukthankar R, Gupta A (2017) Robust adversarial reinforcement learning