



Reward Design for Deep Reinforcement Learning Towards Imparting Commonsense Knowledge in Text-Based Scenario

Ryota Kubo¹, Fumito Uwano²^a and Manabu Ohta²^b

¹*School of Engineering, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama, Japan*

²*Faculty of Environmental, Life, Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama, Japan*

Keywords: Commonsense Knowledge, Reinforcement Learning, Deep Q-Network, Reward Design.

Abstract: In text-based reinforcement learning, an agent learns from text to make appropriate choices, with a focus on addressing challenges associated with imparting commonsense knowledge to the learning agent. The commonsense knowledge requires the agent to understand not only the context but also the meaning of textual data. However, the methodology has not been established, that is, the effects on the agents, state-action space, reward, and environment that constitute reinforcement learning are not revealed. This paper focused on the reward for the commonsense knowledge to propose a new reward design method on the existing learning framework called *ScriptWorld*. The experimental results let us discuss the influence of the reward on the acquisition of commonsense knowledge by reinforcement learning.


1 INTRODUCTION


The emergence of ChatGPT¹ endows us with not only knowledge from big data but also an intelligent communication agent. Such a technique boosts research on artificial intelligence technologies that use text data as input. Text is so valuable that we can learn a lot of complex things from text. However, it is hard for an agent to learn from text-based data. In particular, the text includes commonsense knowledge that confuses the agent learning. The commonsense knowledge requires the agent to understand not only the context but also the meaning of textual data. Generally, commonsense knowledge is the general and widely applicable knowledge that forms the basis for human beings to make everyday judgments, such as “sufficient light is necessary when reading a book” and “turning on a light switch makes a room brighter”. Although this knowledge is taken for granted by humans, it is difficult for the agents to acquire this knowledge because it is not learned from textbooks and is not obtained systematically. On the other hand, the agent needs to acquire a large amount of commonsense knowledge to achieve the same level

of decision-making ability as humans, and the acquisition of commonsense knowledge by reinforcement learning agents is an important issue.

Reinforcement learning (RL) performs in multi-step decision-making where the agent attempts to make a decision at its state. The state transitions to the next state by following the decision, and this cycle repeats until the learning process is terminated (Sutton and Barto, 1998). RL aims to enable the agent to refine its policy which influences the immediate and the future decision. ChatGPT utilizes RL techniques by using the users’ prompt information as human feedback. This research focuses on the decision-making for text-based RL rather than communication between a human and ChatGPT, especially, how RL learns knowledge from text data with commonsense knowledge.

The agent learns from text data in the framework of Interactive Fiction Games (IFG) (Hausknecht et al., 2020). IFG is an interactive game using natural language, in which players take appropriate actions based on sentence input as states to achieve their goals. Specifically, the states, actions and rewards are constructed based on textual descriptions and RL is conducted to learn the relationship between sentences (state transitions) and the commands (actions) that are connected to them. Especially, in IFG, the agents need to acquire commonsense knowledge and achieve

^a <https://orcid.org/0000-0003-4139-2605>

^b <https://orcid.org/0000-0002-9005-2458>

¹<https://chat.openai.com>

a semantic understanding of the text. Recently, Joshi et al. have pointed out that IFG are far from real-world scenarios, and have proposed *ScriptWorld* as a new environment to solve this problem (Joshi et al., 2023). *ScriptWorld* enables reinforcement learning agents to learn commonsense knowledge by using real scenarios based on human life and thus enables learning that assumes application to real-world problems. The commonsense knowledge, which is the goal of *ScriptWorld*, is the knowledge that helps computer agents take appropriate actions based on semantical similarity among actions in a scenario and appears as a parameter of neural networks in deep reinforcement learning. Repeating the learning process and updating parameters enables the acquisition of commonsense knowledge to choose actions based on the real world. This knowledge leads to endowing the agent with the ability for commonsense reasoning.

Although the above works focused on imparting commonsense knowledge to a learning agent, commonsense knowledge was not defined in machine learning, yet. For instance, the agent learns from one of the scenarios in *ScriptWorld*. Joshi et al. attempt to transfer the learning outcome to another scenario such that the agent uses commonsense knowledge for the prior scenario to perform in the next scenario. However, there is no mechanism to accomplish such transfer in the learning agent (Joshi et al., 2023). Therefore, we discuss commonsense knowledge to classify it into some types and propose a new reward design method to enable an agent to learn the commonsense knowledge for such transfer. Concretely, we classify commonsense knowledge into two discrete types: event-level commonsense knowledge and scenario-level commonsense knowledge. In addition, we propose a new method for a preliminary study in terms of the reward design in *ScriptWorld*. In the experiment, we can see the results of introducing deep reinforcement learning into *ScriptWorld*, and then the effect of feeding rewards into it. At the end, we discuss the commonsense knowledge for learning agents and show the practical reward design techniques.

This paper is organized as follows. Section 2 introduces the mechanisms of RL and Deep Q-Network (DQN) (Mnih et al., 2015) which is the deep RL method adopted in the experiment. Section 3 shows *ScriptWorld* in detail. Section 4 shows related works. Then, Section 5 describes the proposal, that is, the introduction method of DQN into *ScriptWorld* and the brief method to impart rewards into it. The experimental results and the discussion are summarized in Section 6. Finally, this paper concludes in Section 7.

2 BACKGROUND

2.1 Reinforcement Learning

Reinforcement Learning (RL) (Sutton and Barto, 1998) attempts to select an appropriate action for each situation in the environment, and learns that selection rule as policy by using a reward. The reward is a value for a clue to evaluate the action comparing the current situation with the goal. Specifically, the agent observes its state s from the environment, selects an action a by the state value $V(s)$ as the value of the state s or by the state-action value $Q(s, a)$ as the value of the action a at the state s , and updates $V(s)$ or $Q(s, a)$ by using the reward resulting from the action a at the state s . Note that the iteration is called “step” in this paper. The state value $V(s)$ is calculated by the following equation as a recursive equation based on the Bellman equation:

$$V^\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} P(s'|s, a)(r(s, a, s') + \gamma V^\pi(s')), \quad (1)$$

where the next state is denoted by s' . The policy π is a set of selection probabilities for all actions at each state, and the selection probability for action a at state s is denoted as follows:

$$\pi(a|s) = P(a_t = a | s_t = s). \quad (2)$$

The state-action value in Q-learning is denoted as the following equation (3):

$$Q^\pi(s, a) = \sum_{s' \in S} P(s'|s, a)(r(s, a, s') + \sum_{a' \in A(s')} \gamma \pi(a'|s') Q^\pi(s', a')), \quad (3)$$

where s , a , s' , and a' denote as the current state, the executed action at the current state, the next state, and the next executed action, respectively. The action a' belongs to the set $A(s')$ at the state s' . The discount rate is denoted as γ .

2.2 Deep Q-Network

Deep Q-Network (DQN) (Mnih et al., 2015) is a reinforcement learning method that introduces a neural network to Q-learning. The neural network consists of three different layers: an input layer, a hidden layer, and an output layer, and a deep neural network with n hidden layers is introduced in this paper. The goal of this neural network is to output in the output layer a Q-value. This represents the value of the action in the input layer for the number of actions k in the input

state. In the hidden layer, the outputs in each layer are passed as inputs to the next layer, and as the number of hidden layers n increases, more complex problems can be dealt with. Learning is achieved by calculating the Q-value for the selected next state, calculating the loss function between the Q-value and the output of the neural network, and updating the weights by performing error backpropagation. The loss function is the mean squared error (MSE), which is calculated as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\bar{y}_i - y_i)^2. \quad (4)$$

where n , \bar{y} , and y denote the number of data, the predicted value, and the target value.

3 ScriptWorld

In this section, we introduce ScriptWorld (Joshi et al., 2023) proposed as a framework for reinforcement learning that gives commonsense knowledge.

3.1 Text Data Modeling and Commonsense Knowledge

We use an existing script corpus (*DeScript* (Wanzare et al., 2016)) to create the *ScriptWorld* environment. DeScript provides ESDs (Event Sequence Descriptions) $(\mathcal{E}_1^{S_i}, \mathcal{E}_2^{S_i}, \dots, \mathcal{E}_N^{S_i})$, which are temporal event descriptions for a given scenario S_i . ESD \mathcal{E}_k^i consists of a short description of events that describe the behavior $(\mathbf{e}_1^{(\mathcal{E}_k^i)}, \mathbf{e}_2^{(\mathcal{E}_k^i)}, \dots, \mathbf{e}_N^{(\mathcal{E}_k^i)})$. In DeScript, events of different ESDs are combined by semantic similarity by humans. For example, in the *Washing Dishes* scenario, the events “put dishes in the sink” ($\mathbf{e}_1^{(\mathcal{E}_1^{Wash})}$) and “take dirty dishes to sink” ($\mathbf{e}_1^{(\mathcal{E}_2^{Wash})}$) are clustered into a single process. ScriptWorld aims to enable agents to acquire commonsense knowledge by creating a reinforcement learning environment by graphically representing DeScript models.

3.2 Creating a Graph

The first step in creating the environment is to graph the scenarios. First, semantically similar events in different ESDs are clustered. Next, the clusters are used as nodes, and the nodes are connected based on the order of the events to create a directed graph. If there is at least one event in node p that follows the event in node q , an edge is created from p to q .

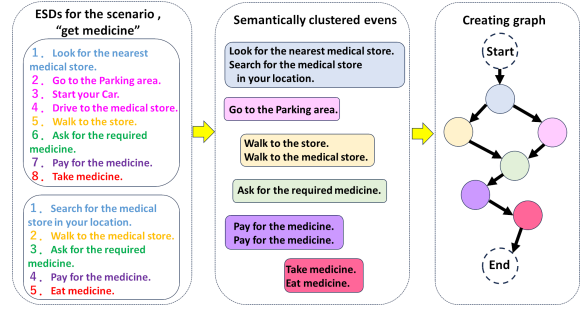


Figure 1: Process for creating an environmental graph from ESDs for the scenario of “getting medicine.”

Figure 1 illustrates the specific flow. ESD for the “getting medicine” scenario consists of the following eight events: 1) Look for the nearest medical store, 2) Go to the Parking area, 3) Start your car, 4) Drive to the medical store, 5) Walk to the store, 6) Ask for the required medicine, 7) Pay for the medicine, 8) Take medicine. There are several possible patterns of ESDs when the number of events is small or when the wording is different. Therefore, we cluster events that are semantically similar between different ESDs (Figure 1 central). The three events, “pay for medicine,” “pay the fee,” and “pay at the cash register,” are events in different ESDs, but they are semantically similar and can be grouped into the same cluster. Next, a graph is created with the clustered events as nodes (Figure 1 right). After the nodes are created, the nodes of the cluster containing successive events in ESD are connected by edges. For example, since the events “going to the parking lot” and “starting the car” are sequential in ESD, the node containing the event “going to the parking lot” is connected to the node containing the event “starting the car” by an edge.

3.3 Creating Environment

An environment is created by using the created graph. Reinforcement learning agents observe nodes in the graph as states and learn to make the correct choice among the possible choices. In a single cluster, there may be multiple actions with similar meanings, so one of them is chosen as the correct choice. Reinforcement learning also requires learning false choices, so we use the temporal properties of the graph to create false choices. As a graph contains the sequence of actions to perform a specific subtask, all actions in nodes that are far from the current node become invalid for the current state. Therefore, we sample invalid actions according to different distances and give them to the agents as choices. This allows the relationship of all alternatives to each scenario to be represented in the learning environment, facilitating un-

derstanding of the order in which events are selected to achieve the objective.

3.4 Setting Rewards

The agent gets a negative reward of -1 for choosing the wrong choice. If the agent make an incorrect choice for five consecutive times, it will receive an additional reward of -5 . No reward (reward: 0) is earned when the correct choice is made. The agent also receives a positive reward of 10 if the task is completed. The training is terminated when the total reward is less than -5 . When a wrong choice is made, the agent returns to the previous state, and the agent learns by choosing the choice again.

3.5 Handicaps

In *ScriptWorld*, the environment may be too complex for agents to learn from scratch. Therefore, handicaps are provided for each state to reduce the complexity. Handicaps give a short sentence hint for the next action in the current state. The title of the scenario and a description of the events in the current state are given to GPT2, which automatically generates hints by sampling from a large number of generated handicaps.

4 RELATED WORK

Reward design is a long-standing topic in RL. Ng et al. proposed potential-based reward shaping as a function (Ng et al., 1999). The reward shaping function guarantees to learn the optimal policy theoretically. Thus, there are many works by inheriting from the potential-based reward shaping (Devlin and Kudenko, 2012; Mannion et al., 2018).

On the other hand, Russell proposed Inverse Reinforcement Learning (IRL) as a modification to the reward function for preserving the optimal policy (Russell, 1998). In particular, IRL is given expert data to enable an agent to demonstrate the expert. For example, IRL trains an agent to manipulate a drone using human-manipulation data. Although the IRL is usually applied to robotics and auto-driving, the expert data is not always effective for all circumstances. For instance, Kuderer et al. applied IRL to learning from an automobile driving data to estimate model parameters and proposed the auto-driving method following each individual driving style (Kuderer et al., 2015). Wu et al. revealed demands in auto-driving: (1) handling the sequentiality, vehicle kinematics, and unstableness contained in automobile trajectory, and (2) in-

terpretability and generalization ability of the element in the reward estimation in IRL (Wu et al., 2020). However, IRL should perform the best in text-based games like *ScriptWorld* where the expert data as the scenario are completed.

As for commonsense knowledge, Brown et al. showed T-REX as a reward function estimation method enabling an agent to learn optimal policy using only sub-optimal expert data. In short, T-REX can complement the reward function by inferring the true reward function for the developer (Brown et al., 2019). Therefore, T-REX can impart the scenario-level commonsense knowledge to an agent in *ScriptWorld*, and the IRL is familiar with the text-based RL because it utilizes the expert data as real-world data. We will introduce the IRL technique into this work to learn both levels of commonsense knowledge in the future.

5 PROPOSAL

5.1 Introducing Deep Reinforcement Learning

The goal of *ScriptWorld* imparts commonsense knowledge to the agent. However, Q-learning, or a typical reinforcement learning method, cannot learn commonsense knowledge. This is because the fact that Q-learning does not have a generalization mechanism, since Q-values are computed for each set of states and actions. Neural networks can be introduced as a method to provide a generalization mechanism. We believe that the introduction of a DQN with a neural network will lead to a generalization mechanism because it will allow learning by adjusting parameters. Figure 2 shows the neural network model used in DQN and input data. In this diagram, the number of actions k is 2 and the number of hidden layers is 2, the number of units in the output layer is k . The input to the neural network is the concatenation of the current state with the data of the choices. Note that when calculating the Q-value in the next state, no concatenation with choices is made. Without concatenation, the output Q-value only represents the value of the number of choices presented in the input state. The concatenated data is tokenised and vectorised, then normalised by dividing by 10000 and used as input to the neural network.

Batch processing is carried out during the learning process. For batch processing, variables representing the current state, choice number, the reward, the next state chosen from choices, and a variable whether the episode has ended are stored in memory for each ac-

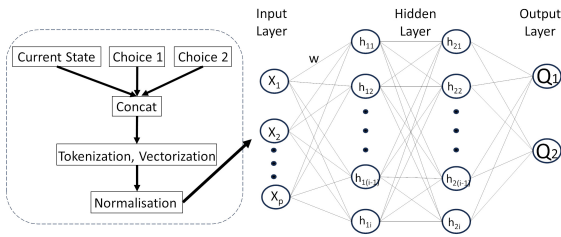


Figure 2: Neural network model used in DQN and input data

tion selection. An episode is defined as the time from the start to the goal or game over in the *ScriptWorld* environment. Information is batched up to a batch size of 10, or the state has reached its goal. Memory is cleared after each learning. As for learning, the loss function is calculated for the output of the neural network, and the weights w are updated by error back propagation, which is repeated.

5.2 Reward Design

In *ScriptWorld*, the agent received positive rewards only when a task was completed. This reward design allows for the evaluation of the entire scenario but does not allow for the evaluation of partial event groups within the scenario. There are common elements in partial event groups that are shared with other scenarios. For example, the event of “Paying money” is common to both “buying a car” and “eating out”. Learning elements shared with other scenarios means that the knowledge acquired in one scenario holds significance in other scenarios as well, leading to the acquisition of commonsense knowledge. Events that are common to other scenarios and contribute to commonsense knowledge are thought to be located within parts of a scenario rather than across the entire scenario. Therefore, we propose a brief method to set a reward in *ScriptWorld*. Specifically, it changes the reward design so that the rewards can be obtained even for events in the middle of graphs.

This time, the node with the highest number of connected nodes in the *ScriptWorld* graph is set as a sub-goal, so that the reward can be obtained there as well, apart from at the goal. The most connected nodes are considered to be the most important actions in the scenario. The main action in a scenario with a high level of importance has a large meaning in itself and is likely to have a large meaning in other scenarios as well, leading to commonsense knowledge.

6 EXPERIMENTS

6.1 Experiment 1: Deep Q-Network in *ScriptWorld*

6.1.1 Experimental Setup 1

We analyze the behavior of DQN on *ScriptWorld* and discuss the function of neural networks in the acquisition of commonsense knowledge. The agent was evaluated with percentage completion as a criterion, an index that expresses the percentage of states reached from the start to the goal.

In this experiment, the number of actions per state was set to 2, the permissible number of mistakes in a single trial was set to 5, the number of nodes to backtrack when a choice was incorrect was set to 1, the seed value was set to 42, and the number of episodes was set to 10,000.

Keras was used to build a deep reinforcement learning model in DQN, which consists of two hidden layers with 256 units each and an output layer with units for the number of action options. ReLU is used as the activation function, mean squared error as the loss function, Adam as the optimisation algorithm. SentencePiece is used to tokenize and vectorize strings for input to the neural network model. As for the parameters for the DQN, we set the learning rate α to 0.001 and the discount rate γ to 0.95. Then, ϵ -greedy selection was used for action selection, with the value of ϵ gradually decreasing, so that random search gradually decreases. The initial value of ϵ is set to 1.0, and 0.995 is multiplied by ϵ each time for training session. This process is repeated for training session until ϵ becomes smaller than 0.001.

6.1.2 Result 1

Figure 3 shows the experimental result by comparing the performances of DQN with Q-learning. The vertical axis indicates the percentage completion, and the horizontal axis indicates the number of episodes. Note that the percentage completion of each result is calculated as a moving average over a window of 30 intervals. The scenario of this experiment is “taking a bath”. The result shows that the percentage completion gradually increases as the number of episodes increases, indicating successful learning for DQN, while Q-learning fails to learn. Figure 4 shows the difference in the performance when the number of mistakes allowed in one trial life was set by 5 and 20. In this experiment, the results show that increasing the life to 20 is more efficient for learning, which ends up with the percentage completion reaching 100.

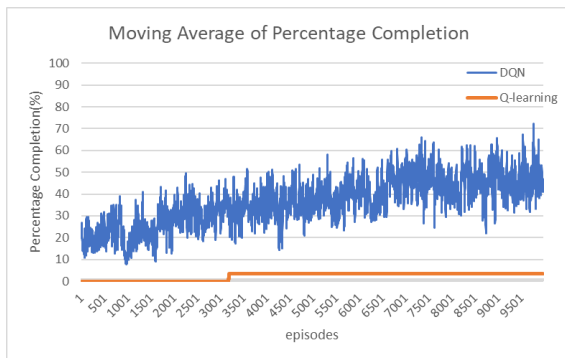


Figure 3: Result of MAPC introducing DQN for the scenario “taking a bath”.

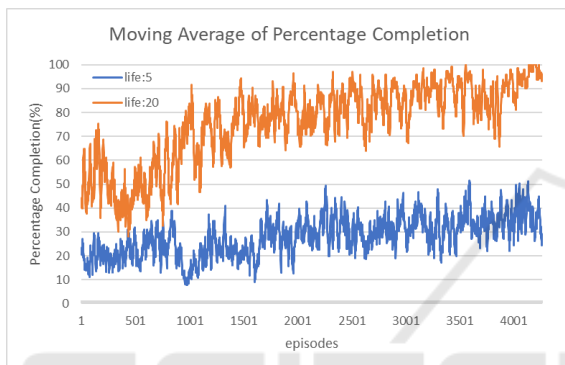


Figure 4: Results of MAPC for the different number of times the agent can fail for the scenario “taking a bath”.

The result of the experiment in which the scenario was changed to “baking a cake” is shown in Figure 5. The moving average interval is set to 100 to make the comparison easier to understand. The result shows that the agent successfully learned in the other scenarios as well.

6.1.3 Discussion 1

From the results, Q-learning estimates appropriate Q-values for all associated patterns of states and actions, resulting in failure to learn in a situation where there exist many patterns. On the other hand, the introduction of DQN has enabled successful learning by allowing the neural network to abstract input data. Thus, the *ScriptWorld* enables the neural learning agent to acquire commonsense knowledge in a way that subsumes the different inputs as the same. This paper calls such knowledge as event-level commonsense knowledge. The event-level commonsense knowledge is not used for transfer learning.

On the other hand, we discuss the influence of the reward by using the Figure 4. In particular, *ScriptWorld* has a positive reward only in the goal, while it has negative rewards for the mistakes. Thus,

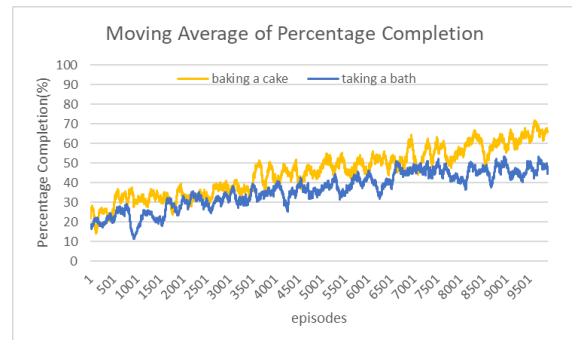


Figure 5: Results of MAPC for the scenario “taking a bath” and “baking a cake”.

Table 1: Percentage of positive rewards and goals.

Life	Positive Reward(%)	Goal(%)
5	0.117	0.117
20	0.348	0.646

increasing the number of lives allowed the agent to explore more, resulting in increasing the probability where it reaches the goal and learns with positive rewards.

Table 1 shows the rates of episodes for resulting in a positive accumulated reward and for resulting in the final goal. The table shows the increasing percentage of positive rewards as increasing the number of mistakes the agent can make. This suggests that the reward function of the *ScriptWorld* is so simple that the agent cannot learn the relationship between events associated with each other. For instance, when the agent learns a policy from the scenario “riding on a bus” to the scenario “going on a train” based on commonsense knowledge, it should describe actions such as boarding the vehicle at the departure and disembarking at the destination. Events occurring before boarding the vehicle or after reaching the destination should be excluded. However, the current reward function imparts the knowledge of entire events to the agent. In contrast, the results show that the precise reward function can enable an agent to utilize its learned commonsense knowledge for any scenario. Such commonsense knowledge is different from event-level commonsense knowledge and is called scenario-level commonsense knowledge in this paper. The reward design in this paper should enable the agent to acquire both commonsense knowledge. In addition, such reward function seems to contribute to the learning efficiency in *ScriptWorld*.

For instance, the learning accuracy is different with scenarios, which is influenced by the complexity of the graph. The reward design might solve the problem. Table 2 shows the number of nodes in the graph for the two scenarios and the percentage of episodes

Table 2: The number of nodes.

Scenario	Nodes	Goal(%)
taking a bath	32	0.117
baking a cake	28	0.239

that reach the goal. The table shows that the “baking a cake” scenario has more nodes and the percentage of goals achieved is smaller. This suggests that as the number of nodes increases, the number of possible routes increases, the graph becomes more complex, and it becomes more difficult to reach the goal. Therefore, the reward design promotes the agent to learn to reach the goal if it adds rewards as sub-goals.

6.2 Experiment 2: Reward Design

6.2.1 Experimental Setup 2

In this section, we discuss the relationship between rewards and commonsense knowledge acquisition by reviewing the reward design to set sub-goals. The parameters are the same as in 6.1.1, and in the scenario “taking a bath”, the node “wash/take shower”, which has the most connected nodes, is set as a sub-goal, and it is rewarded with 3.

If an agent makes a wrong choice and returns to the previous node and reaches the sub-goal more than once, the agent will receive the reward for the sub-goal only once. In the “baking a cake” scenario, a sub-goal is also set and tested. In this scenario, the node “wait” is the most frequently connected, so this node is set as a sub-goal, and agents get reward 3 for passing the sub-goal.

6.2.2 Result 2

Figure 6 shows the results of the experiment with sub-goals. The yellow line represents the percentage completion with sub-goals, while the blue line represents it without sub-goals. The result is presented as the moving averages of the percentage completion over 100 intervals. The findings suggest that the accuracy of learning improves when a sub-goal is set. Furthermore, for another scenario, the experimental results for the “baking a cake” scenario are illustrated in Figure 7. The results indicate that setting a sub-goal leads to a decrease in learning accuracy.

6.2.3 Discussion 2

The results show the effectiveness of the brief sub-goal method such that setting sub-goal at the appropriate state promoted the agent to learn for reaching the goal. Compared to the case where positive rewards are only given at the goal, having positive re-

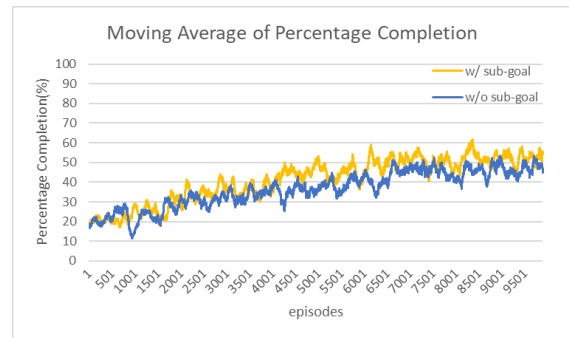


Figure 6: Results of MAPC w/ and w/o a subgoal for the scenario “taking a bath”.

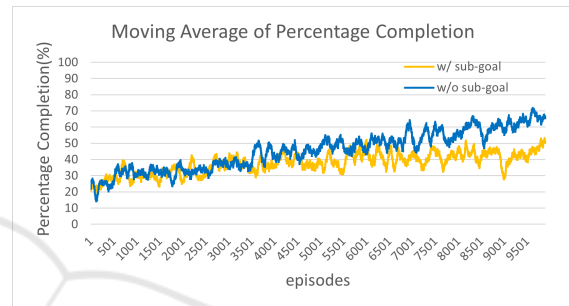


Figure 7: Results of MAPC w/ and w/o a subgoal for the scenario “baking a cake”.

wards provided at intermediate nodes as well leads to an increase in the proportion of learning due to positive rewards. It can be considered that the increase in the total reward value is also attributed to the setting of sub-goals.

In the “baking a cake” scenario, the results suggest that setting sub-goals does not necessarily lead to an improvement in learning accuracy. It is understood that the number of connected nodes is insufficient as a condition for setting sub-goals. As a possible condition for setting sub-goals, the number of bypass routes could be considered. In the “taking a bath” scenario, there is only one edge that is a bypass, moving from a node that is temporally before the sub-goal to a node after the sub-goal without passing through the sub-goal, whereas in the “baking a cake” scenario, there are six edges that do not pass through the sub-goal. Figure 8 shows the results of an experiment with three different sub-goals in the “baking a cake” scenario. The three sub-goals are chosen in order of the number of nodes connected in the scenario. Table 3 also shows the number of connected nodes at each sub-goal and the number of bypass routes that do not pass through the sub-goal. From Figure 8 and Table 3, it can be seen that the “put-cake-oven” with a small number of bypass routes has a higher learning accuracy. This suggests that the number of bypasses is important for setting sub-goals.

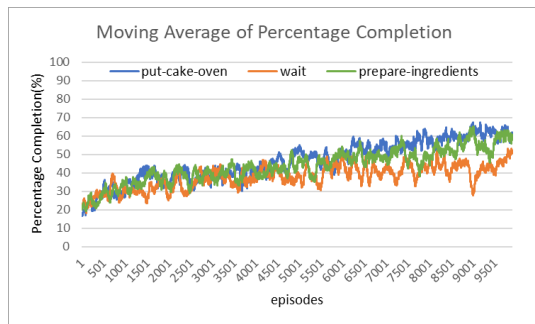


Figure 8: MAPC with various sub-goals for “baking a cake”.

Table 3: Various sub-goals.

Subgoal	Connected nodes	Bypass
put-cake-oven	9	2
wait	11	6
prepare-ingredients	9	4

7 CONCLUSION

This paper focused on the generalization and learning mechanisms of deep reinforcement learning and analyzed the learned commonsense knowledge. In particular, we discussed the impact of neural networks on the acquisition of commonsense knowledge and the challenges of ScriptWorld in terms of reward design. The experiment showed that DQN outperforms Q-learning in ScriptWorld, which indicates that the neural network generalized the input as event-level commonsense knowledge. In addition, we found that the brief method to impart sub-goals sometimes improved the learning accuracy, indicating that the reward design was effective.

One of the future work is to establish the reward design method for imparting scenario-level commonsense knowledge. To acquire such commonsense knowledge, it is necessary to focus not only on the scenario but also on partial actions within the scenario. In fact, it was found that setting sub-goals is effective to some extent, but there are still issues to be addressed regarding the placement of these sub-goals. The method we are considering is to set a reward for every node and change the value of the reward according to the importance of each node. Another aspect of this environment is that the success rate of reaching the goal is low, and the data used for learning often has a negative component. It could be devised in such a way that data with a positive element would be heavily reflected in the learning process. In addition, since Handicap is available in ScriptWorld, we would like to consider how to utilize it to further improve learning efficiency.

REFERENCES

- Brown, D., Goo, W., Nagarajan, P., and Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Proceedings of the 36th International Conference on Machine Learning, ICML '19*, pages 783–792, Long Beach, California, USA.
- Devlin, S. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS '12*, page 433–440.
- Hausknecht, M., Ammanabrolu, P., Côté, M.-A., and Yuan, X. (2020). Interactive fiction games: A colossal adventure. *Proceedings of the AAI Conference on Artificial Intelligence*, 34(05):7903–7910.
- Joshi, A., Ahmad, A., Pandey, U., and Modi, A. (2023). Scriptworld: Text based environment for learning procedural knowledge. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. Main Track.
- Kuderer, M., Gulati, S., and Burgard, W. (2015). Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646.
- Mannion, P., Devlin, S., Duggan, J., and Howley, E. (2018). Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review*, 33:e23.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning, ICML '99*, page 278–287.
- Russell, S. (1998). Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT' 98*, page 101–103, New York, NY, USA.
- Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.
- Wanzare, L. D. A., Zarccone, A., Thater, S., and Pinkal, M. (2016). A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3494–3501.
- Wu, Z., Sun, L., Zhan, W., Yang, C., and Tomizuka, M. (2020). Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving. *IEEE Robotics and Automation Letters*, 5(4):5355–5362.