Learning Basic Skills to Survive the First Day in Minecraft with Life-long Learning

Laige Peng[1,a)] Yoshimasa Tsuruoka[1]

Abstract: In recent years, deep reinforcement learning has had a great impact on agent design in the field of game AI. The success of Atari 2600 gives researchers confidence to attempt to utilize it in more challenging games. Minecraft, as a representative example of complicated and challenging games, has recently attracted a lot of attention from researchers. Considering the complexity of playing Minecraft, in our work, we applied the concept of lifelong learning in deep reinforcement learning to solve a task: learning basic skills to survive the first day in Minecraft. In our experiment, we decomposed this main task into three subtasks. The agent learned the skills from the process of solving the three subtasks and finally managed to finish the main task using pre-learned skills.

1. Introduction

Recent achievements in Atari games and self-play algorithms have shown the power of deep reinforcement learning, which demonstrates human-level or even better performance in games. Deep reinforcement learning has wide applications and can be seen as a means to achieve Artificial General Intelligence (AGI). The publication of Deep Q-Network [1] has had a far-reaching impact on the field of game AI. There have been incessant numbers of new reports on state-of-art solutions to some new games or new adaptions of the DQN algorithms. However, deep reinforcement learning is not omnipotent. Minecraft, a popular but challenging game, is one example that presents some challenges to deep reinforcement learning.

Minecraft creates a 3D world for players to explore various topographies and scenes. Its players can enjoy an exciting experience of ultimate survival or the pleasure of creating their own worlds. Figure 1 shows a screenshot of the game. The world is very complex, containing thousands of different elements, items, blocks and operations. This complexity makes it very difficult to have an artificial intelligence agent learn to act like a human in the Minecraft world. It is hard to define the action space because the actions in Minecraft are continuous, since the action time depends on how long players press their keyboard. The complicated survival system is confusing even to human players. For example, new players may not know where to start or what actions to make, facing such a huge plane when they enter the world for the first time. Besides the complexity of the environment, another difficulty is the property of partial observability. Like many other first-person sight games, the agent sees only the scenes in front of it, which only provide partial information on the whole environment. These two properties of the special environment in Minecraft present a challenge for researchers to create an artificial intelligence agent for it.

Figure 1. A screenshot of the game Minecraft.

Although the task is challenging, Minecraft still attracts the attention of many researchers as a testbed for deep reinforcement learning algorithms. One of the reasons for this is its resemblance to the real world. This indicates the fact that good methods in Minecraft may be adapted into the real world. There are already some excellent studies on Minecraft. David et al. [2] that combined the idea of decomposing large tasks into several subtasks and propose a method of selecting subgoals by using deep learning from raw pixels. Chen Tessler et al. [3] propose a Hierarchical Deep Reinforcement Learning Network (H-DRLN) architecture to learn skills and solve complicated tasks. Shu et al. [4] also propose another kind of hierarchical architecture and add a human language interpretable skill in multi-task reinforcement learning in Minecraft.

In our work, we mainly construct our agent with the hierarchical architecture H-DRLN, which we will describe in

[1] Department of Information and Communication Engineering, The School of Information Science and Technology, The University of Tokyo
[a)] penglaige0417@yahoo.co.jp

the related research section. We aim to solve more realistic tasks in Minecraft, including the task of surviving the first day in the world. Survival is a serious problem because it determines whether players are able to continue their life in this world. We manually decompose this main target into three subtasks: (1) Collect an apple in case of starvation; (2) Collect cobble stone as an important material; (3) Cut the wool in order to make a bed to skip the dangerous night. To decrease the difficulty of the situation, we initially give the agent a diamond pickaxe and shears as tools. According to our experiment, we successfully have the agent learn those three skills and finish the final task. We describe the details of the methods we use to achieve this main task and the final performance.

2. Related Research

2.1 Markov Decision Process (MDP)

In a Markov Decision Process, there are several definitions to describe the terminology:

- S : A set of states.
- A : A set of actions.
- $P_a(s, s\prime)$: The probability that state s at time t will transition to state $s\prime$ at time $t+1$ when given action a.
- $R_a(s, s\prime)$: The immediately received rewards after state s transitioning to state $s\prime$ when given action a.
- $\gamma \in [0, 1]$: The discount factor, represents the priority difference of the future rewards and the present rewards.

A Markov Decision Process is defined as a process in which the future is independent of the past given the present. That is, state $s\prime$ at time $t+1$ is only determined by the state s and the action a at time t.

2.2 Reinforcement Learning

Reinforcement learning [5] is an area of machine learning in which an agent learns what to do—how to map situations to actions—so as to maximize a numerical reward signal. Reinforcement learning focuses on the interaction between the agent and the environment. More specifically, the agent observes a situation or state s from the environment, and then chooses an action a and receives a reward r from the environment. The environment makes a transition to another state $s\prime$ because of the effect of action a, and then the agent repeats to choose a new action. The whole process is described as a MDP. What is different from supervised learning is that, the agent is not told which actions to take or any other extra labeled information, but instead discovers which action will generate rewards by trying them.

In reinforcement learning, the agent at time t always tries to maximize the expectation of the cumulative rewards with a discount factor from time t. The value function of each state s is represented by using the Bellman equation as:

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s].$$

A policy π is represented by $\pi(a \mid s) = P[A_t = a \mid S_t = s]$. The state value function is defined as the expected return starting from state s, and then following policy π:

$$V_\pi(s) = \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s], \text{ for all } s \in S$$

Thus the reinforcement learning agent aims to learn a policy π which maximizes the value function $V(s)$ of a given state s.

Reinforcement learning is widely adapted in various areas such as game AI, control theory and multi-agent systems. It is imperative to recognize that reinforcement learning has a deep influence in game algorithm researches.

2.3 Q-learning

Q-learning [6] is a well-known reinforcement learning method, aiming to learn how to act optimally in controlled Markovian domains. Q-learning is based on the action value function $q_\pi(s, a)$ which defines as:

$$q_\pi(s, a) = \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a]$$

The optimal action value function is represented as $q^*_\pi(s, a)$ when π is the optimal policy. Q-learning aims to approximate the optimal action value function $q^*_\pi(s, a)$ by using a Q-table which is a table that records the value of $q(s, a)$ for all $a \in A$ and $s \in S$.

The process of the Q-learning algorithms is as follows: **1.** Initialize the Q-table arbitrarily. **2.** For a state s at time step t, choose an action a with highest value in the current table of $q(s, a)$, observe the next state $s\prime$ at time t and the reward r. The target value of $q(s, a)$ is $r + \gamma \max_{a\prime} q(s, a\prime)$, then update $q(s, a) := q(s, a) + \alpha[r + \gamma max_{a\prime} q(s, a\prime) - q(s, a)]$, α is the learning rate. **3.** Repeat **2** until learning ends.

Q-learning is limited to solve MDPs with finite states and finite actions. However, many environments have infinite states space, it is impossible to create all the state-action pairs to calculate the $q(s, a)$.

2.4 Deep Q-Network (DQN)

Deep Q-Network [7] is a method that combines reinforcement learning with a deep neural network to approximate the action value function instead of Q-table in infinite states environments. The notable success in Atari 2600 shows the potential of the DQN algorithm.

A DQN takes an observation s as the input of the deep neural network, and calculates the value $q(s, a)$ for all possible action a given s. Assume that the parameters of the neural network is θ. Q-learning updates at iteration i with the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s\prime)\sim U(D)}[(r + \gamma max_{a\prime} Q(s\prime, a\prime; \theta_i^-) - Q(s, a; \theta_i))^2]$$

in which γ is the discount factor and θ_i^- is the parameters of the target network at iteration i. The target network parameters θ_i^- updates with the θ_i only every C steps in order to eliminate the correlations between the network $Q(\theta)$ and the target network $Q(\theta^-)$.

Another key idea to break the correlations present in the sequence of the past observations in DQN is the experience replay. The DQN model creates a memory buffer to store the $(s, a, r, s\prime)$ tuples in the buffer for learning process. When selecting the learning data from the memory buffer, randomly choosing the data reduces the influence of the correlations.

There are some variants of DQNs such as Double DQN [8], Dueling DQN [9] and Distributed DQN [10].

2.5 Lifelong learning

A traditional definition of lifelong learning is "all learning activity undertaken throughout life, with the aim of improving knowledge, skills and competences within a personal, civic, social and/or employment-related perspective" [11]. Although this definition is to describe the lifelong learning in human domain, it is also able to be adapted to the game theory.

A lifelong machine learning system [12] aims to continuously learn tasks from one or more domains over its lifetime. There are two unique properties in lifelong learning systems: (1) sequentially retaining the skills it has learned; (2) selectively applying the learned skills to solve new tasks. These properties provide a lifelong learning system with the ability to solve tasks continuously from simple ones to complex ones. It is not a specific learning algorithms, but a good conception for the system designers to move beyond the learning algorithms to more seriously think about the nature of systems that are able to continuously learn new knowledge over a lifelong time.

2.6 Hierarchical Deep Reinforcement Learning Network (H-DRLN)

The Hierarchical deep reinforcement learning network (H-DRLN) architecture (Figure 1) which aims to learn skills and solve complex tasks was proposed by Chen Tessler et al. [3]. The H-DRLN architecture which utilizes the concept of lifelong learning, has the ability to retain a skill learned from one task, and then know how to reuse the previously learned skill to solve new tasks. This novel architecture was proposed to solve the complex tasks in Minecraft. The architecture consists of a main neural network which determines whether to use skills or take primitive actions and a deep skill module (DSN) as a pool of skills where each skill is represented by a pre-trained neural network model gained from previous tasks. Input of this architecture is the state space that represented as raw image pixels from the last four image frames. If the network selects a primitive action, the agent will take a primitive action as a result. Otherwise, the agent will follow the policy of the skill network model until it terminates. Each pre-trained skill neural network was trained by the Vanilla DQN architecture [7].

There are two different modules to implement the DSN: (1) a simple array of pre-trained skill networks in which each network is represented by a separate DQN; (2) a single deep network that represents multiple skill networks by using policy distillation [13].

In the experiments, Chen Tessler et al. prepared four pre-trained skills: (1) navigation in one room without obstacles; (2) navigation in one room with obstacles; (3) picking up objects; (4) placing objects. When the main network was trained in a two-room multi-task domain, the H-DRLN network achieved the highest success rate of 76%, outperforming a simple DQN or DSN. This results show the advantage of the lifelong learning system and its adaptability to new and complex tasks.

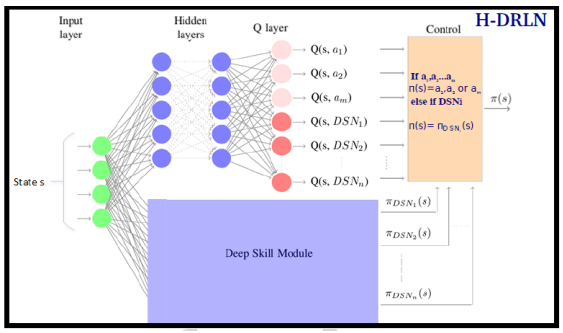

Figure 2. The H-DRLN Architecture[3]

3. Proposed Approach

Our proposed approach in this paper is mainly based on the H-DRLN architecture. Chen Tessler et al. showed us that this architecture has achieved good performance to solve complex tasks in Minecraft. However, the tasks in their experiment were a little bit abstract compared to the real Minecraft world. Thus in this paper, we propose a main task of surviving the first day in Minecraft and try to make our goal more realistic in Minecraft world.

We manually decompose the main task into three subtasks: (1) collect an apple; (2) find a stone or a sheep; (3) use tools to get the cobblestone or wool. Our proposal is to solve the main task by using a H-DRLN network with a pre-trained skill pool. Therefore, we define each subtask as a basic skill and each skill is trained with a DQN architecture. After training the skill networks and saving the models, we then simply put those models in the DSN array for the H-DRLN network to choose. Finally we train the H-DRLN network aiming to solve the main task of collecting the apple, the stone, and the wool to survive the first day in Minecraft. The output of the H-DRLN network contains several primitive actions and a DSN in which we store models of the three skill networks. The training process would make the agent learn when to use primitive actions and when to utilize skills.

We use the vanilla DQN structure [7] to train our three skill networks. Most of the parameters and hyperparameters are the same as those in the vanilla DQN. However, we adjust some parameters such as exploration rate and learning-start time steps to fit our experiments. The same DQN structure is used to train all the three skill networks but with different environment settings, rewards and action spaces. The second skill "find object" aims to find the object for the agent to remain in front of it, and the third skill "use tools" aims for the agent to learn how to use different tools depending on what object is in front of it. In order to avoid using two networks to train the agent to find the stone and the sheep, we prepare two different maps in the training process, which give the agent the ability of recognizing the

objects in front of it. We will discuss the details later in the experiment section.

This research aims to utilize the pre-trained skill networks to solve the main task by applying the H-DRLN architecture.

4. Experiment

4.1 Train skill networks

In this research, we use the Project Malmo as a platform to construct the game environment. The Project Malmo, which is a research platform created by Microsoft for conducting artificial intelligence experiments in Minecraft, has contributed to the new generation of algorithms to challenge more complicated problems in such a unique environment. Hence, we construct the experiment environment basing on the Malmo platform, which is convenient to get environment information and control the agent.

We have three skill networks as mentioned previously. We use the vanilla DQN structure described in the related research section to train those three skills but with different environment settings. The results indicate that the three skill networks are able to finish the subtasks. In the following parts, we describe the experiment details.

4.1.1 Skill network 1: collecting apples

State space - The frames initially obtained from the game environment are the images of the first-person view of the agent, with size 200 × 160. The previous process was used to scale the frames to size 84 × 84 and transform the RGB images to gray images. State space is represented as the raw pixels by combining the last 4 processed frames.

Actions - In the task of collecting apples, in the beginning of each episode, the agent receives a command to continuously move forward. In the training process, there are three actions can be chosen: (1) Continue Moving forward; (2) Stop, turn right by 30° and then move forward; (3) Stop, turn left by 30° and then move forward.

Rewards - The agent gets a non-negative reward when collected an apple.

Game environment setting - The experiment environment is a 8 × 8 place that is restricted by fences in the grassland. In the beginning of each episode, there are 5 position fixed apples drop inside the place. For each episode, the initial position of the agent is fixed, but the direction that the agent faces is randomly set. Once the agent collects one apple, it gets a reward of 100. An episode ends when the agent collects all the apples or uses up the allotted time. Final scores of each episode are calculated by the time average rewards.

Training results - Figure 3 shows the training results of this skill network. It indicates that the agent learns from the experience and collects apples faster. However, there is a decrease in the image from the 200th episode. In order to explain the reason for it, another detail should be mentioned. During the training process, each episode was recorded as a video by using the video recording function provided by the Malmo platform. According to the observations of those episode videos, we find that there are several reasons for this performance.

According to the videos, we observed several phenomena: (1) As the learning goes on, the agent is able to rapidly collect the nearby apples. However, it becomes difficult to collect the farthest apple, especially when it is the last one to collect. (2) When there is no apple in sight of the agent, the latter gets confused and stuck for a period of time. And in our map, the 4 corners of the fence are similar to each other, this is what increases the difficulty for the agent to distinguish its own position with respect to the fence. (3) Because of the game property in Minecraft, the agent is able to collect the apple from aside without necessarily seeing the apple, in lots of videos we observe that the agent collects apples in this manner. This causes a huge problem because this also gives the agent a reward, in turn leading it to learn the wrong policy. (4) Although there are some problems in the learning process, we observe that once there is an apple in the sight of the agent, it is able to quickly collect the apple.

From the previous observations, we consider that in the learning process, the learning curve increases because of the agent learns from the experience buffer, with higher exploration rate (possibility to choose a randomly action rather than a policy) to overcome being stuck in the corner. However, as the exploration rate decreases, the policy spends more time trapped in the corner and affects the score. Thus the curve decreases as the exploration rate gets lower.

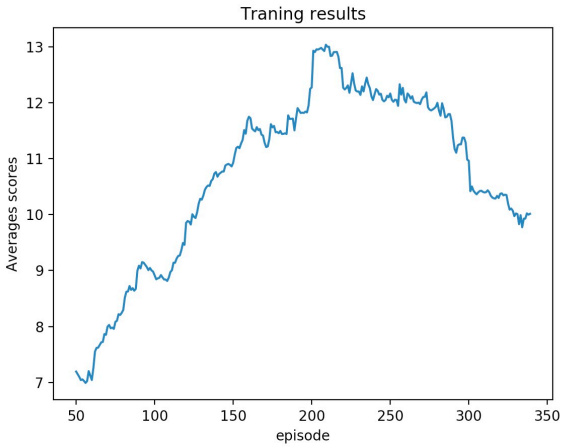

Figure 3. Training results of the apple skill network. The vertical axis represent the average scores of the last 100 episodes.

4.1.2 Skill network 2: find objects

State space - The state space of this task is the same as the task of collecting apples.

Actions - In the task of finding objects, it is a little different from that in the collecting task, in the beginning of each episode, there is no "move forward" command, the agent is initially still, waiting for the action command. There are four actions that can be chosen: (1) Move forward; (2) Turn right by 30°; (3) Turn left by 30°; (4) Stay at the same place.

Rewards - The agent gets a non-negative reward when

stand in front of the objects.

Game environment setting - In this task, we aim to train the agent to find both stone and sheep. In order to avoid using two networks, one for stone and one for sheep, we propose to train these two objects in one network. We prepare two maps, one map with a stone in the middle of the field and the other with a sheep in the middle. In the beginning of each episode, a map is randomly loaded. Once the agent finds the stone or the sheep, it gets a reward of 10. Another consideration is that we add a "stay" action in this task's action space. The reason is that in the final H-DRLN network, we would like the agent to firstly find the object and then stay in front of it so that the agent could use the tool to get items. For each episode, the limited action steps are 500. An episode ends when the agent has taken 500 actions. Final scores of each episode are calculated by the step average rewards.

Training results - Figure 4 shows the training results of this skill network. The curve shows good performance in this task. The higher score means that the agent is able to find the object as quickly as it can and once it had found the object, it remains in front of it in order to get more records. The recording videos we took from the training process also showed this fact.

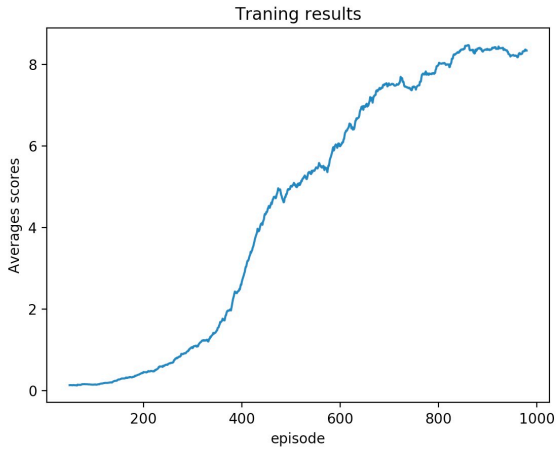

Figure 4. Training results of the finding object skill network. The vertical axis represents the average scores of the last 100 episodes.

4.1.3 Skill network 3: use tools

State space - The state space of this task is a little bit different from the previous two skill networks. State space is represented as the present one raw pixels instead of the last 4 frames. We discuss the reason in the game environment setting section.

Actions - In the task of using tools, as that in the "find objects" skill network, the agent does not get a "move forward" command from the beginning. Only two actions can be chosen: (1) Use a diamond pickaxe, wait 5 seconds and then move forward to pick the cobblestone; or (2) Use a shears, wait 5 seconds and then move forward to pick the wool.

Rewards - The agent gets a non-negative reward when using a diamond pickaxe to mine the stone or using the shears to cut the wool from the sheep.

Game environment setting - In this task, the state space is slightly different. The reason is that the aim of this task is to train the agent to use different tools depending on the object it discovered. Thus, the initial position setting is for the agent to be in front of the object. Moreover, we would like the agent to make only one action in each episode. Therefore, there is no need to use state space with the last 4 frames. This method looks like a classification problem, however, we obtain the labels and the data from the experience replays. We also prepare two maps, one map with a stone in the middle of the field, the other one with a sheep in the middle. In the beginning of each episode, it randomly loads a map and the agent just stands in front of the object, then it tries to choose an action to see if it gets a reward. If the agent choose the right tool, it gets a reward of 100 after collecting the cobblestone or the wool.

Training results - Figure 5 shows the training results of this skill network. The curve also shows a good performance in this task.

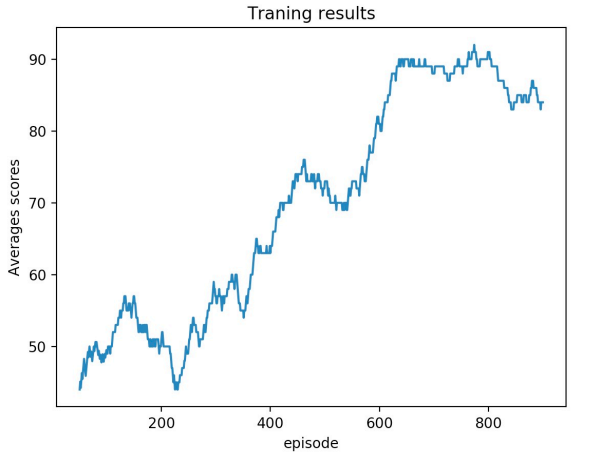

Figure 5. Training results of the using tools skill network. The vertical axis represents the average scores of the last 100 episodes.

4.2 H-DRLN network

State space - The state space of this task is the same as the task of collecting apples. Input to the H-DRLN network is also the input to the pre-trained skill networks. However, it should be noted that the state space of the network for "using tools" is different. Thus, to solve this problem, we extract the last frame of the state space as the input to the network for "using tools".

Actions - In this network, considering that we have a different action space for different skill networks, we separate it into a different action space. There are three primitive actions: (1) Move forward; (2) Turn right by 30°; (3) Turn left by 30°, and three skill networks: (1) Collect apples; (2) Find objects; (3) Use tools. Thus there are six outputs of the H-DRLN network. When the agent chooses to use a skill

network, the same input is fed into that skill network and it predicts an action in the corresponding action space. Next, the agent executes the action in that action space.

Rewards - In this task, the reward for an apple is 50, a block of wool is 50 and a block of cobblestone is 100. The reason of setting the reward this way is that the number of wool that an agent can get from the sheep by using a shears is arbitrary. In general the number of wool it can get is 2 or 3. In order to better evaluate the scores for each episode, we make the reward of the wool half of the reward of the cobblestone. Collecting apples can be done when using primitive actions or the "collect apples" skill, while collecting wool and cobblestone requires firstly to find the object and subsequently use the "use tools" skill. Regarding to the difficulty, we also make the reward for an apple half of the reward of cobblestone.

Game environment setting - In this task we place 3 objects in the map; an apple, a sheep and a stone. All of the objects are in the same position in the beginning of each episode, but during the game, the sheep is able to change its position. The agent is also placed in the same initial position with a fixed direction. Figure 6 shows the beginning scene of this task. As we mentioned before in the explanation of the H-DRLN architecture, each skill network will be executed until it terminates. In our H-DRLN network, skill networks terminate when the agent gets a reward or approaches the limited steps. The limited steps of skill 1 to 3 are respectively 20, 50 and 1. And each episode ends when the total steps in that episode exceed 500 or when the total rewards exceed 200. Final scores of each episode are calculated by the step average rewards.

Figure 6. The initial position setting of the H-DRLN network experiment

Training results - Figure 7 shows the training results of this learning process. When we process the training results data, we find that there are two episode with extremely unusual high scores: 100 and 50 while no other episodes have a score over 10. Upon looking at the videos of these two episodes, we discover the reason: the sheep approaches the agent spontaneously at the beginning of the episode and the agent obtains the wool immediately, causing an extremely high score. Thus when we plot the curve of the training results, we remove these two scores to avoid a huge gap in the average scores. The results indicate the effectiveness of the method to solve this complicated task.

We observe several interesting phenomena from the learning process: (1) As mentioned before, the sheep in the environment is able to move. We observe that the agent learns to track a moving sheep. (2) There is a potential immobility problem in this task. By using the "find object" skill, the agent goes to find an object and finally remains in front of it. However, if the object does not disappear in the map, the agent is likely to stay there, in front of the same object, with a high probability. This occurs when the agent firstly finds the sheep and the sheep does not disappear when agent cuts the wool. On the other hand, if the agent firstly find the stone and mines it, the agent is more likely to continue to find other objects. (3) Another observation is that, the sheep is able to grow new wool by eating grass, which means that the agent can cut wool from the same sheep twice or more times. (4) Although "collect apple" was pre-trained as a skill network in order to finish a subtask in the main task, we find that both primitive actions and the "use tools" actions can lead to collecting apples. Notice that there is also a "move forward" command contained in the action spaces of the "use tools" network. Another point is that we set the position of the apple right in front of the agent, this lets the agent learn to move forward directly to collect it. These two reasons make the influence of the "collect apple" skill not very obvious or useful in the main task. However, the agent is only able to collect cobblestone and wool by using "use tools" skill after it finds the object. Our observation and the results curve show that the "find object" and "use tools" skills make a great contribution to the main task.

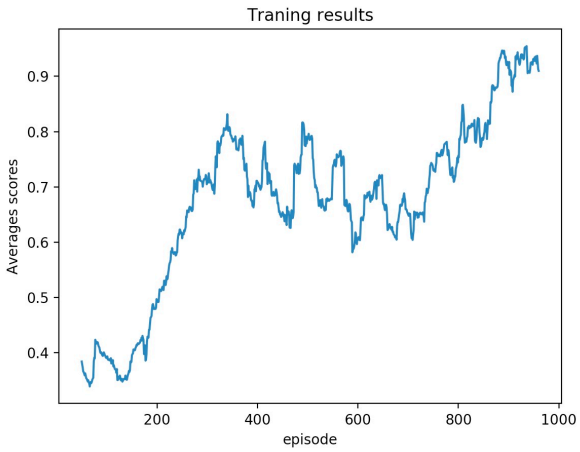

Figure 7. Training results of the H-DRLN network. The vertical axis represents the average scores of the last 100 episodes.

4.3 Discussion

In our experiment, we train three skill networks and then use them in our main task. The good performance of the "find objects" and "use tools" in the main task shows the effectiveness of the skill networks and the idea of lifelong

learning. However, several problems still remain and require our attention for further research in the future, one of these problems is the decrease of the curve in the "collect apple" models. We will try to improve its performance and experiment with more interesting tasks and methods.

5. Conclusion

In this work, we propose a complex task in the world of Minecraft and a feasible solution by applying the concept of lifelong learning to the H-DRLN architecture. Our experiment shows that the pre-trained skill networks actually contribute to the main task, which is presents a certain difficultly when solutions are pursued using flat algorithms.

In future works, we will concentrate on the property of partial observability in Minecraft. We will try to make the agent memorize the history of its experience by using Recurrent Neural Network (RNN) with deep reinforcement learning.

References

[1] Playing Atari with Deep Reinforcement Learning, Mnih, V. and Kavukcuoglu, K. and Silver, D. and Graves, A. and Antonoglou, I. and Wierstra, D. and Riedmiller, M., 2013.

[2] Selecting Subgoals using Deep Learning in Minecraft: A Preliminary Report, David Bonanno and Mark Roberts and Leslie N. Smith and David W. Aha, 2016.

[3] A Deep Hierarchical Approach to Lifelong Learning in Minecraft, Tessler, C. and Givony, S. and Zahavy, T. and Mankowitz, D. J. and Mannor, S., 2016.

[4] Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning, Tianmin Shu and Caiming Xiong and Richard Socher, 2018.

[5] Reinforcement Learning: An Introduction, Sutton, R.S. and Barto, A.G. and Barto, R.S.S.A.G. and Barto, C.D.A.L.L.A.G. and Bach, F., 1998.

[6] Q-learning, Watkins, Christopher J. C. H. and Dayan, Peter, 1992.

[7] Human-level control through deep reinforcement learning, Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Rusu, Andrei A. and Veness, Joel and Bellemare, Marc G. and Graves, Alex and Riedmiller, Martin and Fidjeland, Andreas K. and Ostrovski, Georg and Petersen, Stig and Beattie, Charles and Sadik, Amir and Antonoglou, Ioannis and King, Helen and Kumaran, Dharshan and Wierstra, Daan and Legg, Shane and Hassabis, Demis, 2015.

[8] Deep Reinforcement Learning with Double Q-learning, van Hasselt, H. and Guez, A. and Silver, D., 2015.

[9] Dueling Network Architectures for Deep Reinforcement Learning, Wang, Z. and Schaul, T. and Hessel, M. and van Hasselt, H. and Lanctot, M. and de Freitas, N., 2015.

[10] Distributed Deep Q-Learning, Ong, H. Y., Chavez, K., & Hong, A. 2015.

[11] The Bases of Competence: Skills for Lifelong Learning and Employability, Evers, F.T. and Rush, J.C. and Berdrow, I., 1998.

[12] Lifelong Machine Learning Systems: Beyond Learning Algorithms, Daniel Silver and Qiang Yang and Lianghao Li, 2013.

[13] Distilling the Knowledge in a Neural Network, Hinton, G. and Vinyals, O. and Dean, J., 2015.