Dynamic Skill Acquisition in Multi-Task Policy-Distilled Agents

Itai Peri, Asaf Aharoni
Electrical Engineering Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

We propose a lifelong learning system that utilizes knowledge obtained from previously learned skills in order to help generalize and accelerate the learning of new skills. A skill is a strategy learned by the agent in order to perform a certain task. We focus on skill learning in the domain of Minecraft, a popular video game which is an unsolved and high-dimensional lifelong learning problem. We accumulate knowledge in the form of skills and enable the learning of new skills, using our online form of policy distillation, eliminating the need for a domain expert required in traditional distillation setups, and speeding up the training process. Moreover, when learning new skills, we enable knowledge retention of previously learned skills, i.e., overcoming catastrophic forgetting. We provide an ablative analysis that details the conditions under which the learning system can mitigate catastrophic forgetting whilst learning new skills.

Introduction

Lifelong Learning is a sub-domain of Machine Learning and AI, focused on developing systems that continually and gradually learn how to perform new tasks over the course of time. Lifelong Learning is an open research problem, and is a crucial milestone towards achieving General Artificial Intelligence (GAI). A formal definition of Lifelong Learning is given by Silver, Yang, and Li (2013):

Definition 1. Lifelong Learning is the continued learning of tasks, from one or more domains, over the course of a lifetime, by a lifelong learning system. A lifelong learning system efficiently and effectively (1) retains the knowledge it has learned; (2) selectively transfers knowledge to learn new tasks; and (3) ensures the effective and efficient interaction between (1) and (2)

In real-world domains, lifelong learning hinders from the curse of dimensionality. That is to say, that the state space and action space are infinitely large, and therefore cause solving problems in such domains a very difficult task. Minecraft, as a high-dimensional and open-world video game, serves as a good approximation of real-world domains on one hand, and a flexible experimental sandbox on the other. Since Minecraft is an open-world game, many complex scenarios and tasks may be incorporated into a single domain, in order to examine the ability of Lifelong Learning agents to adapt to different scenarios and tasks in such domain. Due to the high complexity and flexibility of the game, Minecraft is an open research problem as it is impossible to solve the entire game using a single AI technique (Bonanno et al., 2016). Instead, the solution to Minecraft may lie in solving sub-problems (noted as *tasks*) using a modular approach, and then providing a synergy between the various components. Once an agent learns to solve a task, we say it has acquired a *skill* that can then be reused when a similar task is subsequently encountered.

Accumulating knowledge in a single agent is a non-trivial problem. When examining the lifelong learning field and specifically the ability to learn new skills, we take into account the following traits: (1) Knowledge Transfer, meaning the ability to transfer knowledge from one skill to another. In the case of Deep Reinforcement Learning, this knowledge is transferred via shared network weights among skills. This trait is crucial in order to make the lifelong learning process scalable memory- and time-wise, and also help it generalize better; (2) Dynamic Skill Acquisition (DSA), meaning the ability to learn new skills after an initial training on previous skills, with reasonable learning resources. This trait is important for lifelong learning agents, in order to evolve and scale through their lifetime. This is called an *online setting*, since new skills may be learned while the agent is still "online". Opposite to this, in the *offline setting* an agent must be re-trained from scratch in order to adapt to a new skill; (3) Knowledge Retention, meaning the preservation of relevant knowledge regarding previously learned skills. This is done in order to prevent catastrophic forgetting, which damages the ability to perform the skills in a future scenario. This trait is relevant only in the online setting.

In the work done by Tessler et al. (2016), a Deep Skill Module is suggested in order to incorporate multiple skills into a single network. This module is then able to perform all the skills incorporated into it, via the usage of a wrapping mechanism which decides which skill to perform at which time. The architecture used is a Distilled Skill Network, which along with the Deep Skill Module are brought in more detail in the Background Section.

The Deep Skill Module allows for knowledge transfer, but

lacks the ability of DSA. Since the distillation is performed offline, the module lacks the ability to learn new skills once the initial distillation process is done. In order to learn a new skill, the distillation process has to be re-run with all other skills, and also requires a teacher network (domain expert) for the new skill.

We suggest a novel approach to true lifelong skill learning, which enables transfer of knowledge among skills and dynamic acquisition of new skills once the initial skills have been learned, while retaining knowledge relevant to previously learned skills. Our approach extends the Distilled Skill Network architecture of the Deep Skill Module to the online setting in order to enable DSA, while retaining the knowledge transfer ability of this module. We show that by using our methodology, we enable learning of new skills, eliminating the need for a domain expert required in traditional distillation setups, and speeding up the training process. Moreover, we enable knowledge retention of previously learned skills – overcoming the problem of catastrophic forgetting.

Background

Reinforcement Learning (RL): RL is a subfield of Machine Learning which focuses on agents operating in dynamic environments. At each time step t the agent observes a state $s_t \in S$ which is dependent on the environment and the agent's internal state, takes an action $a_t \in A$, is rewarded a bounded reward $r_t \in [0, R_{max}]$ by the environment, where R_{max} is the maximum possible reward, and then transitions to a state $s_{t+1} \in S$ based on the action taken. The cumulative reward (or return) of the agent at time t is defined as $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$, where γ is a discount factor, which is used to discount rewards further along the time horizon, preferring short-term rewards over long ones. In infinite horizon problems, we require $\gamma \in (0, 1)$ in order to guarantee convergence of R_t. A policy $\pi : S \rightarrow \Delta_A$ is a mapping from states $s \in S$ to a probability distribution over the actions A. The action-value (Q) function given that the agent performs an action a and then follows a policy π is defined by $Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$, which represents the expected return at state s. The optimal Q function obeys a fundamental recursion known as the Bellman equation (Sutton, Precup, and Singh (1999)): $Q^*(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')\right]$. The goal of the agent is to maximize its expected return by learning a policy π which achieves Q^*.

Deep Q Networks (DQN): The DQN algorithm (Mnih et al., 2015) approximates the optimal Q function via a Convolutional Neural Network (Krizhevsky, Sutskever, and Hinton, 2012), by optimizing the network weights such that the expected Temporal Difference (TD) error of the optimal Bellman equation is minimized:

$$\mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \left\| Q_\theta(s_t, a_t) - y_t \right\|_2^2$$

where

$$y_t = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q_{\theta_{target}}(s_{t+1}, a') & \text{otherwise} \end{cases}$$

The DQN algorithm maintains two separate Q networks. The policy network has parameters θ and is the one being optimized by the algorithm. The target network with parameters θ_{target} is used as a baseline for computation of the TD error. The parameters θ_{target} are fixed in time, and are updated every fixed number of iterations to the current value of θ, in order to reflect the learning that has occurred since the last update.

The DQN is an offline learning algorithm, meaning that the tuples $\{s_t, a_t, r_t, s_{t+1}, \gamma\}$ for each time step t are collected from the agent's experiences and are stored in the Experience Replay (Lin, 1993). Every fixed number of steps where the agent operates in the environment and gains experience in the ER, mini-batch training is applied in order to optimize θ according to tuples sampled at random from the ER.

Skills and Options (Sutton, Precup, and Singh, 1999): A skill σ is a temporally extended control structure defined by a triplet $\sigma = \langle I, \pi, \beta \rangle$, where I is the set of states where the skill may be initiated, π is the intra-skill policy, which determines how the skill behaves for every $s \in S$, and β is the set of termination probabilities determining when a skill will stop executing.

Multi-Task Agent: Multi-task agent is an agent which has the ability to perform multiple tasks, based on a task selector switch.

Policy Distillation (Rusu et al., 2015): Distillation (Hinton, Vinyals, and Dean, 2015) is a method to transfer knowledge from a teacher model T to a student model S. In the case of deep neural networks, distillation is the process of a student learning to predict its teacher's outputs, which serve as the ground truth for the student in a supervised setting. In our experiments, we choose to work with a MSE loss function: $cost(s, a) = \| Q_T(s, a) - Q_S(s, a) \|^2$, where $Q_T(s, a), Q_S(s, a)$ are the values of the Q function of the teacher and student for state s and action a, respectively. Policy distillation is further extended to transfer knowledge from multiple teachers to a single student, by switching the teachers every fixed number of iterations during the training process. In order to be compatible with all tasks, a separate output layer exists in the student for each task, which is also switched upon when the teacher is switched.

Deep Skill Network (DSN): a Deep Skill Network is a single DQN network which is designed to perform a single skill.

Deep Skill Module: in abstract terms, the Deep Skill Module is a multi-task agent capable of performing multiple skills by incorporating multiple DSNs. Given an input state $s \in S$ and a desired skill i, the Deep Skill Module selects the action chosen by the i^{th} DSN given the state s. One architecture of the deep skill module is **Distilled Skill Network:** a single deep network that represents multiple DSNs, incorporating multiple skills in one network. Here, the different DSNs share all of the hidden layers while a separate output layer is trained for each DSN via policy distillation (Rusu et al., 2015), as shown in Figure 1. In this case, the network outputs an appropriate action for all the skills incorporated in it, and an external mechanism decides which action to select.

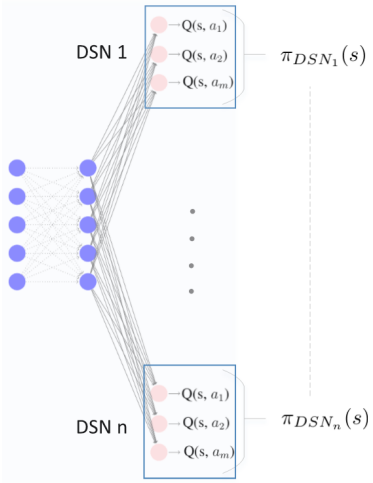

Figure 1: Distilled Skill Network architecture as suggested by Tessler et al. (2016)

Dynamic Multi-Skill Networks

In this section, we present an in-depth description of the Dynamic Multi-Skill Network (DMSN); a new training methodology that extends the Deep Skill Module and facilitates DSA while utilizing knowledge transfer between skills in lifelong learning scenarios.

First, we lay the conceptual grounds of the DMSN, and then dive into details regarding architecture. The DMSN may be analogized to a smart phone, where each application represents a specific skill of the network. The user of the smart phone may choose, at any given time, which application to use in order to perform the task at hand. Moreover, the user can freely switch applications in order to perform various tasks, instead of using multiple separate tools, such as a dedicated calculator, camera, etc. The ability of all applications to share the same hardware, operating system and some data regarding the user, instead of different components for each application, is analogous to the knowledge transfer trait. The DSA trait is represented by the ability of the user to install a new application on the smart phone which allows new functionality and the ability to solve new problems. Lastly, the knowledge retention trait is represented by the fact that each application has its own data stored on disk, which is not shared with other applications.

Layer Abstraction Level in Neural Networks: In general, the neurons of a neural network represent abstract features of the state space. Each layer, which is composed of multiple neurons, represents a different level of abstraction, which gets more task specific the deeper the layer is in the network. For convenience, we define the following sets: (1) *head*, the layers which contain task-related features; and (2) *body*, the layers which contain domain-related features. Therefore, we can say the network is comprised of a body and a head, and all layers of the network belong to one of the two.

The Domain-Task Partitioning Hypothesis: in accordance with the layer abstraction level, we offer our hypothesis that during the training of networks for different tasks in a specific domain, there exists an index i, denoted the *head index*, such that under the serial partitioning: $B = \{L_1, \ldots, L_{i-1}\}, H = \{L_i, \ldots, L_n\}$, where L_j is the j^{th} layer in the network, the set B includes only layers which belong to the body and H includes only layers which belong to the head.

DMSN Methodology

The multi-task agents used in our approach are an extension of the Distilled Skill Network, which is based on the vanilla DQN algorithm. Our implementation resembles the distillation architecture, where the body is shared among all tasks, and each task has its own head instance. The different heads may be connected to the body in order to form a fully-operating network that solves a specific task. The task selector switch controls which head is connected to the body at any given time.

For clarification reasons, we define two different types of tasks: (1) base tasks, which are learned in an offline manner, via policy distillation using domain experts; and (2) dynamic tasks, which are learned online by the multi-task agent itself, with no domain expert involved in the process.

Achieving knowledge transfer in a multi-task lifelong agent faces several challenges during training: (1) Compatibility between the single body to the different heads, so that each head performs its task correctly; (2) Richness of domain features represented in the body, so that heads have enough meaningful input from the body in order to be able to learn their task. Using policy distillation overcomes these challenges, and allows learning multiple base tasks a priori to the online learning setting. Compatibility is achieved by constantly switching between the domain experts and their corresponding heads during distillation, allowing the network to adjust the body by small steps for all heads; richness of domain features is achieved by the fact that each task relies on different features of the domain, and by the constant switching during training, all features which are important to any one of the base tasks are incorporated into the body. The more diversity of base tasks included in the distillation process, the more diverse features of the domain are incorporated into the body, and the better the network is able to generalize to new, unseen tasks.

Policy distillation is an offline process, while learning to perform dynamic tasks requires an online one. Therefore, base tasks which have domain experts may be learned during distillation, in contrast to dynamic tasks, which require a different process. In order to learn a dynamic task, the agent must interact with its environment via observations and rewards, in order to learn how to perform the task, since a domain expert does not exist. To do so, we add a new head per each dynamic task, and train the agent in its environment as we would train an agent in a regular RL setting.

Training an agent through this procedure leads to learning throughout the whole network, body and head alike, which may results in some cases – as detailed in the Experiments Section – in catastrophic forgetting, since the body is now incompatible with the heads of the base tasks learned during distillation. In order to overcome this, and achieve both the

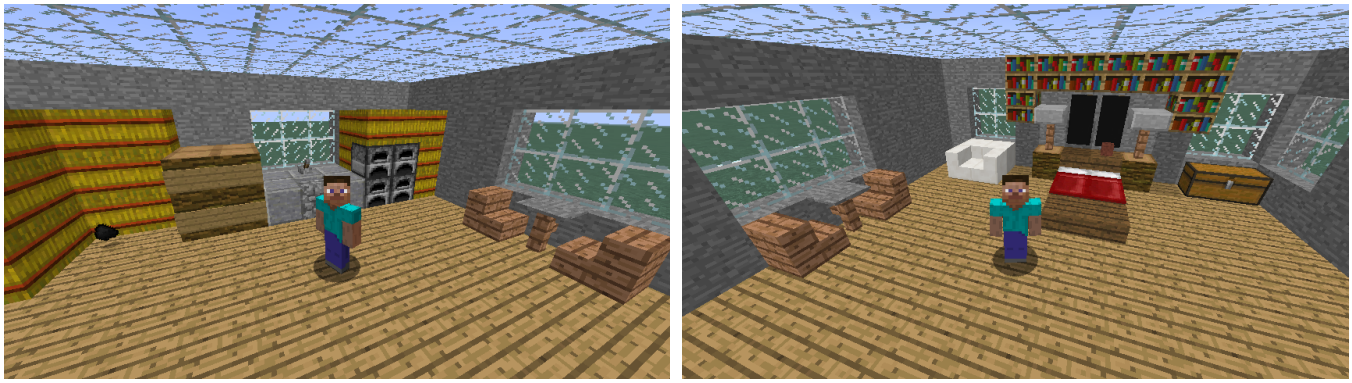

Figure 2: Minecraft domain – a home with everyday furniture and items. Left: (1) mining task is performed by picking up the coal in the left corner; (2) cooking task is performed by interacting with furnaces in the right corner. Right: hunting task is performed by taking a piece of chicken out of the chest in the right corner.

DSA and knowledge retention traits, we suggest the following procedure:

Training a DMSN First, we distill the network with some base tasks in order to incorporate domain features into the body, during which we train heads to solve the base tasks. This enables us to store knowledge in the body that is used both by the base tasks which are trained offline, and by the dynamic tasks which are trained online. Next, during training of each dynamic task, we: (1) create a new head for the dynamic task, and connect it to the body, to create a well-formed network; (2) In the second stage, we consider two approaches, which we test throughout our experiments to determine which one yields better results. We denote *freezing* of a network's layers as zeroing-out the gradients of said layers during backpropogation, resulting in the layers' weights unchanging throughout the training process. In one approach, we freeze the network's body, and in the second approach, we leave the whole network unfrozen; (3) Train the agent for the dynamic task in a regular RL setting based on interaction with the environment via states, actions and rewards, the same as we would train a domain expert. In the case of the frozen body, only the head is optimized during stage (3).

We test two approaches in the second stage to test their effect on the performance of the network on both the base and dynamic tasks, in terms of scores and training times. On one hand, the frozen body scenario allows us to retain the knowledge distilled into the body during the distillation stage as-is, and also ensures compatibility of the body with the heads related to already learned base tasks, since both the body and the heads of the base tasks remain the same. However, this might have a negative effect on the training time and performance of the dynamic tasks. On the other hand, the unfrozen body scenario lets the training process of the dynamic tasks also optimize the body, which affects the operation of the network on all tasks, including the base tasks, and might damage the performance on them.

Next, we give an analysis of our method showing that our approach fulfills the three lifelong learning traits. Knowledge transfer is achieved firstly by using policy distillation, which constructs the body as a shared knowledge base among the base tasks. Both in the frozen and unfrozen settings knowledge is shared among tasks, but in the unfrozen setting it may also be altered by the dynamic tasks. DSA is achieved through the use of new heads for the dynamic tasks and the training of the agents for the specific task in the domain, while also making the new heads compatible with the body, since the body is given as the input of the heads. Lastly, knowledge retention can be achieved either by a correct use of the Domain-Task Separation Hypothesis, or artificially via the freezing of the body, which does not change during the dynamic tasks' training, and thus retains all knowledge and structure of features required to be compatible with base tasks'/previously learned dynamic tasks' heads in order to solve these tasks. Moreover, we achieve acceleration of learning by two means. The first, is that the domain features needed to train the dynamic task is already present in the network, and thus the network does not need to learn the features from scratch, resulting in shorter training time in terms of number of epochs until convergence of the network (until mastering the specific task). The second, is that we compute the gradients and the weight updates only for the layers included in the heads, and not for those included in the body. This speeds up the back-propagation process and the compute time of the optimization steps.

Experiments

In this section, we present several results of experiments conducted in order to show that our approach indeed shares and retains knowledge, is able to learn skills in an online fashion and learns faster than a regular agent trained for the task at hand, using transferred knowledge.

Our domain, as noted earlier, is of Minecraft. We constructed a room as shown in Figure 2, comprising of standard house furniture and items, in order to be able to perform everyday tasks in said domain. We designed three tasks for the agents to perform: (1) pick up coal from a specific location of the room; (2) take out a piece of chicken from a chest; (3)

cook said chicken with the coal inside a furnace. For convenience, we note these tasks as *mine*, *hunt*, *cook* respectively. In all tasks, the reward function is as follows: every action taken by the agent is rewarded with -1 points, and achieving its goal rewards the agent with 1000 points. The agents are able to make around 55 steps per episode.

In our experiments, the mining and hunting tasks are used as base tasks, while the cooking task is a dynamic one. Therefore, we train domain experts from-scratch for the mining and hunting tasks, then we proceed by using these two experts in order to train a distilled agent which can solve both base tasks, and finally train the agent in an online manner in order to solve the cooking task.

We consider 5 types of agents for analysis: (1) a distilled agent, with head index of 4, and an unfrozen body. This agent is noted *cooker-head-4-unfrozen*; (2) a distilled agent, with head index of 3, and an unfrozen body. This agent is noted *cooker-head-3-unfrozen*; (3) a distilled agent, with head index of 4, and a frozen body. This agent is noted *cooker-head-4-frozen*; (4) a distilled agent, with head index of 3, and a frozen body. This agent is noted *cooker-head-3-frozen*. After distillation, the distilled agents are trained online for the cooking task. (5) domain experts, trained from-scratch to perform the mining, hunting and cooking tasks, and used as performance baselines. These agents are noted, respectively, *miner-baseline*, *hunter-baseline* and *cooker-baseline*. miner- and hunter-baseline are also used during the distillation process in order to learn the base tasks. It is important to note that cooker-head-4-unfrozen/frozen may be thought of as agents which have been distilled by the vanilla policy distillation process, and then trained online for another task. We note *convergence time* as the time it takes an agent, during training on a dynamic task, to reach and maintain a score which represents success in performing a task. In our experiments, this score is over 950. Below, we give an analysis of the performance of the agents in several aspects. In each analysis, we analyze all agents in comparison to one another and to the baselines.

First, we investigate the performance per-task of all agents on all three tasks, as shown in Figure 3. These scores are evaluated post-training. First, we examine cooker-head-3-frozen, which exhibits the best results. As expected from its frozen nature, its performance on the mining and hunting tasks are on-par with the baselines' performance. Moreover, we see that it achieves baseline-level performance also for the cooking task, which yields initial evidence that task-domain separation done with head index of three leads to good separation of body and heads. This evidence is strongly reinforced by examination of the results of cooker-head-3-unfrozen. It, too, achieves baseline-level performance for the cooking task, and suffers only a slight degradation of 6.5-7% in its performance for the mining and hunting tasks. This indeed provides further evidence due to the fact that in spite of cooker-head-3-unfrozen's ability to optimize its body layers to its needs, it does not alter the body in a way that changes the features represented in it and used by the mining and hunting heads, meaning that the features represented in the body are indeed those which represent the domain, and are shared among all tasks. Next, we examine cooker-head-4-unfrozen. In spite its ability to perform the cooking task well, it suffers from catastrophic forgetting for both mining and hunting, which takes the form of very low scores in these tasks. This is due to bad domain-task partitioning, which was done with head index of four, as can be seen in contrast to the results obtained when using a head index of three. The main difference between the two head indexes is the sharing of layer L_3 among all tasks as part of the body, as is the case in cooker-head-4-unfrozen. It is evident that the layer L_3 is crucial for the representation of task features. With head index of three, this layer is part of the head of the network, and as such, is optimized separately for each task, enabling it to learn task features related only to the task that the head corresponds to. On the other hand, with head index of four, this layer is part of the body of the network, and as such, is shared among all tasks. During distillation, this layer is trained in an interleaving fashion, such that although it is shared among the mining and hunting tasks, the rapid alternation between the two allows them to distill both tasks' features into the layer. However, when training for the cooking task alone, with no alteration among the mining and hunting task, this layer is heavily optimized for the cooking task, rendering the features of the layer mostly cooking-related, with little sign of relevance for the other tasks. Thus, when evaluating the agent on the base tasks, the heads receive incorrect features as input and are unable to perform their task. Last, we see that cooker-head-4-frozen does not reach good results for the cooking task, in contrast to its results on the base tasks, which derives from the frozen setting, since the base tasks are not affected by the training of the dynamic task, but only by the initial distillation. The low score is due to its inability to converge in a reasonable time frame as will be shown in the second experiment.

In our second experiment, we examine the convergence time it takes all agents in order to master the dynamic cooking task, by a comparison of the validation scores as a function of the number of epochs that have passed in the training process, for all 5 agent types, as shown in Figure 4. Cooker-head-3-frozen converges after 17 epochs, which is a 50% improvement over the baseline. This can be explained by the initial state of the agents when the training process is initiated: the baseline agent DQN's parameters are randomly initialized, in contrast to cooker-head-3-frozen's DQN which brings to the table the knowledge distilled and shared by its base tasks in its body layers, giving it prior knowledge regarding important domain features. More evidence for this is obtained by looking at cooker-head-3-unfrozen. We see that even though this agent is able to optimize all of its layers during training for the dynamic task, it converges roughly at the same time, with a convergence time of 20 epochs, which is only 17.6% slower than cooker-head-3-frozen, and still 41.2% faster than the corresponding baseline. This slight slowdown may be attributed to the agent making further optimizations to its body layers, which as seen in previous analysis, does not greatly affect the domain features represented by the body, hence still resulting in similar convergence times. Next, looking at cooker-head-4-unfrozen, we see that it converges after only 13 epochs, which is 61.8% faster than the baseline, and 35% faster than cooker-head-

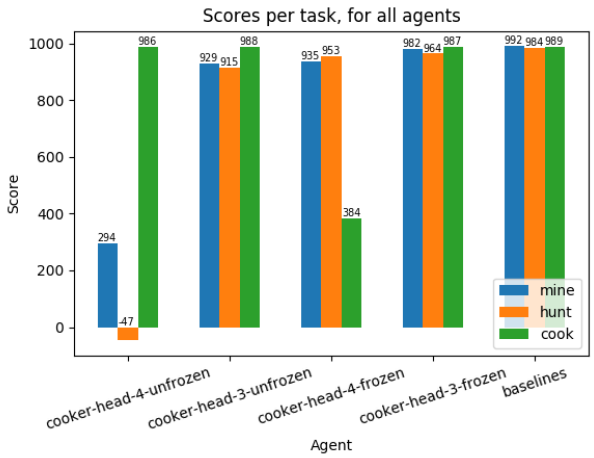

Figure 3: Validation score per-task, after training, for agents: (1) cooker-head-4-unfrozen, (2) cooker-head-3-unfrozen, (3) cooker-head-4-frozen, (4) cooker-head-3-frozen, (5) baselines, which represent the score achieved by each separate domain expert in its task, averaged on all experts, and for tasks: (i) mine (blue), (ii) hunt (orange), (iii) cook (green).

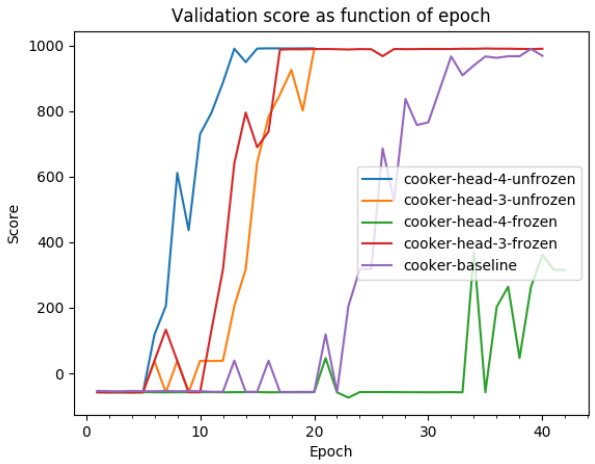

Figure 4: Validation score as a function of the number of epochs during training for the cooking task, for agents: (1) cooker-head-4-unfrozen (blue), (2) cooker-head-3-unfrozen (orange), (3) cooker-head-4-frozen (green), (4) cooker-head-3-frozen (red), (5) cooker-baseline (purple)

3-unfrozen. In order to explain this, we must consider the main difference between the two agents, which is layer L_3 of the DQNs of the agents. In the case cooker-head-3-unfrozen, layer L_3 is initialized randomly at the beginning of the training for the dynamic task, rendering layer L_4 *blind* at the beginning of the training, until layer L_3 begins to learn relevant features. In contrast, cooker-head-4-unfrozen's L_3 layer is part of the body, and hence also takes part during the distillation process. In spite of the differences between the base tasks and the dynamic one, some task features learned for the base tasks which are present in layer L_3, may also be similar to task features required by the dynamic task, and may only need small optimization to fit the dynamic task's needs. Moreover, the norms of the new randomly-initialized layers for the dynamic task differ by a significant magnitude compared to the corresponding norms which were obtained during distillation. The additional difference of 7 epochs in convergence time may be attributed to the fact that the network has to overcome these norm differences, and moreover – overcoming the problem in layer L_3 is not enough, and only after that the agent is able to also handle problems in its fourth and last layer. Last, cooker-head-4-frozen has not managed to converge in a reasonable time of 42 epochs, which are still 31.3% more epochs than the baseline agent. Therefore we can safely say that it is unable to learn dynamic tasks in a reasonable time, which is a requirement for scalable multi-task agents. As we concluded in the first experiment, the L_3 layer is crucial for task-related features representation, and since this agent is frozen, it cannot use this layer to represent features relevant to the cooking task, and is left only with features relevant for the hunting and mining tasks, resulting in the inability to learn the cooking task in a reasonable time frame.

Following on the previous experiments and conclusions, we examine qualitatively the performance on average of the 5 types of agents on all three tasks, as shown in Figure 5. This examination allows us to determine which agents are eligible to be considered true lifelong learning agents, and compare, in general, between different head indexes and the frozen/unfrozen settings. Moreover, we will establish a link between upholding the traits we defined as critical to lifelong learning agents, and the average score of all tasks of an agent.

Among all agents, cooker-head-4-unfrozen has the lowest average score, of 411, which is significantly below the 988 score of the baselines. This is due to the inability of the agent to retain its knowledge of the base tasks, leading to catastrophic forgetting. This is avoided in the case of cooker-head-3-unfrozen in accordance with the Domain-Task Separation Hypothesis, since the body and head partitioning is correct, resulting in appropriate task-related features for all tasks and their corresponding heads. This is ensured because of the correct choice of the head index, since the task-related features of the base tasks are not overwritten during the training process of the dynamic tasks. However, the scores of cooker-head-3-unfrozen are still lower than the baselines' and cooker-head-3-frozen's (discussed below), since during training, the features in the body are still being optimized for the dynamic task, in the price of its generalization ability.

Looking at cooker-head-4-frozen, we see that it scores relatively high. This is due to its knowledge retention, which enables it to perform the two base tasks with baseline-level performance, but lacks the ability to perform the dynamic task, and therefore scores shy above 2/3 of the baseline agents score. Comparing also cooker-head-4-frozen with cooker-head-3-frozen, we see that the latter exhibits better

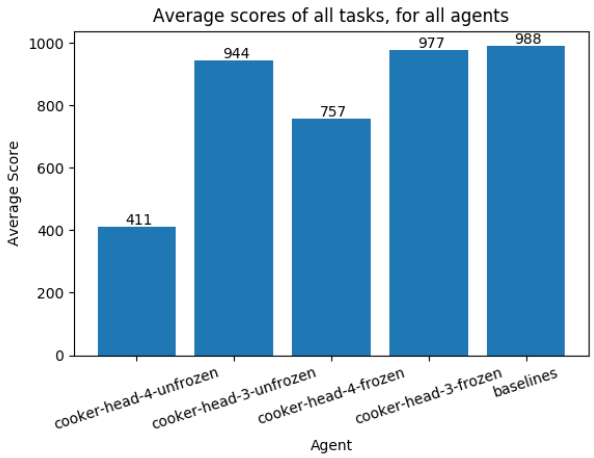

Figure 5: Average validation score of all tasks, after training, for agents: (1) cooker-head-4-unfrozen, (2) cooker-head-3-unfrozen, (3) cooker-head-4-frozen, (4) cooker-head-3-frozen, (5) baselines, which represent the score achieved by each separate domain expert in its task, averaged on all experts.

Table 1: Comparison of agents regarding the three traits of lifelong learning agents

Agent Type	Knowledge Transfer	DSA	Knowledge Retention
cooker-head-4-unfrozen	V	V	X
cooker-head-3-unfrozen	V	V	Partial
cooker-head-4-frozen	V	X	V
cooker-head-3-frozen	V	V	V

performance on average. This serves as additional evidence that the L_3 layer contains task-related features which are crucial for performing the tasks, and that the head index which is right for our network is three.

Last, we investigate cooker-head-3-frozen. From the previous experiment and by peaking an average score of 977, we conclude that this agent, a compressed multi-task and significantly quicker-to-learn agent, scores as good as 3 separate domain experts, while upholding the three lifelong learning traits.

To sum up, we compare all agents according to their fulfillment of the three lifelong learning traits, as shown in Table 1. All first three agents experience difficulties, such as catastrophic forgetting, only partial knowledge retention, and an inability to learn a dynamic task in reasonable time , in contrast to the last agent, cooker-head-3-frozen, which upholds all the three traits of lifelong learning agents.

Discussion

We have showed that by using our novel combination of distillation and online training called Dynamic Multi-Skill Network (DMSN), it is possible to create multi-task agents which uphold all three lifelong learning traits: (1) Knowledge transfer; (2) Dynamic skill acquisition; (3) Knowledge retention. This combination includes our novel variation of policy distillation, which, based on our Domain-Task Separation Hypothesis, leads to a distillation process which can be further trained in an online manner to perform additional, previously unseen tasks, with no domain experts. This is in contrast to the vanilla policy distillation, which as can be seen in our experiments, does not scale well to the online setting, and fails to uphold the three traits both in the frozen and unfrozen settings. It is important to note that the success of our process could be heavily influenced by choice of number and variation of the base tasks. Although our process is simple and is based on two widely-used RL techniques (policy distillation and Q-learning), it adds an overhead of an additional hyper parameter which is the head index by which to partition the network. In our current research, we have not managed to find a deterministic method to determine the optimal head index for a given problem and network architecture, and are left to set this parameter via empiric experimentation.

This work was done in a very specific and limited domain, and further research is required to determine whether DSA is applicable among different domains, such as different Minecraft environments, such as rooms, outdoors, etc. It is possible that integration with the H-DRLN solution depicted inTessler et al. (2016) is required in order to allow for adaptation across multiple domains.

We also offer an approach we call *Evolutionary Multi-Task Agents* to improve on our results in terms of better generalization, faster convergence times, and possible cross-domain adaptation, meaning the ability to perform tasks from different domains in one multi-task agent. We believe that such cross-domain adaptation might be possible thanks to its presence in the vanilla policy distillation technique, which is similar to the Evolutionary Multi-Task Agents technique. Our approach consists of a very simple iterative process, resembling the process in this work: Given a multi-task agent A_n trained for n tasks, like the ones presented in our paper, the process is as follows: (1) Add a new randomly-initialized head $n+1$; (2) Train the new head in an online fashion for a new task in a frozen setting; (3) Distill a new agent A_{n+1} for the $n+1$ tasks using the multi-task agent as a teacher of said tasks by using the relevant head for each task. Each agent A_i serves as an *ancestor* of the next agent A_{i+1} which is distilled by it, transferring its knowledge to A_{i+1} via policy distillation, which also inserts some "mutations" to the process compared to the original knowledge of A_i. Then the A_{i+1} agent evolves to learn a new skill, and the cycle repeats as necessary. Further research needs to be conducted regarding this approach in order to investigate its contributions.

References

Bonanno, D.; Roberts, M.; Smith, L. N.; and Aha, D. W. 2016. Selecting subgoals using deep learning in minecraft: A preliminary report.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. *ArXiv e-prints*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. 1097–1105.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, DTIC Document.

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* 518(7540):529–533.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation.

Silver, D. L.; Yang, Q.; and Li, L. 2013. Lifelong machine learning systems: Beyond learning algorithms. In *in AAAI Spring Symposium Series*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112(1-2):181–211.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2016. A deep hierarchical approach to lifelong learning in minecraft. *CoRR* abs/1604.07255.