Knowledge Transfer for Deep Reinforcement Learning with Hierarchical Experience Replay

Haiyan Yin and Sinno Jialin Pan
School of Computer Science and Engineering
Nanyang Technological University, Singapore
{haiyanyin, sinnopan}@ntu.edu.sg

Abstract

The process for transferring knowledge of multiple reinforcement learning policies into a single multi-task policy via distillation technique is known as *policy distillation*. When policy distillation is under a deep reinforcement learning setting, due to the giant parameter size and the huge state space for each task domain, it requires extensive computational efforts to train the multi-task policy network. In this paper, we propose a new policy distillation architecture for deep reinforcement learning, where we assume that each task uses its task-specific high-level convolutional features as the inputs to the multi-task policy network. Furthermore, we propose a new sampling framework termed hierarchical prioritized experience replay to selectively choose experiences from the replay memories of each task domain to perform learning on the network. With the above two attempts, we aim to accelerate the learning of the multi-task policy network while guaranteeing a good performance. We use Atari 2600 games as testing environment to demonstrate the efficiency and effectiveness of our proposed solution for policy distillation.

Introduction

Recently, the advances in deep reinforcement learning have shown that policies can be learned in an end-to-end manner with high-dimensional sensory inputs in many challenging task domains, such as arcade game playing (Mnih et al. 2015; Van Hasselt, Guez, and Silver 2016), robotic manipulation (Levine et al. 2016; Finn, Levine, and Abbeel 2016), and natural language processing (Zhang et al. 2016; Li et al. 2016; Guo 2015). As a combination of reinforcement learning with deep neural networks, deep reinforcement learning exploits the ability of deep networks to learn salient descriptions of raw state inputs, and thus bypasses the need for human experts to handcraft meaningful state features, which always requires extensive domain knowledge. One of the successful algorithms is Deep Q-Network (DQN) (Mnih et al. 2015), which learns game playing policies for Atari 2600 games by receiving only image frames as inputs. Though DQN can surpass human-expert level across many Atari games, it takes a long time to fully train a DQN. Meanwhile, each DQN is specific to play a single game.

To tackle the stated issue, model compression and multi-task learning techniques have been integrated into deep reinforcement learning. The approach that utilizes distillation technique to conduct knowledge transfer for multi-task reinforcement learning is referred to as policy distillation (Rusu et al. 2016). The goal is to train a single policy network that can be used for multiple tasks at the same time. In general, it can be considered as a transfer learning process with a student-teacher architecture. The knowledge is firstly learned in each single problem domain as *teacher* policies, and then it is transferred to a multi-task policy that is known as *student* policy. Such knowledge transfer is conducted via the distillation technique (Bucilu, Caruana, and Niculescu-Mizil 2006), which uses supervised regression to train a student network to generate the same output distribution as taught by the teacher networks.

Though some promising results have been shown in (Rusu et al. 2016; Parisotto, Ba, and Salakhutdinov 2016) recently, policy distillation for deep reinforcement learning suffers from the following three challenges. First, the existing architectures involve multiple convolutional and fully-connected layers with a giant parameter size. This leads to a long training time for the models to converge. Second, for some task domains, the compressed multi-task student network may not be able to achieve comparable performances to the corresponding teacher networks, or even performs much worse. This phenomenon is referred to as negative transfer (Pan and Yang 2010; Rosenstein et al. 2005). Last but not least, to learn from multiple teacher policy networks, the multi-task network needs to learn from a huge amount of data from each problem domain. Therefore, it is essential to develop an efficient sampling strategy to select meaningful data to update the network, especially for those domains where even training a single-task policy network costs a long time.

Our contributions are two-fold. First, a new multi-task policy distillation architecture is proposed. Instead of assuming all the task domains share the same statistical base at the pixel level, we adopt task-specific convolutional features as inputs to construct the multi-task network. It not only reduces the overall training time, but also demonstrates performance with considerable tolerance towards negative transfer. Second, we propose hierarchical prioritized experience replay to enhance the benefit of prioritization by regularizing the distribution of the sampled experiences from each domain. With the proposed experience replay, the overall learning for multi-task policy is accelerated significantly.

Related Work

This work is mainly related to policy distillation for deep reinforcement learning, and prioritized experience replay. Policy distillation is motivated by the idea of model compression in ensemble learning (Bucilu, Caruana, and Niculescu-Mizil 2006) and its application to deep learning, which aims to compress the capacity of a deep network via efficient knowledge transfer (Hinton, Vinyals, and Dean 2014; Ba and Caruana 2014; Tang et al. 2015; Li et al. 2014; Romero et al. 2015). It has been successfully applied to deep reinforcement learning problems (Rusu et al. 2016; Parisotto, Ba, and Salakhutdinov 2016). In previous studies, multiple tasks are assumed to share the same statistical base for pixel-level state inputs so that the convolutional filters are shared by all the tasks to retrieve generalized features from all tasks. Due to the sharing part, it takes a long training time for the resultant models to converge. Meanwhile, in the Atari 2600 games domain, the pixel-level inputs for different games differ a lot. Sharing the convolutional filters among tasks may result in ignorance of some important task-specific features, and thus lead to negative transfer for certain task(s). Therefore, in this work, we propose a new architecture for multi-task policy network. Different from the existing methods (Rusu et al. 2016; Parisotto, Ba, and Salakhutdinov 2016), we remain the convolutional filters as task-specific for each task, and train a set of fully-connected layers with a shared output layer as the multi-task policy network.

Besides the architecture, we also propose a new sampling approach, termed hierarchical prioritized experience replay, to further accelerate the learning of the multi-task policy network. Numerous studies have shown that prioritizing the updates of reinforcement learning policy in an appropriate order could make the algorithm learn more efficiently (Moore and Atkeson 1993; Parr 1998). One common approach to measure these priorities is using the Temporal Difference (TD) error (Seijen and Sutton 2013). The scale of TD error tells how 'surprising' the experience is to the underlying policy. Such prioritization has also been used in deep reinforcement learning (Schaul et al. 2016), resulting in faster learning and increased performance in the Atari 2600 benchmark suite.

Different from training a DQN for a single task domain (Schaul et al. 2016), to train a multi-task policy network with policy distillation, instead of using the TD error, the scale of gradient value for the distillation loss function is used to measure the priority for each experience, which tells how well the underlying student network could deal with the experiences. Furthermore, instead of purely sampling based on the prioritization, we additionally wish the sampled experiences to preserve the original distribution. To this end, we keep track of a state visiting distribution for each task domain to regularize the sampled experiences, which is estimated based on the state values predicted by the corresponding teacher networks. There are a number of studies on DQN that have used the state values to account for the state visiting distribution of DQN, e.g., (Zahavy, Zrihem, and Mannor 2016; Mnih et al. 2015).

Background

Deep Q-Networks

A Markov Decision Process is a tuple $(S, A, \mathcal{P}, R, \gamma)$, where S is a set of states, A is a set of actions, $\mathcal{P}$ is a state transition probability matrix, where $\mathcal{P}(s'|s,a)$ is the probability for transiting from state s to s' by taking action a, R is a reward function mapping each state-action pair to a reward in $\mathbb{R}$, and $\gamma \in [0,1]$ is a discount factor. The agent behavior in an MDP is represented by a policy π, where the value $\pi(a|s)$ represents the probability of taking action a at state s. The Q-function $Q(s,a)$, also known as the action-value function, is the expected future reward starting from state s by taking action a following policy π, i.e., $Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a\right]$, where T represents a finite horizon and r_t is the reward obtained at time t. Based on the Q-function, the state-value function is defined as:

$$V(s) = \max_a Q(s,a). \tag{1}$$

The optimal Q-function $Q^*(s,a)$ is the maximum Q-function over all policies, which can be decomposed using the Bellman equation as follows,

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a'|s,a)\right]. \tag{2}$$

Once the optimal Q-function is known, the optimal policy can be derived from the learned action-values. To learn the Q-function, the DQN algorithm (Mnih et al. 2015) uses a deep neural network to approximate the Q-function, which is parameterized by θ as $Q(s,a;\theta)$. The deep neural network can be trained by iteratively minimizing the following loss function,

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s,a}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i))^2], \tag{3}$$

where θ_i are the parameters from the i-th iteration. In the Atari 2600 games domain, it has been shown that DQN is able to learn the Q-function with low-level pixel inputs in an end-to-end manner (Mnih et al. 2015).

To train a DQN, a technique known as experience replay (Lin 1992) is adopted to break the strong correlations between consecutive state inputs during the learning. Specifically, at each time-step t, an experience is defined by a tuple $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, where s_t is the state input at time t, a_t is the action taken at time t, r_t is the received reward at t, and s_{t+1} is the next state transited from s_t after taking a_t. Recent experiences are stored to construct a replay memory $\mathcal{D} = \{e_1, ..., e_N\}$, where N is the memory size. Learning is performed by sampling experiences from the replay memory to update the network parameters, instead of using online data in the original order. To balance between exploration and exploitation, given an estimated Q-function, DQN adopts the ϵ-*greedy* strategy to generate the experiences.

Policy Distillation

Policy distillation aims to transfer policies learned by one or several teacher Q-network(s) to a single student Q-network via supervised regression. To utilize the knowledge of teacher networks during the transfer, instead of using the DQN loss derived from Bellman error as shown in (3) to update the student Q-network, the output distribution generated by the teacher networks is used to form a more informative target for the student to learn from. Suppose there is a set of m source tasks, $S_1, ..., S_m$, each of which has trained a teacher network, denoted by Q_{T_i}, where $i = 1, ..., m$. The goal is to train a multi-task student Q-network denoted by Q_S. For training, each task domain S_i keeps its own replay memory $\mathcal{D}^{(i)} = \{e_k^{(i)}, \mathbf{q}_k^{(i)}\}$, where $e_k^{(i)}$ is the k-th experience in $\mathcal{D}^{(i)}$, and $\mathbf{q}_k^{(i)}$ is the corresponding vector of Q-values over output actions generated by Q_{T_i}. The values $\mathbf{q}_k^{(i)}$ serve as a regression target for the student Q-network to learn from. Rather than matching the exact values, it has been shown that training the student Q-network by matching the output distributions between the student and teacher Q-networks using KL-divergence is more effective (Rusu et al. 2016). To be specific, the parameters of the multi-task student Q-network θ_S are optimized by minimizing the following loss:

$$\mathcal{L}_{KL}\left(\mathcal{D}_k^{(i)}, \theta_S\right) = f\left(\frac{\mathbf{q}_k^{(i)}}{\tau}\right) \cdot \ln\left(\frac{f\left(\mathbf{q}_k^{(i)}/\tau\right)}{f\left(\mathbf{q}_k^{(S)}\right)}\right), \quad (4)$$

where $\mathcal{D}_k^{(i)}$ is the k-th replay in $\mathcal{D}^{(i)}$, $f(\cdot)$ is the softmax function, τ is the temperature to soften the distribution, and $\cdot$ is the dot product.

Proposed Multi-task Policy Distillation

Architecture

In this paper, we propose a new multi-task policy distillation architecture as shown in Figure 1. In the new architecture, each task preserves its own convolutional filters to generate task-specific high-level features. Each task-specific part consists of three convolutional layers with each followed by a rectifier layer. The outputs of the last rectifier layer are used as the inputs to the multi-task policy network. A set of fully-connected layers are defined as the shared multi-task policy layers. Knowledge from the teacher Q-networks is transferred to the student Q-network through the shared policy layers. The final output of the student network is a set of all the available actions (e.g., 18 control actions for Atari). Instead of using gated actions to separate the actions for each task, the proposed architecture shares the final outputs, so that the shared actions go through the same path. For example, if two games both consist of the action *fire*, as their inputs are forwarded towards *fire* along the same path, the weights will be shared by both games in the forward path and being updated by both games in the backward path. Therefore, as the shared policy layers are trained by multiple source tasks, they can learn a generalized reasoning about when to issue what action under different circumstances.

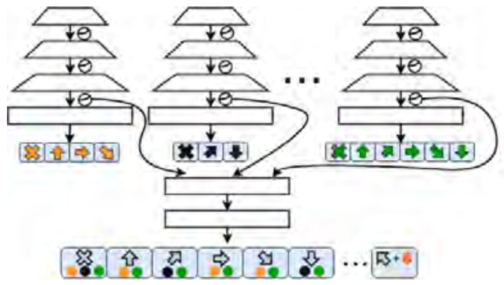


Figure 1: Multi-task policy distillation architecture

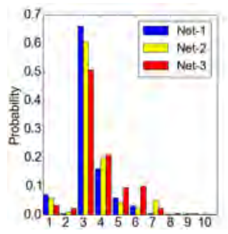
Overall, the new architecture concatenates a set of task-specific convolutional layers and shared multi-task fully-connected layers. The task-specific parts are available from the single-task teachers, but the multi-task fully-connected layers are trained from scratch. Using task-specific high-level features as the inputs to the multi-task architecture is crucial for the proposed policy distillation approach which involves end-to-end training. Studies over the state representation learned by multi-task deep policy network have shown that the low-level state representation is quite game-specific due to the diversity of pixel-level inputs, but the embedding over higher-level state representation shows higher within-game variance, which means that the games are more mixed out (Rusu et al. 2016). Therefore, sharing the convolutional filters among tasks may result in losing important task-specific information. Thus, we use task-specific high-level features to prevent negative transfer effect. Meanwhile, sharing the entire layers makes the model difficult to encorporate useful pre-trained knowledge. The proposed approach utilizes the existing knowledge from the convolutional filters, which helps to significantly improve the time efficiency for training the proposed architecture.

Hierarchical Prioritized Experience Replay

In this section, we introduce hierarchical prioritized experience replay to sample experiences from the replay memories of multiple task domains to train the multi-task student Q-network. The proposed approach is motivated by the design of replay memory for DQN and the prioritized experience replay approach proposed for a single DQN by Schaul et al. (2016). In a standard DQN, instead of using online generated data in original order, experiences are first stored in a replay memory, and then sampled uniformly for updating the network. This is to break the correlation between consequent states from online data, and benefit DQN by searching through the potentially huge state space more efficiently (Mnih et al. 2015).

The experiences stored at the replay memory form a distribution. For some games, such distribution varies a lot through the training time, as the ability of the policy network changes. For instance, in the game *Breakout*, DQN will not visit the state shown in Figure 2(a) unless the agent has learned how to dig a tunnel. Histograms over the state distributions generated by three *Breakout* policy networks with different playing abilities are also shown in Figure 2(b). The

(a) An example state



(b) State statistics

Figure 2: DQN state visiting for *Breakout*.

playing ability increases from *Net-1* to *Net-3*. The state distribution is computed according to the state value predicted by a fully-trained teacher network based on (1). The entire range of state values is evenly divided into 10 bins. It is interesting to find out that as the ability of the policy network increases, the distribution shifts towards visiting higher valued states more frequently. When performing sampling, it is important to preserve the state distribution in order to balance the learning of the policy network.

To improve the sampling efficiency for DQN, Schaul et al. (2016) proposed prioritized experience replay, which samples experiences according to the magnitude of their TD error (Sutton and Barto 1998). The higher the error is, the probability for the experience to be sampled becomes larger. By referring to TD error-based prioritization on experiences, prioritized replay intends to select more meaningful data to update the network. It turns out that such prioritization could accelerate the learning of policy network and lead DQN to a better local optima. However, prioritized replay introduces distribution bias to the sampled experiences, which means that the original state distribution cannot be preserved. Though importance sampling weights are post-processed to the sampled experiences for correcting the bias of updates on the network parameters in (Schaul et al. 2016), breaking the balance between learning from known knowledge and unknown knowledge may not be a good choice.

Therefore, directly applying the TD-based prioritized experience replay to multi-task policy distillation is not ideal. First, as described, the multi-task student network is updated using the distillation technique by minimizing the loss (4) between the output distributions of the student and teacher networks, rather than using the Q-learning algorithm. Thus, policy distillation requires for a new prioritization scheme. Second, the experience samples generated by solely using prioritized experience replay are not representative to preserve the global population of experiences for each domain.

To address the above two issues, we propose hierarchical prioritized experience replay, whereby a sampling decision is made in a hierarchical manner: which part from the distribution to sample, followed by which experience from that part to sample. To this end, each replay memory is first divided into several partitions, with each partition storing the experiences from a certain part of the state distribution. Within each partition, there is a priority queue to store the experiences according to their priorities. The partition sampling is done uniformly. This helps to make the sampled experiences preserve the global state visiting distribution for each task domain. Within a sampled partition, experiences are further sampled according their priorities, and importance sampling is performed to correct the bias of updates for student network parameters.

Uniform Sampling on Partitions For each problem domain S_i, a state visiting distribution is created according to the state values for each experience, which is computed by the teacher network Q_{T_i} following (1). The range for each state distribution is measured by generating some playing experiences by the teacher network in the problem domain, which is denoted as $[V_{min}^{(i)}, V_{max}^{(i)}]$. Then each state distribution range is evenly divided into p partitions, $\{[V_1^{(i)}, V_2^{(i)}], (V_2^{(i)}, V_3^{(i)}], ...(V_p^{(i)}, V_{p+1}^{(i)}]\}$. For each partition, a prioritized memory queue is created to store the experiences. Therefore, for each task domain S_i, there are p prioritized queues, with the j-th queue storing the experience samples whose state values fall into the range $(V_j^{(i)}, V_{j+1}^{(i)}]$. At runtime, the program keeps track of the exact number of experiences assigned to each partition j for each task S_i within a time window, denoted by $N_j^{(i)}$. When selecting which partition to sample from, uniform sampling is performed. Therefore, for task domain S_i, the probability for partition j to be selected is: $P_j^{(i)} = \frac{N_j^{(i)}}{\sum_{k=1}^{p} N_k^{(i)}}$.

Prioritization within Each Partition After a partition is selected for a task domain, e.g., partition j is selected for task S_i, all the experiences in the partition are prioritized based on the absolute gradient value of KL-divergence between the output distributions of the student network Q_S and the teacher network Q_{T_i} w.r.t. $\mathbf{q}_{j_{[k]}}^{(S)}$:

$$|\delta_{j_{[k]}}^{(i)}| = \frac{1}{|A_{T_i}|} \left\| f\left(\frac{\mathbf{q}_{j_{[k]}}^{(i)}}{\tau}\right) - f\left(\mathbf{q}_{j_{[k]}}^{(S)}\right) \right\|_1, \quad (5)$$

where $|A_{T_i}|$ is the number of actions for the i-th source task, $j_{[k]}$ is the index of the k-th experience in the j-th partition, and $|\delta_{j_{[k]}}^{(i)}|$ is the priority score assigned to that experience. Within the j-th partition for task domain S_i, the probability for an experience k to be selected is defined as:

$$P_{j_{[k]}}^{(i)} = \frac{\left(\sigma_j^{(i)}(k)\right)^{\alpha}}{\sum_{t=1}^{N_j^{(i)}} \left(\sigma_j^{(i)}(t)\right)^{\alpha}}, \quad (6)$$

where $\sigma_j^{(i)}(k) = \frac{1}{\text{rank}_j^{(i)}(k)}$ with $\text{rank}_j^{(i)}(k)$ denoting the ranking position of experience k in partition j determined by $|\delta_{j_{[k]}}^{(i)}|$ in descending order, and α is a scaling factor. The reason why we use the ranking position of an experience rather than proportion of its absolute gradient value to define probabilities is that prioritization on experience using rank-based information has been shown to be more robust for learning a single DQN (Schaul et al. 2016).

Bias Correction via Importance Sampling Consider experience k in partition j of the replay memory $\mathcal{D}^{(i)}$. Then the overall probability for the experience to be sampled is

$$P_j^{(i)}(k) = P_j^{(i)} \times P_{j_{[k]}}^{(i)}. \quad (7)$$

Though the sampling on partitions is uniform, the sampling on particular experiences within a partition is based on their priorities. As a result, the overall sampling still introduces bias to the updates of the student network parameters. Here, we introduce importance sampling weights to correct the bias brought by each experience,

$$w_j^{(i)}(k) = \left(\frac{\frac{1}{\sum_{t=1}^{p} N_t^{(i)}}}{P_j^{(i)} \times P_{j_{[k]}}^{(i)}} \right)^{\beta} = \left(\frac{1}{N_j^{(i)}} \times \frac{1}{P_{j_{[k]}}^{(i)}} \right)^{\beta}, \quad (8)$$

where β is a scaling factor. For stability reason, the weights are normalized by dividing $\max_{k,j} w_j^{(i)}(k)$ from the mini-batch, denoted by $\hat{w}_j^{(i)}(k)$. Thus, the final gradient used for mini-batch gradient update is $\hat{w}_j^{(i)}(k) \times \delta_{j_{[k]}}^{(i)}$.

In summary, with hierarchical prioritized experience replay, uniform sampling is performed over the partition selection to make the sampled experiences preserve a global structure of the original data distribution, while prioritization on experiences within each partition utilizes the gradient information to select more meaningful data to update the network. Though there is a requirement for a trained teacher network to perform partition sampling as an additional step, as policy distillation naturally falls into a student-teacher architecture where a teacher is already trained for single-task in advance, the requirement for teacher network should not be considered as a big external cost.

Experiments

Experimental Setting

To evaluate the efficiency and effectiveness of the proposed multi-task architecture, a multi-task domain is created with 10 popular Atari games: *Beamrider*, *Breakout*, *Enduro*, *Freeway*, *Ms.Pacman*, *Pong*, *Q*bert*, *Seaquest*, *Space Invaders*, and *River Raid*. To evaluate the impact of hierarchical prioritized experience replay on each single domain, we used a subset of 4 games from the multi-task domain: *Breakout*, *Freeway*, *Pong* and *Q*bert*.

The network architecture used to train the single-task teacher DQN is identical to (Mnih et al. 2015). For student network, we used the proposed architecture as shown in Figure 1, where the convolutional layers from teacher networks are used to generate task-specific input features with a dimension size of 3,136. Moreover, the student network has two fully connected layers, with each consisting of 1,028 and 512 neurons respectively, and an output layer of 18 units. Each output corresponds to one control action in Atari games. Each game uses a subset of outputs and different games may share the same outputs as long as they contain the corresponding control actions. During training, the outputs that are excluded in the game domain are discarded.

There is a separate replay memory to store experiences for each game domain. All the experiences are generated by an ϵ-greedy strategy following the student Q-network. The value for ϵ linearly decays from 1 to 0.1 within first 1 million steps. At each step, a new experience is generated for each game domain. The student performs one mini-batch update by sampling experience from each teacher's replay memory at every 4 steps of playing. When using hierarchical prioritized experience replay, the number of partitions for each replay memory is set to be 5. Each partition can store up to 200,000 experiences. When using uniform sampling, the replay memory capacity is set to be 500,000. Overall, the experience size for hierarchical experience replay is greater than uniform sampling. But this size has neutral effect on the learning performance empirically.

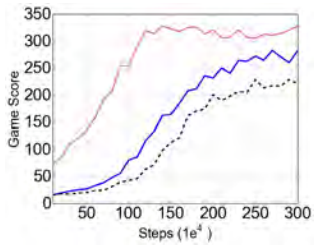
During the training, the network is evaluated once after every 25,000 times of mini-batch updates on each game domain have been performed. To avoid the agent from memorizing the steps, a random number of null operations (up to 30) are generated at the start of each episode. For each evaluation, the agent plays for 100,000 control steps, where the behavior of the agent follows an ϵ-greedy strategy with ϵ set as 0.05 (a default setting for DQN evaluation (Rusu et al. 2016)). The average episodic rewards over all the completed episodes during evaluation are recorded to report the performance of each policy network.

Evaluation on Architecture

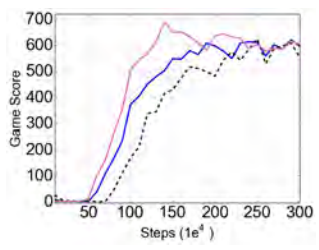
The proposed architecture is compared with two baseline architectures. The first baseline is proposed by Rusu et al. (2016), denoted by DIST, where a set of shared convolutional layers is concatenated with a task-specific fully-connected layer and an output layer. The second baseline is the Actor-Mimic Network (AMN) proposed by Parisotto, Ba, and Salakhutdinov (2016), which shares all the convolutional, fully-connected and the output layer.

During evaluation, a policy network is created according to each architecture on the multi-task domain. To make a fair comparison on the architectural effectiveness, all networks adopt uniform sampling for experience replay and use the same set of teacher networks. The RMSProp algorithm (Tieleman and Hinton 2012) is used for optimization. For statistical evaluation, we run each approach with three random seeds and report the averaged results. The networks under each architecture are trained for up to 4 million steps. A single optimization update on the DIST architecture takes the longest time. With modern GPUs, the reported results for DIST consumes approximately 250 hours of training time without taking into account of the evaluation time.

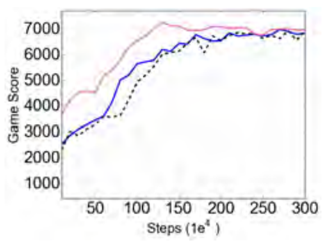
The performance for the best multi-task networks under each architecture for each task domain is shown in Table 1. For the multi-task networks, the performance value is evaluated as the percentage of the corresponding teacher network's score. For all the task domains, the proposed architecture could stably yield to performance at least as good as the corresponding teacher DQN. Therefore, it demonstrates considerable tolerance towards negative transfer. However, for DIST, the performance of the multi-task student network falls far behind the single-task teacher networks ($<75\%$) in



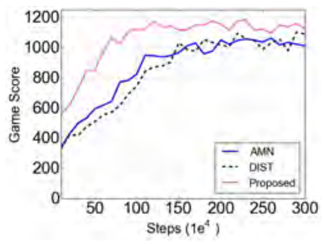
(a) Breakout



(b) Enduro



(c) River Raid



(d) Space Invaders

Figure 3: Learning curves for different architectures on the 4 games that requires long time to converge.

games *Beamrider* and *Breakout*. AMN does not learn well in *Beamrider* compared to its single-task teacher network, either. Moreover, the results in Table 1 demonstrate that the knowledge sharing among multiple tasks from the proposed architecture brings significant positive effect to the game *Enduro*, where a performance increase of >15% is shown.

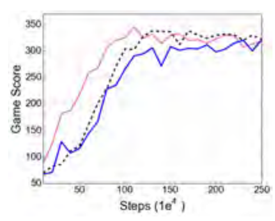
	Teacher	DIST	AMN	Proposed
	(score)	(% of teacher)		
Beamrider	6510.47	62.7	60.3	**104.5**
Breakout	309.17	73.9	91.4	**106.2**
Enduro	597.00	104.7	103.9	**115.2**
Freeway	28.20	99.9	99.3	**100.4**
Ms.Pacman	2192.35	103.8	**105.0**	102.6
Pong	19.68	98.1	97.2	**100.5**
Q*bert	4033.41	102.4	101.4	**103.9**
Seaquest	702.06	87.8	87.9	**100.2**
Space Invaders	1146.62	96.0	92.7	**103.3**
River Raid	7305.14	94.8	95.4	**101.2**
Geometric Mean		92.4[1]	93.5	**103.8**

Table 1: Performance scores for policy networks with different architectures in each game domain.

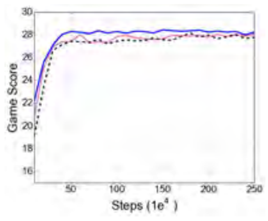
The proposed architecture also demonstrates significant advantage in terms of time efficiency for the learning. Among the 10 Atari games, *Breakout*, *Enduro*, *River Raid* and *Space Invaders* take longer time to train than others, as the proposed architecture converges within 1 million minibatch steps in all other domains but those four. We show the learning curves for different architectures on those four games in Figure 3. Even in those games which require for long training time, the proposed architecture could converge significantly faster than the other two architectures. For all of the 10 games, it could converge within 1.5 million steps, while the other two architectures require at least 2.5 million steps to get all games converge.

Evaluation on Hierarchical Prioritized Replay

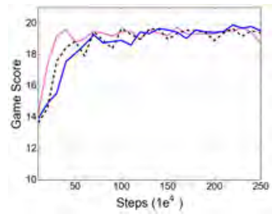
To evaluate the efficiency of the proposed hierarchical prioritized replay, denoted by H-PR, we compare it with two other sampling approaches: uniform sampling, denoted by Uniform, and rank-based prioritized replay (Schaul et al. 2016), denoted by PR. The four games are chosen so that the impact of sampling on games with both slow convergence (*Breakout* and *Q*bert*) and fast convergence (*Freeway* and *Pong*) could be shown. Note that when $p = 1$, H-PR is reduced to as PR, and when p is set to be the size of the replay memory, H-PR is reduced to as Uniform. All sampling approaches are implemented with the proposed architecture.



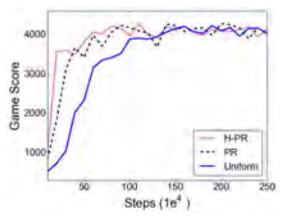
(a) Breakout



(b) Freeway



(c) Pong



(d) Q*bert

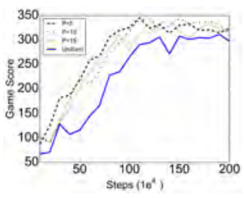
Figure 4: Learning curves for the multi-task policy networks with different sampling approaches.

The performance of the policy networks learned with different sampling approaches is shown in Figure 4. The games *Freeway* and *Pong* are very easy to train. Therefore, H-PR does not show significant advantage on these two tasks. However, for *Breakout* and *Q*bert* which require a relatively long time to converge, the advantage for H-PR is more obvious. Especially for *Breakout*, as the overall state visiting distribution for that game is changing quite dynamically during the learning phase, the effect of H-PR is large. For *Breakout* and *Q*bert*, H-PR only requires for approximately 50% of the steps taken by Uniform to reach a performance level of scoring over 300 and 4,000 respectively.

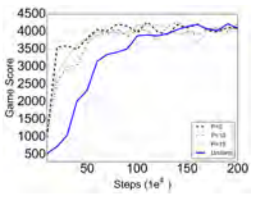
Sensitivity of Partition Size Parameter To investigate the impact of the partition size parameter, p, on the learn-

[1]Thanks for the comments from anonymous reviewer. We rerun this baseline with three random seeds and report averaged result.

ing performance of the multi-task policy network, H-PR is implemented on the proposed architecture with varying partition size, 5, 10 and 15. From the results shown in Figure 5, we observe that with different values of p, H-PR shows obvious acceleration impact on the learning. This indicates that the partition size parameter has a moderate impact on the learning performance for H-PR. However, when the capacity of each partition remains to be the same, the memory consumption increases with the partition size. Therefore, we chose 5 as the default value.



(a) Breakout



(b) Q*bert

Figure 5: Learning curves for H-PR with diff. partition sizes.

Conclusion

In this work, we investigate knowledge transfer for deep reinforcement learning. On one hand, we propose a new architecture for policy network, which introduces significant reduction in terms of training time, and yields to performance surpasses single-task teacher DQNs over all the task domains. On the other hand, we propose hierarchical prioritized experience replay to further accelerate the learning of multi-task policy network, especially for those tasks that even take very long time for single-task training. A direction of future work is to further accelerate the learning by incorporating efficient exploration strategy.

Acknowledgments

This work is supported by the NTU Singapore Nanyang Assistant Professorship (NAP) grant M4081532.020.

References

Ba, J., and Caruana, R. 2014. Do deep nets really need to be deep? In *NIPS*, 2654–2662.

Bucilu, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In *SIGKDD*, 535–541. ACM.

Finn, C.; Levine, S.; and Abbeel, P. 2016. Guided cost learning: Deep inverse optimal control via policy optimization. *arXiv preprint arXiv:1603.00448*.

Guo, H. 2015. Generating text with deep reinforcement learning. *arXiv preprint arXiv:1510.09202*.

Hinton, G.; Vinyals, O.; and Dean, J. 2014. Distilling the knowledge in a neural network. In *NIPS Workshop on Deep Learning and Representation Learning*.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *JMLR* 17(39):1–40.

Li, J.; Zhao, R.; Huang, J.-T.; and Gong, Y. 2014. Learning small-size dnn with output-distribution-based criteria. In *Interspeech*, 1910–1914.

Li, J.; Monroe, W.; Ritter, A.; and Jurafsky, D. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

Lin, L.-J. 1992. *Reinforcement Learning for Robots Using Neural Networks*. Ph.D. Dissertation, Pittsburgh, PA, USA. UMI Order No. GAX93-22750.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; a Rusu, A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; C. Beattie, A. S.; Antonoglou, I.; H. King, D. K.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*.

Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13(1):103–130.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22(10):1345–1359.

Parisotto, E.; Ba, J.; and Salakhutdinov, R. 2016. Actor-mimic deep multitask and transfer reinforcement learning. In *ICLR*.

Parr, D. A. N. F. R. 1998. Generalized prioritized sweeping. In *NIPS*.

Romero, A.; Ballas, N.; Kahou, S. E.; Chassang, A.; Gatta, C.; and Bengio, Y. 2015. Fitnets: Hints for thin deep nets. In *ICLR*.

Rosenstein, M. T.; Marx, Z.; Kaelbling, L. P.; and Dietterich, T. G. 2005. To transfer or not to transfer. In *NIPS Workshop on Inductive Transfer: 10 Years Later*.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2016. Policy distillation. In *ICLR*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay. In *ICLR*.

Seijen, H. V., and Sutton, R. S. 2013. Planning by prioritized sweeping with small backups. In *ICML*, 361–369.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Tang, Z.; Wang, D.; Pan, Y.; and Zhang, Z. 2015. Knowledge transfer pre-training. *arXiv preprint arXiv:1506.02256*.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *AAAI*, 2094–2100.

Zahavy, T.; Zrihem, N. B.; and Mannor, S. 2016. Graying the black box: Understanding dqns. In *ICML*, 1899–1908.

Zhang, M.; McCarthy, Z.; Finn, C.; Levine, S.; and Abbeel, P. 2016. Learning deep neural network policies with continuous memory states. In *ICRA*, 520–527. IEEE.