Generalization Tower Network: A Novel Deep Neural Network Architecture for Multi-Task Learning

Removed for Review

Abstract

Deep learning (DL) advances state-of-the-art reinforcement learning (RL), by incorporating deep neural networks in learning representations from the input to RL. However, the conventional deep neural network architecture is limited in learning representations for multi-task RL (MT-RL), as multiple tasks can refer to different kinds of representations. In this paper, we thus propose a novel deep neural network architecture, namely generalization tower network (GTN), which can achieve MT-RL within a single learned model. Specifically, the architecture of GTN is composed of both horizontal and vertical streams. In our GTN architecture, horizontal streams are used to learn representation shared in similar tasks. In contrast, the vertical streams are introduced to be more suitable for handling diverse tasks, which encodes hierarchical shared knowledge of these tasks. The effectiveness of the introduced vertical stream is validated by experimental results. Experimental results further verify that our GTN architecture is able to advance the state-of-the-art MT-RL, via being tested on 51 Atari games.

Introduction

Reinforcement learning (RL) is an active research area of artificial intelligence (AI). It aims at making a computer, as the agent, take actions in an environment to maximize the cumulative reward. The past decades have witnessed the evolution of RL (Sutton and Barto 1998), especially the recent development of deep RL (DRL) (Mnih et al. 2015). The recent DRL almost achieves human-level intelligence in mastering single task, for a small yet growing set of scenarios. Unfortunately, DRL is still in its infancy for handling more than one task, i.e., multi-task RL (MT-RL) (Caruana 1998).

For MT-RL, recent works have already achieved notable advances on inter-task transfer, showing the feasibility of transferring learned knowledge from one task to another. Actor-mimic (Parisotto, Ba, and Salakhutdinov 2016) and policy distillation (Rusu et al. 2015) leverage techniques from model compression to perform state-of-the-art inter-task transfer. Beyond inter-task transfer, multi-task generalization has also been considered in MT-RL. For example, shared representations have been verified to be formable, when facing diverse tasks (Borsa, Graepel, and Shawe-Taylor 2016; Romoff, Bengio, and Pineau 2016). The approach of progressive networks (Rusu et al. 2016) is insightful to transfer knowledge from the hidden layers of previously learned model to learn a never-experienced task. Yet, its multi-task ability is accomplished by storing all learned models and relying on inference to specify a corresponding model when switching to a specific task.

Despite all recent achievements for MT-RL, the multi-task generalization is still limited, or requires all learned models to be stored and then be manually selected according to the task. To avoid such limitation, this paper proposes[1] a novel deep neural network architecture for MT-RL, which is capable of handling all tasks within a single model. It is worth pointing out that our architecture can be easily combined with the existing MT-RL algorithms, e.g., actor-mimic (Parisotto, Ba, and Salakhutdinov 2016), policy distillation (Rusu et al. 2015), progressive networks (Rusu et al. 2016) and EWC (Kirkpatrick et al. 2017), as our work mainly focuses on the architecture of deep neural network for MT-RL. To be more specific, this paper proposes the generalization tower network (GTN), a novel deep neural network architecture for MT-RL. The proposed GTN is inspired by our finding in Figure 1, which implies that similar tasks in MT-RL share some hierarchical common knowledge, called *hierarchical shared knowledge*. As shown in Figure 1, we find that all shooting games[2] in Atari (Bellemare et al. 2013) need to *shoot continuously*, and some of them may further require *shooting to targets* or even *shooting to valid targets*. Thus, knowledge needs to be encoded in hierarchy for different tasks of MT-RL. Accordingly, our GTN architecture has both horizontal and vertical streams. Horizontal streams learn to extract multi-layer representations from

[1] Due to double blind reviewing policy, we will release our code later.

[2] For obtaining scores, we asked 10 volunteers aging from 18 to 26 to play these *shooting games*. All of them are naive to these games, avoiding the influence of prior knowledge. The volunteers were told to play the games only with three kinds of knowledge: I. *shoot continuously*, II. *aim to targets and then shoot continuously* and III. *aim to valid targets and then shoot continuously*. Scores of games are not displayed (by occlusion) for making knowledge valid. With each knowledge, they play three shooting games in a random order, each for 90 seconds.

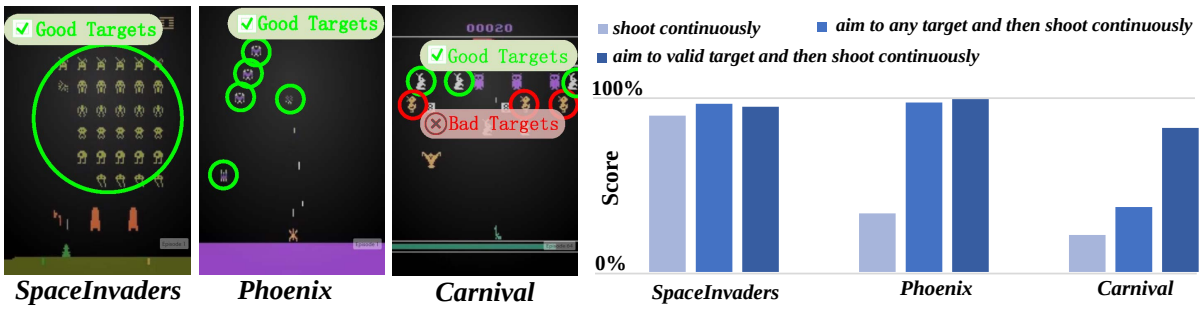


Figure 1: *Hierarchical shared knowledge* in shooting games. ***SpaveInvaders*** can obtain high scores by knowledge *shoot continuously*, since there are substantial and intensive targets. As for ***Phoenix***, the targets are much fewer but are all valid targets, high scores are obtained with knowledge *aim to targets and then shoot continuously*. ***Carnival*** is much more demanding for the players, since there are both bad targets and good targets. Thus, we should *aim to valid targets and then shoot continuously*. Thus, all three shooting games share knowledge of *shoot continuously*. Both ***Phoenix*** and ***Carnival*** games have knowledge of *aim to targets and then shoot continuously*. Only ***Carnival*** holds knowledge of *aim to valid targets and then shoot continuously*.

high-dimensional input, whereas vertical streams generalize *hierarchical shared knowledge*. The functionality of the introduced vertical steams is two-fold: (1) When facing multiple tasks, low-level features may be shared in the vertical stream; (2) Some tasks can be learned by short streams, while others may be learnt by long streams. We further verify through experiments that vertical streams of GTN are capable of improving the performance of mastering multiple tasks within a single model for MT-RL. More importantly, experimental results show that our GTN approach is better than the state-of-the-art asynchronous advantage actor-critic (A3C) for MT-RL (Romoff, Bengio, and Pineau 2016) on 51 games of Atari, with higher scores.

Related Work

Deep Reinforcement Learning

Taking advantage of recent success in deep learning (DL) (LeCun, Bengio, and Hinton 2015) and long-existing research in RL (Sutton and Barto 1998), DRL has made great progress, starting from deep Q-learning (Mnih et al. 2015). Notable advances include double Q-learning (Van Hasselt, Guez, and Silver 2016), prioritized experience replay (Schaul et al. 2015), dueling networks (Wang et al. 2016), and asynchronous methods (Mnih et al. 2016). Utilizing advances from these key achievements, A3C in (Mnih et al. 2016) achieves the state-of-the-art performance in mastering RL tasks, reaching human level intelligence in various domains. However, in spite of the existing achievements, agents in above RL works are still limited to learning and mastering one task at a time.

Multi-Task Reinforcement Learning

MT-RL is crucial in the RL area, which learns more than one task at a time. In this regard, the ultimate goals for MT-RL can be classified as: (1) **Inter-Task Transfer**, i.e., the RL model learned for one task can help in learning other tasks; (2) **Multi-Task Generalization**, i.e., multiple tasks can be handled by learning a single RL model. Above topics have long been reputed as a critical challenge in many AI works (Ring 1994; Silver, Yang, and Li 2013; Taylor and Stone 2011). Recent advances in this topic can be generally divided into following two directions.

Inter-Task Transfer. Inter-Task transfer works primarily focus on how to transfer knowledge from the learned model to a new one, when facing a new task. Upon such transfer, the new learned models are capable of mastering more tasks, in a way that different tasks are learned one by one. However, the main challenge for inter-task transfer is *catastrophic forgetting* (McCloskey and Cohen 1989; McClelland, McNaughton, and O'reilly 1995; Ratcliff 1990), which means that the newly learned knowledge may completely break previous learned knowledge of old tasks. There are some novel advances to this issue (Rusu et al. 2016; Fahlman and Lebiere 1990; Kirkpatrick et al. 2017; Rusu et al. 2015; Terekhov, Montone, and ORegan 2015; Parisotto, Ba, and Salakhutdinov 2016).

Multi-Task Generalization. Multi-Task generalization (Borsa, Graepel, and Shawe-Taylor 2016; Romoff, Bengio, and Pineau 2016; Sermanet, Xu, and Levine 2016) generally concentrates on how to make an agent master several RL tasks only with a single learned model. Linking preknowledge towards any tasks, the agent is greatly confused by the diversity of state representations and strategies. In short, multi-task generalization in RL mainly aims at learning a generalized model across diverse tasks. In this direction, previous works have been proposed with novel models and algorithms that are more capable of generalizing and transferring with shared representations. For example, the latest literature (Borsa, Graepel, and Shawe-Taylor 2016) verifies the practicability of learning shared representations of value functions. However, it has been demonstrated that a completely shared model performs poorly (Romoff, Bengio, and Pineau 2016). Furthermore, to deal with the problem of *task diversity*, the agent in (Romoff, Bengio, and Pineau

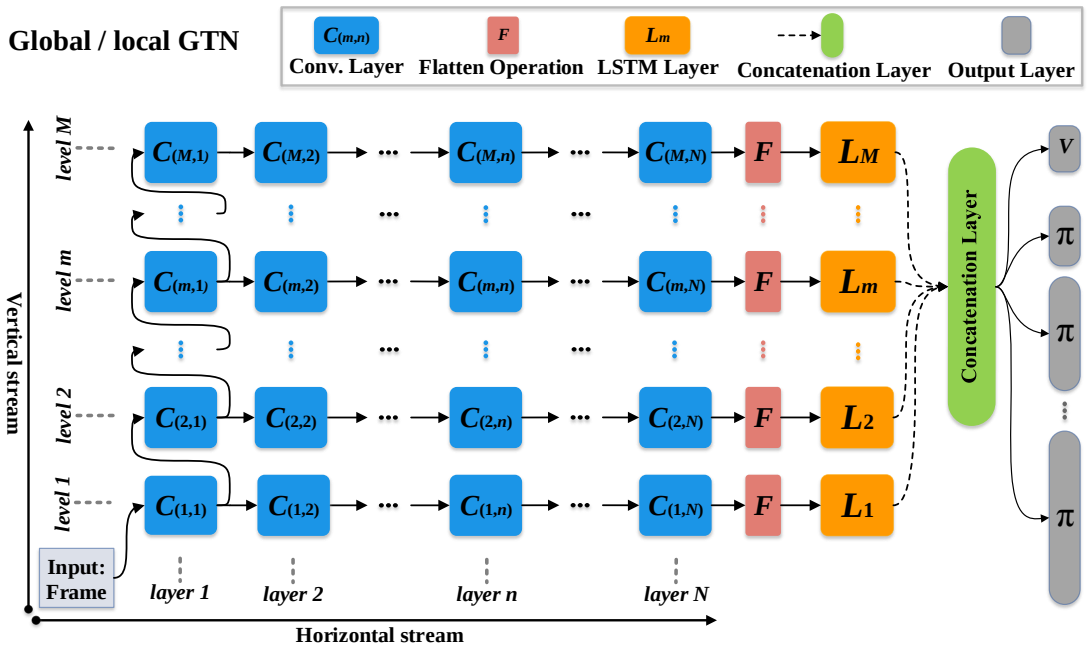


Figure 2: Architecture of Generalization Tower Network (GTN). As shown in this figure, GTN contains both horizontal and vertical streams. Specifically, horizontal streams at different levels are similarly composed of convolutional layers, flatten layer and LSTM layer. As for the vertical streams, they are achieved by sharing layers of horizontal streams in hierarchy.

2016) is designed to have different hidden layers and output layers for each task, remaining shared convolutional layers across all tasks. However, it can only learn to master 3 Atari games. An additional network (Sermanet, Xu, and Levine 2016) has been designed to detect which task is the agent facing, and then to favor or select a corresponding stream in the model.

In summary, recent works have made great advances in MT-RL. However, they still use conventional deep neural network architecture, which does not consider *hierarchical shared knowledge* across multiple tasks. This leads to poor generalization ability in mastering diverse tasks, which can be solved by the proposed GTN architecture presented in the following.

Generalization Tower Network

In this section, we introduce our GTN approach for MT-RL, which learns to master multiple RL tasks in a single learned model. Specifically, the architecture of GTN is proposed at first. Then, we present how GTN is implemented to work on multi-task scenarios in an asynchronous way, so that all tasks go through the same GTN model without requiring any task labels.

Architecture of GTN

Figure 2 shows the architecture of GTN. We can see from this figure that the input to GTN is frame content from the game emulator, and the output is state value V and policy π. Here, π is the set of vectors with different size, corresponding to the action space of different tasks. As shown in Figure 2, GTN is a two-dimensional network, including horizontal and vertical streams. In the horizontal stream, representations features are extracted "layer by layer", the same as the traditional deep neural networks. To model *hierarchical shared knowledge*, the vertical stream is introduced in GTN to master multiple tasks, in which higher levels[3] learn more representations features than lower ones. As such, our GTN architecture is suitable for diverse tasks, which can be well mastered by different numbers of features. In the following, we present the horizontal and vertical streams of our GTN architecture, respectively.

Horizontal streams with multi-layers. To present the horizontal streams of GTN, we take the horizontal stream at the *level 1* in (Figure 2) as an example. The horizontal stream of GTN contains multiple convolutional layers, i.e., $C_{(1,1)} \rightarrow C_{(1,2)} \rightarrow, ..., \rightarrow C_{(1,n)} \rightarrow, ..., \rightarrow C_{(1,N)}$, to learn features from the high-dimensional input. Thus, the horizontal stream contains layers $\{1, \cdots, n, \cdots, N\}$. Finally, the convolutional layer is flattened into a vector with flatten layer (F), followed by a single LSTM layer (L_1) to model the temporal correlation of RL tasks. For the LSTM chunk, we use the standard LSTM (Hochreiter and Schmidhuber 1997) which includes input, forget and output gates. Thus, the horizontal stream of our GTN mainly produces traditional representations features of deep convolutional neural network (Simonyan and Zisserman 2014) and learns tempo-

[3]In this paper, we use "level" and "layer"to denote information flows of the vertical and horizontal streams in GTN, respectively.

ral information with LSTM, the same as (Hausknecht and Stone 2015). Similar structure holds for horizontal streams at other levels.

Vertical stream with multi-levels. Our GTN architecture includes the vertical streams with levels $\{1, \cdots, m, \cdots, M\}$. Specifically, after the first convolutional layer at *level 1*, i.e., $C_{(1,1)}$, the stream of GTN splits into two sub-streams. One continues to go through the remaining horizontal layers in *level 1*, while the other is connected to *level 2*, acting as the input to *level 2*. As such, every *level m* uses the feature map from the first convolutional layer at *level m-1* as input. This way, the vertical stream is formed in GTN. Then, the last LSTM layers of the multiple horizontal streams, i.e., L_1, L_2, $\cdots$, L_m, $\cdots$, L_M are combined together for yielding the final output. Let the activation of LSTM layer L_m be a_m. The activation of the concatenation layer (H) is obtained by the Rectified Linear Units (ReLU) operation $\text{ReLU} = \max(0, x)$, as follows,

$$H = \text{ReLU}(\sum_{m=1}^{M} a_m \mathbf{T}_m), \tag{1}$$

where $\{\mathbf{T}_1, \cdots \mathbf{T}_m \cdots, \mathbf{T}_M\}$ are learned parameters, to concatenate M horizontal streams into a single layer. Note that $\mathbf{T}_m \in \mathbb{R}^{S \times A}$, where S is the size of LSTM layer and A is the size of concatenation layer. The vertical stream aims to model *hierarchical shared knowledge*, making our GTN fit to different tasks.

Implementation of GTN in asynchronous method

To learn multiple tasks with a single GTN model, we develop an asynchronous method to implement GTN based on (Mnih et al. 2016). Our asynchronous method is summarized in Figure 3. Specifically, there is one thread (thread 0) holding a global GTN and each of other multiple threads (thread $1, ..., i$) holding a local GTN and a game emulator. Note that both global and local GTNs use the same architecture as depicted in Figure 2. In a single episode for each thread, the following procedure is conducted.

- Stage 1: The thread copies all parameters from global GTN to its own local GTN.
- Stage 2: The local GTN interacts with the game emulator in the same thread and stores the experiences.
- Stage 3: Once gathering enough experiences (i.e., meet terminal condition t_{max}), the thread updates all parameters in the global GTN via accumulating gradient.

At Stage 2, the local GTN and the game emulator interact with each other for several steps $\{1, \cdots t \cdots, t_{max}\}$. At step t, GTN obtains *observation* o_t from the game emulator, which is the frame content gray-scaled and down-sampled to 42×42. Then, observation o_t is provided to GTN, together with the LSTM feature from the GTN at last step f_{t-1}. Afterwards, GTN produces a set of policies with different size, corresponding to the size-varied actions of MT-RL. Given policy π, action a_t can be generated with standard deviation ε to ensure exploration. According to a_t, the game emulator updates from o_t to o_{t+1}, returning a step reward r_t. At the end of step t, a set of experiences $\{o_t, f_{t-1}, a_t, r_t\}$ is stored for updating parameters at Stage 3. At Stage 3, we apply the accumulating gradient to update parameters as follows. We assume that θ and θ_v are parameter vectors that generate policy π and state value V, in global GTN. In local GTN, θ' and θ'_v are parameter vectors for π and V. Then, the accumulating gradients are calculated as

$$d\theta \leftarrow d\theta + \sum_{t=1}^{t_{max}} \nabla_{\theta'} \log \pi(a_t|o_t, f_{t-1}; \theta')(R_t - V(o_t, f_{t-1}; \theta'_v)), \tag{2}$$

$$d\theta_v \leftarrow d\theta_v + \sum_{t=1}^{t_{max}} \partial (R_t - V(o_t, f_{t-1}; \theta'_v))^2 / \partial \theta'_v. \tag{3}$$

In the above equation, R_t denotes the discounted reward:

$$R_t = \sum_{i=t}^{t_{max}} \gamma^{i-t} r_i. \tag{4}$$

where r_i is the step reward, and γ (=0.99) is the discount factor for future rewards.

Generalization Analysis

Now, we analyze the generalization ability of our GTN approach across multiple tasks. In this paper, generalization ability refers to the performance of GTN, when mastering different numbers of tasks within a single learned model. Since our GTN introduces the vertical stream for learning multiple tasks, we investigate the generalization ability of GTN at different levels of the vertical stream. For such investigation, we measure the average perturbation sensitivity (APS) (Rusu et al. 2016), which indicates how much the final output relies on a specific level of GTN. To obtain APS, we first inject Gaussian noise $\mathcal{N}(0, 1)$ to a specific horizontal stream at one level for producing perturbation. Then, we measure the impact of this perturbation on the performance, using APS. In this paper, APS is measured by the percentage of score dropping, due to the noise perturbation. We compute the relative APS (RAPS) via dividing APS by the sum of the APSs of all streams, to investigate the relative dependency of different levels of GTN.

Figure 4(a) shows the RAPS results of different levels in GTN, when learning 1 to 18 tasks. The results are obtained by running our GTN approach, averaged over all *shooting games* of Atari as list in Figure 5. From Figure 4(a), we can see that RAPS of high-level (e.g., levels 3 and 4) is increased at more tasks. This indicates that when learning to master more tasks, GTN turns to rely on higher levels. Thus, the proposed architecture of GTN, especially the vertical stream, is effective in improving the generalization ability for MT-RL. Figure 4(b) further draws the RAPS results of different levels versus training episodes, when running our GTN approach to handle all *shooting games* of Atari. One may observe that RAPS can be increased at more training episodes, for high-level in GTN (e.g., levels 3 and 4). By contrast, RAPS of low-level decreases alongside increased episodes. In a word, more levels in the vertical stream and larger number of training episodes are required for improving the generalization ability for MT-RL.

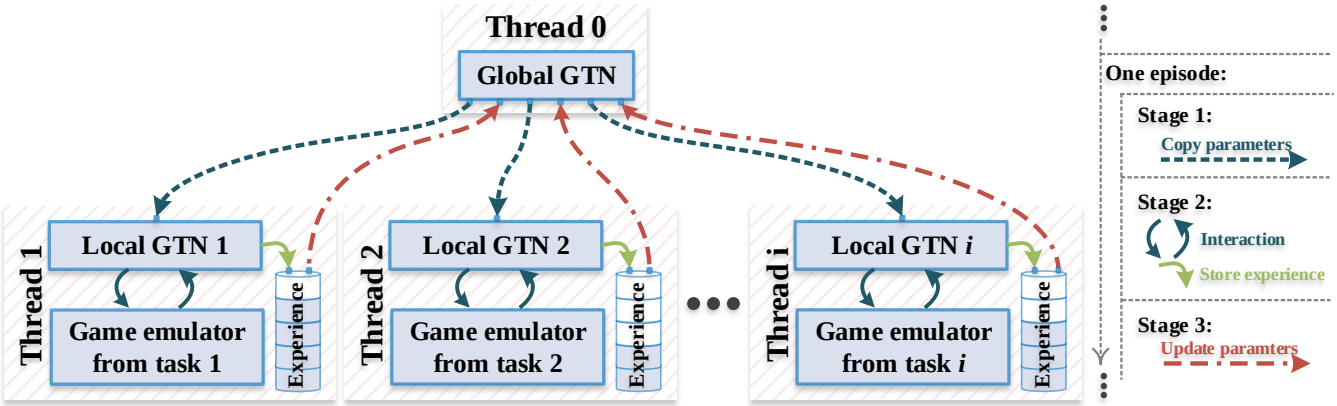
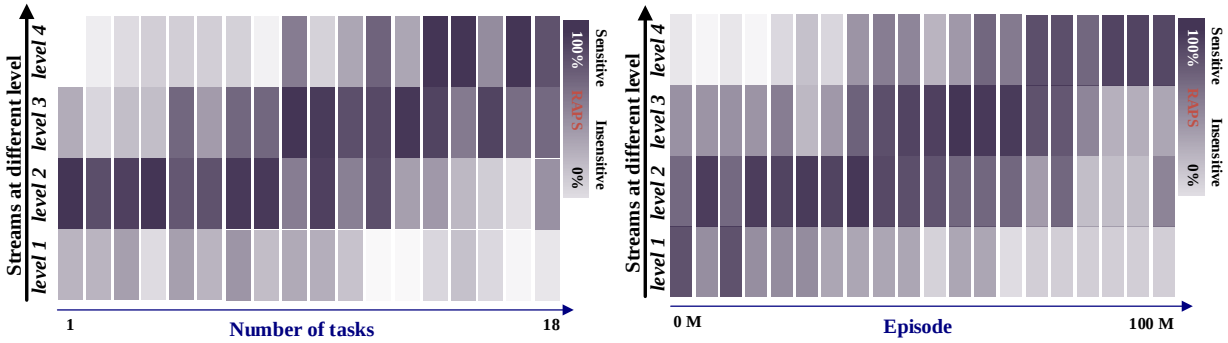


Figure 3: Mastering multiple tasks with a single GTN model: implement GTN asynchronously.



(a) (b)

Figure 4: (a): RAPS variance at different number of tasks. For obtaining RAPS, the experiment was conducted on a series of GTNs with $N = 4$ and $M = 4$, learning to master 1 to 18 *shooting games* from Atari. The results were obtained at 100M training episodes. (b): RAPS variance alongside training episodes, obtained in learning all 18 *shooting games*.

Experiment

Setup

In our experiments, we manually divide the Atari games into 6 categories, according to the similarity of tasks. All tasks belonging to each category can be found from Figures 5 and 6. They cover 51 of total 57 Atari games, and the remaining 6 games are hard to be classified. Thus, our experiments were conducted on those 51 games.

GTN Setup. The GTN is simply set with 4 levels ($M = 4$) in vertical streams and 4 layers ($N = 4$) in horizontal streams. In our GTN, the architecture of each horizontal stream is the same as that of A3C (Mnih et al. 2016), for fair comparison. Specifically, for all convolutional layers, the kernel size is 3×3 and the number of kernels is 32. The stride during convolution is set to 2. Besides, the size of the LSTM layers is all set to 288. For playing the Atari games, the minimal and maximal numbers of *actions* are 5 and 18, respectively. To master diverse tasks in a unified form, during the first 2 episodes, the agent detects and records the maximal step reward. Afterwards, every step reward is normalized by dividing this stored maximal step reward. We follow (Mnih et al. 2016) to configure other unmentioned settings.

Evaluation Metric. To evaluate the multi-task generalization ability, we measure how much a multi-task agent achieves or exceeds the scores performed by the corresponding single-task agent. To this end, we measure the relative final scores (RFS):

$$\text{RFS} = \frac{\text{Score}_{\text{multi-task}}}{\text{Score}_{\text{single-task}}}, \tag{5}$$

which eliminates the diverse scales of scores across multiple tasks[4]. Note that the same metric is also used in most recent works of MT-RL (Rusu et al. 2015; Parisotto, Ba, and Salakhutdinov 2016).

[4]Since the baseline of raw scores from different tasks is diverse, we follow (Mnih et al. 2015) to subtract the corresponding scores of random actions from the raw scores.

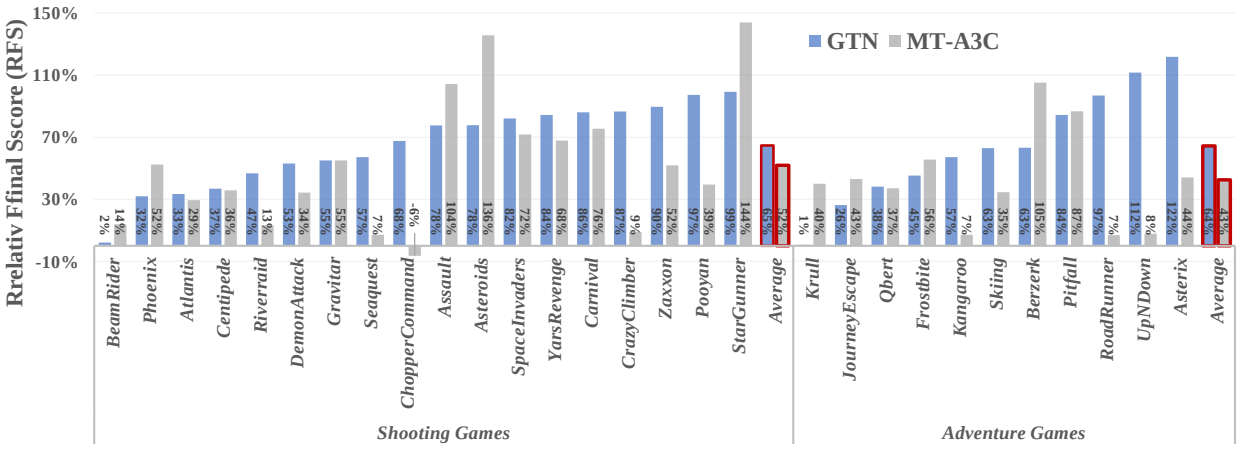


Figure 5: Comparison of RFS over classified Atari games: *Shooting Games* and *Adventure Games*.

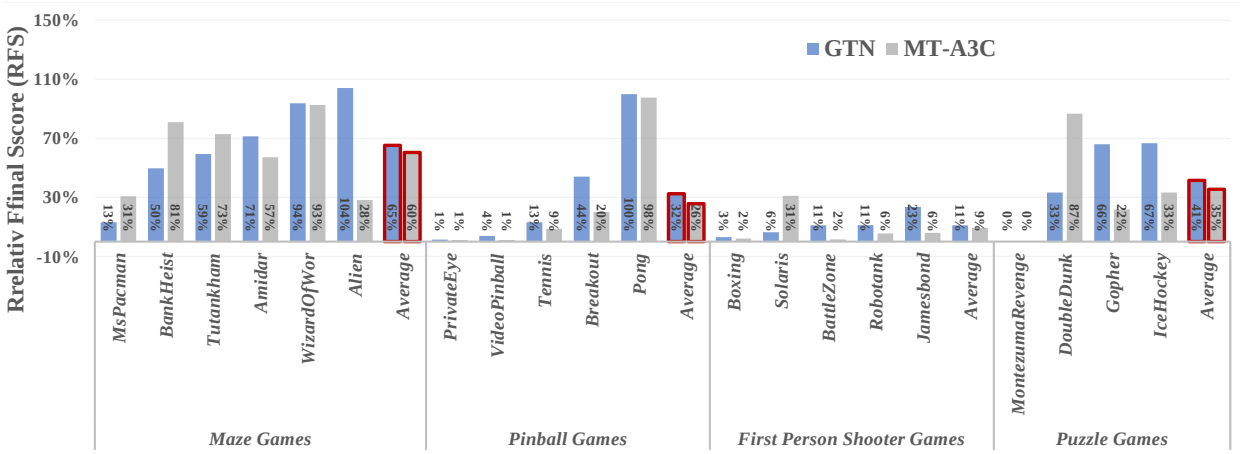


Figure 6: Comparison of RFS over classified Atari games: *Maze Games*, *Pinball Games*, *First Person Shooter Games* and *Puzzle Games*.

Performance Analysis of GTN

Since the vertical and horizontal streams are the main contribution of the proposed GTN architecture, we analyze the performance of our GTN approach, with different numbers of levels (M) and layers (N). In the following, the performance of *agent* is evaluated on single task and multiple tasks, when M ranges from 1 to 4 and N varies from 1 to 6.

Figure 7(a) shows the performance of GTN on single task, along with increased N and M. The performance in this figure is measured by RFS, averaged over all *shooting games* of Atari. We can see from this figure that when making GTN work on single task, the averaged RFS dramatically changes at different N and it can achieve 104% at $N = 4$ and $M = 2$. In contrast, M slightly improves the performance of our GTN on single task. This implies the horizontal stream is more effective in handling single task in RL. For multiple tasks, the averaged RFS values of playing all *shooting games* are used for performance evaluation. Figure 7(b) shows the averaged RFS values of GTN on handling multiple tasks of all *shooting games*, with different N and M. We can find from this figure that enhancement of M from 1 to 4 improves the averaged RFS from 52% to 65% at $N = 4$, whereas N has little impact on the averaged RFS. In a word, the above analysis indicates that the horizonal stream in our GTN approach is effective in mastering single task, and the vertical stream is "good at" learning multiple tasks.

Evaluation on Performance.

This section compares the RFS results between the proposed GTN and conventional MT-A3C (Romoff, Bengio, and Pineau 2016), both based on (Mnih et al. 2016). In particular, MT-A3C means that the asynchronous MT-RL framework is implemented with the same traditional deep neural network architecture as A3C, instead of our GTN. It is worth mentioning that we do not compare against progres-

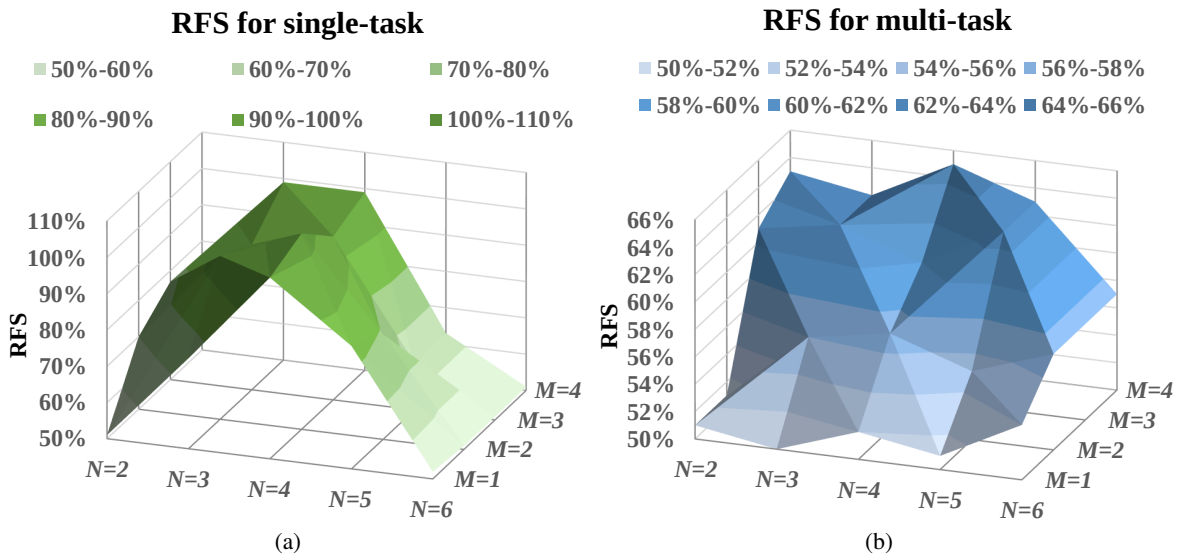


Figure 7: RFS results of our GTN approach at different M and N for mastering (a) single task and (b) multiple tasks. Note that the RFS results are averaged over all *shooting games* of Atari. For single task, each game is individually played by *agent* of GTN. For multiple tasks, all games are synchronously played by *agent* of GTN.

sive network (Rusu et al. 2016). It is because progressive network (Rusu et al. 2016) requires manual inference to select corresponding stored model for different tasks, whereas our GTN approach requires no task labels and enables mastering multi-tasks in one model. In other words, it is unfair to compare our approach with (Rusu et al. 2016).

Figures 5 and 6 plot the RFS results of MT-A3C and GTN for each game of different categories. We can see from these figures that GTN performs better than MT-A3C in all 6 categories, with up to **23%** improvement (*shooting games*) in average RFS. Interestingly, the performance gap between GTN and MT-A3C is especially large for *Shooting Games* and *Adventure Games*, which have more tasks than other 4 categories. This indicates that our GTN approach is more suitable for MT-RL with larger number of similar RL tasks, thus showing the high generalization ability of our GTN approach. This also makes our approach practical in real-world application.

Conclusion

This paper has proposed a novel deep neural network architecture called GTN for MT-RL, which incorporates both horizontal and vertical streams within a single learned model. Different from the conventional MT-RL approaches, the vertical stream proposed in our GTN architecture enables better performance when dealing with multiple tasks. This is mainly motivated by the fact that multiple tasks may share common knowledge in hierarchy. Experimental results also showed that our GTN approach improves the performance of A3C in MT-RL, over 51 Atari games.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Borsa, D.; Graepel, T.; and Shawe-Taylor, J. 2016. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

Caruana, R. 1998. Multitask learning. In *Learning to learn*. Springer. 95–133.

Fahlman, S. E., and Lebiere, C. 1990. The cascade-correlation learning architecture. In *NIPS*, 524–532.

Hausknecht, M., and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. In *AAAI*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 201611835.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

McClelland, J. L.; McNaughton, B. L.; and O'reilly, R. C. 1995. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review* 102(3):419.

McCloskey, M., and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation* 24:109–165.

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.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *ICML*.

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

Ratcliff, R. 1990. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review* 97(2):285–308.

Ring, M. B. 1994. *Continual Learning in Reinforcement Environments*. Ph.D. Dissertation, University of Texas at Austin.

Romoff, J.; Bengio, E.; and Pineau, J. 2016. Deep conditional multi-task learning in atari. In *ICML Workshop*.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295*.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. In *ICLR*.

Sermanet, P.; Xu, K.; and Levine, S. 2016. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*.

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

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taylor, M. E., and Stone, P. 2011. An introduction to inter-task transfer for reinforcement learning. 32(1):15.

Terekhov, A. V.; Montone, G.; and ORegan, J. K. 2015. Knowledge transfer in deep block-modular neural networks. In *CBBS*.

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

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H. v.; Lanctot, M.; and Freitas, N. d. 2016. Dueling network architectures for deep reinforcement learning. In *ICML*, 1995–2003.