利用對抗式目標與資料擴增於深度强化學習間的遷移

許書軒　　沈奕超　　陳炳宇

國立臺灣大學

Abstract

在最近幾年內，深度强化學習已經被充分證明可以用來解決高維度的複雜問題，深度强化學習的下一個重點將會是該如何讓神經網路學會不同環境間的核心概念、或是藉由已經學過的知識來加速學習新環境的速度。現今的强化學習訓練出來的模型大部分都有無法很好的處理新環境的缺陷，即便這個新環境與曾經學過的環境十分相近依舊無法有好的成績。我們提出的方法可以讓强化學習的模型從單一環境中學習到較為核心的特徵，並且可以透過半監督式學習來加速模型於新環境中的學習速度。最後我們在一個非常受歡迎的强化學習環境 —— Arcade Learning Environment (ALE) [Bellemare et al. 2013]檢驗我們提出的方法，並且可以發現我們的方法可以打敗常見的標準方法像是使用預訓練模型以及微調網路權重等。

Keywords: Machine Learning, Reinforcement Learning and Domain Adaption

1 Introduction

Deep Reinforcement Learning (DRL), the combination of reinforcement learning methods and deep neural network function approximators, has recently shown considerable success in high dimensional challenging tasks, such as arcade games [Mnih et al. 2015] and robotic manipulation [Levine et al. 2016]. These methods can learn features that often better than hand-craft features that require more domain knowledge. For example, the Deep Q-Network (DQN) [Mnih et al. 2015], one of the most famous DRL method, has achieved super human performance on the Arcade Learning Environment (ALE) [Bellemare et al. 2013], a benchmark of Atari 2600 arcade games.

Although the DRL algorithms can usually learn how to take the best action based on the state of the environment, but it can only learn a single environment at a time, despite the existence of similarities between those environments. For example, the tennis-like game of pong and the squash-like game of breakout are both similar in that each game consists of trying to hit a moving ball with a rectangular paddle, but an agent that is good at pong can not handle breakout well. Another issue of DRL is that training DRL agents can be very time-consuming, many researcher studies on the methods that can speed up training time [Mnih et al. 2016; van Hasselt et al. 2015].

Some research speed up learning on new tasks by perform cross environment transfer [Rusu et al. 2016; Parisotto et al. 2015], but they all need to pre-train an agent on multiple source environments to generalize the knowledge, which is very time consuming. In this work, we trying to leverage the prior knowledge that learn by a single source environment agent to speed up agent to handle new environment, using only one source environment prior knowledge can minimize the time to train on new environment, and can also solve some issues of reinforcement learning, including unable to handle similar tasks and long training time problems.

We proposed a semi-supervised domain adaption that use the concept of Generative Adversarial Network [Goodfellow et al. 2014]. More specifically, we learn a mapping from target observations to source feature space by fooling a domain discriminator that tries to distinguish the encoded target observation from source examples. We also found it is helpful for transfer knowledge by doing some augmentation when training on source task. While our approach can integrate into any DRL algorithm, we show results of our approach by combining it with DQN [Mnih et al. 2015] algorithm and showing results on Atari 2600 domain.

Our contributions are two-fold. First, we proposed a method that can leverage knowledge from a single source task agent to help speed up training on new target environment. Second, we found that perform augmentation on environment to train a source agent and use it as target task initialization often can help better result. With these proposed method, the overall learning on target task is accelerated compared with baselines that we have considered.

The rest of this paper organized as follows. In Section 2, we surveyed several previous method for reinforcement learning, domain adaption and multi-task agent. Section 3 we detailed the knowledge of Deep Q Network, as it is the most popular reinforcement learning algorithm and is used in our method. Section 4 we describe the two main method used in our approach, including *Adversarial objective* and *Augmentation*. Section 5 we do a series of experiments to evaluate our method, and further discuss in detailed. Finally, in Section 6 concludes the paper.

2 Related Work

Reinforcement Learning Reinforcement learning is a method that can learn how to map situations to actions, try to maximize the reward. Unlike other machine learning will usually provide correct action (label) used for direct feedback, the reinforcement learner is not told which action is best, but instead need to discover which actions yield the most reward by trying them. Actions taken by agent may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning.

In recent years, thanks to the significant progress in deep learning, numerous researchers have attempted to use deep learning to solve the reinforcement learning tasks [Mnih et al. 2015; Mnih et al. 2016; van Hasselt et al. 2015; Mnih et al. 2013]. Mnih *et al.* [2015] first presented Deep Q-Network (DQN) that uses a deep network to approximate the state-action value function, solving the storing space issue in traditional state-action tabular representation (Q-table) [Watkins and Dayan 1992]. Van Hasselt *et al.* [2015] futher extend the DQN by solved an overestimate action values issue suffered for DQN algorithm and leads to much better performance. Mnih *et al.* [2016] introduced an deep reinforcement learning framework that uses asynchronous gradient descent for optimization of deep neural network controllers, that achieve state-of-the-art performance in training time. Fortunato *et al.* [2017] introduce a method called NosiyNet to replacing the conventional exploration heuristics for A3C, DQN, and yield higher scores for a wide range of Atari games.

Domain Adaptation Deep neural networks are able to learn powerful representations from large quantities of labeled input data, however they usually fail to handle when the input distribution have some changes.

Numerous studies on domain adaption methods have been proposed over the recent years [Tzeng et al. 2017; Sun and Saenko 2016; Ganin and Lempitsky 2015; Sun et al. 2015; Xiao et al. 2016]. Instead of collecting labelled data and training a new classifier for every possible scenario, unsupervised domain adaptation methods are try to compensate for the degradation in performance by transferring knowledge from labelled source domains to unlabelled target domains. Sun *et al.* [2015] proposed CORAL method that can align the second-order statistics of the source and target distributions with a linear transformation. Sun and Saenko [2016] extends CORAL to incorporated it into deep networks by define a differentable loss function that can minimize the CORAL loss, it works well for unsupervised domain adaptation and can easily integrate with other deep nural networks. Ganin and Lempitsky [2015] use a domain classifier with a gradient reversal layer that multiplies the gradient by a negative constant, which can ensures that the feature distributions over the two domains are made similar, thus resulting in the domain-invariant features. Tzeng *et al.* [2017] present a method that use the concept of Generative Adversarial Network [Goodfellow et al. 2014], first pre-train a source encoder CNN with labeled data, then perform adversarial adaptation by learning a target encoder CNN such that a discriminator that sees encoded source and target examples cannot reliably predict their domain label, resulting a target encoder CNN that can produce features that similar to source features.

Multi-task Agent Although DRL can suppress human-expert level across many Atari games, but each agent can only play a single game. To solve this issue, some researchers attempt to train a single agent to handle multiple tasks by integrate model compression technique to deep reinforcement learning, Parisotto *et al.* [2015] first present "Actor-Mimic", exploits the use of deep reinforcement learning and model compression techniques to train a single policy network that learns how to act in a set of distinct tasks by using the guidance of several expert teachers. Yin and Pan [2017] propose a new multi-task policy distillation architecture, concatenates a set of task-specific convolutional layers and shared multi-task fully connected layers, the shared multi-task fully connected layers enable the agent to learn a generalized reasoning about when to issue what action under different circumstances. Sharma and Ravindran [2017] propose a simple yet efficient multi-task learning framework which solves multiple goal-directed tasks in an online or active learning setup without the need for expert supervision, solving the needs of training large task-specific teacher (expert) networks.

3 Background: Deep Q Networks

Estimates for the optimal action values can be learned using Q-learning [Watkins and Dayan 1992], but instead of learning all action values in all states, which may be too large to learn in complex environments, we can learn a parameterized value function $Q(s, a; \theta)$, where θ are the parameters of the network. For updating the network parameters after taking action A_t in state S_t and observing the reward R_{t+1} and next state S_{t+1} is:

$$\theta_{t+1} = \theta_t + \alpha(Y_t - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \quad (1)$$

and Y_t is defined as:

$$Y_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \quad (2)$$

where α is learning rate and $\gamma \in [0, 1]$ is a discount factor that trades off the importance of immediate and later rewards.

A most famous deep reinforcement learning algorithm is deep Q network (DQN) that introduced by Mnih et al. [Mnih et al. 2015]. DQN is a multi-layered neural network that for a given state outputs a vector of action values. For an n-dimensional state space and an action space containing m actions, the neural network is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.

There have two important components that used in DQN, a target network and replaying memory. The target network is same as online network, and the parameters of target network $\theta^{'}$ will copy from the online network every t steps, and kept fixed on all other steps. The target network is used to predict the true Q value when learning (Y_t). For the replaying memory, observed transitions are stored for some time and sampled uniformly from this memory buffer to update the network. Both the target network and the replaying memory improve the training stability and overall performance of the algorithm [Mnih et al. 2015]. And to explore environment more stable and faster, DQN also using ϵ-greedy[Sutton and Barto] agent's policy, that agent will select random action with certain possibilities.

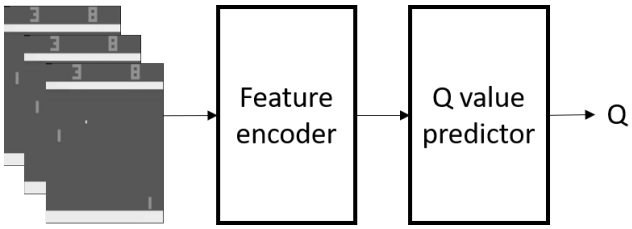

Figure 1: *Agent is divided into two components, a feature encoder and a Q value predictor, agent is able to select action based on the output Q values.*

In this work, we split the agent into two network as shown in Figure 1, a feature encoder F and a Q value predictor V. Feature encoder extract features based on the input observations, and the Q value predictor use these encoded features to predict the corresponding action Q values.

4 Approach

In this section, we present a transfer learning architecture that can speed up learning progress when facing new target environment. Furthermore, we found a novel data augmentation method that can help single task agent to avoid over-fitting and thus learn more core features.

4.1 Transfer with Adversarial Objective

We present a framework for unsupervised adaption between deep reinforcement learning tasks. Assume a source environment Env_s and a target environment Env_t, there have some domain shift between Env_s and Env_t. We have access to the reward and next state after performed an action on Env_s.

The overview is shown in Figure 2, we first pre-trained the source task agent on Env_S, training a feature encoder F_S and Q value predictor V_S that can return a vector of Q values for all possible actions by given features of observation, thus can select best action a. And our goal is to learn a target feature encoder F_T and Q value predictor V_T that can handle Env_T as fast as possible.

We integrate generative adversarial network [Goodfellow et al. 2014] concept into transfer progress, as shown in Figure 2 (b), the target feature encoder F_T plays the role of generator, and a domain

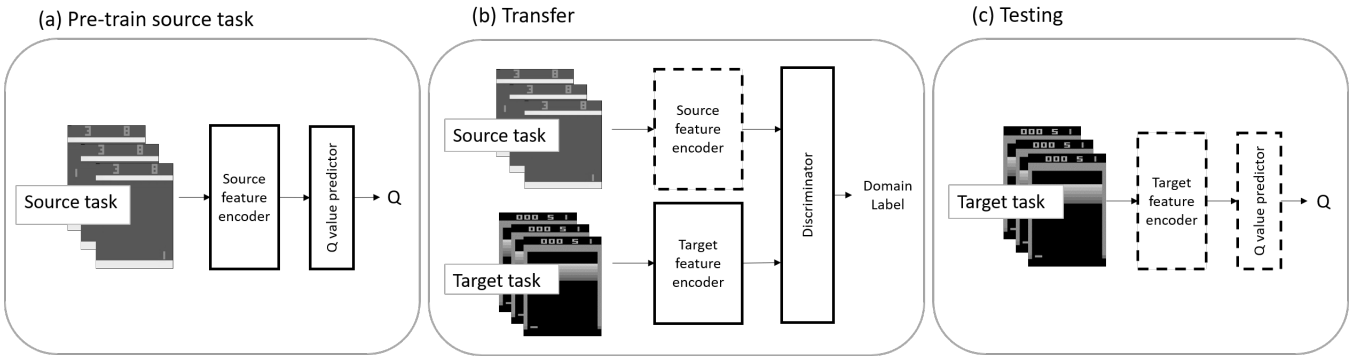

Figure 2: *An overview of our transfer process. (a) We first well-trained an agent on source task environments, then (b) we train a domain classifier and target feature encoder that have adversarial objective to learn a map that maps target feature encoder to source feature encoder. (c) At testing time we can use source task Q value predictor directly because the target feature encoder's output feature are similar to source feature encoder. Dashed line means fixed parameters.*

classifier D that can predict the domain label (source or target domain) by seeing the encoded features output by F_S and F_T. We then perform adversarial adaption by train the F_T to fool the domain classifier, let it cannot easily predict their domain label by sees the encoded source and target features. The target feature encoder (Generator) and domain classifier (Discriminator) are playing counterparts, and at the end of training, the target feature encoder F_T will learn a domain map to the source domain.

In the adversarial objective approach, the main goal is to regularize the learning of the source and target mappings, we can minimize the distance between $F_S(S_s)$ and $F_T(S_t)$ distributions, where S_s and S_t is the states of Env_S and Env_T. If the distribution between $F_S(S_s)$ and $F_T(S_t)$ are similar, then we can directly apply source task Q value predictor V_s to the F_t, skipping the need to learn a V_t and instead setting $V_s = V_t$.

We first describe the domain classifier, D, which classifies whether encoded features are drawn from the source or the target domain. Thus D is optimized according to a standard cross entropy loss, $L_D(S_s, S_t, F_s, F_t)$ where the labels indicate the origin domain, defined below:

$$L_D(S_s, S_t, F_s, F_t) = -\log(D(F_s(S_s))) - \log(1 - D(F_t(S_t))) \tag{3}$$

And for generator, we train with the standard loss function with inverted labels [Goodfellow et al. 2014], then the loss can be describe as:

$$L_G(S_s, S_t, D) = -\log(D(F_t(S_t))) \tag{4}$$

Then, the source and target mappings are optimized according to an adversarial objective, they are optimized to confuse D to unable to predict reliable domain label.

4.2 Augmentation

Most deep reinforcement learning algorithm can achieve great performance in single environment, but agent often can not handle a new environment even with the new environment just slightly different. Some examples are shown at Figure 3, a DQN agent that pretrained on Pong would get very worse performance when add some Gaussian noise or invert the color on Pong.

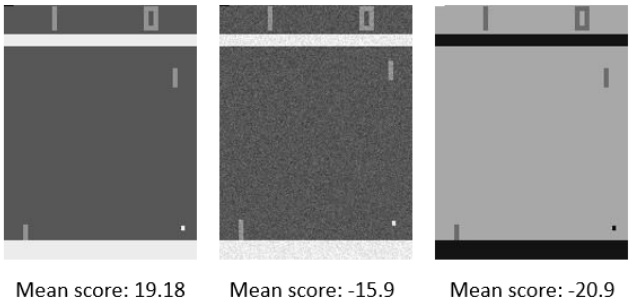

Figure 3: *The agent are trained on Pong for 10 million frames, and test the agent on both Pong (left), Pong with Gaussian noise (middle) and invert color Pong (right). We use $\epsilon = 0.05$ for ϵ-greedy policy and $\sigma = 15$ for noise, the mean scores are average of 10 episodes game play.*

Rusu *et al.* [2016] have analyze the Pong to Pong with noise case and found that the high level filter on the clean task is not sufficiently tolerant to the added noise. We believe that this cause by over-fitting when training on source task, the agent sees too many local features that has little contribution for getting good performance on the source environment, and these local feature are so sensitive that will harm performance when input distribution slightly changed. We found that add some augmentation when training on source would help to avoid over-fitting and thus learn more key features of the tasks. we add a data augmentation layer before feed the input data into replay memory, the data augmentation layer will randomly transform the input. For example, Eq. (5) demonstrates a data augmentation layer that will invert the color of environment state (screen) with the probability of 30%, and remain unchanged otherwise.

$$S = \begin{cases} 1 - S & \text{prop } 0.3 \\ S & \text{otherwise} \end{cases} \tag{5}$$

While this method did increase the difficulty of training the agent, but with proper augmentation setting, like Figure 5(a) shows, the difficulty of training with augmentation are almost same as without it.

Figure 5(b) shows three standard DQN agent training progress on "Pong with Gaussian noise", with different weight initialize, *Pong*

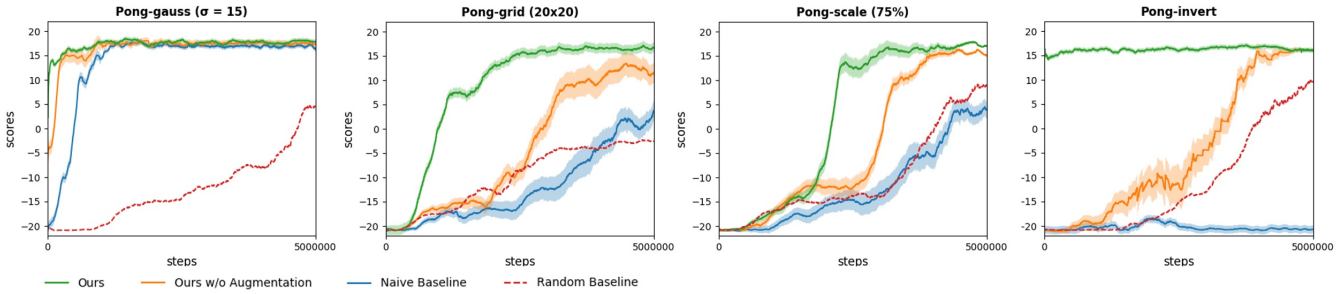

Figure 4: *Transfer progress of Pong variants. Pong variants include noisy, inverted color and scaled transforms. The results are averaged over 3 runs and the shadow represent standard deviation.*

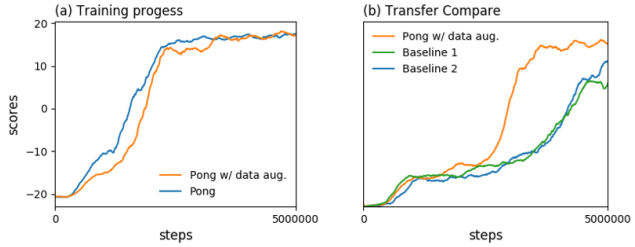

Figure 5: *(a) The training progress of Pong and Pong with augmentation using Eq.*(5) *(b) Training progress on a new environment using different pre-trained model as initial. The environment in (b) is Pong with Gaussian noise and* $\sigma = 50$*. We use constant* $\epsilon = 0.1$ *for* ϵ*-greedy policy.*

with data aug. is using the pretrained model of Pong with augmentation defined as Eq.(5), *Baseline 1* use random initial and *Baseline 2* use pretrained model of Pong. it shows that using pretrained model of Pong with augmentation outperform other baseline on training time, indicate that the knowledge with augmentation is more helpful when facing new similar environment.

5 Experiments

In the following experiments, we evaluate our method by demonstrating its effectiveness at transfer learning in the Arcade Learning Environment (ALE). We first consider synthetic versions of Pong, altered to have some visually difference. Next we experiment on more challenging setting that transfer between different Atari games. Figure 6 shows the selected tasks sample frames on different domain. For our experiments, the augmentation method are use 30% inverted input frame color, and we combine our method with Deep Q network (DQN) that introduced in [Mnih et al. 2015], with hyper-parameters set constant $\epsilon = 0.1$ for ϵ-greedy policy and replay memory size of 100,000.

5.1 Pong Variants

The first evaluation domain is a set of synthetic variants of the Atari game of Pong where the visuals have been altered, thus providing a setting where we can be confident that there are transferable aspects of the tasks. The variants of Pong are Noisy (Gaussian noise is added to the inputs), Grid (fixed grid lines are add on input), Invert (input color is inverted), Scale (input is scaled by 75% and with black padding). Example frames are shown in Figure 6 (a).

Figure 4 shows the transfer progress in each variants, *Random*

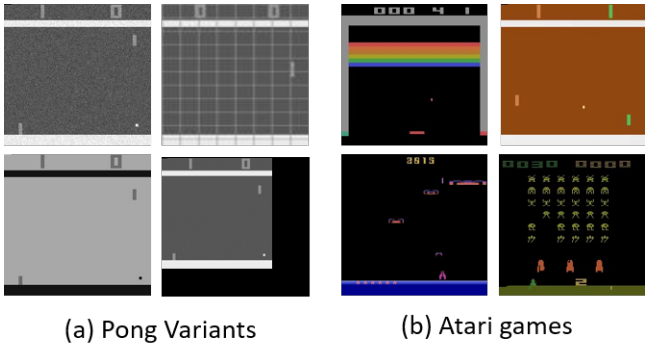

Figure 6: *Example frames from different domain. (a) Pong variants including noisy, recoloured and scale transforms; (b) Atari games offer a more challenging setting for transfer.*

Baseline is random initialized DQN and directly train on target task; *Naive Baseline* is naive transfer approach by initialize the well-trained source task model parameters to target task network, this approach is very common in computer vision domain that use ImageNet [Deng et al. 2009] pre-trained model as new task initial and perform full model fine-tune; *Ours w/o Augmentation* represent using only the method describe in Sec 4.1 to help learning on new environments; and *Ours* indicate using both our method when transferring and source environment pre-trained with augmentation as initialization.

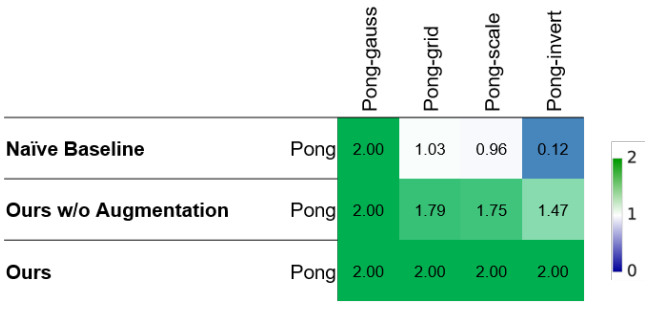

Figure 7: *Transfer score matrix. The higher score means the better transfer performance. Colours indicate transfer scores (clipped at 2).*

For a more clearly look, we measure the transfer performance by measuring area under the learning curve [Hanley and McNeil 1983], and the transfer score is then defined as the area under spec-

ify method divided by the area under *Random Baseline*, in other words the transfer score is the relative performance of an architecture compared with random initialize baseline (*Random Baseline*).

We can make some observations from these results. Overall speaking, as shown in Figure 7, we can see that our method get a better transfer scores in all experiments. Interestingly, the Naive Baseline (initialize with Pong pre-trained) got transfer score are not pretty good, indicate there would have negative transfer effect, especially for **Pong-invert** case, when directly use the Pong pre-trained.

Our method provided an increase in learning speed, both with and without augmentation used are much better than the Navie Baseline approach. For our method with augmentation, the learning speed get a significant improved compared to without it, showing that using source task pre-trained with augmentation provide a better initial guess in parameter space, thus can converge more faster.

We further look close at specified experiment cases, for **Pong-gauss**, the difference between source and target environment is shortest, we can find the prove that even with the Naive Baseline method provide a very good transfer effect, although the transfer scores reached the max (2.0), but using our method still beat baseline on coverage time as shown in Figure 4; And for **Pong-invert** case, it shows that Naive approach fails to learn on target task, means that the source task parameters are not helpful (even be painful) for this task thus have result on transfer score that close to zero, means that in this case it is worse than random baseline, on the other hands, our method without augmentation can minimize the negative effect because the generator will trying to produce features that similar with source task's features, it help speed up learning a better target task feature encoder. And for our method, it achieve great performance at starting time is because that the source task is trained with 30% inverted frame, in other words, the source agent have learn **Pong-invert** at training time, thus it can handle this specified case well. For both **Pong-grid** and **Pong-scale**, the difficulty of transfer is between gauss case and invert case, in general speaking, our method get the best result, following by our method without augmentation, and naive approach get worst results.

5.2 Cross Games Transfer

We next investigate the transfer between different Atari games, we select 4 different games from Atari 2600 to perform cross game transfer experiments, including Pong, Breakout, SpaceInvader and DemonAttack. Pong and Breakout are consider as having some similarity because both gameplay consists of trying to hit a moving ball with a rectangular paddle. And SpaceInvader is consider similar to DemonAttack since they both need to shoot some moving enemies. In this experiments, we perform transfer between these selected games.

Since in reinforcement learning scenario, it is normally able to get reward on target environment, unlike standard transfer learning setting [Sun and Saenko 2016; Tzeng et al. 2017; Ganin and Lempitsky 2015] that use unlabelled data to perform unsupervised learning on target domain, we allow getting reward when training on target environment.

The summary results are reported as transfer scores matrix, shown in Figure 8, in this matrix we compare three different approaches , including naive baseline, ours w/o augmentation and ours method.

Overall, naive approach often got transfer score that smaller than 1, means that it need more training time compare to training on target task directly, thus have negative transfer effect. For ours approaches, the transfer scores is much higher than naive approach, in most of cases, ours approach helps learning faster on target tasks,

Figure 8: *Cross game transfer score matrix. The higher score means the better transfer performance. Colours indicate transfer scores (clipped at 2). The X sign mean source and target task are same and no need to transfer.*

and in some cases although transfer score are not high, but compare to naive approach that causing negative effect, ours method eliminate the negative effect.

For our method with augmentation, the benefit from augmentation are not so significant like our experiments in Pong variants described in Sec. 5.1, in some case like **Pong to Breakout** and **Pong to DemonAttack**, augmentation still helps the model learn faster than without it, and in other cases, the performance with and without augmentation did not shows a noticeable benefit.

Although we selected some similar games (Pong and Breakout, SpaceInvader and DemonAttack), the results shows that the transfer performance are not having obvious improve whether source and target are consider similar, we believe that this is because these Atari games are too different that the some similarity between games could not take advantage when transferring.

6 Conclusion

In this works, we investigate the knowledge transfer for deep reinforcement learning scenario. Unlike previous works [Rusu et al. 2016; Parisotto et al. 2015] that need multiple source task for generalize and transfer to target task, we proposed a method that can accelerate the training progress on new task with a single prior task knowledge, furthermore, we found that using a deadly simple augmentation method can help target task have a better initial guess of model parameters, thus can have some improvement on learning new tasks. And we proved that our apporach outperform baselines in both easy and challenge cases by evaluating on popular benchmark Atari 2600 domain.

Source	Target	Method	Training Steps									
			1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Pong	Breakout	Random	26	**57**	**80**	81	85	83	70	81	83	80
		Naive	13	42	66	79	106	115	114	127	132	130
		Ours	**23**	56	72	**131**	**152**	**186**	**179**	**174**	**181**	**191**
	Space Invader	Random	233	350	501	555	622	650	622	670	650	688
		Naive	234	376	522	578	615	648	630	656	650	588
		Ours	**269**	**405**	**524**	**613**	**623**	**676**	**643**	**708**	**685**	**720**
	Demon Attack	Random	270	1164	1770	1664	1865	2232	2265	2144	2052	2154
		Naive	187	443	843	1109	1039	1035	1099	1030	914	1021
		Ours	**576**	**2333**	**2457**	**2618**	**3477**	**3370**	**3879**	**3655**	**3688**	**3852**
Breakout	Pong	Random	**-15.9**	-8.3	**9.6**	**13.8**	14.2	14.7	14.2	15.2	**15.0**	14.3
		Naive	-20.7	-16.8	-1.3	12.6	14.4	**15.2**	15.3	15.2	14.9	15.2
		Ours	-19.6	**-6.0**	9.0	13.4	**14.9**	14.9	**16.4**	**15.4**	14.1	**15.4**
	Space Invader	Random	233	350	**525**	555	**622**	**650**	622	670	**680**	**688**
		Naive	272	253	452	**645**	551	517	590	572	632	640
		Ours	**301**	**409**	489	588	611	628	**670**	**681**	650	670
	Demon Attack	Random	**270**	1164	1770	1664	1865	2232	2265	2144	2052	2154
		Naive	116	940	1366	980	1158	1222	1150	1269	1302	1252
		Ours	183	**1688**	**2064**	**2614**	**2011**	**2376**	**2269**	**2550**	**2367**	**2262**
Space Invader	Pong	Random	-15.9	**-8.3**	**9.6**	**13.8**	14.2	**15.7**	14.2	**15.2**	15	14.3
		Naive	**-14.5**	-12.2	8.8	12.2	12.3	14.1	13.9	14	14.2	13.3
		Ours	-19	-17.4	0.3	12.5	**15.5**	14.2	**14.5**	15	**15.1**	**16.8**
	Breakout	Random	26	**57**	80	81	85	83	70	81	83	80
		Naive	31	47	71	100	125	126	100	122	125	133
		Ours	12.8	37.6	**87**	**112**	**171**	**168**	**173**	**188**	**191**	**178**
	Demon Attack	Random	**270**	1164	1770	1664	1865	2232	2265	2144	2052	2154
		Naive	220	**1354**	1675	1589	1514	2107	2049	1820	2241	2194
		Ours	173	933	**1902**	**1707**	**1995**	**2255**	**2520**	**2444**	**2387**	**2414**
Demon Attack	Pong	Random	-15.9	**-8.3**	**9.6**	13.8	14.2	15.7	14.2	15.2	15	14.3
		Naive	-16.1	-13.4	0	8.5	10.6	10.7	9.9	9.4	8.1	8.9
		Ours	**-13**	-12.4	1	**15.8**	**15.2**	**17.7**	**16.2**	**17.2**	**17.1**	**16.3**
	Breakout	Random	**26**	57	**80**	81	85	83	**70**	**81**	**83**	80
		Naive	7	31	49	54	54	71	69	80	73	82
		Ours	20	**62**	71	**82**	**95**	**88**	65	**81**	75	**85**
	Space Invader	Random	233	350	525	555	622	**650**	622	**670**	**680**	**688**
		Naive	220	381	502	580	562	574	620	581	618	621
		Ours	**275**	**393**	**540**	**625**	**638**	575	**632**	621	611	680

Table 1: *The full training progress of cross game transfer of 4 different games. "Random" means directly train on target task; "Naive" means naive transfer by initialize the well-trained source task model parameters to target task network; "Ours" means using method describe in Sec 4 to help learning on new environments. We report the reward by average two runs of experiments. For each experiment, the bold text indicate the highest average reward for that particular column.*

References

BELLEMARE, M. G., NADDAF, Y., VENESS, J., AND BOWLING, M. 2013. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR) 47*, 253–279.

DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

FORTUNATO, M., AZAR, M. G., PIOT, B., MENICK, J., OSBAND, I., GRAVES, A., MNIH, V., MUNOS, R., HASSABIS, D., PIETQUIN, O., BLUNDELL, C., AND LEGG, S. 2017. Noisy networks for exploration. *CoRR abs/1706.10295*.

GANIN, Y., AND LEMPITSKY, V. 2015. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, JMLR.org, ICML'15, 1180–1189.

GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2672–2680.

HANLEY, J. A., AND MCNEIL, B. J. 1983. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology 148*, 3, 839–843.

LEVINE, S., FINN, C., DARRELL, T., AND ABBEEL, P. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research 17*, 39, 1–40.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VE-

NESS, 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. *CoRR abs/1602.01783*.

PARISOTTO, E., BA, J. L., AND SALAKHUTDINOV, R. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

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*.

SHARMA, S., AND RAVINDRAN, B. 2017. Online multi-task learning using active sampling. *CoRR abs/1702.06053*.

SUN, B., AND SAENKO, K. 2016. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision–ECCV 2016 Workshops*, Springer, 443–450.

SUN, B., FENG, J., AND SAENKO, K. 2015. Return of frustratingly easy domain adaptation. *CoRR abs/1511.05547*.

SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*, vol. 1.

TZENG, E., HOFFMAN, J., SAENKO, K., AND DARRELL, T. 2017. Adversarial discriminative domain adaptation. *arXiv preprint arXiv:1702.05464*.

VAN HASSELT, H., GUEZ, A., AND SILVER, D. 2015. Deep reinforcement learning with double q-learning. *CoRR abs/1509.06461*.

WATKINS, C. J., AND DAYAN, P. 1992. Q-learning. *Machine learning 8*, 3-4, 279–292.

XIAO, T., LI, H., OUYANG, W., AND WANG, X. 2016. Learning deep feature representations with domain guided dropout for person re-identification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, IEEE, 1249–1258.

YIN, H., AND PAN, S. J. 2017. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *AAAI*, 1640–1646.