CS394R Final Project Report

Jinde Yang, **Shiyu Wu**
Department of Computer Science
University of Texas at Austin
jinde.yang97@gmail.com, swu@cs.utexas.edu
jy23444, sw37625

Abstract—**We utilize and extend the previous work on deep reinforcement learning to teach the agents to play Pong and Tennis Atari games directly from high-dimensional sensory input. We investigate the impact of action space size of Tennis game on the agent training process, in particular, the convergence speed. Furthermore, we propose a image-translation based transfer learning method of training Tennis-playing agents from Pong-playting agents. We achieve a large convergence speed gain by a small amount of transfer learning pretraining. Project Github repository[1] and the project video[2] are available for review.**

I. Introduction

Building autonomous systems and training the agent with a higher level understanding of the visual world is a long-standing challenges and long-term goals in the field of Artificial Intelligence (AI). In the meanwhile, most recent advances in neural networks based deep learning have contributed to several significant breakthroughs in the domains including computer vision, machine translation and time series prediction. These deep learning methods utilize a range of neural network architectures, such as convolutional neural networks, multilayer perceptrons, and are able to extract high-level features from raw sensory data, such as image pixels, text, and etc. More recently, the deep learning methods have been combined with reinforcement learning (RL), enabling it to scale to problems that were previously intractable, such as learning to play video games directly from pixels [1], mastering Go to defeat a human world champion [2].

Deep reinforcement learning, the deep learning based reinforcement learning, demonstrates several advantages over traditional methods in the context of learning to control the agent using the visual inputs. Traditional methods heavily depend on the hand-crafted features, while deep reinforcement learning is able to extract the high-level features with stronger generalizability. On the other hand, the Deep Q-Network algorithm (DQN) proposed in [1] is capable of handling the context in which the rewards are delayed for a long time after the action is executed.

However, the successful applications of deep reinforcement learning are still limited by many factors, among which the data efficiency is one big concern. Recently, transfer learning has drawn a lot of attention in deep learning domain given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained. The proper application of transfer learning could become effective in the domain where data efficiency is the bottleneck, such as some of robotics applications.

In this paper, we utilize and extend the Deep Q-Network (DQN) algorithm [1] to train the agent to play Pong and Tennis games in the simulated Atari environment based on OpenAI Gym framework [3]. Furthermore, we investigate the methods of performing transfer learning from Pong-playing agents to Tennis-playing agents.

We train and evaluate the models based on our proposed approaches. The agents trained with standard DQN algorithms are able to greatly outperform the opponents after a certain number of training steps. Our results also suggests that reducing the action space of Tennis game enables the agent to master the game more quickly. In the end, we show that with transfer learning, only a small amount of pretraining using well-tuned Pong model on Tennis model could give a relatively large improvement in terms of convergence speed.

II. Background

A. Reinforcement Learning

Reinforcement learning (RL), with the scope of machine learning, is a computational method to automatize goal-directed learning through interaction with environment [4]. Specifically, an *agent* learns to act with an *environment*, to maximize a *reward* signal returned by the environment. In general, a reinforcement learning task is formulated as a *Markov Decision Process* (MDP), which is a discrete-time stochastic model for optimizing sequential decision making.

We could define an MDP for RL as a tuple $M = (S, A, p, r, \gamma)$, where S is the set of state s of environment, A is the set of action a of agent, $p(s_{t+1} = s' | s_t = s, a_t = a)$ is the state-transition function depending on environment dynamics, $r(s, a) = \mathbb{E}[r_t | s_t = s, a_t = a]$ is the reward function, and γ is the discount factor. A wide range of reinforcement learning algorithms seek to maximize *return* rather than reward. Return R is usually defined as

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots + \gamma^{T-t} \cdot r_T,$$

where T is the terminal time step in episodic case.

B. Atari Environment

We formulate our target application scenario in which an agent learns to play Pong and Tennis games on Atari 2600

[1]Github repository: https://github.com/mirageyjd/dqn-transfer
[2]Video URL: TODO

platforms as one finite MDP. In particular, the player agent interacts with an environment ξ, which is the Atari environment emulator in our application, with a sequence of observations, rewards when the agent performs a sequence of actions. At each time-step, the agent is able to observe the raw video game frame from the emulator, represented as $x_t \in \mathbb{R}^d$, one flatten vector with each element corresponding to each pixel value for each color channel in the image frame. Notice that the agent is only able to observe the raw output video game frame, while the internal state of the game is non-observable to the agent, which requires agent's inference. At each time-step, given the current observation from the emulator, the agent is able to pick one action a_t from the set of legitimate actions at this time-step, $A_t = \{1, 2, .., K\}$. The selected action a_t will be passed into the Atari game emulator, modifies the internal state of the game, and outputs the new observation x_{t+1}, and corresponding rewards r_{t+1} related to the game score back to the agent.

In general, the rewards r_t and the game score depends not only on the current observation and action, but also on the whole prior sequence. For example, the agent may only get the positive reward of winning 1 point after hitting the ball back and forth several times in the tennis game. Thus, we formulate the sequence $s_t = x_0, a_0, x_1, a_1, ...a_{t-1}, x_t$ as one state in MDP process. The MDP process is finite in the sense that both the Pong and the Tennis will eventually terminate with one player winning the point, while the opponent losing the point. This formulation leads to a sequential decision process in the sense that the agent needs to select proper action to perform given the current state, namely the prior sequence.

Further, we formulate the return of our application as normal discounted return, as many other reinforcement learning problem formulation. The discounted return at time t is defined as $R_t = \sum_{h=t}^{T} \gamma^{h-t} r_h$, where T is the time-step at which the game terminates. Our goal here is to teach the agent learn the best game strategies or policy for maximizing the return R_t given the prior sequence.

C. Deep Reinforcement Learning and DQN

Inspired by the breakthroughs in computer vision and speech recognition based on deep neural networks, Mnih et al. introduced the idea of applying deep neural network to reinforcement learning, called *deep reinforcement learning*, and derive DQN algorithm [1].

DQN is derived from Q-learning [5], an old but simple off-line algorithm. Q-learning only performs update for state-action pair on sampling trajectory, but uses bootstrapping for each update. More specifically, Q-learning samples trajectory by ϵ-greedy policy π based on function $Q(s, a)$, and update Q-value as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Q-learning applies a tabular setting (that is, stores Q-values of all state-value pairs in a table), which is impractical in tasks with large state space, such as playing Atari games with image inputs. DQN extends it by introducing a non-linear approximation of Q function, a deep neural network called *Q-network*. Let $Q(s, a; \theta)$ denote the Q function approximated by Q-network with weights θ. We can train Q function by iteration: optimize it by minimizing the loss function $L_i(\theta_i)$ at each iteration i,

$$L_i(\theta_i) = (y_i - Q(s, a; \theta_i))^2, \tag{1}$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$.

Since y_i varies from iteration to iteration, $Q(s, a)$ is updated towards a moving target. Therefore, DQN introduces *Target network* $\hat{Q}(s, a; \theta^-)$, which is updated every C steps, allowing a stable regression to perform within C steps.

Moreover, DQN utilizes *experience replay* technique [6], which holds a large number of recent transitions (s_t, a_t, r_t, s_{t+1}) in a *replay buffer*. At each iteration, DQN trains Q function with a minibatch randomly sampled from replay buffer. This technique improves the efficiency of updates by reducing the correlation between sampled transition in a minibatch.

With the above techniques, the loss function in Equation 1 can be rewritten as

$$L_i(\theta_i) = \frac{1}{n} \sum_{j=1}^{n} (y_{ij} - Q(s_j, a_j; \theta_i))^2, \tag{2}$$

where $y_{ij} = r_j + \gamma \max_{a'_j} \hat{Q}(s'_j, a'_j; \theta^-)$, and n is the size of a minibatch.

D. Transfer Learning

Transfer learning (TL) is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. More specifically, as explained in [7], "transfer learning and domain adaptation refer to the situation where what has been learned in one setting is exploited to improve generalization in another setting". In other words, transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned, which is usually adopted in problems such as multi-task learning and concept drift.

More specifically, the definition of transfer learning can be formulated in terms of domain and task. As specified in [8], "The domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Given a specific domain, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task consists of 2 components: a label space space $\mathcal{Y}$ and an objective predictive mapping function $f(\cdot)$ (denoted as $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$) learned from the training data which consists of pairs $\{x_i, y_i\}$, where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$." Then, given a source domain $\mathcal{D}_\mathcal{S}$ and learning task $\mathcal{T}_\mathcal{S}$, a target domain $\mathcal{D}_\mathcal{T}$ and learning task $\mathcal{T}_\mathcal{T}$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_\mathcal{T}$ using the knowledge in $\mathcal{D}_\mathcal{S}$ and $\mathcal{T}_\mathcal{S}$, where $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$, or $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$.

There are 3 typical benefits the transfer learning can bring about for the target task:

1) Higher start. The initial skill (before refining the model) on the source model is higher than it otherwise would be.

2) Higher slope. The rate of improvement of skill during training of the source model is steeper than it otherwise would be. In other words, the agent is able to learn faster.
3) Higher asymptote. The converged skill of the trained model is better than it otherwise would be.

Two commonly used transfer learning approaches in deep learning includes i) Pre-trained Model Approach, ii) Develop Model Approach. These two approaches both involve the step of

1) Select source task and model, where a related predictive modeling problem and model is chosen.
2) Reuse Model. The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3) Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest with further training.

The main distinction of two approaches is that Develop Model Approach involes developing source model step after selecting the proper source model. Extra steps need to be done for tailoring the model to be used in the target domain, such as performing dimensionality reduction. In general, Pre-trained Model Approach is more widely used in deep learning domain, and will be our main focus.

III. Related Work

A. Deep Reinforcement Learning

Many of successful applications and scaling of reinforcement learning in recent years were driven by the Deep Q-Network algorithm (DQN) [1]. It combines the traditional Q-learning [4] with convolutional neural networks which has led to great improvements in computer vision, speech recognition,and and etc, and experience relay which eases the training of deep networks for RL and enables the agent to learn to play Atari2600 games from raw image pixels.

Our work is built upon DQN algorithm to train the agent for playing Atari games, and investigate potential methods for performing transfer learning to similar Atari games. We will experiment our algorithms on OpenAi Gym [3], a toolkit generalizing reinforcement learning environment including Atari Learning Environment (ALE) [9] applied in [1].

B. Transfer Learning in Reinforcement Learning

There have been many works managing to apply transfer learning techniques in reinforcement learning tasks, such as [10], [11], [12], [13]. [10] developed the Actor-Mimic method that trains a multi-task network with assistance of expert networks from individual games. For TL, they treat the multi-task network as a DQN, and they transfer all the weights except the final softmax layer to a new DQN. This leads to much faster learning on the target task than random weight initialization. [11] proposed progressive neural network that applies transfer learning without fine tuning on the source network, by using the activation of each layer of the source network as an extra input for the next layer in target network.

TABLE I
SOME DQN HYPER-PARAMETERS

Hyper-parameters	**value**
Observation size	(4, 84, 84)
Gray scale	True
Pixel value normalization	True
Frame skipping	4
No-op max	30
Reward clipping	[-1, 1]
Adam α	6.25×10^{-5}
Adam ϵ	1.5×10^{-4}
Target network update frequency	10K
Replay buffer size	1M
Minibatch size	32
Learning start	50K
Exploration ϵ	$1 \rightarrow 0.1$
Discount factor γ	0.99

Different from the previous two works, [12] and [13] exploited the visual similarities between Atari environment, taking extended approaches by performing image-to-image translation alongside with TL. With the recent advancement of Generative Adversarial Networks (GAN) [14], variants of GAN, such as CoGAN [15] and UNIT GAN [16], perform for high-quality unsupervised image-to-image translation. [13] and [12] both adopted UNIT GAN to overcome visual differences between Atari games, but preprocessed raw image in different ways. Whereas [13] only added a constant noise to raw image, [12] used a thorough preprocessing, extracting key features of games by removing background of image and then turning it into a binary image. Particularly, [12] further investigates the transferring knowledge between completely different games by utilizing i) data transfer by pretraining ii) Continuous data transfer iii) Distillation.

Our work also utilizes UNIT GAN to achieve image-to-image translation between Pong and Tennis, and applies modest preprocessing method: only removes the background of Pong and Tennis. Moreover, we adopt similar transfer learning technique to i) data transfer by pretraining in [12].

IV. Experiment Setting

We proceed with our project in two main phases. In the first phase, we train the agent on Pong and Tennis Atari games using DQN separately. In the second phase, we investigate the possible approach of doing transfer learning from pre-trained Pong-playing agent to Tennis-playing agent. The details of our method and experiment settings are included in the following sections.

A. Training Pong & Tennis Playing Agent with DQN

We follow the general approaches of DQN and experiment settings as discussed in [1] with some preprocessing variations. Some specific experiment settings and preprocessing we make during the experiment are discussed below.

1) Hyper-parameters: The hyper-parameters below are from [1] unless specified. A list of hyper-parameters is provided in Table I.

i) The raw Atari frames, which are RGB images with size 210×160 pixels, are converted to gray-scale and then

TABLE II
NETWORK ARCHITECTURE

Layer	characteristics
Convolution layer 1	32 8×8 filters with stride 4
Convolution layer 2	64 4×4 filters with stride 2
Convolution layer 3	64 3×3 filters with stride 1
Hidden fully-connected layer	512 units
Output fully-connected layer	#action units

compressed to 84 $\times$ 84 pixels by bilinear interpolation. After that, all pixel values are normalized inside the range [0, 1]. Each observation from environment, or the input of Q-network, contains 4 recent frames, and thus the observation is 4 $\times$ 84 $\times$ 84 pixel image. As a result, the size and dimension of observation are significantly decreased.

ii) Frame-skipping technique proposed in [9] is applied to reduce computation. The agent select an action on every fourth frame and repeat it until next selection. This accelerates the learning process without seriously affecting the performance of the agent.
iii) Each game, or episode, starts with a random number of no-ops. The number of no-ops does not exceed 30.
iv) Rewards are clipped to +1 for positive rewards and -1 for negative rewards to reduce the effect of error derivatives.
v) We use the same network architecture as in [17]. A Q-network consists of three convolutional layers and two fully-connected layers (see Table II). Each layer, except the output layer, is activated by ReLU function.
vi) We use Adam optimizer with learning rate $\alpha = 6.25 \times 10^{-5}$ and $\epsilon = 1.5 \times 10^{-4}$ as suggested in [17], rather than RMSProp optimizer.
vii) We utilize a replay buffer of size 1M to break the strong correlation between consecutive transitions, and trains Q-network with a random minibatch of size 32 from replay buffer at each step. Learning is not performed at first 50K step to ensure sufficient transitions in replay buffer.
viii) For behavior policy, we apply ϵ-greedy policy, where ϵ is linearly annealed from 1 to 0.1 at first million steps, and remains 0.1 at rest steps.

2) Changing Sides of Tennis: By nature, the Tennis and Pong games share many similarities with each other. One noticeable difference is that for the tennis game, after one player wins the game, players will change the sides to continue. In order to avoid the confusion caused by changing sides and make it easier for training the agent, we will terminate the game after finishing one game. In other words, whenever one player wins the first game, the game will terminate and restart from the beginning.

3) Action Space of Atari Environment: The Atari environment provided by Gym provides redundant actions which may slow down the learning process of agent. 6 actions are provides in Pong, whereas only 3 actions {NO_OP, LEFT, RIGHT} are valid actions. Hence, we preprocess the environment and provide a smaller action space including only 3 valid actions.

Tennis is much more complex since it involves 18 actions, including NO_OP, FIRE, moving in 8 different directions as {LEFT, RIGHT, UP, DOWN, UPLEFT, UPRIGHT, DOWNLEFT, DOWNRIGHT}, plus move and FIRE in 8 different directions. However, move and FIRE have the same effect as the corresponding move after serving. To reduce action space, we removes 8 move actions and no-op actions, leaving 9 available actions for Tennis-playing agent. Moreover, we further investigate training agent with smaller action space with only {FIRE, UPFIRE, RIGHTFIRE, LEFTFIRE, DOWNFIRE} 5 actions. The comparison is shown in result section.

B. Transfer Learning from Pong-playing Agent to Tennis-playing Agent

We further investigate potential methods of performing transfer learning from Pong-playing agent to Tennis-playing agent. We adopt the Pre-trained Model Approach which is similar to the pretraining method proposed in [12]. In particular, the steps are summarized below.

i) Train the Pong-playing agent using DQN with model M_1
ii) Apply model M_1 on Pong games. Translate the raw image from Pong into Tennis. Feed the translated image into model M_2. Optimize M_2.
iii) Further train model M_2 on Tennis game.

Step ii, or *pretrain step*, is where we adopt transfer learning approach. We use the same framework as in standard DQN, but replace the target network $\hat{Q}$ with pretrained Pong network M_1. In other words, the behavior policy depends on M_1, and the update target is also computed based on M_1. We use a fixed $\epsilon = 0.05$ for exploration during 1M pretrain steps, in order to make full use of pretrained Pong model but keeps a small amount of exploration.

One big step inside the transfer learning is performing the translation between Pong images and Tennis images. In order to achieve the translation, we utilize the UNIT-GAN [16]. UNIT-GAN is one type of generative adversarial networks able to perform unsupervised image-to-image translation. We pretrain the UNIT-GAN with Pong images and Tennis images for the translation. Since UNIT-GAN satisfies the cycle-consistency property [16], the pretrained model is able to perform both Pong to Tennis, and Tennis to Pong translations. However, we will focus on the transfer learning from Pong to Tennis scenario, which only utilizes part of the UNIT-GAN model.

Other details of transfer learning are discussed below.

1) Image Preprocessing: In order to make the UNIT-GAN tranining easier and more efficient, we perform more complex image preprocessing including removing background, and etc:

i) Pre-calculate the background image by calculating the mean pixel values across multiple Pong and Tennis video game frames.
ii) Substract the background image pixel values from the raw image pixels, and rescale the pixel value to be inside the range $[0, 255]$.
iii) Crop the game score.
iv) Rotate Pong image by 90 degrees counter-clockwise such that the player agent controls are at the same side as Tennis.
v) Turn the image into greyscale.

TABLE III
ACTION MAPPING IN TRANSFER LEARNING FROM PONG-PLAYING AGENT TO TENNIS-PLAYING AGENT

Pong action	**Tennis action**
NO_OP	FIRE
RIGHT	LEFTFIRE
LEFT	RIGHTFIRE
NONE	UPFIRE
NONE	DOWNFIRE

vi) Resize the image to be 84×84.
vii) Normalize the pixel values to be inside the range $[0, 1]$ easy for training.

2) Action Mapping: Since Tennis game has a much larger action space than Pong's, we consider a simple action mapping between two games. First, we restrict the action space of Tennis to be just 5 actions, involving {FIRE, UPFIRE, RIGHTFIRE, LEFTFIRE, DOWNFIRE}. Then, we map the action from Pong to Tennis as in Table III. One noticeable setting is that there is no mapping to UPFIRE and DOWNFIRE (that is, these two actions are excluded form pretrain). The other is that we map RIGHT to LEFTFIRE and LEFT to RIGHTFIRE, for the reason that after rotating the Pong image by 90-degrees counter-clockwise, the RIGHT key essentially moves the player to the left side of the screen.

Intuitively, this pretraining using Pong's model teaches the agent how to play the tennis just at the court baseline. Though the agent will never move back and forth, only moving left and right on the baseline can still lead to a well-played agent.

C. Evaluation

We follow the evaluation method in [1]. For every 250K training steps, we evaluate the agent for 125K steps under an ϵ-greedy policy with $\epsilon = 0.05$, by the average reward per episode. Whereas an episode starting at the end of 125K steps may have an impact on the average reward, it turns out that this effect is limited due to i) at the beginning (end) of training process, the agent wins (loses) very fast, resulting in a large number of episode during evaluation; ii) in the middle, the agent and its opponent are neck and neck, which gives an average reward close to 0.

V. RESULT

A. Training Pong & Tennis Playing Agent with DQN

We train our Pong-playing and Tennis-playing agent based on the settings mentioned in previous section, with 10M steps and 25M steps respectively. Fig. 1 demonstrates the learning curve of standard DQN in Pong and Tennis.

Pong has a tiny aciton space and thus the agent converges fast at about 4M steps. The agent reaches a final score of around +18 points, which is slightly lower than the one presented in [1]. The network model of this agent will be used as M_1 in the experiment of transfer learning.

Tennis has a complex action space, and the agent learns much slower even with a compressed action space with 5 actions. The learning is unstable in the middle of training, compared to that of Pong, but converges to a final score of around +3.8 points. Recalling that we have preprocessed the Tennis environment to terminate at the end of first game, the maximum score in a game (episode) is +4. In this sense, our Tennis-playing agent outperforms its opponent and reaches a high performance. This agent will be treated as the baseline of the experiment of transfer learning.

To illustrate the significant improvement of reducing action space, we compare two agents trained with different size of action space: one takes 5 actions, while the other takes 9 actions (see experiment setting section). As shown in Fig. 2, smaller action space requires little exploration, and thus essentially speedup the learning process. Specifically, the performance of agent with 5 actions significantly increases after 9M steps, whereas the performance of agent with 9 actions starts increasing after 15M steps. Moreover, with careful design of reduction, agent with 5 actions achieves the same final performance as the agent with 9 actions. A reasonable explanation is that since the speed of tennis is slower than that of players' movement, the removal of diagonal movement has little impact on players' moving capability. To conclude, our reduction on action space on Tennis successfully speedup the learning without deduction on final performance.

B. Transfer Learning from Pong-playing Agent to Tennis-playing Agent

We use the original author's implementation of UNIT-GAN to train the image translation from Pong to Tennis. We manually prepare the datasets on our own. More specifically, we prepare 3K preprocessed 84×84 images for Pong and Tennis respectively, in which 2.1K images are used as training sets, while the remaining 900 images are used as test/validation sets. Since the resized image is only 84×84, we believe that 3K images are enough to cover different variations of the game settings. We further train the UNIT-GAN on TACC for 450K steps. The result is shown in Fig. 3.

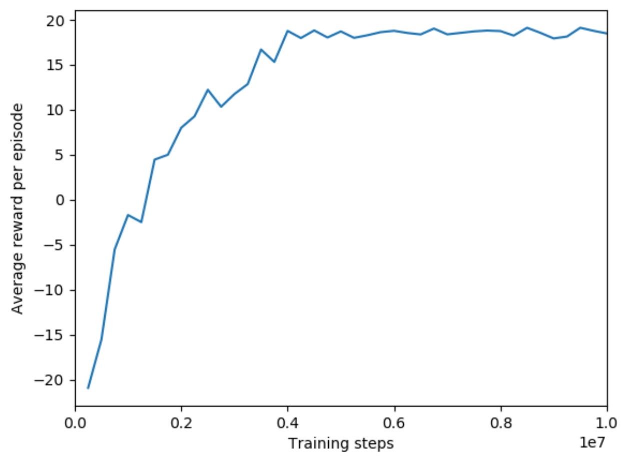
Though the UNIT-GAN may not generate very meaningful images in some cases, in most cases, the generated Tennis images are relatively decent, reflecting the relative positions of players and the ball. We consider this to be enough for doing transfer learning.

Then, we train the Pong-playing agent model M_1 using the DQN algorithm with the same settings as the previous section for 10M steps.

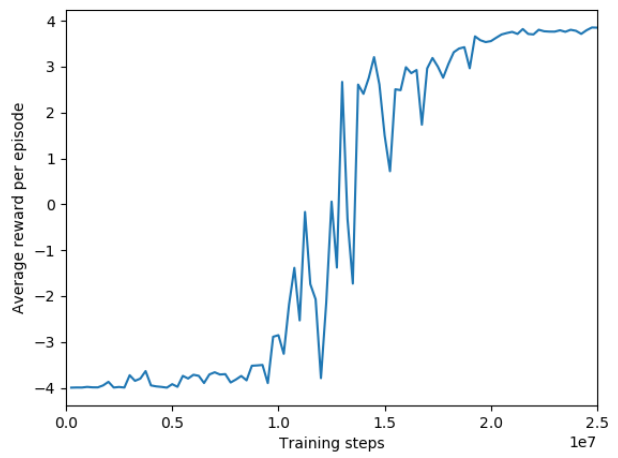
Further, we pretrain the Tennis-playing agent model M_2 by applying model M_1 on Pong, translate the raw image from Pong into Tennis, and optimize M_2 for 1M steps.

Finally, we train the Tennis-playing agent model M_2 on Tennis for another 15M steps using DQN algorithm with $\epsilon = 1$. In other words, we treat M_2 as a brand new model for training.

Fig. 4 shows the comparison between the transfer DQN and standard DQN as the change of average reward per episode with respect to the training steps. We can see that both methods will eventually converge to around $+3.8$ points per episode, meaning that the agent greatly outperforms the opponent given the maximum reward is $+4$. However, it is notable that the



(a)



(b)

Fig. 1. The learning curve of standard DQN in (a) Pong with 3 actions, and (b) Tennis with 5 actions.

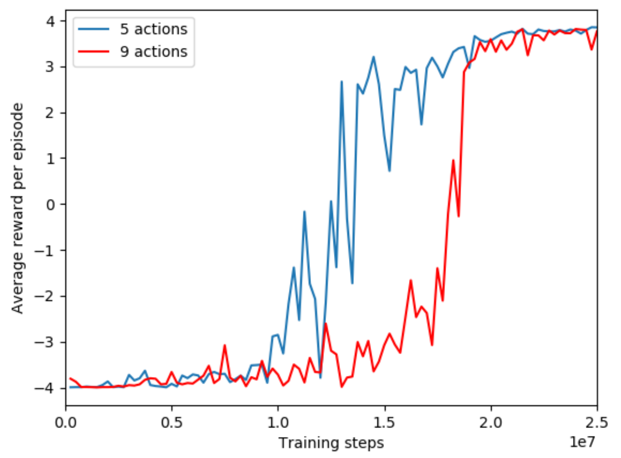
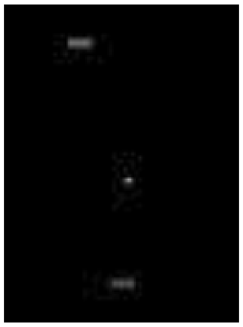


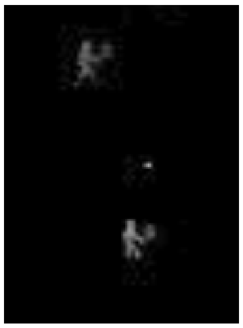
Fig. 2. Comparison of Tennis-playing agent trained with 5 actions and 9 actions

transfer DQN reaches the convergence at a much faster pace around 12M steps, saving around 8M steps compared with standard DQN, especially given the fact that we only pretrain the transfer DQN for only 1M steps. Only a small number of pretraining using the transfer learning on Tennis model could give a relatively large improvement in terms of convergence speed.

It is also worth noticing that there is a sequence of ripples in the initial stage of transfer DQN, which could be the joint effect of ϵ-greedy method and pretraining. Through the pretraining, the agent learns FIRE, LEFTFIRE, RIGHTFIRE very well, while almost never learns UPFIRE, DOWNFIRE. With large ϵ value in the beginning, the agent may sometimes randomly pick the action leading to poor performance, but may also exploit {FIRE, LEFTFIRE, RIGHTFIRE}. But whenever the agent learns {UPFIRE, DOWNFIRE}, the performance



(a)



(b)

Fig. 3. (a) Preprocessed pong images. Only relative positions of players and ball are kept. (b) Translated tennis image from UNIT-GAN based on (a)

will steadily and rapidly improve with a much smoother curve as shown in Fig. 4.

VI. Discussion

A. Training Pong & Tennis Playing Agent with DQN

i) Whereas our Tennis-playing agents achieves an expected high converged performance, they oscillate sharply in the middle of training process, in both 5-action case and 9-action case. More importantly, we observe that the level of oscillation in 5-action case is much higher than that in 9-action case. The oscillation may be caused by several possible reasons: nature of complex Tennis environment; stochastic factor from ϵ-greedy policy; removal of diagonal movements. Our current experiment cannot lead to any conclusion on the reason of oscillation. To figure it out, we may repeat the same experiments several times, and present the results by average of results from repeated experiments.

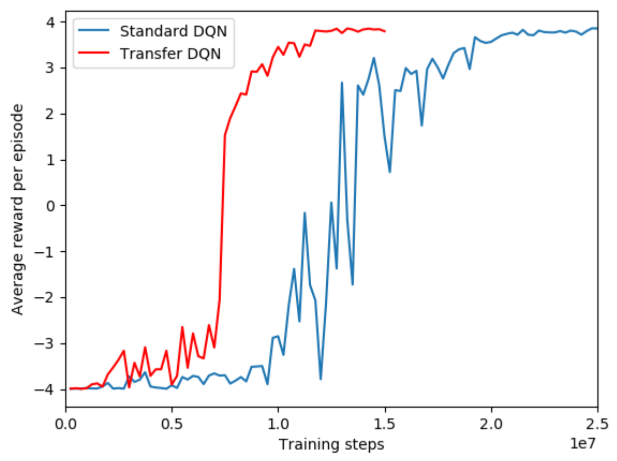


Fig. 4. Comparison of Tennis-playing agent trained with 5 actions between standard DQN and transfer DQN

B. Transfer Learning from Pong-playing Agent to Tennis-playing Agent

i) We believe that by nature, Pong game is simpler than Tennis game mostly in the sense that the action space of Pong is much smaller. We decide to proceed the transfer learning from simple game to more complex game. However, it remains an interesting topic to explore the difference between transfer learning from Pong to Tennis and from Tennis to Pong.

ii) In the final training of Tennis model M_2, we set $\epsilon = 1$ in the beginning to force the agent to widely explore the actions, particularly to learn {UPFIRE, DOWNFIRE}. It remains a question whether we could use smaller ϵ for faster convergence speed.

iii) Currently, we use very intuitive action mapping from Pong to Tennis in terms of moving left, right, and firing. More interesting action mapping is worth exploring.

VII. Conclusion

In this paper, we utilize and extend the Deep Q-Network(DQN) algorithm to train the agent to play Pong and Tennis games in the simulated Atari environment based on OpenAI Gym framework. We adopt the general settings in [1], but train the agent in Pong and Tennis with less training steps and smaller action space. In particular, we propose a action space of size 5 compared to the original one of size 18 in Tennis game, which significantly speedup the learning process without reducing the converged performance.

Furthermore, we propose a image-to-image-translation based transfer learning method of training Tennis-playing agents from Pong-playing agents. More specifically, we pre-train Tennis-playing agent under Pong environment, with the assistance of our expert Pong-playing agent and image-to-image-translation network UNIT GAN. Our results show that the transfer learning enables the Tennis-playing agents to master the game at a faster pace with a few pretrain steps.

Due to limited time and resource, We are not able to present statistical results, as well as further investigate image preprocessing and pretrain technique of transfer learning. Moreover, we have not extended our approach of transfer learning in other Atari games. Those would be the possible focus of our future work.

VIII. Acknowledgement

We are greatly thankful to Prof. Scott Niekum, Prof. Peter Stone, and all the teaching assistants for all the insightful feedback of the project and providing the support of computing resources on TACC. We can not make it without the help from all of you.

IX. Code Usage & Credit

We implemented the baseline DQN algorithm for training Pong and Tennis playing agent, the image preprocessing, and all the transfer learning algorithm in Pytorch on our own.

We manually prepared the datasets for training UNIT-GAN for image translation. The source code of UNIT-GAN was adopted from the original author's implementation at https://github.com/mingyuliutw/UNIT.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[6] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[8] Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in human neuroscience*, 11:334, 2017.

[9] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[10] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[11] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[12] Doron Sobol, Lior Wolf, and Yaniv Taigman. Visual analogies between atari games for studying transfer learning in rl. *arXiv preprint arXiv:1807.11074*, 2018.

[13] Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. *arXiv preprint arXiv:1806.07377*, 2018.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477, 2016.

[16] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems*, pages 700–708, 2017.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.