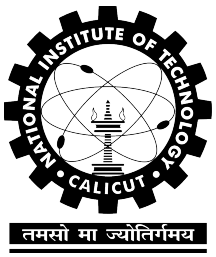
Transfer Learning: Deep Reinforcement Learning with Atari Games

CS4090 Project Final Report

Submitted by

Gazala Muhamed	**B150028CS**
Kalyanam Srikanth	**B150531CS**
Pala Tanusha Durga	**B150603CS**

Under the Guidance of
Dr. Saleena N

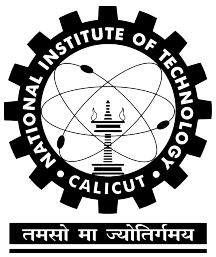


Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

April 24, 2019

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT, KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



2019

CERTIFICATE

Certified that this is a bonafide record of the project work titled

TRANSFER LEARNING: DEEP REINFORCEMENT LEARNING WITH ATARI GAMES

done by

Gazala Muhamed
Kalyanam Srikanth
Pala Tanusha Durga

of eighth semester B. Tech in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology Calicut

Project Guide
Dr. Saleena N
Associate Professor

Head of Department
Dr. Saleena N
Associate Professor

DECLARATION

We hereby declare that the project titled, **Transfer Learning: Deep Reinforcement Learning with Atari Games**, is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Place : NIT Calicut
Date : 24-04-2019

Signature :
Name : Gazala Muhamed
Reg. No. : B150028CS

Signature :
Name : Kalyanam Srikanth
Reg. No : B150531CS

Signature :
Name : Pala Tanusha Durga
Reg. No : B150603CS

Abstract

Transfer learning can improve the ability of an agent to act in multiple environments and transfer previous knowledge to new situations. A deep reinforcement learning algorithm typically only does a single task. Even slightly modified tasks require re-training from scratch. The conditions under which the application of Transfer Learning to the Deep Reinforcement Learning domain can improve the learning of a new task are evaluated. The common baseline for such algorithms, Atari games, is used.

ACKNOWLEDGEMENT

We would sincerely like to thank our guide, Dr. Saleena N, who took the time and effort to guide us through this project. We also appreciate the guidance given by Dr. Saidalavi Kalady and Dr. Jay Prakash who evaluated the project periodically and gave valuable comments and suggestions. In addition, we thank our wonderful parents who supported us, as well as our fellow classmates who helped us in times of need.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to generalize across similar problems is arguably the most important part of human intelligence. The notion was originally introduced as "transfer of practice" in 1901 by Edward Thorndike and Robert S. Woodworth[1]. This quality of learning between tasks is the basis for transfer learning. Deep neural networks trained on similar datasets show similarities on the first few layers[2], indicating that transfer learning is applicable for deep learning.

1.1 Deep Reinforcement Learning

The main difference between Deep Learning and traditional Machine Learning algorithms is that feature extraction is done automatically in Deep Learning. Deep Reinforcement Learning is a branch of machine learning where the neural net is trained by awarding 'rewards' for making the correct 'actions' i.e. reinforcing the right behaviour.

Since each task typically has different actions and rewards, current Deep Reinforcement Learning (DRL) architectures get less effective, given more tasks. DRL algorithms suffer 'catastrophic forgetting', that is when previously unseen information is fed into the neural net, it 'forgets' the previously

learned behavior. Another issue is state aliasing, where different frames are mapped as the same state. They appear similar but require different responses.

1.2 Transfer Learning

Transfer learning is defined as a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem [3]. Under changes in conditions like distribution and feature space, most statistical models need to be rebuilt from scratch by collecting new training data. Thus transfer learning is increasingly important in real life applications where it may be expensive or even impossible to collect complete training data.

To define it formally, we need a few notations:

1. A domain, denoted by $\mathcal{D} = \{\chi, P(X)\}$, where χ is the feature space, and $P(X)$ is the edge probability distribution where $X = \{x_1, ..., x_n\} \in \chi$.

2. A task is denoted by $\mathcal{T} = \{y, f(x)\}$, where y is the label space and $f(x)$ is the target prediction function, or conditional probability function $P(y|x)$.

(Transfer Learning) [4]. Given a target task $\mathcal{T}_t$ based on a target domain $\mathcal{D}_t$, we can use a source domain $\mathcal{D}_s$ to learn a source task $\mathcal{T}_s$. Transfer learning aims to improve the performance of predictive function $f_{\mathcal{T}}(\cdot)$ for target task $\mathcal{T}_t$ by discovering and transferring latent knowledge from $\mathcal{D}_s$ and $\mathcal{T}_s$, where $\mathcal{D}_s \neq \mathcal{D}_t$ and/or $\mathcal{T}_s \neq \mathcal{T}_t$. In addition, in most cases, the size of $\mathcal{D}_s$ is much larger than the size of $\mathcal{D}_t$.

Transfer learning usually provides a higher start, a steeper slope and reaches a higher asymptote as compared to normal algorithms [5], except in cases of 'negative transfer'.

Chapter 2

Literature Survey

In the past few years, deep reinforcement learning has shown great results for problems with complex states such as board games like *Go*, or video games like *Breakout*, that can be used without vast expert knowledge. Similar to *ImageNet*, the image database that led to huge strides in the machine learning field of image classification, the Atari games are the standard baseline to test Deep Reinforcement Learning.

2.1 Q-Learning

Watkin's Q-learning [6] is considered one of the breakthroughs in reinforcement learning algorithms. Each possible action for each possible state has its Q value, where Q stands for a quality of a given move. For each state experienced by the agent, the state, action and reward are remembered and an experience replay is performed. Q value is updated with the cumulative discounted future rewards. The Q-function is defined as follows:

$$Q(s_t, a) = r_t + \gamma * max_a Q(s_{t+1}, a) \tag{2.1}$$

where r_t is the current reward, γ is the discount factor, s_t is the current state, s_{t+1} is the next state, a is an action, and $max_a Q(s_{t+1}, a)$ is the estimate of the optimal future value of all possible actions.

2.2 DQN

Using Q-learning and neural networks, Mnih, V. et al. [7] from Google's DeepMind presented the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning - DQN. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards.

DQN uses the Q-function as well as experience replay, which prevents the network from diverging. Experience replay is a biologically inspired process that uniformly (to reduce correlation between subsequent actions) samples experiences from the memory and for each entry updates its Q value.

$$Q(s_t, a) = (1 - \alpha) * Q(s_t, a) + \alpha * (r_t + \gamma * max_a Q(s_t + 1, a)) \tag{2.2}$$

where α is the learning rate of the experience replay.

2.3 Double DQN

Van Hasselt et al. [8] show that DQN overestimates action values that may be due to statistical error. Regular DQN is not guaranteed to converge since it tends to overestimate Q-values of potential actions in a given state. Once one specific action becomes overestimated, its more likely to be chosen in the next iteration making it very hard for the agent to explore the environment uniformly and find the right policy.

Double DQN aims to avoid this by estimating the calculation of the target Q-values of the next states. The action choice is decoupled from the target Q-value generation. Two separate networks will be used: the primary network to select an action and a target network to generate a Q-value for that action.

We now have two Q-functions: Q_a and Q_b. Q_a gets updated using Q_b for the next states. The update consists of finding the action a^* that maximises Q_a in the next state ($Q_a(s', a^*) = MaxQ_a(s', a)$), then use a^* to get the value of $Q_b(s', a^*)$ in order to update $Q_a(s, a)$.

The paper proves that $E(Q_b(s, a^*)) \leq MaxQ_a(s, a^*)$. So over enough number of experiments the expected value of $Q_b(s, a^*)$ is always less than or equals $MaxQ_a(s, a^*)$, which means that $Q_a(s, a)$ is not updated with a maximum value.

$$Q_a(s_t, a) = r_t + \gamma * Q_a(s_{t+1}, argmax(Q_b(s_{t+1}, a)) \tag{2.3}$$

where $argmax(Q_b(s_{t+1}, a)$ returns the action that maximizes the Q-value of the second Q-function, Q_b.

2.4 A3C

Mnih, V. et al [9] later presented A3C (Asynchronous Advantage Actor Critic) that uses asynchronous gradient descent for optimization of deep neural network controller. It outperformed DQN on many Atari games.

A3C runs many workers in parallel. These workers asynchronously push their gradient updates to a central "target network". This allows a greater exploration to occur over the search space, since different agents will likely experience different states and transitions.

2.5 Transfer Learning

Hsu et al. [10] discusses a semi-supervised approach using Transfer Learning which uses a single source task and claims to outperform common baselines after fine-tuning. They test a configurable Atari game *Breakout* to evaluate multi-level learning.

Chapter 3

Problem Definition

The problem is to identify a general model for a deep reinforcement agent that should be able to play different modifications of a game using transfer learning. In transfer learning, we first train a base network on a base domain and task, and then we re-purpose the learned features, or transfer them, to a second target network to be trained on a target domain and task. In order to test various modifications, a configurable game API is required.

Chapter 4

Design

In our project we aim to provide a direct comparison between the popular algorithms mentioned in the literature survey: DQN, Double DQN and A3C. We can then infer which algorithm provides a model that works across domains and tasks, by training each model an equivalent number of frames, and then testing on different targets.

4.0.1 Input specification

The input to the program is the game API written in Python, that provides each frame of the game as a state. We use the *Tensorflow* and *Keras* libraries to implement the deep learning models for each algorithm.

4.0.2 Output specification

The output is a trained model that can play that game by performing actions on each state. This can be saved as a *.tar* file readable by *Tensorflow*.

4.0.3 Project Outline

The project consists of the following parts:

1. Creating the configurable game API for the Atari games *Breakout* and *Pong*
2. Training a few state-of-the-art DRL algorithms on the game on a fixed configuration
3. Testing each trained model on a different configuration (cross-task transfer learning), or a different game (cross-domain transfer learning)
4. Comparing the performance of each algorithm for transfer learning
5. Adding improvements to the model by introducing negative rewards as well as positive rewards

Chapter 5

Implementation

5.1 Training Data

In Deep Reinforcement Learning Algorithms, rather than a labelled dataset, the dataset is dynamically produced while the agent makes choices or 'actions'. The input to the program is the game API, that provides each frame of the game as a state. Most of the referenced papers use a licensed version of the Atari games, which is not configurable and requires MuJoCo (multi-joint dynamics in contact) physics simulator, which is proprietary and requires binaries and a license.

We created an open source, configurable implementation of the Atari game *Pong*[11], which can be used without proprietary software. The original game *Breakout* only has one paddle, $paddle_1$. We implemented a second paddle, $paddle_2$, whose movement is hard-coded into the game. The motion is defined to follow the current position of the ball, which does not take into account the bouncing of the ball from walls. The first paddle, $paddle_1$ will be controlled by the DRL agent. Additionally, we made modifications to both *Pong* and *Breakout* such that, depending on the parameters passed to the API, the frames returned may be color inverted or rotated. This is done by manipulating the rendering of the arrays that represent frames. This allows

us to test cross-task transfer learning.

The *Pong* API returns the current state as a frame. Each frame in the game is represented by a 210x160 matrix. Each value in the matrix is an integer between 0-255, indicating the grayscale value of each respective pixel. Each game object (ball, paddle, etc) is implemented by creating a bounding box for collision detection. We used *OpenCV* to display the pixels as the model is being tested to visualize the game.

We aim to improve the performance of each of the algorithms (DQN, Double DQN and A3C) by introducing a 'negative reward'. When the agent loses a life, we penalize the agent with a -1 reward. This should ideally teach the agent not to repeat those actions that result in a negative reward, while reinforcing those actions that result in positive rewards.

Chapter 6

Experimental Results

We conducted the following experiments to test for cross-domain and cross-task transfer learning. For each algorithm (DQN, Double DQN and A3C), the source model was trained for 200,000 frames before testing on the target.

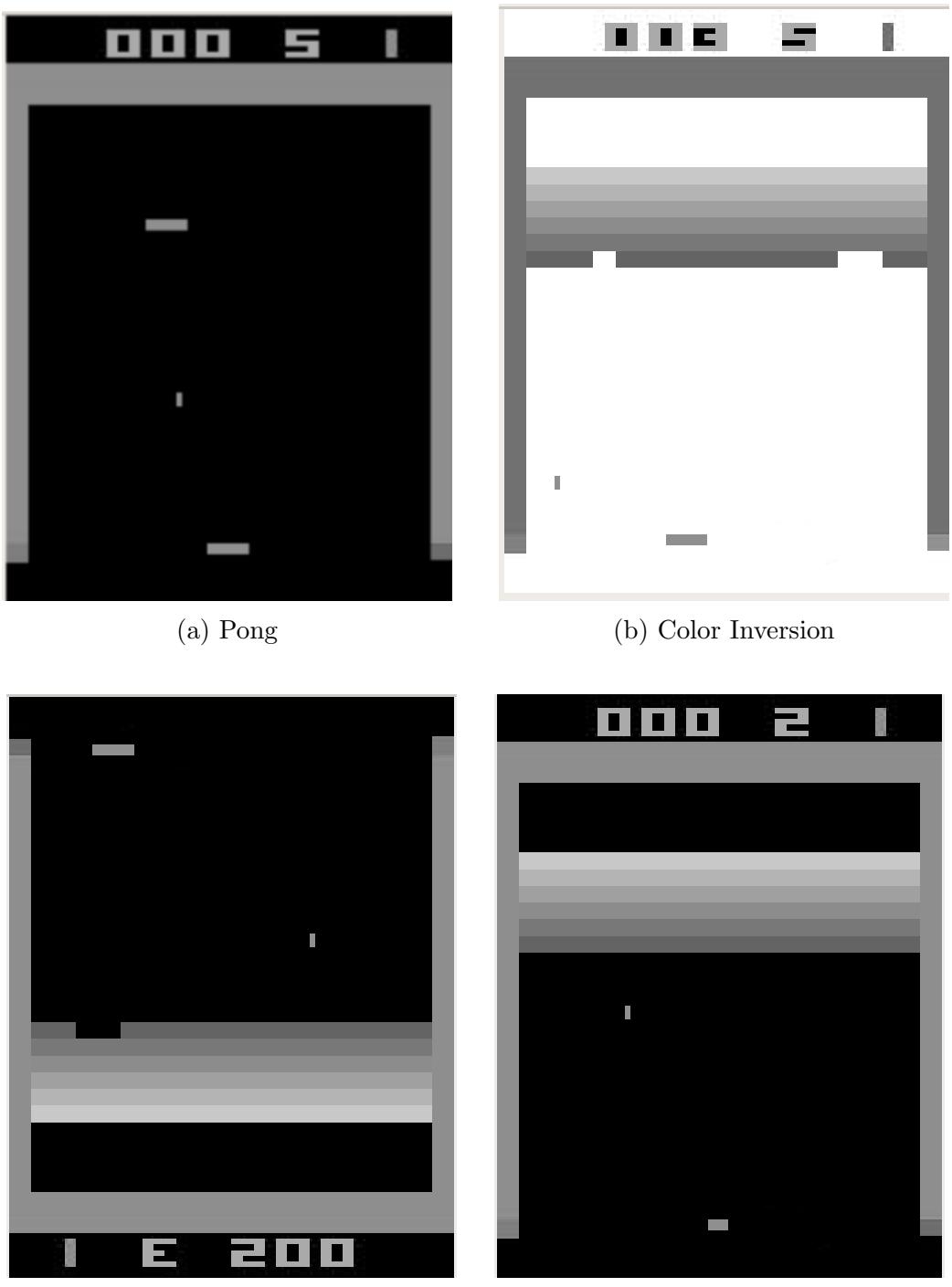
A sample frame from each experiment's target is shown in Figure 6.1. We show the resulting rewards in the tables 6.1, 6.2, 6.3 and 6.4.

6.1 Experiment 1

Our first experiment was for cross-domain transfer learning between two Atari games *Pong* and *Breakout*.

Source Domain: *Breakout* The source domain consists of 160x210 matrices representing frames of the game. The difference from the target domain is that there is no second moving paddle. Instead, the agent has to hit the line of bricks to increment score.

The task is to hit all the bricks with the ball to increase points. The action space is similar to *Pong*: LEFT, RIGHT, FIRE, NOOP.



(a) Pong

(b) Color Inversion

(c) Screen Rotation

(d) Paddle size decrease

Figure 6.1: Visualisation of targets

Target Domain: *Pong* The basic gameplay is as follows: There are two agents, each controlling the movement of their paddle. The **source task** is the choice of action from the action space LEFT, RIGHT, FIRE, NOOP such that the score is maximized. The agent controls the bottom paddle while the movement of top paddle is a simple hard-coded function that follows the current location of the ball.

The agent scores a point when the ball hits the opposite edge of the frame, without being blocked by the other paddle. If the agent misses the ball, they lose a life. The game ends when the number of lives reaches 0.

Here, hitting the opposite edge of the frame does not increase the score. This is the one of the biggest changes across the domains, which may cause the learned model to show 'negative transfer'.

6.2 Experiment 2

In the next few experiments, we test cross-task transfer learning. We expect better results since the domain is the same between the two tasks : The *Breakout* API.

Source Task: Default Configurations The source task was chosen to be the default configurations for the game *Breakout*, which has a black background.

Target Task The target task was chosen to test whether the model was correctly learning based on the colors of the paddle and ball, irrespective of features that do not influence gameplay, like the background color. We inverted the colours of frame's background to be white, as seen in Figure 6.1b

6.3 Experiment 3

Source Task: Default Configurations The source task was chosen to be the default configurations for the game, which has the paddle at the bottom of the frames.

Target Task The target task was chosen to test if the model learned based on the relative positions of the paddle and ball. The absolute positions should not influence the gameplay. The input frames were rotated 180 degrees, as seen in Figure 6.1c

6.4 Experiment 4

Source Task: Default Configurations The source task was chosen to be the default configurations of for the paddle, which is 15 px.

Target Task The target task was chosen to test whether the model was correctly learning the expected task: to hit the ball with the paddle. We decreased the paddle size from 15 px to 7 px, as seen in Figure 6.1d. Since there is less surface area for the ball to hit, it is more difficult for the agent to position the paddle correctly in time.

6.5 Results

DQN The results show that DQN only performed half as well in the cross-domain transfer. The score reduced from 1.16 to 0.83. This could be an indication that the DQN model converged too early to a local optimum, and was not able to learn the given task effectively enough to transfer the knowledge across domains.

Table 6.1: Results of Experiment 1

	Mean Scores over Six Games		
	DQN	DDQN	A3C
On Source Task	1.16	2.33	2.53
On Target Task	0.83	2.83	2.79

Table 6.2: Results of Experiment 2

	Mean Scores over Six Games		
	DQN	DDQN	A3C
On Source Task	1.16	2.33	2.53
On Target Task	1.67	5.17	3.05

Table 6.3: Results of Experiment 3

	Mean Scores over Six Games		
	DQN	DDQN	A3C
On Source Task	1.16	2.33	2.53
On Target Task	1.5	1.5	2.14

Table 6.4: Results of Experiment 4

	Mean Scores over Six Games		
	DQN	DDQN	A3C
On Source Task	1.16	2.33	2.53
On Target Task	0.33	1.33	1.97

As for the cross-task transfer experiments, DQN was able to show positive transfer in the case of color inversion and screen rotation. This shows that the model was able to effectively learn the paddle-ball interaction.

But the results of experiment 4 show that the movement of the paddle is not optimum since a decrease in paddle size dramatically reduced the score from 1.16 to 0.33.

We can conclude that DQN was not able to effectively transfer the latent knowledge from a source to a target, making it ineffectual for transfer learning.

Double DQN Double DQN is expected to perform better than DQN by avoiding over-estimating Q-values. The results varied based on the difference in the state of the target task from the source task. In experiment 1, the cross-domain transfer to *Pong* (score 2.83) had better performance than *Breakout* (score 2.33).

Experiment 2 (color inversion) showed significant improvements (doubled from 2.33 to 5.17), which may indicate that the edge-detection was able to run more effectively on the white background with higher contrast against the ball and paddle.

In experiment 3, the model showed a lesser score on the target task (1.5) than on the source task (2.33). Thus the screen rotation significantly impacted the model's ability to perform. In this experiment, Double DQN's performance is comparable to DQN. In experiment 4, the model was able to outperform DQN's score.

A3C The agent was able to retain the learning quite well, with an average reward of 2.53 (over DQN's 1.16 and Double DQN's 2.33). This shows us that the initial model 'learned' the expected task correctly, that is, to hit the ball with the paddle.

Experiment 1 shows that positive cross-domain transfer learning was possible (from 2.53 to 2.79). Experiment 2 also shows improvement, though not

as much as Double DQN.

Experiment 2 and 3 showed a slight decrease in performance on screen rotation and paddle size decrease. But compared to DQN and Double DQN, the A3C model was able to retain more latent knowledge across tasks.

Final Results We can conclude that for transfer learning, A3C outperforms DQN and Double DQN in both cross-domain and cross-task transfers. Surprisingly, Double DQN showed better performance than A3C in experiment 2 (color inversion), which may be indicate that edge detection is better in Double DQN.

The training loss of A3C and DDQN is shown in Firgure 6.2, plotting the loss per episode of the game. We see that DDQN quickly tries to reduce it's loss value, and has greater variation over time. Whereas in A3C, it is slower to converge, and periodically finds new optimum loss values. This results in a smoother graph, and reduces over-estimation of Q-values.

One more interesting result comes up when we view the frames: the game prioritizes a technique called 'tunneling' as seen in Figure 6.3. The ball is repeatedly aimed at a particular spot on the brick layer, which creates a tunnel. This allows the ball to bounce between the top edge of the frame and the bricks, which accumulates a lot of rewards very quickly. This fact is exploited by the agent, which causes the higher scores in this domain, *Breakout*, that cannot be transferred to other domains like *Pong*.

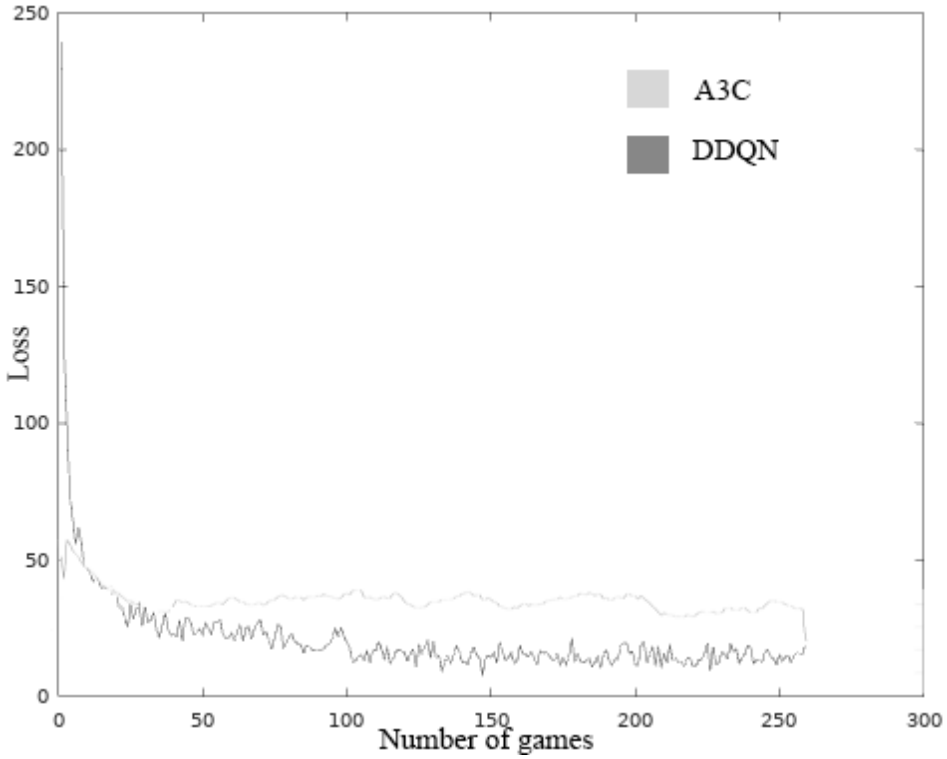


Figure 6.2: Training Loss of A3C and DDQN

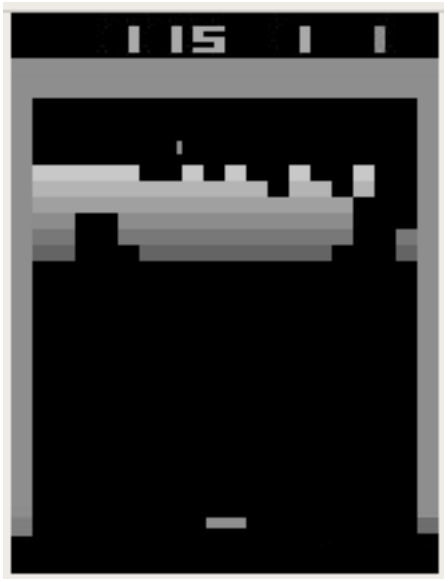


Figure 6.3: Tunneling

Chapter 7

Conclusion and Future work

By implementing a configurable game API for *Breakout* and *Pong*, we were able to successfully compare the performance in the ability of each algorithm to transfer across tasks and domains. The results across each algorithm (DQN, Double DQN, A3C) showed us that A3C could provide a good general model for Atari games. The reason could be because the model learned techniques like tunneling that are specific to the domain, that result in better performance across tasks.

Research in the Deep Learning field is ongoing, and newer models could outperform A3C. Espeholt's 2018 paper [12] describes a natural progression of A3C, IMPALA, which claims to be scalable and have positive transfer between tasks as a result of its multi-task approach. As for future work, it is possible to conduct further tests in both cross-task and cross-domain experiments, as well as newer algorithms such as IMPALA.

References

[1] R. S. Woodworth and E. Thorndike, "The influence of improvement in one mental function upon the efficiency of other functions.(i).," *Psychological review*, vol. 8, no. 3, p. 247, 1901.

[2] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[3] J. West, D. Ventura, and S. Warnick, "Spring research presentation: A theoretical foundation for inductive transfer," *Brigham Young University, College of Physical and Mathematical Sciences*, vol. 1, p. 32, 2007.

[4] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International Conference on Artificial Neural Networks*, pp. 270–279, Springer, 2018.

[5] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez, "Handbook of research on machine learning applications and trends: Algorithms, methods and techniques-2 volumes," 2009.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[10] S.-H. Hsu, I. Shen, B.-Y. Chen, *et al.*, "Transferring deep reinforcement learning with adversarial objective and augmentation," *arXiv preprint arXiv:1809.00770*, 2018.

[11] G. Muhamed, "Configurable breakout api." https://github.com/gazlaws-dev/baby-a3c, 2018. Online; last accessed 22-April-2019.

[12] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[13] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," *GitHub, GitHub repository*, 2017.