Transfer learning between Atari games

Arpit Gupta	Ajinkya Rawankar	Amrit Kataria	Pranav Verma	Suraj Garg
2018201048	*2018201020*	*2018201067*	*2018201047*	*2018202003*
M.Tech(CSE) 2nd year	*M.Tech(CSE) 2nd year*	*M.Tech(CSE) 2nd year*	*M.Tech(CSE) 2nd year*	*M.Tech(CSIS) 2nd year*

***Abstract*—In this project, we use Deep Q learning to train two similar games Atari Pong and Atari Tennis, and verify if we can transfer the knowledge from one game to another and vice-versa.**

***Index Terms*—DQN, Atari Games, Reinforcement Learning, Transfer learning**

I. Introduction

In this project, we explore if the model which is trained for Pong environment could be used for the Tennis environment and check its performance. If it can be achieved, this will indicate the usefulness of transfer learning, as a general model can be developed and the knowledge can be transferred to different models which will use less computational resources than developing a model for each specific task.

A. Transfer Learning

It is a machine learning method that uses the knowledge gained while solving a problem and applies it to a different but related problem. The key distinction between standard machine learning method and transfer learning is the use of pre-trained models that have been built from previous task to jump start the development process on a new task without having to apply the same technique over again. This saves the effort of re-training a problem similar to the previously solved problem since the already developed modules are used and hence the overall training time is reduced thereby accelerating the results.

In this project, we aim at achieving the goal of training the game Atari "PONG" from the existing pre trained model of Atari "TENNIS" through transfer learning by means of reinforcement learning and vice versa.

B. Reinforcement Learning

Reinforcement Learning(RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. In reinforcement learning, the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. Fig.1 represents the basic idea and elements involved in a reinforcement learning model.

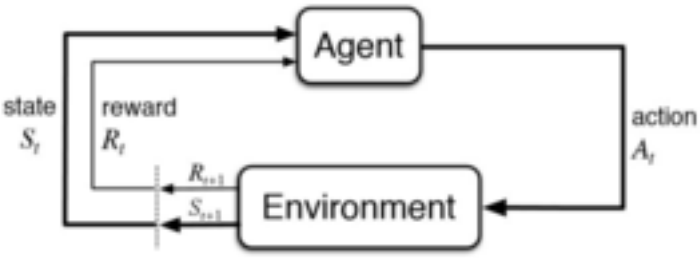

Fig. 1. Elements involved in a basic reinforcement learning model.

II. Implementation

In order to perform transfer learning between two similar games we need to learn each game individually. To achieve this we used reinforcement learning technique. There are various reinforcement learning approaches available. We have used the concept of Deep Q-Network (DQN) for our project. Implementation of reinforcement learning can be broadly divided into three phases –

- Runnning environment of that game.
- Creating Convolutional Neural Network (CNN).
- Applying DQN algorithm.

A. Environment

To test our implementation we needed an graphical interface of the game which can visually show the effectness of the algorithm. To fulfil this need we used OpenAI Gym. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. This is the gym open-source library, which gives you access to a standardized set of environments. There are two basic concepts in reinforcement learning: the environment (namely, the outside world) and the agent (namely, the algorithm you are writing). The agent sends actions to the environment, and the environment replies with observations and rewards (that is, a score).

The core gym interface is Env, which is the unified environment interface. There is no interface for agents; that part is left to you. The following are the Env methods we used:

- reset(self): Reset the environment's state. Returns observation.
- step(self, action): Step the environment by one timestep. Returns observation, reward, done, info.

- render(self, mode='human'): Render one frame of the environment. The default mode will do something human friendly, such as pop up a window.

B. Environment

Deep Q-Network (DQN) algorithm demands construction of a neural network for its implementation. But, we cannot use any neural network in our project as the openAI gym interface we used returns one frame of the environment which is an image. Thus, we used CNN for our project.

Atari-playing DQNs make sense of the screen with the help of a convolutional neural network (CNN) — a series of convolutional layers that learn to detect relevant game information from the mess of pixel values. We pre-process the raw input by downsizing from the original size to a more manageable 84 x 84 and convert to grayscale.

We used Torch, an open-source machine learning library, for constructing CNN in an efficient and accurate way.

C. Deep Q-Netwok (DQN)

DQN is a RL technique that is aimed at choosing the best action for given circumstances (observation). Each possible action for each possible observation has its Q value, where 'Q' stands for a quality of a given move. DQN is based on another more basic RL algorithm called as Q-learning.

Q-learning learns the action-value function Q(s, a): how good to take an action at a particular state. In Q-learning, we build a memory table Q[s, a] to store Q-values for all possible combinations of s and a.

Technical speaking, we sample an action from the current state. We find out the reward R (if any) and the new state s' (the new board position). From the memory table, we determine the next action a' to take which has the maximum Q(s', a').

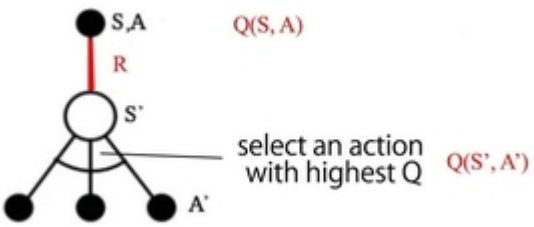

Fig. 2. Selecting an action in Q-learning.

We can take a single move a and see what reward R can we get. This creates a one-step look ahead. R + Q(s', a') becomes the target that we want Q(s, a) to be.

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

However, if the combinations of states and actions are too large, the memory and the computation requirement for Q will be too high. To address that, we switch to a deep network Q (DQN) to approximate Q(s, a). The learning algorithm is called Deep Q-learning. With the new approach, we generalize the approximation of the Q-value function rather than remembering the solutions.

III. Problems Faced

In reinforcement learning, both the input and the target change constantly during the process and make training unstable. In DQN, we build a deep network to learn the values of Q but its target values are changing as we know things better that is we are chasing a non-stationary target.

In RL, we often depend on the policy or value functions to sample actions. However, this is frequently changing as we know better what to explore. As we play out the game, we know better about the ground truth values of states and actions. So our target outputs are changing also. Now, we try to learn a mapping f for a constantly changing input and output. Solutions are -

A. Experience Replay

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

- reduces correlation between experiences in updating DNN
- increases learning speed with mini-batches
- reuses past transitions to avoid catastrophic forgetting

B. Target Netwoks

In TD error calculation, target function is changed frequently with DNN. Unstable target function makes training difficult. So Target Network technique fixes parameters of target function and replaces them with the latest network every thousands steps.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_{p} Q(s_{t+1}, p) - Q(s_t, a) \right]$$

Fig. 3. Target Q function in the red rectangular is fixed.

IV. Result

A. Atari Pong

We ran our DQN agent on several different environments such as Pong-Deterministic-v4, Pong-Deterministic-v0, Pong-v0, and Pong-v4. Each environment has different frame skip rate and action probabilities. We got least training time on Pong-Deterministic-v4 and best performance on Pong-Deterministic-v0. See Fig. 4 for game play.

Average Reward - 19 on PongDeterministic-v4
Average Reward - 17 on PongDeterministic-v0
Total Episodes - 2000

B. Atari Tennis

We ran our DQN agent on Tennis-Deterministic-v4 environment. Training time required was too high and we couldn't complete the training. We ran our agent continuous for four days on gpu environment. Reason for such a long time in training is that input for CNN is frame of game play and our

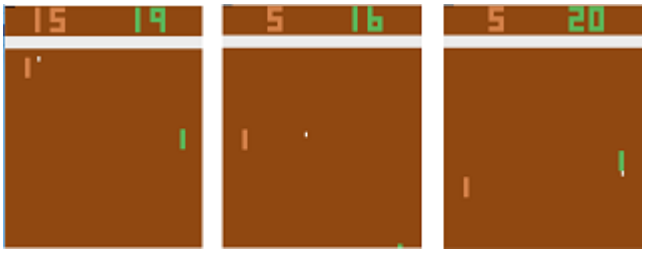

Fig. 4. Pong Game Play

agent flips its side in each set of a game, because of constant flipping of sides the CNN is having difficulties finding position of the agent in the game play. See Fig. 5 for game play.

Average Reward - 5

Total Episodes - 10000

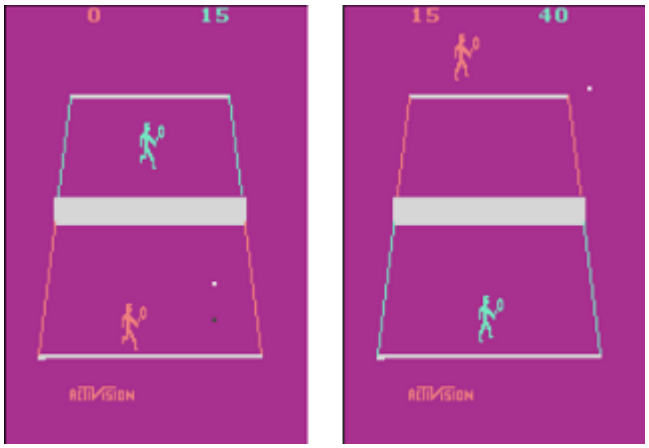

Fig. 5. Tennis Game Play

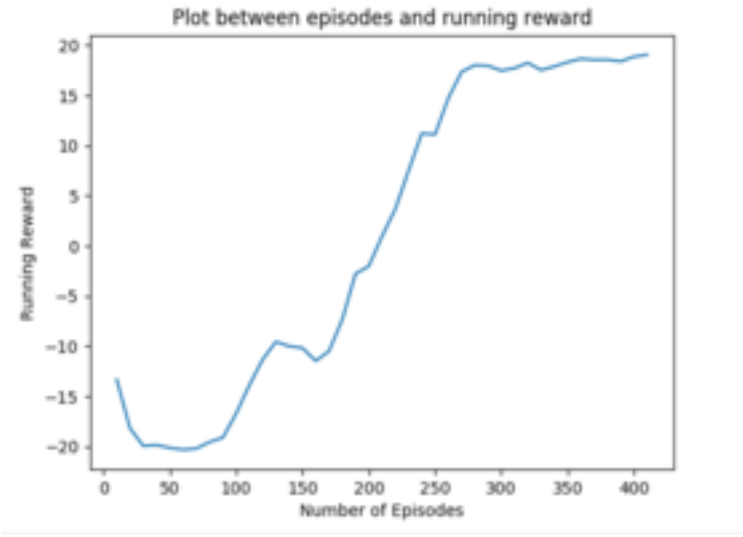

Fig. 6. Plot between Episodes and Running Reward for Atari Pong

V. CONCLUSION

We successfully trained DQN model to play Pong, whereas Tennis agent is able to play a decent game but not as good as Pong. Besides having good agents for Pong and Tennis we couldn't do successful transfer learning because of following possible reasons – In Atari Pong, agent's playing side is fixed in the game means it always appears on right side, whereas in Atari Tennis, agent flips its side on each set in a game. Also the playground of Pong is horizontal and Tennis has vertical playground.

For transfer learning we should not choose games just because they look similar in game play. The games we choose should have similar playground, agent behavior.

REFERENCES

[1] Chaitanya Asawa, Christopher Elamri, and David Pan, "Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability"

[2] Deep Q-Learning - https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4

[3] Reinforcement Learning introduction- https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

[4] https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

[5] https://en.wikipedia.org/wiki/Reinforcement_learning

[6] OpenAI GYM environment and documentation - https://gym.openai.com/envs/atari http://gym.openai.com/docs/

[7] What is transfer learning - https://www.hackerearth.com/practice/machine-learning/transfer-learning/transfer-learning-intro/tutorial/

[8] Implementation references - https://github.com/Vetal1977/pong-rl https://paperswithcode.com/paper/playing-atari-with-deep-reinforcement https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/ https://braraki.github.io/research/2018/06/15/play-openai-gym-games/ https://braraki.github.io/research/2018/06/15/play-openai-gym-games/ https://github.com/navjindervirdee/2048-deep-reinforcement-learning https://towardsdatascience.com/what-is-transfer-learning-8b1a0fa42b4 https://github.com/Vetal1977/pong-rl