CSCI 599: ML for Games

Final Project Report

DEEP RL for Zed

Team: FireBoy and WaterGirl

Naman Gupta

Nigrah Bamb

Aditi Swaroop

Yash Shah

Mitali Mehta

Table of Content

Abstract

Our 599 project is focused on developing reinforcement learning based agents to play a single-player, 2D platformer, pre-existing game named 'Zed'. 'Zed' is a goal-based video game, where the agent has to start at one point, and to complete the level, must reach the end while collecting coins and avoiding obstacles. Our project focuses on learning, based on agent's actions and we implemented Reinforcement learning with an Advantage Actor-Critic Model to make our agent make its own decisions. While this can be taken forward for more levels of this game and even for multi-player games, as our Future Scope will describe, we have currently succeeded in training our agent for the first level of the game. To calculate an apt reward function, we have used a combination of four indicators that show the success-failure level of the agent.

Background and Overview

'Platformers' or 'platform games' are games that mainly revolve around a character controlled by the player, which runs and jumps to avoid obstacles and/or to defeat enemies. Platformers are often classified as a subgenre of action games, and is considered to be one of the first game genres [1]. The game we have chosen for our project is a 2D platformer game, with a single agent that we will apply two state-of-the-art Deep Reinforcement learning strategies on. This project is mainly divided in three parts - determining the correct reward function for our game, applying Deep Q-Network to the agent, and finally applying an Advantage Actor-Critic (A2C) Model to train the agent, and comparing these two Reinforcement learning techniques, to see which one fits our use case better.

Considering the complex nature of 2D platformer games, Deep Reinforcement learning strategies were chosen to model the agent of 'Zed', our game. Two well known types of learning strategies, Deep Q-Learning and Advantage Actor-Critic Model were explored in modeling the main AI. Both of these methods will be explained in detail in our Methodology subsection. When dealing with Deep Reinforcement models, representation of the state of the game with which the models train, plays an important role. The game state is expected to encapsulate the spatial and temporal features. The spatial features include the player, resource, and enemy locations. The temporal features include the resource depletion, game score, player health, allies alive etc. Typically, a RGB snapshot of the game at a certain instant in the game is fed as the input to the models.

Overview of the Game Environment

The game we have chosen to train with Deep Reinforcement learning is called 'Zed' [2]. Zed is a 2D platformer game owned by MiniClip, the world's largest privately owned gaming website. It is a single player game consisting of ten levels, with increasing levels of difficulty. Our project is focused on Level 1. Zed was near-perfect for our project due to the tasks contained in the game, and some well-defined parameters, that would help form a strong reward function for our network.

Zed is an Android. Zed's mission is to collect three gold coins from the whole maze, while avoiding obstacles in order to reach a golden door. On reaching the golden door, Zed goes on to the next level of the game. At the end of successfully crossing all ten levels, Zed wins a Golden Suit. The

game quest begins in gold mines, and as levels progress, the mazes get more challenging. The level that our project focuses on contains several features, to guide us through training. Firstly, there are Blue Coins, which are similar to Gold Coins. The amount of blue coins collected contributes to the total score of that level. Then, there are Gold Coins. There are three of these, spaced out through the level. On collection of all three, the Golden door opens, and Zed has to navigate back to that door, and pass through it to complete the level. Level 1 gives Zed three lives. If Zed comes in contact with the fire-throwing dinosaur, Zed loses a life and has to start at the beginning. On losing all three lives, the player loses the level. Lastly, there is an energy level. In the beginning of the level, the energy bar is full. The longer Zed stays in that maze and stalls, the lesser the energy gets. If the energy level goes to zero, Zed loses a life.

Our project uses these features to come up with a reward function to train Zed. All these features are needed for Zed to successfully pass this level. Hence, the points, the gold coins, the lives and the energy, all contribute to Zed's reward policy, as explained in detail in our Methodology.

Additionally, our code is set to read the game window's coordinates assuming that it will open at the middle of the system's screen. It thus grabs screenshots of the game window and a lot of other information using these coordinates.

Figure 1 | Zed[2]

Related Work

The game environment is a complex environment with many factors to consider. While we were sure of Reinforcement learning, we had multiple options to go with. Some of the options we considered, that is, some methods that exist to implement deep reinforcement learning are elaborated below. Using these and others as a reference, we decided to implement two strategies.

Double Q Learning

In his paper Double Q-Learning, Hado van Hasselt explains how Q-Learning performs very poorly in some stochastic environments. He pointed out that the poor performance is caused by large overestimation of action values due to the use of Max Q(s',a) in Q-learning. The proposed solution is to maintain two Q-value functions QA and QB, each one gets an update from the other for the next state. The update consists of finding the action a* that maximises QA in the next state (Q(s', a*) = Max Q(s', a)), then use a* to get the value of QB(s', a*) in order to update QA(s, a) [3].

REINFORCE: Policy Gradients

The DQN based solution basically implements the Q-Learning algorithm that aims to learn the values of actions in a particular state and then an action is selected based on their estimated action values. Unlike Q-Learning, Policy Gradients (PG) based solutions attempt to learn a parameterized policy that can select actions without consulting a value function estimate. A value function may still be used to learn the policy parameter, but is not required for action selection. The REINFORCE algorithm is a Monte Carlo based policy gradient method to learn the policy parameter based on the gradient of the reward measure with respect to the policy parameters. The main idea is that at each increment of the policy parameters, the update is proportional to the product of a return G , includes rewards from time *t* until the episode ends, and a vector, the gradient of the probability of taking the action actually taken divided by the probability of taking that action. The update causes the parameter to move the most in directions that favor actions that yield the highest return [4].

Dueling DQN

(Wang et al.) presents the novel dueling architecture which explicitly separates the representation of state values and state-dependent action advantages via two separate streams. The key motivation behind this architecture is that for some games, it is unnecessary to know the value of each action at every timestep. The authors give an example of the Atari game Enduro, where it is not necessary to know which action to take until collision is imminent. By explicitly separating two estimators, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. Like the Enduro example, this architecture becomes especially more relevant in tasks where actions might not always affect the environment in meaningful ways [5].

Methodology

1. Reward Function

One of the key pieces of information that plays a vital role in developing a good model to learn the game is the reward function that the agent aims to optimize. Analyzing sparse rewards after every game does not capture the intermediate rewards at various instances in a game. Thus, we must develop some function to evaluate the state of the game at each time step.

To better quantify the state of the game, we introduce the notion of an intermediary reward, which is computed at every time step and is used by the model to critique its policies [6]. Our reward function was a combination of several features that indicated the player's success at each instant. The factors are as follows:

- Blue coins collected (BlC)

  There were a set of blue coins along the whole maze, at the end of the level, the number of blue coins got added to the total score of that level. We used Python's 'Pytesseract' library and performed Optical Character Recognition to parse the score.

- Gold coins collected (GC)

  The gold coins collected were needed for the player to progress to the next level, the ultimate success of the player was calculated with the three gold coins obtained.

- Lives remaining (L)

  Each player was given 3 lives at the beginning of a game, once the three lives were exhausted, the player lost the game. Lives could be exhausted by either coming in contact with the fire breathing dragon, or by reaching a zero energy level, due to staying in the game for a long period of time.

- Energy level remaining (E)

  In each life the player starts with a full energy level, we quantify the full energy level as 3000pts as that is the bonus added to the player score at the end of the level if the player has full energy.

  The energy bar changes the color shade as the energy goes on decreasing. Hence, we applied colour clustering to convert the energy bar into a number from 0 to 3000.This value is one of the inputs of the reward function.

The factors weighed in as follows:

RF = (1 - L) * Previous State + E + GC + BIC

2. Deep Q-Network

Q-Learning, where Q stands for 'Quality', is a variation of reinforcement learning. In Q learning, there is an agent having states and corresponding actions. At any moment, the agent is in some feasible state. In the next time step, the state is transformed to another state(s) by performing some action. This action is accompanied either by reward or a punishment (negative reward). The goal of the agent is to maximize the reward gain [7]. Deep Q Network (DQN) is a form of reinforced learning in which the output of a CNN is not classification, but Q values (rewards) given to actions resulting from the input states. The DQN agent learns successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. Recently, DQN has proved successful in the challenging domain of classic Atari 2600 games [8].

Given below, is an explanation and a flowchart of how we have implemented Deep Q-learning in this project.

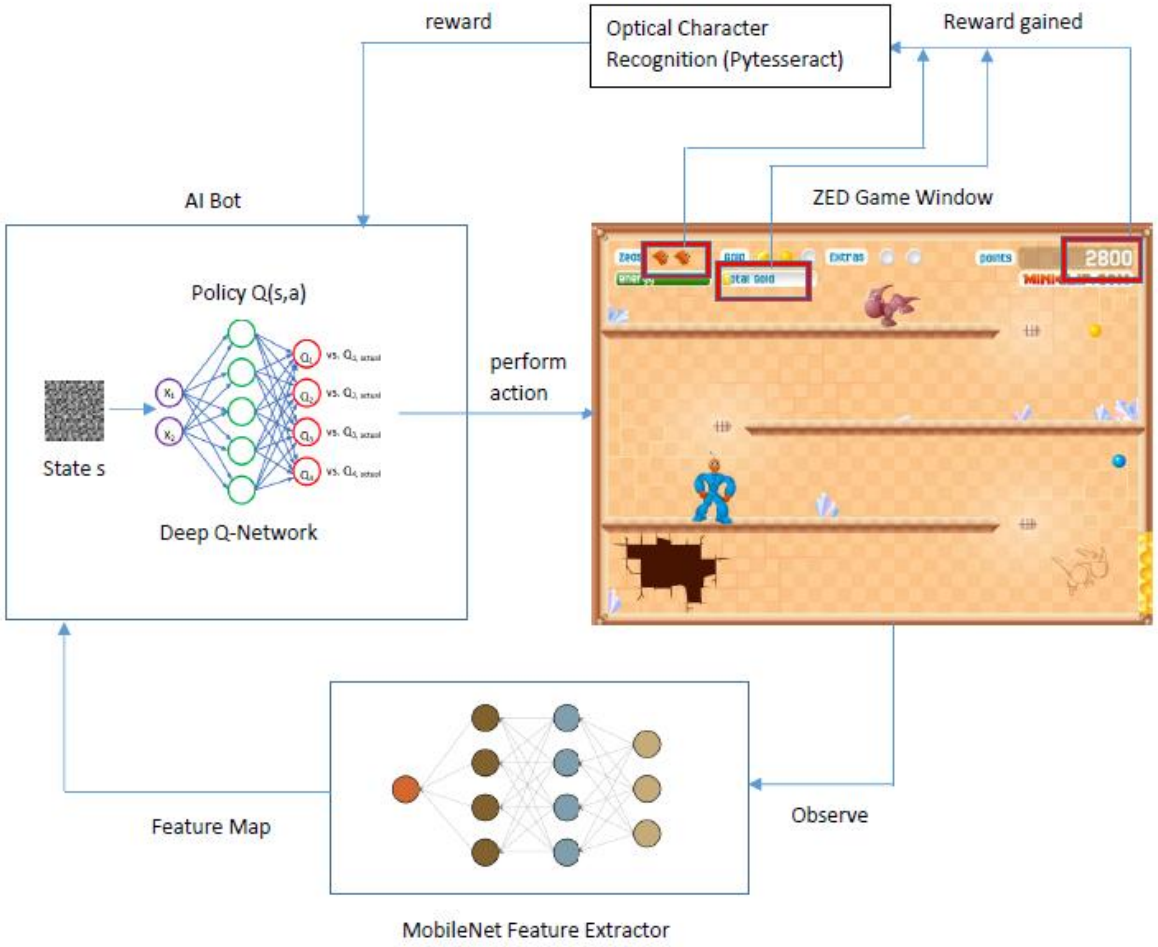

Figure 2 | Schematic illustration of the Deep Q-Network

2.1 Experience Replay:

We consider tasks in which the agent interacts with an environment through a sequence of observations, actions and rewards. The goal of the agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\, a_t = a,\, \pi\right], \quad [8]$$

which is the maximum sum of rewards r_t discounted by **γ** at each time-step t, achievable by a behavior policy **π = P(a|s)**, after making an observation (s) and taking an action (a). Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $\boldsymbol{r + \gamma\, max\, Q(s', a')}$.We address the instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed **'**Experience Replay' that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Second, we used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target [8].

2.2 Epsilon-Greedy Strategy:

Balancing the ratio of exploration/exploitation is a great challenge in reinforcement learning (RL) that has a great bias on learning time and the quality of learned policies. On the one hand, too much exploration prevents from maximizing the short-term reward because selected "exploration" actions may yield negative reward from the environment. But on the other hand, exploiting uncertain environment knowledge prevents from maximizing the long-term reward because selected actions may not be optimal [11]. To get this balance between exploitation and exploration, we use what is called an *epsilon greedy strategy*. With this strategy, we define an *exploration rate* ϵ that we initially set to 1. This exploration rate is the probability that our agent will explore the environment rather than exploit it. With ϵ=1, it is 100% certain that the agent will start out by exploring the environment. As the agent learns more about the environment, at the start of each new episode, ϵ will decay by some rate that we set so that the likelihood of exploration becomes less and less probable as the agent learns more and more about the environment. The agent will become *"greedy"* in terms of exploiting the environment once it has had the opportunity to explore and learn more about it.

$$p = min + (epsilon + min) * e(-decayRate * \text{epoch})$$

To determine whether the agent will choose exploration or exploitation at each time step, we generate a random number between 0and 1. If this number is greater than epsilon, then the agent will choose its next action via exploitation, i.e. it will choose the action with the highest Q-value for its current state from the Q-table. Otherwise, its next action will be chosen via exploration, i.e. randomly choosing its action and exploring what happens in the environment [13].

2.3 Updating the Q-Value:

The optimal action-value function obeys an important identity known as the Bellman equation. This is based on the following intuition: if the optimal value Q*(s', a') of the sequence s' at the next time-step was known for all possible actions a' ,then the optimal strategy is to select the action a' maximizing the expected value of $\boldsymbol{r + \gamma\, max\, Q(s', a')}$. The basic idea behind our algorithm is to estimate the action-value function by using the Bellman equation as an iterative update [8].

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s', a')\right] \quad [13]$$

We want to make the Q-value for the given state-action pair as close as we can to the right hand side of the Bellman equation so that the Q-value will eventually converge to the optimal Q-value q∗.This will happen over time by iteratively comparing the loss between the Q-value and the optimal Q-value for the given state-action pair and then updating the Q-value over and over again each time we encounter this same state-action pair to reduce the loss [13].

$$q_*(s,a) - q(s,a) = loss$$

$$E\left[R_{t+1} + \gamma \max_{a'} q_*(s', a')\right] - E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] = loss \quad [13]$$

2.4 Learning rate:

The learning rate **α** is a number between 0 and 1, which can be thought of as how quickly the agent abandons the previous Q-value in the Q-table for a given state-action pair for the new Q-value. So, for example, suppose we have a Q-value in the Q-table for some arbitrary state-action pair that the agent has experienced in a previous time step. Well, if the agent experiences that

same state-action pair at a later time step once it's learned more about the environment, the Q-value will need to be updated to reflect the change in expectations the agent now has for the future returns. We don't want to just overwrite the old Q-value, but rather, we use the learning rate as a tool to determine how much information we keep about the previously computed Q-value for the given state-action pair versus the new Q-value calculated for the same state-action pair at a later time step. The higher the learning rate, the more quickly the agent will adopt the new Q-value. For example, if the learning rate is 1, the estimate for the Q-value for a given state-action pair would be the straight up newly calculated Q-value and would not consider previous Q-values that had been calculated for the given state-action pair at previous time steps [13].

2.5 Calculating new Q-value:

Our new Q-value is equal to a weighted sum of our old value and the *learned value.* We multiply the old value by (1−α). Our learned value is the reward the agent receives from moving right from the starting state plus the discounted estimate of the optimal future Q-value for the next state-action pair (s',a') at time t+1. This entire learned value is then multiplied by our learning rate.

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}} \quad [13]$$

2.6 Training:

- 'Zed' is kept running in a 800 x 600 window right in the center of the screen.
- We utilized Python's 'ImageGrab' library that took screenshots of the game screen and fed it to MobileNet, a CNN feature extractor. The feature-map extracted from MobileNet represents the current state of the game environment.
- A dense neural network is used as a function-approximator that takes this state as an input and decides what action to take (output). This Deep Q-Network takes one of four actions. [left, right, jump left, jump right] based on the Epsilon-Greedy strategy described above. Once an action was taken to interact with the game, an outcome was observed and a net reward was calculated based on the reward function described above. This information (reward) is fed back to the Deep Q-Network. The network adjusts its policy by updating the Q-values for the state-action pairs as described above, based on thousands of such interactions with the game.
- Game is restarted every 100 steps to reset the environment. This goes on for 100 games/epochs.

3. Advantage Actor Critic (A2C) and ACKTR Model

The Actor-Critic algorithm uses two neural networks to approximate the policy. One is a neural network that approximates policy, and an object that selects an action using this network is called an Actor. This neural network that approximates the policy is called a policy network. The other is a neural network that judges whether the action selected by the Actor is good or bad behavior. Using this network, an object that predicts the value of the action that the Actor selected is called the value network. The value network approximates a Q function that directly represents the value of an action that an actor chooses in a specific state [9].

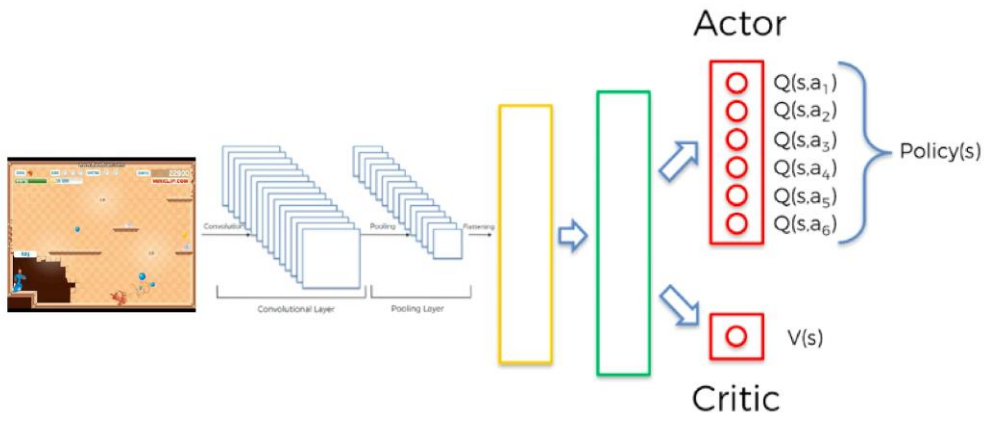

Figure 3 | Actor-Critic Model Flow Diagram

The above diagram explains our actor-critic implementation for Zed online game. Before implementing different variants of actor-critic on Zed, we implemented ACKTR variant of actor-critic on Atari games and Montezuma's revenge - a similar game for Zed. Actor Critic using Kronecker-Factored Trust Region (ACKTR) uses scalable trust region natural gradient method for actor-critic methods. It is also a method that learns non-trivial tasks in continuous control as well as discrete control policies directly from raw pixel inputs.

We used similar settings and environmental variables to tune our hyper-parameters for Zed actor-critic implementation based on openAI baselines for Actor Critic using Kronecker Factored approximation. As researched, we found out that ACKTR is a more sample efficient reinforcement learning algorithm than A2C - synchronous version of Asynchronous Advantage Actor-Critic method.

The step model is updated using the following steps in actor-critic environment:

1. It creates a runner object that handles the different environment state.
2. The step model generates different experiences from the environment.
3. Using these experiences, we train our model to capture the learning.
4. When the AI runner agent takes a step or performs an action, each environment is updated with corresponding action or step.
5. The resulting output is a batch of experiences which is further passed for gradient calculation.

6. The trained model and the batch output is used to calculate the gradient all at once.
7. Simultaneously, the step model is updated with newly calculated weights. Below diagram explains this process with multiple worker agents:

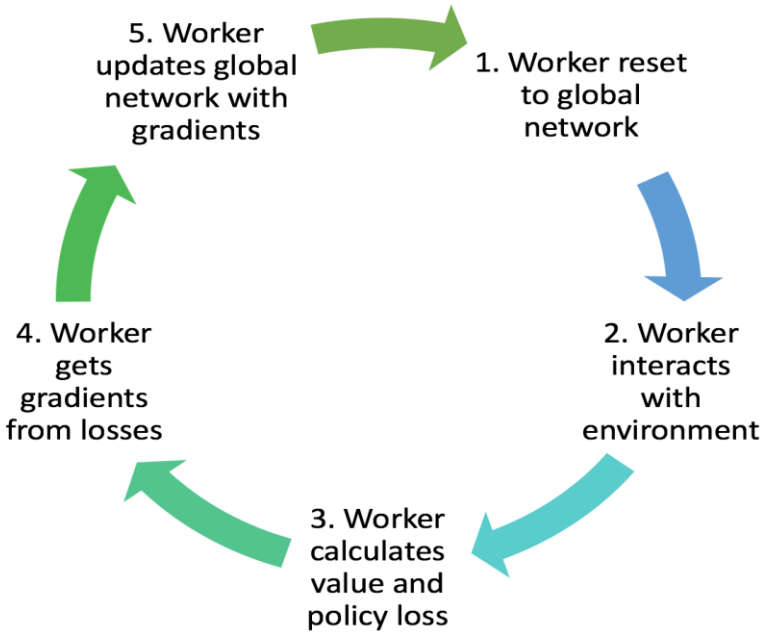

Figure 4 | Actor-Critic Lifecycle

Advantages of using Actor-Critic Kronecker-Factored Trust Region:

1. It is based on the underlying actor-critic approach.
2. Provides capability of optimizing trust regions.
3. Improves sample efficiency and is highly scalable.

Experiments & Results

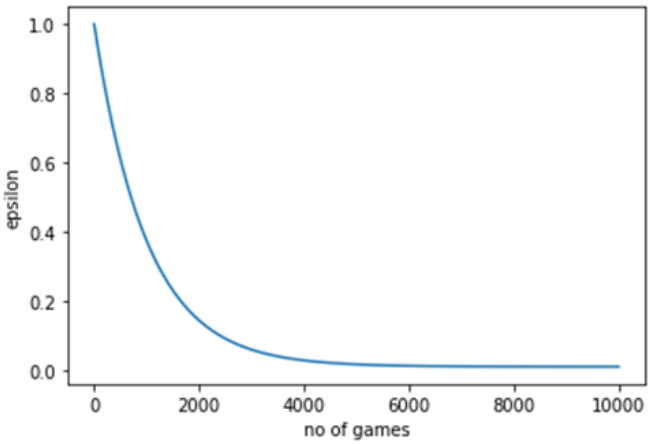

Figure 5 | Curve tracking the epsilon value throughout training

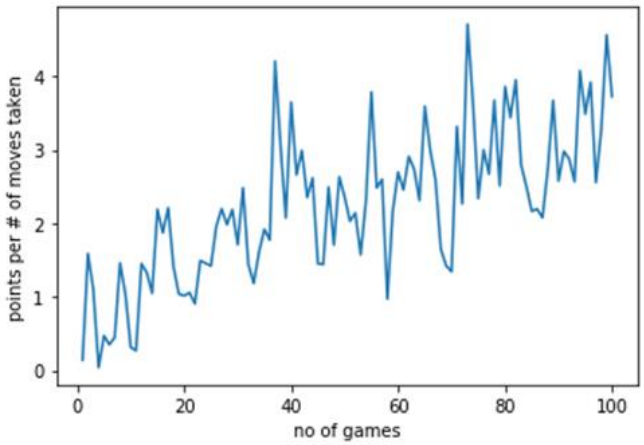

Figure 6 | Curve tracking the average score per move. Each point is the average score achieved per episode after the agent is run with ε-greedy policy for 100k frames

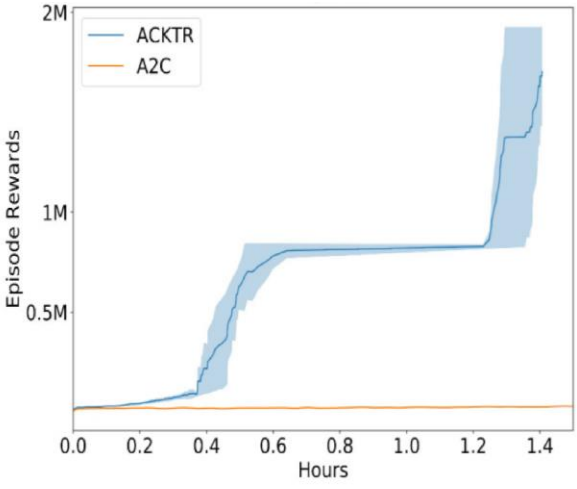

Figure 7 | ACKTR agent in Atari game starts earning Rewards in a few hours compared to A2C agent.

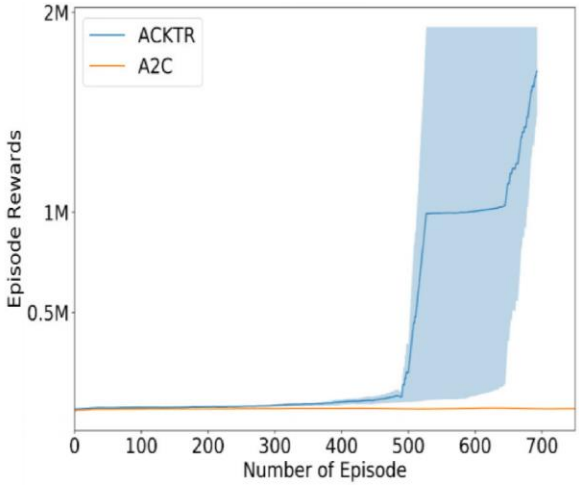

Figure 8 | Curve tracking the average reward as number of episodes increases to 700 for both the agents.

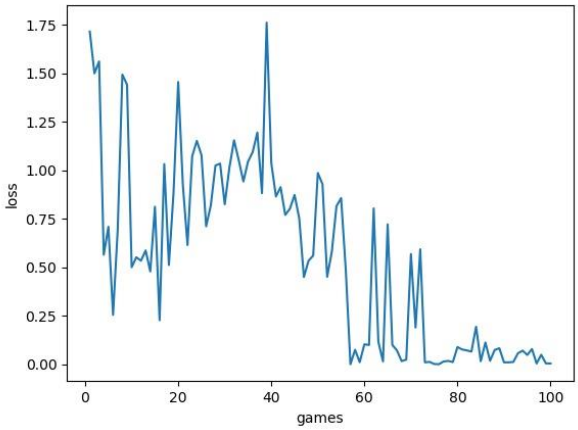

Figure 9 | Curve tracking the loss throughout the training

We tested variants of actor-critic on Atari game Montezuma's Revenge and our game Zed, and we observed 2- to 3-fold improvements in sample efficiency on average compared with a first-order gradient method (A2C) and the traditional deep Q approach. Due to the scalability of the ACKTR algorithm, we could train several non-trivial tasks in continuous control directly from raw pixel observation space.

Future Scope

This report outlines and summarizes the two Deep Reinforcement Learning techniques that we have applied for 2D platformer game agent training. In our project, we have used Deep Q-learning and Advantage Actor Critic (A2C) Model to train Zed. Some improvements that could be applied to this project, and some future scopes are that we could use Deep reinforcement learning techniques for multi-agent 2D platformer gaming environments.

We've observed that in applied RL settings, the question of whether it makes sense to use multi-agent algorithms often comes up. Compared to training a single policy that issues all actions in the environment, multi-agent approaches can offer a more natural decomposition of the problem and potential for more scalable learning [10]. Multi-Agent reinforcement learning is an interesting challenge in the gaming industry. Moreover, multi-agent Reinforcement learning is not as challenging when the agents have to play against each other. Rather, when there are agents that help each other via Synchronization to reach the goal, that's what poses a bigger hurdle. If we look at this issue on a broader scope, we can think of how it can help automate traffic congestion with automated vehicles. This problem would need different vehicles to communicate with each other, and if we can achieve that effectively in our project, its scope is limitless. So, as far as the scope of this project is concerned, we can build on our single agent Reinforcement learning techniques and use it for a game that has two or more agents, who work

with a synchronization network to communicate with each other and collaboratively, while keeping the others' state in mind, reach a common goal.

References

[1] Minkkinen, T., 2016. *Basics Of Platform Games*. Bachelor of Business Administration, Business Information Technology. Kajaanin Ammattikorkeakoulu University of Applied Sciences.

[2] Miniclip.com. n.d. Zed - A Free Puzzle Game. [online] Available at: https://www.miniclip.com/games/zed/en/ [Accessed 19 February 2020].

[3] Medium. n.d. Double Q-Learning The Easy Way. [online] Available at: https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3 [Accessed 25 March 2020].

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

[5] Medium. n.d. *Double Q-Learning The Easy Way*. [online] Available at: <https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3> [Accessed 25 March 2020].

[6] Justesen, Niels, et al. "Deep learning for video game playing." IEEE Transactions on Games (2019).

[7] T. Okuyama, T. Gonsalves and J. Upadhay, "Autonomous Driving System based on Deep Q Learning," 2018 International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, 2018, pp. 201-205.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G.Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[9] S. Kim, J. Kim and D. Suh, "Game Controller Position Tracking using A2C Machine Learning on Inertial Sensors," 2019 IEEE Games, Entertainment, Media Conference (GEM), New Haven, CT, USA, 2019, pp. 1-6.

[10] Seita, D., 2018. *Scaling Multi-Agent Reinforcement Learning*. [online] The Berkeley Artificial Intelligence Research Blog. Available at: https://bair.berkeley.edu/blog/2018/12/12/rllib/ [Accessed 24 February 2020].

[11] Michel Tokic[1,2], "Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences", [1]Institute of Applied Research, University of Applied Sciences Ravensburg-Weingarten, 88241 Weingarten, Germany. [2]Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany.

[12] James D McCaffrey. n.d. The Epsilon Greedy Algorithm. [online] Available at: https://jamesmccaffrey.wordpress.com/2017/11/30/the-epsilon-greedy-algorithm/ [Accessed 26th February 2020].

[13] DeepLizard. N.d. Reinforcement Learning - Goal Oriented Intelligence. [online] Available at: https://deeplizard.com/learn/video/mo96Nqlo1L8 [Accessed 27th February 2020].

[14] Expert-augmented actor-critic for ViZDoom and Montezuma's Revenge. [online] Available at:https://arxiv.org/pdf/1809.03447.pdf [Accessed 20th March 2020].