



FILA PROJECT

CS 747

Transfer Learning to improve Reinforcement Learning

Tanmaya Shekhawat

160050051

Rajat Majoka

160050031

Prateek

160050110

Aditya Kumar Mahto

160050094

INTRODUCTION

Transfer learning is a machine learning method which focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

Using Reinforcement Learning or Deep Q-Learning to learn different games is fairly popular at present. We explore transfer learning to see if we can use a pre-trained network on one game to another. As mentioned in [1] transfer learning could also have a negative impact on sharing knowledge. For this task, we used games like snake and Pacman because in both of them the task is similar, which is to move to some sort of target while avoiding some object, therefore, we believe that transferring the knowledge from one game to another will give some positive results.

Goals

The following are some of the targets that we tried to accomplish in this project:

1. Improve the performance of a game using information learned from some other but related game i.e we will transfer the knowledge from one game to another.
2. Reduce the training time by training only some of the layers and with better initialisations.
3. Show that it also maintains greater performance stability during the process of learning as we would like to deploy a model with reliable performance.

To fulfil the aforementioned goals, we implemented certain novel things i.e. approximating Q-values of snake and Pacman using features instead of the whole state space, tried transfer Learning between snake and Pacman and Use of Dual experience Replay in transfer learning.

Related Work

Related work has been done in the paper [4] (Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability). In this paper, the transfer learning is performed between snake and puckworld using deep Q-learning network.

In the paper [2](Exploration of Reinforcement Learning to SNAKE), reinforcement learning and convolutional neural network has been combined to train the agent of snake game.

In a paper [6] “Playing Atari with Deep Reinforcement Learning” (Mnih et al., 2013), Mnih et al. developed a single Deep Q-Network (DQN) that is able to play multiple Atari games, in many cases surpassing human expert players. The model takes in the raw image pixels of a game and outputs Q-values estimating future rewards. The model is trained with a variant of Q-learning with a target network and experience replay. The target network helps reduce oscillations or divergence of the policy, and experience replay removes correlation in the observations and prevents the model from getting stuck in bad local minima.

In the paper [5] “Autonomous Agents in Snake Game via Deep Reinforcement Learning” they proposed the idea to use HSV color image as input to the model instead of RGB as they eliminated the unnecessary grids in the game background by applying bitmask in HSV model and outputting the result to RGB. They also Dual Experience Replay to sample from the memory which we used in our task.

Approach

In Q-learning, we are trying to estimate for a given state and action pair, what is the expected discounted sum of future rewards is if we took that action from the state and followed the optimal policy after. Once we have these $Q(s, a)$ values, the action we can take from a given state s is $\text{argmax}_a Q(s, a)$.

When we have an incredibly large state space, however, it is hard with traditional update methods to determine these Q-values. Thus, neural networks have proven to, with training, be a great estimator for these Q-values. We will use deep Q-learning convolution neural network models with ϵ -greedy exploration decaying ϵ over time[3] so initially, we are exploring and trying to understand the environment, but after some time we want to mostly use what we have learned with only a little bit of exploring to maximise our reward.

We also tried experience replay[3] and dual experience replay[5] to smooth out learning and avoiding oscillations or divergence in the parameters. In Experience replay, we store the agent’s experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset named as replay memory. In dual experience replay we keep 2 independent experience sets MP_1 and MP_2 , experiences with higher rewards will be stored in MP_1 and experiences with lower rewards are stored in MP_2 because MP_1 experiences are more important but we keep MP_2 as well for more exploration just like ϵ -greedy exploration. The selection between the two memory pools is regulated by a proportion parameter η i.e., setting different sampling proportion η and $1-\eta$ for MP_1 and MP_2 respectively.

Technical Details

Pacman

We implemented the Pacman game from scratch, In our Pacman game there is only one food for a player to eat which occurs at random positions. There are 3 enemies each of them have random movement.

There are two kinds of walls in the game i.e one as a boundary of the game and the other is in between the game. The player is declared dead on colliding with the boundary walls but not with the walls inside the boundary, it just has to avoid it. So our final task is to train our player so that it can eat food avoiding walls and enemies.

To approximate the values of $Q(s, a)$ we used Deep Reinforcement Learning algorithm. For defining a state In the beginning, we took a screenshot of the game and converted the raw pixel values as an input to our CNN but it was taking too much time to compute as there is a large number of states i.e we have different enemies positions with different player positions and different food positions.

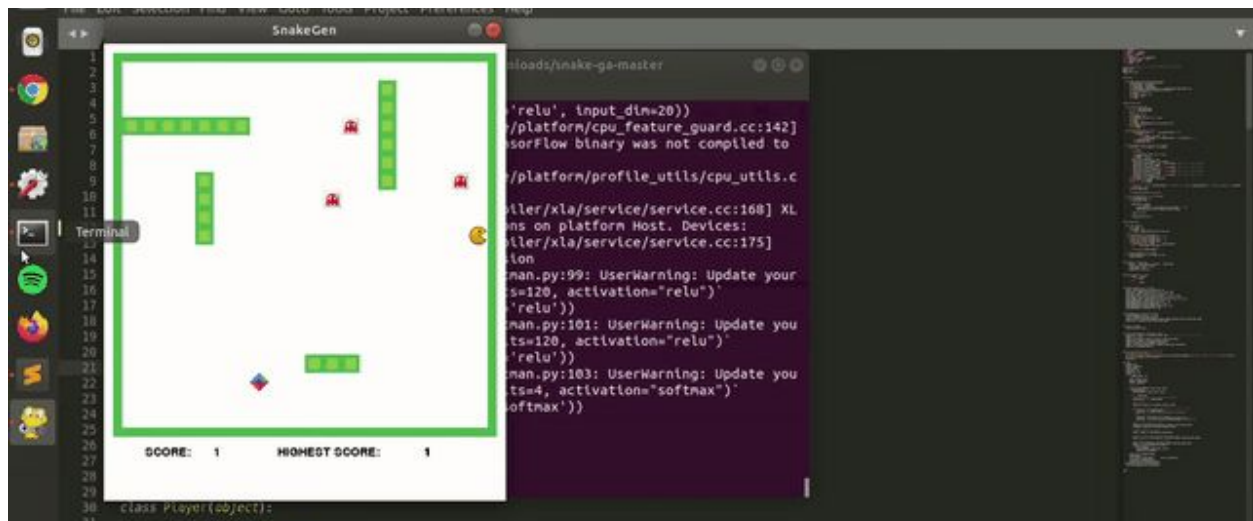
So instead, we used the input as some set of boolean feature, such as is there a danger

In either of the 4 directions, the relative direction of food with respect to the player, danger after taking an action, is there a wall after taking any direction and current moving direction of the player is taken as the input features(total 20 features).

Using positive reward for food and negative reward for enemies and boundary walls the model was trained to output optimal action.

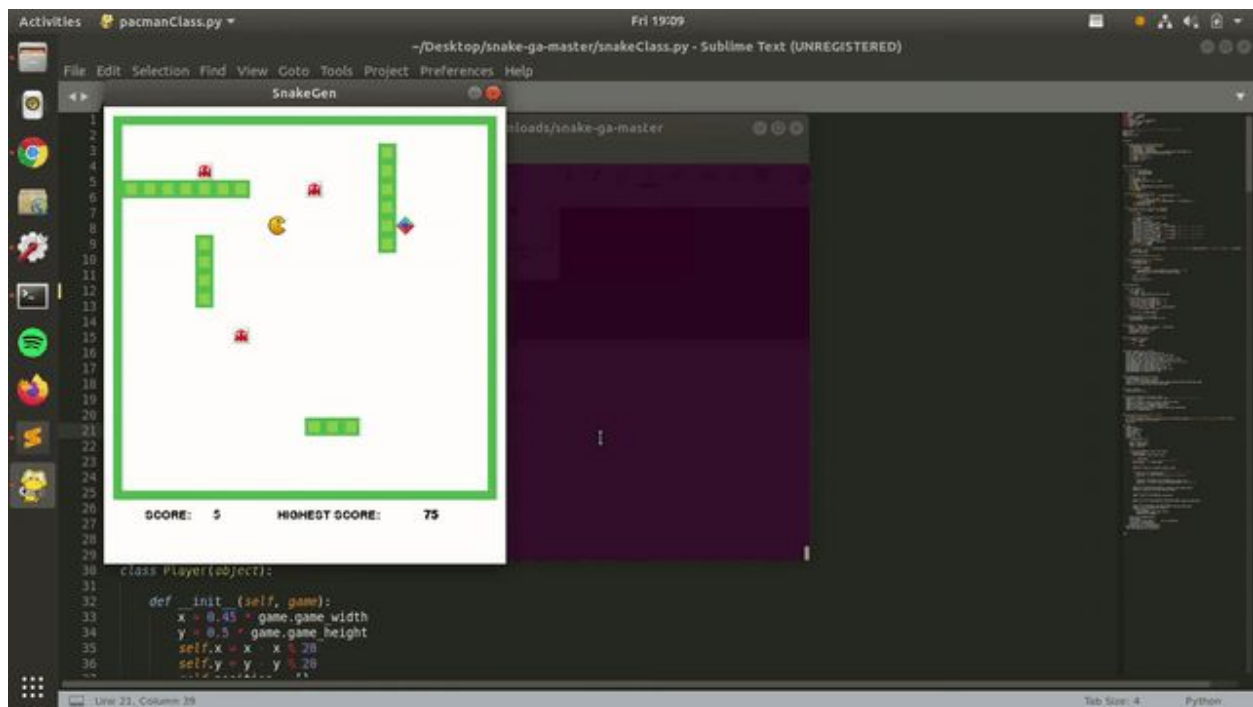
Pacman initially before training

Enemies are in red colour, Pacman is in yellow colour, food is rhombus-shaped and rest are green walls. Initially, the movement of Pacman is random as it hasn't learned anything



Pacman after some training steps

We can see that Pacman has started learning so it is eating food now.



Snake

We implemented the Snake game from scratch. In our snake, there was only one player and food with random positions and a boundary wall where the player will die on colliding.

To approximate the values of $Q(s, a)$ we used a Deep Reinforcement Learning algorithm. For defining a state in the beginning, we took a screenshot of the game and converted the raw pixel values as an input to our CNN but it was taking too much time to compute as there is a large number of states i.e. we have different snake positions with different shape and size and different food positions.

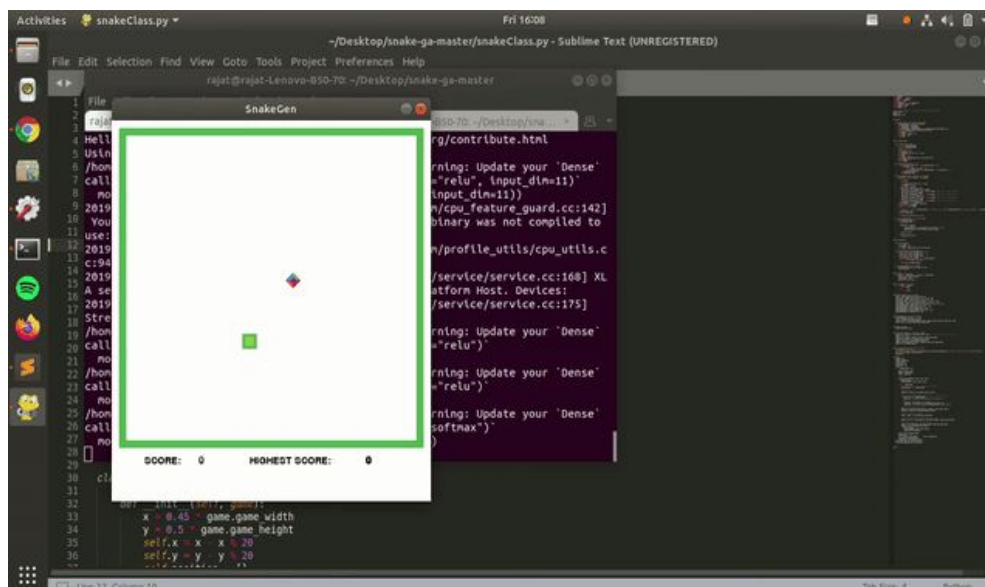
So instead, we used the input as some set of boolean features, such as is there a danger

In either of the 4 directions, the relative direction of food with respect to the player, danger after taking an action, and as we have to maintain the same type of features in snake and Pacman so features related with the walls kept as 0.

Using positive reward for food and negative reward for the boundary walls, the model was trained to output optimal action.

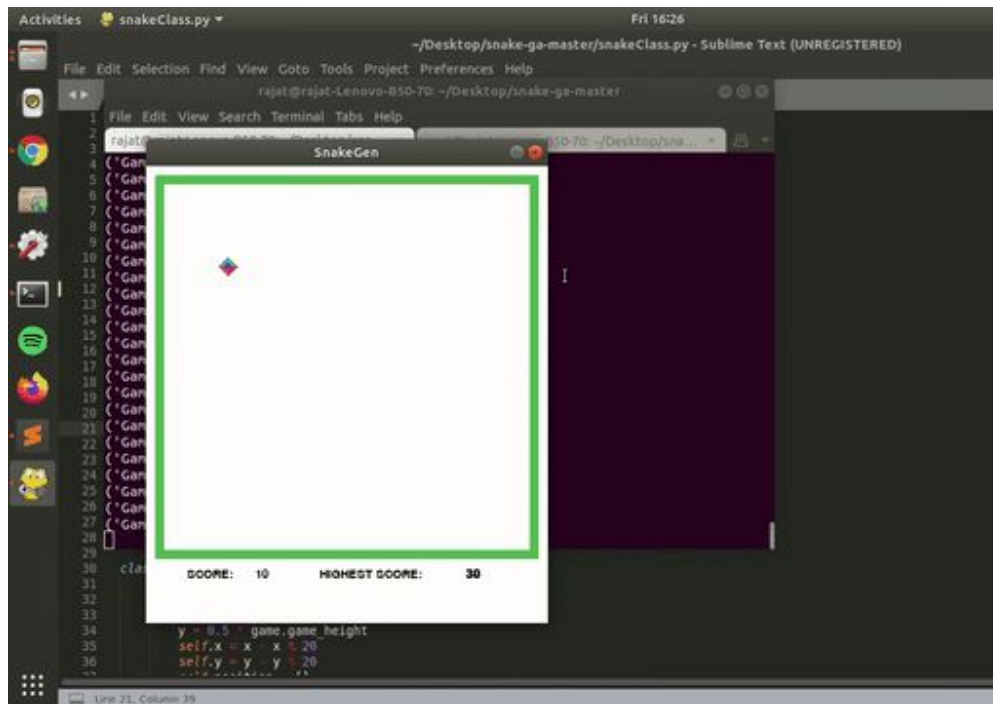
Snake initially before training

We can see that the snake is moving randomly as it hasn't learned anything.



Snake after some training steps

We can see that the snake has learned better now and is going for food.



Neural Network model in use

Dense(output_dim=120, activation='relu', input_dim=20)

Dropout(0.15)

Dense(output_dim=120, activation='relu')

Dropout(0.15)

Dense(output_dim=120, activation='relu')

Dropout(0.15)

Dense(output_dim=4, activation='softmax')

Transfer Learning

Transfer in reinforcement learning is a novel research area that focuses on the development of methods to transfer knowledge from a set of source tasks to a target task. Whenever the tasks are *similar*, the transferred knowledge can be used by a learning algorithm to solve the target task and significantly improve its performance. The idea behind transfer learning is that generalization can occur not only within tasks but also across tasks.

In our case, we would like to use networks that have performed very well at estimating Q-values for one game and try to adapt them, with a little more training, to estimate Q-values for another game. As we have both pre-trained Deep-Q network models of Snake and Pacman so we can apply transfer learning from one model to another and vice-versa as the game has similar objectives of chasing an object and avoiding some other object(s).

We are initializing layers of a target model with the weights of a pre-trained model and continuing to update these weights with backpropagation as we keep the layers fixed and then we are training the target model.

We used experience replay and dual experience replay to improve our model. In experience replay we store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset usually named as replay memory and at the end of every epoch(game) we sample some experiences from it and train our model on these experiences. In dual experience replay we keep 2 independent experience sets MP_1 and MP_2 , experiences with higher magnitude rewards will be stored in MP_1 and experiences with lower magnitude rewards are stored in MP_2 because MP_1 experiences are more important but we keep MP_2 as well for more exploration just like ϵ -greedy exploration. In our case high magnitude rewards are only when player dies or eats food so we kept those experiences in MP_1 and all other experiences in MP_2 .

We gave a reward of -10 on colliding with the wall or enemy(in pacman)/itself(in snake) and a reward of 1 on eating the food. In all other cases there is a reward of 0.

Observations

We plotted the below graphs with x-axis as the number of training steps and y-axis as the average of 5 training steps window. The graph which is shown on the left has been trained on neural networks using the baseline model (i.e based on the reward earned throughout the game). The graphs on the right show the reward values obtained by learning from the model transferred from the other game.

The model obtained for Pacman show a sudden jump for the rewards at certain intervals which shows the instability of baseline Pacman. The overall reward graph of the transferred learned Pacman is higher than the self-trained Pacman as there are more 25+ average reward in transferred Pacman. We got better results in Pacman compared to snake probably because in Pacman the size of the player remains the same and in the snake, we observed that most of the time it dies by colliding with itself when the snake's body is in the way of food instead it should try to circumvent this. Also, the learning imparted by the snake game to the Pacman is similar to the self-trained pacman and vise-versa as the number of features in snake are lesser than pacman so for features present only in pacman it randomly initialises the weights which negatively affects the performance and nullifies the information learned.

Some of our observations in particular for the high scores observed are:

Baseline Pacman: 60

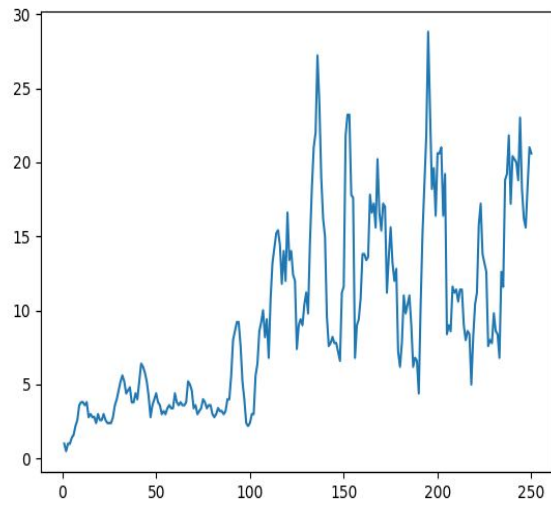
Pacman Transferred from Snake: 91

Baseline Snake: 33

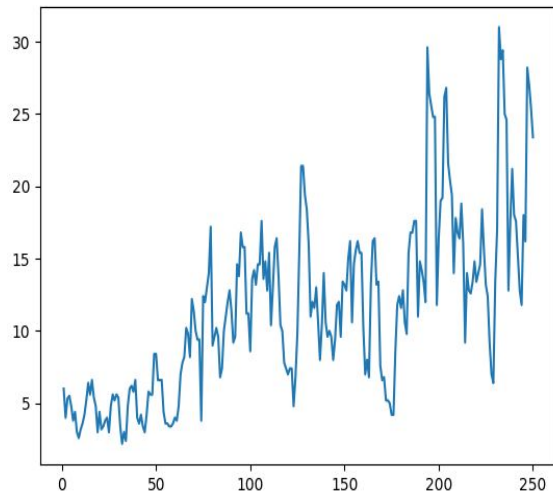
Snake transferred from Pacman: 37

Here we can see that transferred learned model performs better in high score and almost the same average score.

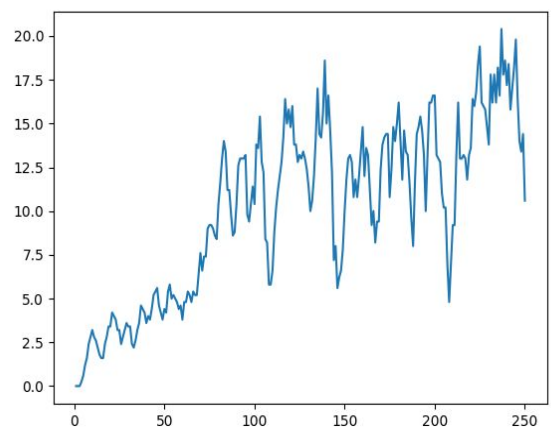
Pacman



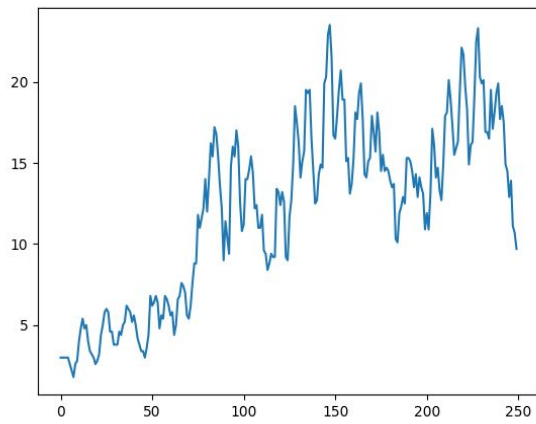
Pacman transferred from snake



Snake



vs Snake Transferred from Pacman



Future Work

In future, there is a lot of things which researchers could try to check if that would improve the performance of the model like using CNN instead of FC layers. Features can't extract all the information from the current state so instead of using features using the whole image is very much likely to improve the performance of the model.

There is also a lot of research happening right now to use prioritized experience replay in Reinforcement Learning tasks we could use that to improve the baseline as well as transferred model.

We could also check whether our model is able to remember the knowledge learned from the first model after using transfer learning to train the second model i.e. keeping a single network for both the games(snake and Pacman) of Y type structure where on one side is the action for snake and other side actions for Pacman, so we first freeze snake side of the model and training Pacman side and common part of the model and then training it on snake by freezing Pacman side of the model and training snake side & common part of the model using common part learned from the Pacman training and finally checking how it performs on Pacman in testing. This would tell us whether our agent is remembering what it learned in the past. For this task we need to use image as the input and not features because the snake has fewer features than Pacman so on switching agent might forget some of the information previously learned.


References

[1] [arXiv:1511.06342v4](#), Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning

[2]<http://cs229.stanford.edu/proj2016spr/report/060.pdf>, Exploration of Reinforcement Learning to SNAKE

[3] [arXiv:1312.5602v1](#), Playing Atari with Deep Reinforcement Learning

[4]<https://pdfs.semanticscholar.org/4a55/2a11f7cfe38cc06b516fa4e7e48a2b489692.pdf>, Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability



[5][http://www.ntulily.org/wp-content/uploads/conference/Autonomous Agents in Snake Game via Deep Reinforcement Learning_accepted.pdf](http://www.ntulily.org/wp-content/uploads/conference/Autonomous_Agents_in_Snake_Game_via_Deep_Reinforcement_Learning_accepted.pdf), Autonomous Agents in Snake Game via Deep Reinforcement Learning

[6]<https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf> Exploration of Reinforcement Learning to SNAKE

[7]<https://github.com/maurock/snake-ga> for snake implementation from scratch