Deep Q Learning for Space Invaders

Ruochen Liu (rl2841), Yinxiang Wu (yw2920)

Yifei Lin (yl3594), Vanessa Huang (yh2873)

1. Introduction

After running Air Raid on OpenAI gym and playing it using the environment, the Atari emulator for Air Raid, we noticed that there were many blank frames in observations. The environment returned blank frames even if it was running and responding to input action. We tried several different available environments, like "AirRaidNoFrameskip" environment, but the problem still remained. Since those blank frames carried no information, we have to switch our game into Space Invaders, which has a similar system and same action space.

During this project, Yinxiang and Ruochen built up and trained models on a laptop with a GPU instance of GTX850M and on AWS with a GPU instance of p2.xlarge at the same time. Yifei and Yuping added summaries and reports to trained models. There are many models we have tried, but we will just include some typical ones.

More specifically, we consider training an agent to play the Space Invaders, with the goal of achieving as higher scores as possible. The agent interacts with the environment given by the OpenAI gym for Space Invaders. The environment can respond to an action the agent chooses by returning a screen image as a state and a gained reward. The game play is discretized into time-steps and at each time step, the agent chooses an action from action space {0, 1, 2, 3, 4, 5} and the environment takes the chosen action and brings a new state and a reward. We can formalize the problem as follows:

- State: a RGB screen image of size (210, 160, 3) and it is represented by a 3-dimensional array.
- Action: an integer in the range of [0, 5]. 0 represents NOOP (no action), 1 represents SHOOT (shoot), 2 represents LEFT (move left), 3 represents RIGHT (move right), 4 represents LEFTSHOOT (shoot while moving left) and 5 represents RIGHTSHOOT (shoot while moving right).
- Reward: an integer from the set {5, 15, 20, 25, 30, 35} and a random bonus.

There are two environments used in this project:

- 'SpaceInvaders-v4': each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from {2,3,4}.

- 'SpaceInvaders-deterministic-v4': each action is repeatedly performed for a duration of k frames, where k is 3.

The main objective of this project is to implement deep Q-learning to train an agent that could play the Space Invaders. Some secondary objectives are to properly pre-process data and to explore effects of different architectures of convolutional neural networks (CNNs) on the performance of deep Q-learner.

2. Preprocess and Frame skipping

The raw observations are 210×160×3 RGB images. Given the color of objects, the landscape and the score bar contain nothing useful for choosing action. We convert raw images into smaller and gray scale ones. The widely-used output size is 84×84×1.

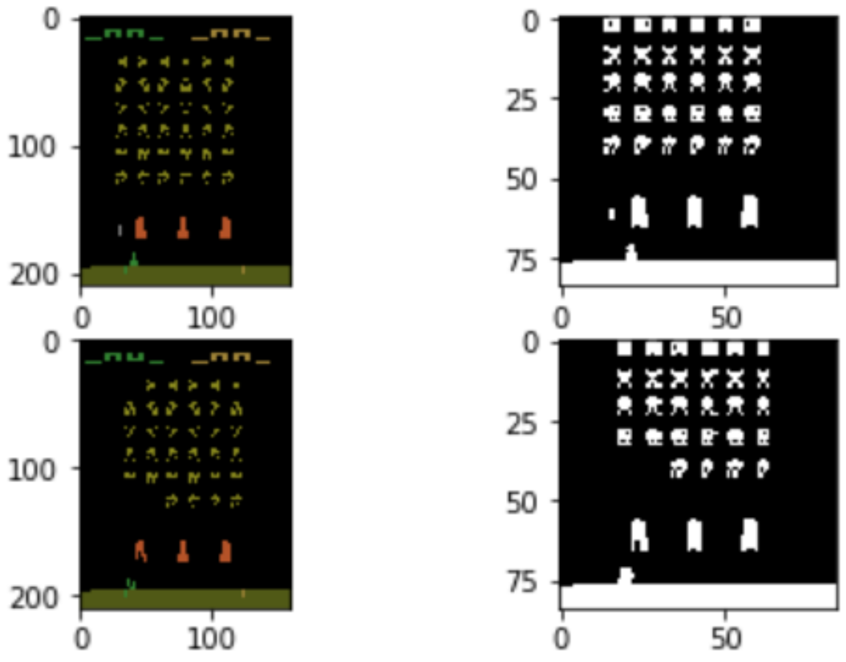

Figure 1: Grayscale images from preprocess[1].

Frame skipping[2] is a popular trick that has been used in many reinforcement learning algorithms for playing games. The idea is that the agent can only see and choose action for every k screen image and an action is repeated for (k-1) times until the agent sees the next screen image. This trick is useful for learning most video games. All video games, like the Space Invaders, display discrete screen images at a rate per second. For example, the Space Invaders returns 60 screen images per second. For players, it looks continuous because human

[1] Note: This is the preprocessing suggested by https://www.pinchofintelligence.com/openai-gym-part-3-playing-space-invaders-deep-reinforcement-learning/

[2] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning":1312.5602 (2013)

eyes cannot distinguish those discrete images at once. For the same reason, a specific sequence of consecutive screen images actually carries very similar information. Hence, it could be not only inefficient but uninformative for a deep Q-learner to learn from every time step. In this project, we used an environment from OpenAI gym for the Space Invaders, called “SpaceInvadersDeterministic-v4” as described above, in which a frame-skipping is used with k=3 as default. By contrast, the default number of frames skipped in SpaceInvaders-v4 is randomly chosen from {2,3,4}.

Apart from the in-built frame skipping, we also used a programmed frame-skipping[3]. We stacked every four “consecutive” screen images to construct an input to the neural networks to approximate the Q-function. For example, given four “consecutive” observations from the environment (x1, x2, x3, x4), each of size (84, 84), they are stacked together as a state of size (84, 84, 4). To choose action for x5, the Q-functions is calculated through the neural network with previous state as input and an action that maximizes the Q-function is chosen. Though stacking observations significantly increases the complexity of the network, it allows the Q-learner to learn from experience up to prior 12 steps, which is expected to have a better performance. We called this trick stack-frame.

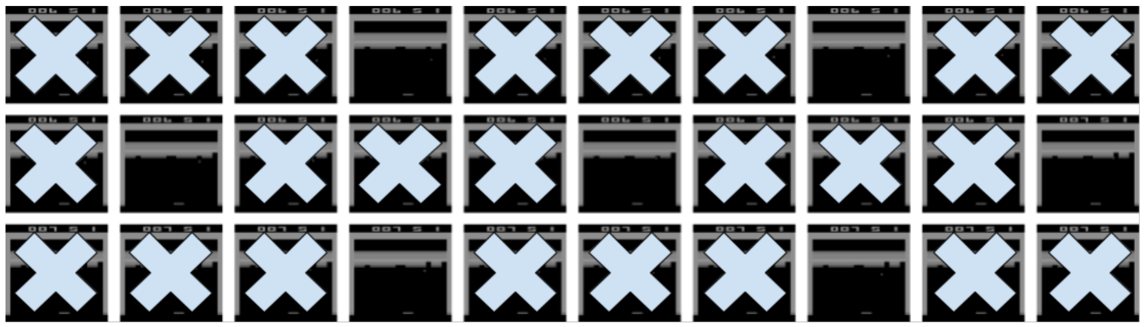
Figure 2: Consider the updated sequence of screenshots, with the skipped frames denoted with an “X” over them. https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

3. Trained Models

We used deep Q-learning for this project. The main procedure of implementing deep Q-learning is similar to the one for Cart-pole taught in class. However, unlike Cart-pole whose states are simply a vector, states in Space Invaders are images. Following the pre-processing and frame-skipping steps, we used convolutional neural networks (CNN) to extract features from the images and map them to corresponding actions. The main bulk of this project is to find out a proper architecture for the CNN and experiment different choices of hyperparameters. Below are several typical models we tried in this project.

3.1. Baseline

[3] Suggested by https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

The environment used to train this model is “SpaceInvader-v4”. Baseline model contains two convolutional layers with 8 and 16 as the number of channels and two fully connected layers with 128 and 6 as the number of outcomes. The model is trained for 1000 episodes.

3.2. Baseline with 3000 episodes

The architecture and environment settings of this model is the same as baseline model, but we trained this model for 3000 episodes.

3.3. Delayed Actions

The environment used to train this model is also “SpaceInvaders-v4”. Training takes 1500 episodes. We used a Q-learner with 4-layer neural network. There are 2 convolutional layers, with the number of respective channels equal to 8 and 16. Each convolutional layer is followed by a maximum pooling. Following a flattening step, the last two layers are fully connected layers, with size 32 and size 6. The difference between this model and other models is that the agent can only see and choose action on every three screen images instead of every screen image and its last action is repeated twice until next action is chosen. Due to the in-built frame skipping, this model is actually repeating an action over about twelve screen images.

3.4. Five layers with stack-frame

The environment used to train this model is “SpaceInvaders-determinisitc-v4”. Training takes 3000 episodes. We used the stack-frame as described above. The input image is of size 84×84×4. We used a Q-learner with a 5-layer neural network. There are three convolutional layers and two fully connected layers. The number of activation maps in three convolutional layers are 32 (filter 5×5 with stride 2), 32 (filter 5×5 with stride 2) and 64 (filter 5×5 with stride 2), each followed by a ReLU non-linearity. The first fully connected layer has 256 units. We also transform rewards to {0, 1}, i.e., getting 1 if a reward is greater than 0 and 0 if a reward is 0.

4. Results

4.1. Baseline

Because of random frame skipping rate, testing results are wavy. The average score is around 240, which is could be regarded as the baseline of test. The network is simple and not very heavy, so the training process only took hours. The simple network also prevents the player agent from being stuck with local optimal strategies, which is proved during the game play of our trained agent. The spaceship is moving around and shooting at aliens, and under its only strategy, it will shoot down all the aliens in the second column first.

Then it will start to look for a good chance to remove the first line. However, we find that the player agent is not that smart to avoid the bullets from aliens. We put it down to the random frame skipping during the training and testing process, which may cause missing bullets.

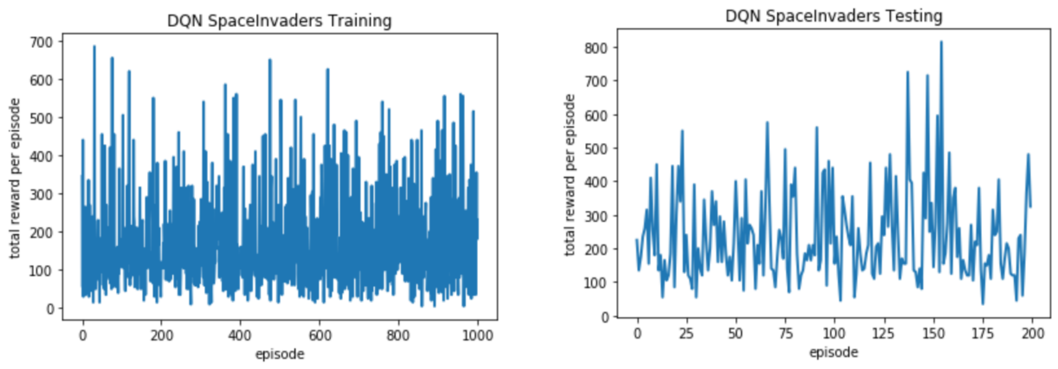

Figure 3: training and testing results for the baseline model.

4.2. Baseline with 3000 episodes

Though the baseline model performs reasonably well with the average score over 100 test episodes being about 240, we want to find a way to further improve it. The first adjustment is to increase the number of training episodes from 1000 to 3000. We expected that training based on more experience should have improved the model. However, it turned out the baseline model trained on 3000 episodes performed worse than the one trained on 1000 episodes. Another observation is that the agent seems to get specific amounts of rewards in a game. One possible reason is that the Q-Networks trained on 1000 episodes did not converge. Since we used the update formula[4] for epsilon, upon termination of 1000 episodes, the value of epsilon is about 0.46, which means that during first 1000 episodes, there are more exploration than exploitation. When the episode goes to 3000, the epsilon is almost 0.1 and the agent began playing following the learned policy. During testing procedure, the agent trained over 3000 episodes clearly is more confident in choosing actions and hence the plot looks less stochastic.

We were also thinking about why the agent did not perform that well, i.e., scoring over 300, even after 3000 episodes training. By observing gameplay by the trained agent. A guess is that we used original rewards from the environment so that shooting down different space ships leads to different rewards. Since the space ship with the highest reward is located at the last row of the enemy, the agent always tried to shoot as many that space ships as possible. However, this results in an agent that emphases too much on shooting instead of moving and avoiding being attacked. Therefore, this trained agent

[4] epsilon = 0.1 + 0.99×exp(-0.001×episodes), suggested by *https://jaromiru.com/2016/10/03/lets-make-a-dqn-implementation/*

usually died very quickly, though it can frequently shoot down the highest-reward space ship. We will experiment converting all rewards to 1 and hence eliminate differences between difference space ships.

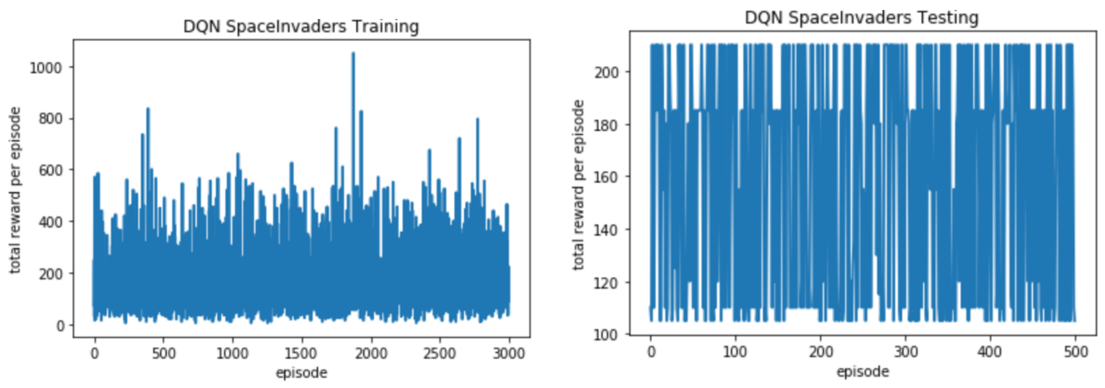

Figure 4: training and testing results for the baseline model with 3000 episodes.

4.3. Delayed Actions

The result of this model is very interesting since the agent can achieve 235 scores in every episode. By monitoring the gameplay by the trained agent, we observed that the agent has learned a playing strategy that is to move all the way down to the right, shoot while moving, keep shooting once the plane reaches the right end. Following this strategy, the agent can always shoot down some enemies in the first row and all enemies in the last column. However, the agent did not learn how to avoid lasers and could die very quickly. This happens possibly for the same reason described in last model. Another possible reason is that repeating an action for many times prevents the agent from learning and choosing action to avoid lasers. It can be also noticed from the plot of scores against episodes that scores achieved in an episode gradually converge to 235. This also makes sense because with epsilon going down, the agent tends to exploit the policy. Perhaps 235 is the maximum score the Q-learner can achieve under this specific setting.

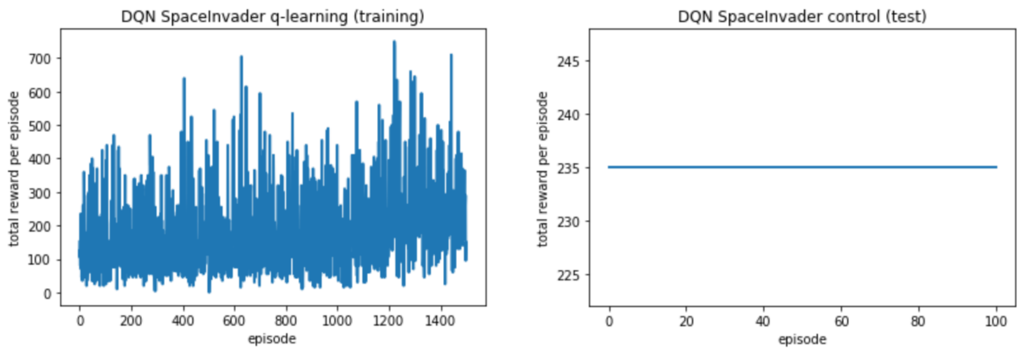

Figure 5: training and testing results for the delayed-action model.

4.4. Five layers with stack-frame

This is the most complex model we have tried. It can be seen from Figure 6 that there is a gradual growth in the total reward per episode and the highest reward is about 800. Unlike the other models, this model takes into account last four states to choose an action for next time-step. Since the agent can gather more information before choosing an action, it performed better than the other models on testing. In addition, it seems the agent has learned a strategy and been very confident in every action it chose. That is probably why during testing, the agent can gather the same score in every episode.

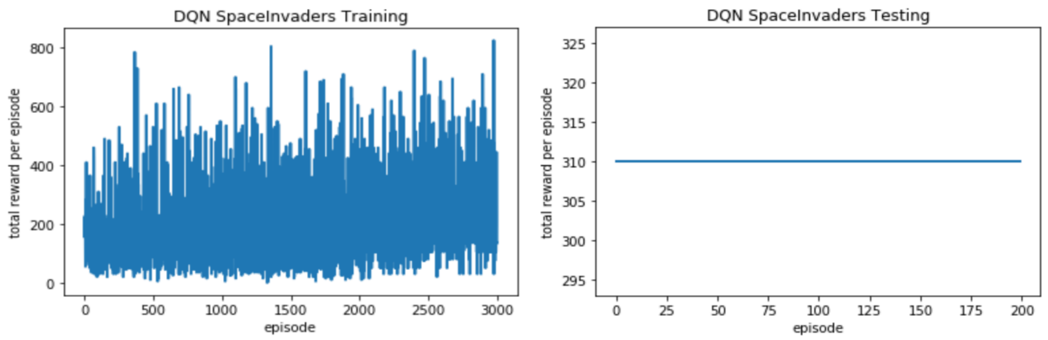

Figure 6: training and testing results for 5-layer with stack-frame.

Below is a table summarizing results of different models.

Table 1: summary of models

Model	Training Average	Testing Average	Training Time
Random action	150	150	N/A
Baseline+1000 ep	178	240	7 hours
Baseline+3000 ep	180	195	22 hours
Delay-action+1500 ep	195	235	12 hours
5-layer+stack-frame	205	310	45 hours

5. Conclusions

Compared with other model, the baseline model could be the simplest model we trained, which has only two convolutional layers and 1000-episode training process. However, it performed reasonably well on the gameplay. In terms of maximum average testing scores, the most complex model is better than others and its performance is quite stable.

Based on the baseline model, we tried training for more steps, on a more complicated neural network, and under different frame skipping rate. We find sometimes the agent is overfitting through heavy network because it could be stuck with the local strategies which are not good enough, so we need to choose the parameters of neural network wisely and try different structures. We also tried to use players' replay as the training start point but found that the agent can gather enough experience during training process without human experience.

We have done different experiments and show that Deep Q-learning algorithm can learn playing games. We explored different network architectures for deep Q-learning and found that the 5-layer plus stack-frame model (3 convolutional layers and 2 fully connected layers) outperforms other models. However, due to constraints of time and money, we did not try other possibilities.

Clearly, the biggest obstacle that prevents our models from doing better is that the agent did not learn how to avoid being attacked well. For future work, we want to further improve the pre-processing step so as to make the laser more visible and impose some penalties on the agent for being attacked.

There are certainly better models that can play extremely well for this game and even surpass human performance. We are still learning and experimenting. All in all, through this project, we do learn a lot about Deep Q-learning and have a deeper understanding of neural networks.