Pacman with Reinforcement Learning

Karim Kousa 202102687

I. Abstract

Various strategies for reinforcement learning based on the classic Pacman game; comparing Q-learning and Deep Q-learning in terms of the total reward. In large grid-layouts, finding the best policy with features from a convolution neural network (CNN) is inefficient. Features were incorporated into the game in Q-learning. However, Deep Q-Network (DQN) can implicitly "extract" significant features and interpolate Q-values of enormous state-action pairs without referring to large data tables, which is a more powerful capability. The fundamental motivation behind this undertaking is to explore the viability of Deep Q-learning in light of the setting of the Pacman game having Q-learning and as a benchmark.

II. Introduction

The Atari 2600 was the first video game console to be widely popular with gamers and featured a slew of well-known titles. It was released in 1977. Ms. Pacman is an arcade video game that was released in 1981 and is a "no-authorized" sequel to the original Pacman game. Namco licensed it as an official title after it was a resounding success. The game itself has a lot of similarities to Pacman. The difference is that this game's protagonist is a woman, and it also created new maze designs and the gameplay.

I want to teach the Pacman agent to act smartly by eating food and avoiding ghosts as much as possible (in order to get higher scores). I am working on this project with the Gym environment because, in addition to wanting to conduct reinforcement learning, build a neural network of my own and apply a convolutional neural network (CNN) trying to use the features extracted from it, I find it appealing to see a trained Pacman agent.

III. Problem

First we need to understand our environment. We have the environment as a RGB image that is displayed to human players as an observation. Image is an array of (210,160,3) dimensions.

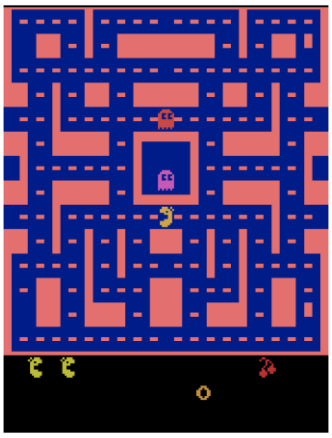

Figure 1: Pacman gym environment

The game involves navigating Pac-Man through a maze. The objective is that Pac-Man eat all of the pellets (circles), while avoiding the ghosts that pursue him.

IV. Reinforcement Learning Solution

In reinforcement learning three main things that have to be defined. Action, State and Reward.

In the Gym Pacman environment, we have 9 actions that could be used to take a choice for the next move

0	NOOP
1	UP
2	RIGHT
3	LEFT
4	DOWN
5	UPRIGHT
6	UPLEFT
7	DOWNRIGHT
8	DOWNLEFT

Table 1- Actions

Pacman with Reinforcement Learning

Karim Kousa 202102687

We have the option to use only 4 actions from them, which are the actions 1,2,3 and 4, they also make more sense in this problem.

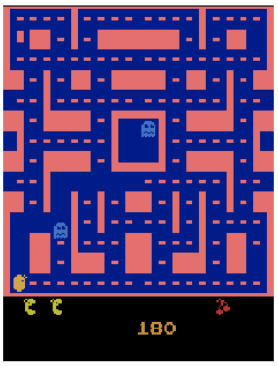

Figure 2: An image of a state

The main focus when observing the state is the position of Pacman and the ghosts, how many lives still remaining, pellets eaten already and current score. A final state is where the game ends and there are mainly two possible outcomes:

- Pac-Man was able to eat all the pellets without running out of lives.
- The ghosts were able to eat Pac-Man three times before Pac-Man was able to eat all the pellets.

Reward is mainly getting a positive reward for eating a pellet and a negative reward when getting eaten by a ghost.

Two different approaches where taken. The first method trains with a normal Q-network algorithm, the second method trains with a Deep Q-Network (DQN) algorithm. Both some image preprocessing where in both methods the image was cropped in order to get faster training. In the first approach (method) the image is cropped and downsized, normalized between -128 and 127, saved into memory as int8 type and this gives an image of (84,80,3) dimensions, also only 4 actions were considered to be taken in this method. In the second method the mage was also cropped and downsized, converted to grayscale, then the contrast of the image was improved, normalized between -1 and 1 and this give an image of (88,80,1) dimensions. One training step in the first method is one move or action taken by Pacman, while in the second method one training step is one complete game.

After that in both methods the image was passed to a CNN. The first method had a CNN of the shape:

- An 8 x 8 filter applied to the (84,80,3) image with 4 strides and no padding.
- A 4 x 4 filter applied to the result of the previous step with 2 strides and no padding.
- A 3 x 3 filter applied to the result of the previous step with 1 stride and no padding.
- The result of the previous step is reshaped to a 7040 vector.
- The result of the previous step is passed to a fully connected neural network with Relu as an activation function will give a 512 vector.
- The output layer is made out of 4 neurons to choose the action.

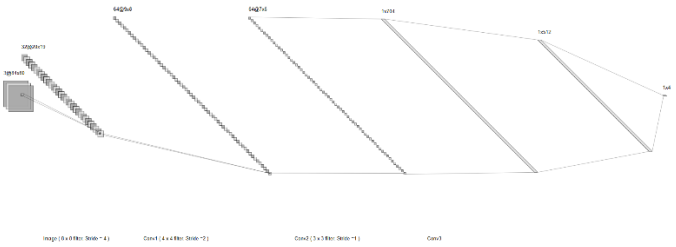

Figure 3: First method CNN

The second method is the same except for the image dimension which is (88,80,1) and the output layer consists of 9 neurons as all the possible actions can be chosen. In both methods batches were used to train.

Pacman with Reinforcement Learning

Karim Kousa 202102687

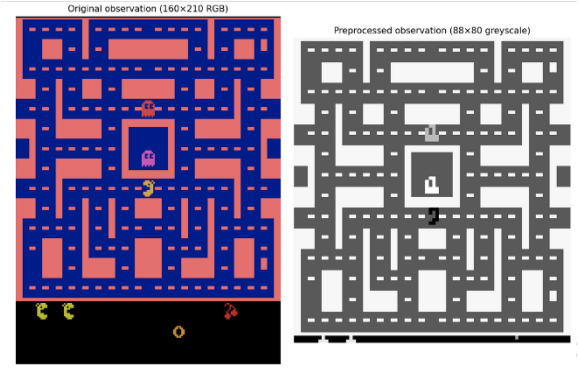

Figure 4: Original image and grayscale image

From the CNN we get online_q_values, online_vars, target_q_values and target_vars.

The first approach (method) had Reward function where epsilon was calculated by:

epsilon = max(epsilon_min, epsilon0 - (epsilon0 -epsilon_min)(step / eps_decay_steps))*

And based on that equation an optimal or a random action was taken. Determining the action using an e-greedy policy on online q_net.

q_vals_eval = q_vals_online.eval(feed_dict={ X_state: state.reshape(1, input_height, input_width, n_channels)})[0]

After the online q_net was trained and estimating the q_online values using reward and q_target value.

q_target_eval = q_vals_target.eval(feed_dict={X_state: batch_next_state})#[batch_size, n_output

q_target_eval_max = np.max(q_target_eval, axis=1).reshape(-1, 1) # [batch_size, 1]

*q_est = batch_reward + discount_rate * q_target_eval_max * batch_continue*

loss_eval, _, b_grads_norm = sess.run([loss, training_op, bottom_grads_norm],feed_dict={X_state: batch_state, X_action: batch_action.reshape(-1), y: q_est}

In the second method

*q_value = tf.reduce_sum(online_q_values * tf.one_hot(X_action, n_outputs), axis=1,keepdims=True)*

error = tf.abs(y - q_value)

clipped_error = tf.clip_by_value(error, 0.0, 1.0)

*linear_error = 2 * (error - clipped_error)*

loss = tf.reduce_mean(tf.square(clipped_error) + linear_error)

optimizer = tf.compat.v1.train.MomentumOptimizer

training_op = optimizer.minimize()

Epsilon was calculated the same way as the first method.

Online DQN evaluates what to do:

q_values = online_q_values.eval(feed_dict={X_state: [state]})

action = epsilon_greedy(q_values, step

Online DQN plays:

obs, reward, done, info = env.step(action)

next_state = preprocess_observation(obs)

Sample memories and use the target DQN to produce the target Q-Value:

X_state_val, X_action_val, rewards, X_next_state_val,continues = (sample_memories(batch_size))

next_q_values = target_q_values.eval(feed_dict={X_state: X_next_state_val})

max_next_q_values = np.max(next_q_values, axis=1, keepdims=True)

Pacman with Reinforcement Learning

Karim Kousa 202102687

*y_val = rewards + continues * discount_rate * max_next_q_values*

Train the online DQN:

_, loss_val = sess.run([training_op, loss], feed_dict={

X_state: X_state_val, X_action: X_action_val, y: y_val})

The first method has the following parameters used:

Name of Variable	Value	Description
max_mem_size	2000	
epsilon0	1.0	
epsilon_min	0.9	
eps_decay_steps	2000000	
learning_rate	0.001	
n_steps	500000	Number of steps to train for
n_steps_train	15	Train online q-net after these many steps
n_steps_copy	3000	Copy online net to target after these many steps
discount_rate	0.999999	
batch_size	3	
skip_start	85	Skip the start of every game (it's just waiting time in Pacman)
training_start	60	Start training after these many steps

Table 2 - First method parameters

Name of Variable	Value	Description
learning_rate	0.001	
momentum	0.95	
eps_min	0.1	
eps_max	1.0	
eps_decay_steps	2000000	
n_steps	500000	Total number of training steps
training_start	10000	Start training after 10,000 game iterations
training_interval	4	Run a training step every 4 game iterations
save_steps	1000	Save the model every 1,000 training steps
copy_steps	1000	Copy online DQN to target DQN every 10,000 training steps
training_start	10000	Start training after 10,000 game iterations
training_interval	4	Run a training step every 4 game iterations
discount_rate	0.99	
skip_start	90	Skip the start of every game (it's just waiting time)
batch_size	50	
iteration	0	Game iterations
gcheckpoint_path	"./my_dqn.ckpt“	
max_mem_size	2000	
done	True	
loss_val	np.infty	
game_length	0	
total_max_q	0	

Table 2 – First method parameters

First method steps:

- Determine the action using e-greedy policy on online q_net
- Make 1 training step = 1 step of Pacman.
- Update the action every 8 life steps

- Execute the action
- Record this step in game memory
- Train the online q_net
- Queue the loss
- Store the model
- Q-Net

Second method steps:

- Play Pac-Man Using the DQN Algorithm.
- Make 1 training step = 1 full game.
  - Iteration, Training step, Loss, Mean, Max-Q.
  - Skip the start of each game
  - Online DQN evaluates what to do
  - Online DQN plays (does one move)
  - Memorize what happened
  - Compute statistics for tracking progress
  - Sample memories and use the target DQN to produce the target Q-Value
  - Train the online DQN
  - Regularly copy the online DQN to the target DQN
  - Save regularly

V. Results

The results were saved at the end in .mp4 format videos. In the first method while training we print the mean of last 10 train losses, bottom gra ds norm , etc. Which looks like:

300 : 0.8 {3: 15} 438.00638 600 : 0.4 {3: 15} 442.93353

In the second method while training printing will show the following:

Training step 340899/500000 (68.2)% Loss 4.287439 Mean Max-Q 16.529545

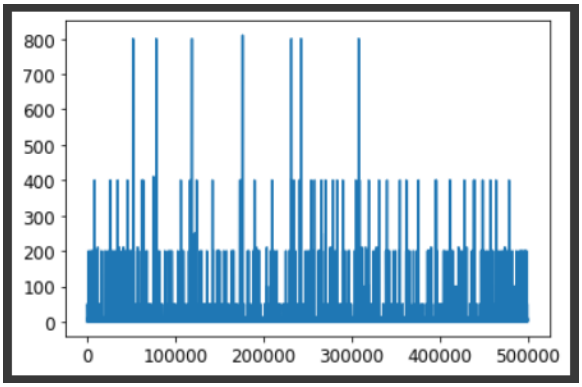

Figure 5: Reward per step of first method

The Gym Pacman environment was done with 10 random runs and the videos were observed and chose the video that represents the average score out of them, the score was 240. The first method gave a score of 350, the second method gave a score of 440. No method was able to win the game.

Figure 6: Result of random run

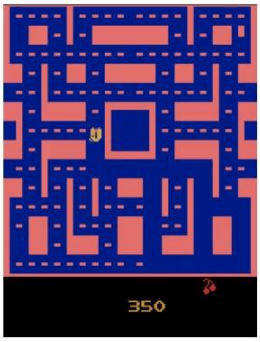

Figure 7: Result of first method

Pacman with Reinforcement Learning

Karim Kousa 202102687

Figure 8: Result of second method

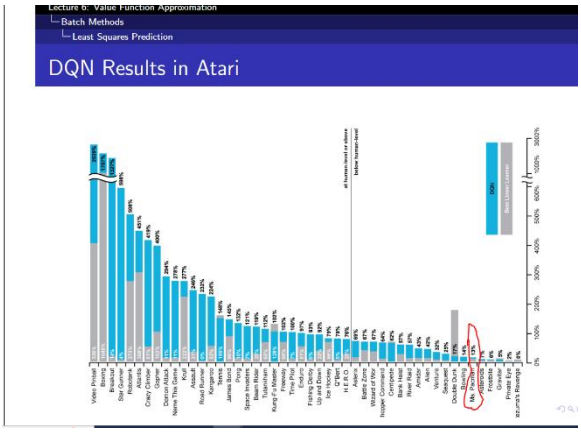

Figure 9: DQN in Atari

VI. Conclusions

Gym Pacman environment is very difficult to train as it needs a lot of time and computational resources. While training the second method, I ran out of RAM after 340899 steps while trying to train for 500000 steps. It is also better to use Gym environment with games of behavior or nature of linear environments like the game Riverraid.

Since the second method gives the best result (highest score) this means that DQN gives the best results and training with 1 full game as one step is better than training with only a single action of Pacman. The second method performed better but needs much more memory even though it is using a much smaller image, which leads to the conclusion that the image size to train is not the main issue to consider when applying reinforcement learning with Pacman in the Gym environment.

The DQN Results in Atari graph shows clearly how the Pacman environment will not give the best results easily.

VII. References

1. https://www.gymlibrary.dev/environments/atari/ms_pacman/
2. https://www.gymlibrary.dev/environments/atari/
3. https://atariage.com/software_page.php?SystemID=2600&SoftwareLabelID=924
4. https://notebook.community/me-surrey/dl-gym/16_reinforcement_learning
5. https://medium.com/analytics-vidhya/how-to-train-ms-pacman-with-reinforcement-learning-dea714a2365e