DQN Reinforcement Learning to play

Space Invaders

ISY5005 Intelligent Software Agents: Practice Module 2

Team members:

- Gu Lijian
- Li Sheng

catalogue

Abstract

In this practice module, we explore algorithms that use reinforcement learning to play the game space invaders. The Q-Learning algorithm for reinforcement learning is modified to work on states that are extremely high dimensional(images) using a convolutional neural network and is called the Deep-Q learning algorithm. We also experiment with training on states represented by the RAM representation of the state. We also look at an extension of Q-learning known as Double Q learning, explore optimal architectures for learning.

Problem

In this project, our target is to train an agent to play Space Invaders (a popular Atari game). To meet the target, we have two problems need to solve:
First, analyze how different representations of the input (image frame pixels v/s emulator RAM state) impact training time and performance. improvements to CNN architectures and the Fully connected networks for the RAM states.
Second, find how to get a high score in the game using Deep Reinforcement Learning.
It can be summarized as follows: train an agent to interact with an environment in a sequence of actions, observations and rewards.

- Environment: The Atari emulator for Space Invaders.
- States: A sequence of observations, where an observation, depending on the state space we are operating in, is a matrix representing the image frame or a vector representing the emulator's RAM state.
- Action: An integer in the range of [1; L]. In the case of Space Invaders L = 6 and the actions are FIRE (shoot without moving), RIGHT (move right), LEFT (move left), RIGHTFIRE (shoot and move right), LEFTFIRE (shoot and move left), NOOP (no operation).
- Reward: Reward returned by the environment, including Aliens (5, 10, 15, 20, 25, 30 points) and Mothership (350 points).

Systems Design

Approach

In this project, we use OpenAI gym to emulate the Atari Game Space Invaders. This emulator provides with environment to play space invaders. The environment represents each state by a raw RGB image of the screen which has a size (210; 160; 3) (SpaceInvaders-v0). Our task is to learn a policy for the agent so that the agent can play the game. In order to accomplish

this task, we use reinforcement learning. However, learning from high dimensional representation of states such as images is difficult for traditional reinforcement learning techniques with handcrafted features. Instead, we use an approach which uses DQNs and extend their approach in a few ways. We introduce the concept of Deep Q-Networks, experience replay which are foundation of our approach.

Deep Q-Networks

During handle of high dimensional data such as an image, it is hard to extract features manually since it is not obvious what the features are relevant to extract from the image. To handle high dimensional inputs where the agent learns from raw inputs such as image frames, we introduced Deep Q-Networks which uses a convolutional neural network to approximate the Q-function. If the state is n dimensional and the number of actions is m then the CNN is a mapping from R^n to R^m. We refer to a neural network function approximator with weights θi as a Q-network. The objective is to now learn the weights θi Q-network.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Formular. 1

With DQN we can save the last N experience tuples in replay memory.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Fig. 1

Hyperparameters

In order to train the networks, we tried the Adam algorithm with learning rate of e^{-4}. The policy behavior during training was ε - greedy with ε annealed linearly from 1 to 0.1 over the duration of the training. We experimented with replay memory of varying sizes, from 100K to 1M but saw little difference in running time or performance of the model. Also, we decided to set window size as 3 which will make 3 observations concatenated to form a "state".

Proposed Model & Parameters

We implement the CNN with some variations in the last layers and using ADAM. The first attempt was a simple CNN with 3 Conv 2D layers, with the Q-Learning algorithm, we obtain a slow learning process for Space Invaders. Then, we try modifying using Dense 512 and 256 networks at the last layer with linear activation and relu.

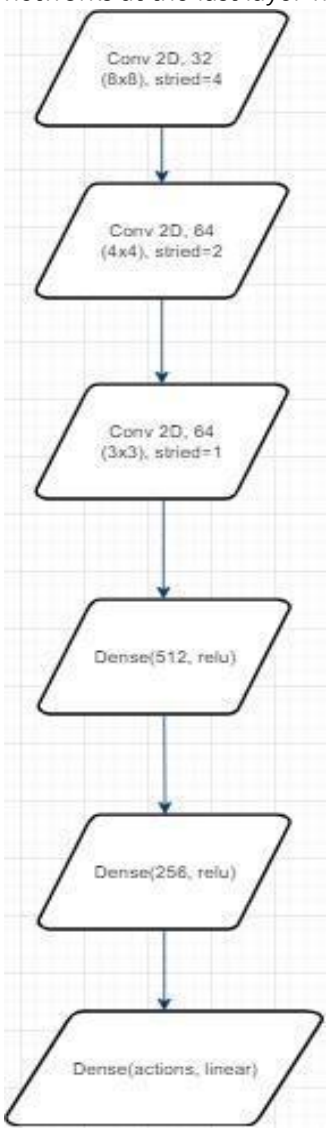

Fig. 2 Model

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 3, 51, 39, 32)	6176
conv2d_7 (Conv2D)	(None, 3, 24, 18, 64)	32832
conv2d_8 (Conv2D)	(None, 3, 22, 16, 64)	36928
flatten_2 (Flatten)	(None, 67584)	0
dense_8 (Dense)	(None, 512)	34603520
dense_9 (Dense)	(None, 256)	131328
dense_10 (Dense)	(None, 6)	1542

Total params: 34,812,326
Trainable params: 34,812,326
Non-trainable params: 0

Fig. 3 Parameters

Results

Scoring summarize comparison shows to us that for 1m steps trained model the testing outcome is higher for average, lowest and highest score to 10k steps trained model.

Model trained For	Avg score	Low score	High score	Observation
10k steps	128	5	355	At this stage the model is still learning what to do, figuring out what is a wise move and what isn't based on reward.
100k steps	168	35	560	The agent started to learn to dodge, it's getting more effective than before.
1m steps	270	40	660	Besides learn to dodge and take cover, now agent realized it can maximize reward by shooting down alien craft, also agent starts to aware by shooting alien mothership, it will get high reward compare to alien normal craft.

Fig. 4

We can see obvious improvements compare 1m steps trained model to 10k steps trained model that agent has improved strategy including:

- learn to dodge & take cover
- try to shoot alien craft more pro-actively
- put more priority on shooting mothership to get higher rewards. (Fig. 5)

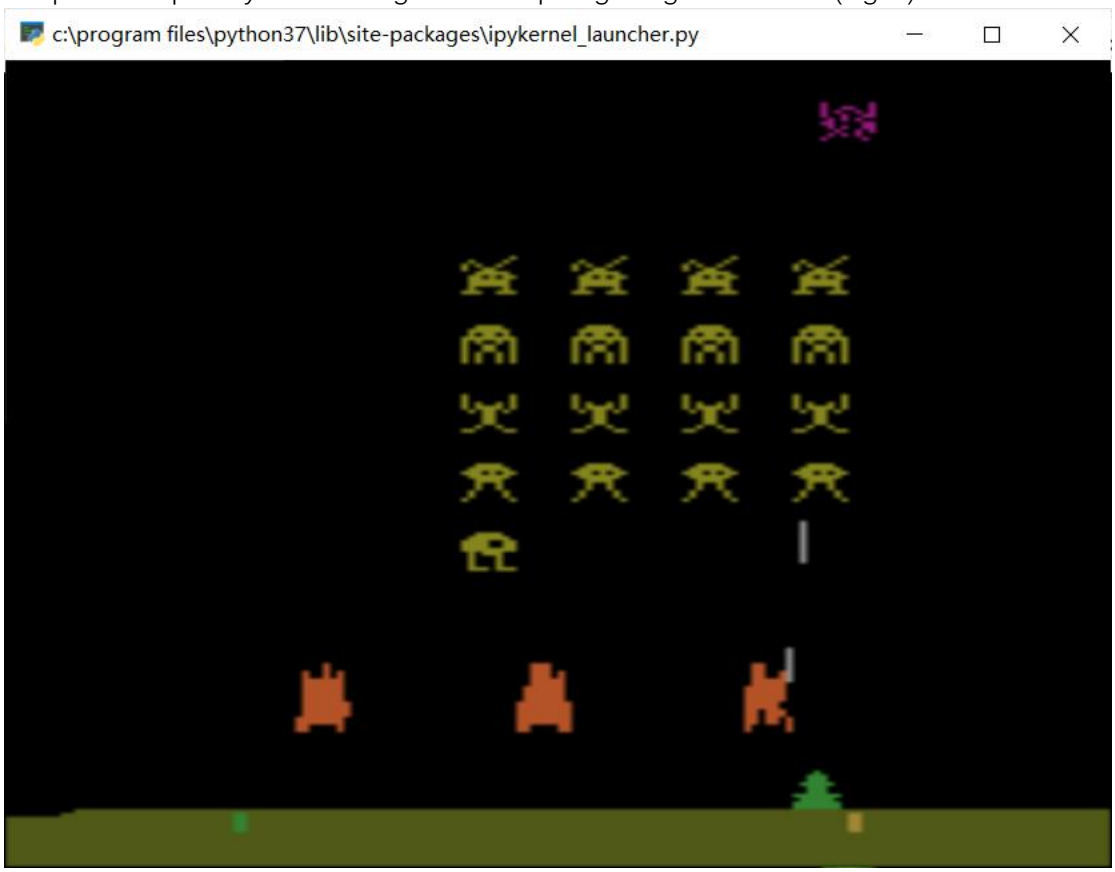

Fig. 5

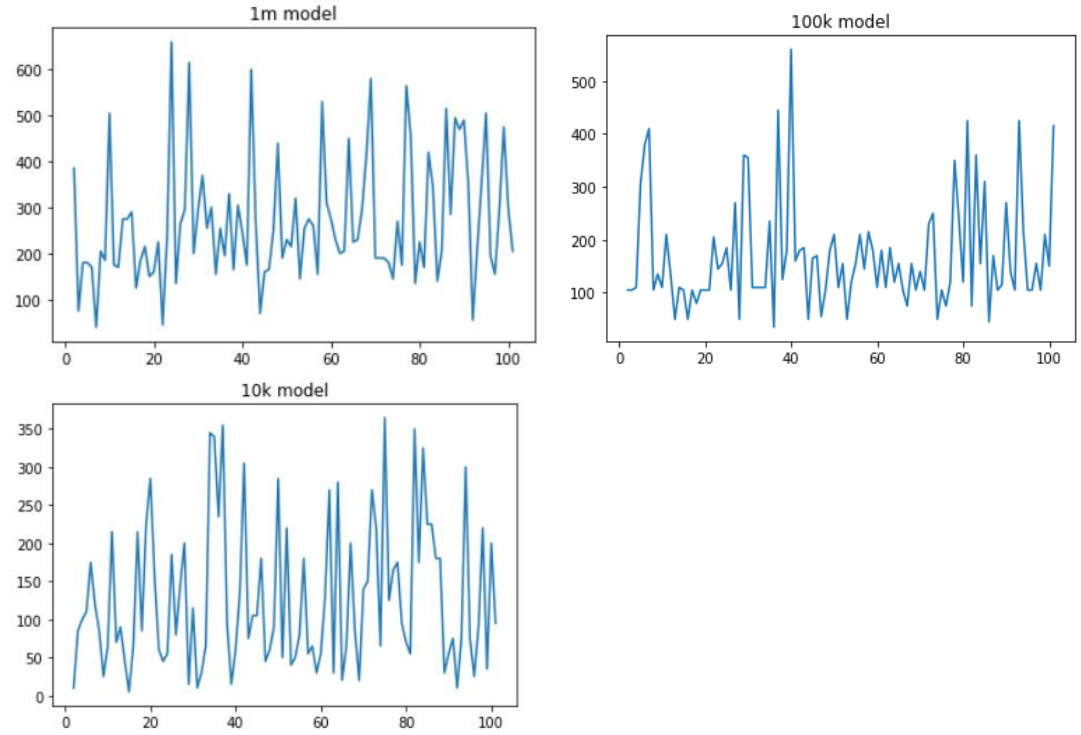

Fig. 6

In Fig.6, Plot of Scoring distribution between 10k, 100k and 1m model by executing 100 testing for each.
We can see clearly 1m model is able to gain higher score in aspects including min max and average.

Understanding/findings and future work

We have done different experiments that show that the Deep-Q learning algorithm can learn from gameplay even with very high dimensional state representations. It's critical to have more steps training which does help in learning better policies but one must keep checkpoints of the weights stored every 100K or so steps because early stopping seems to be crucial in learning a good policy.
It seems that training a Deep-Q learning algorithm well is quite difficult and seems to depend a lot on choosing a good explore vs exploit policy.
Due to computing resource limitations, we were not able to train and build a model with 10m and 50m steps. Each time by using Colab GPU we are able to train maximum 25K steps, by loading the pre-trained model and retrain again we can gradually increase the model trained steps, in future we can target to reach a 10M steps model by using accumulated training process.