### LAB 4: Deep Q Network

Tai-Hsiang Huang (mtk\_06584)

MediaTek Inc.

1. **Introduction**

In this assignment, a Reinforcement Learning (RL) based neural network is designed and constructed to play CartPole-v0, which is gaming environment in OpenAI-gym (Section 2.1). Specifically, the Q-learning architecture is adopted with a deep neural network as the learning kernel. The correlations and variance of the updates are break and reduced with an experience replay mechanism.

The rest of this report is arranged as follows. The experiment setup would be described in Section 2. The experimental results and discussions are described in Section 3.

1. **Experiment Setup**

This section describes the setup of the experiment. The gaming environment is introduced in section 2.1, and the methodology is described in sections 2.2.

## **Gaming Environment — CartPole-v0**

In the CartPole-v0 gaming environment, a pole, is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent if from falling over. A reward of +1 is provided for every time step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

The actions, observation, and reward of this gaming environment are described as follows:

* Two discrete actions:
  + 0(left) or 1(right)
* Four observations:
  + Cart position, Cart velocity, Pole angle, and Pole velocity at Tip
* Reward:
  + +1 every timestep

## **Methodology**

Fig. 1 shows the pseudo-code of the implementation, which exactly follows the requirements of the assignment. The details are described in the following subsections.



**Fig. 1.** Pseudo code of the Deep Q learning.

## **Network Structure & Loss Function**

The neural network adopted in this assignment consists of two fully connected layers with a ReLU lay in between of them. The details are as follows:

* Input (4 elements)
  + Observations from the CartPole-v0
* FC1 (fully connected layers, input:4, output: 32)
  + Weight init: Kaiming He’s weight init
* Relu1
* FC2 (fully connected layers, input:32, output: 2)
  + Weight init: Kaiming He’s weight init
* Output (2 elements)
  + Actions

The loss function adapted in this assignment is Adam with learning rate 0.0005.

## **Epsilon-Greedy Action Select Method**

The Epsilon-Greedy algorithm is adopted for action selection. That is, with probability , instead of the action decided by the neural network, a random action would be selected.

In my implementation, the is initialized with 1, and it decays from 1 to 0.1 with a decay ratio of 0.995 for each epoch. That is, we tend to trust more on the decision made by the neural network as the network becomes mature (knowing how to play the game). Nevertheless, a random mechanism is still needed so that the network has the chance to improve itself by the possibly better random selections.

## **Training process of deep Q Learning**

Basically, two networks with exactly the same network structure are used in this implementation: Target\_Net and Eval\_Net. The Target\_Net is used to make the decision based on the input data from the gaming environment. During the gaming process, a transition data consists of following information would be recorded in the memory (with experience buffer size of 5000) for each step:

* The before state *st*
* The action *at*
* The reward *rt*
* The after state *st+1*

After that, a set of Batch-size transitions would be randomly sampled from the memory and be used as the training data of the Eval\_Net, which aims at minimizing the difference between the expected and real Q values. Specifically, the pseudo codes are as follows:



The detailed hyper-parameters of the training are listed in Table I.

## **How the whole codes work**

The game would be played 1000 times (Training episode), and the Eval\_Net would be trained and updated 1000 times as well. After each training, the Target\_Net would be replaced by the newly trained Eval\_Net.

The gaming video of the training process would be recorded and the episode rewards of 100 testing episodes would be plotted.

1. **Experimental Results and Discussion**

Fig.5 shows the disentanglement result of VAE is shown in. It can be seen that each latent code does represent a special generation factor. Take the results inside the red box for example. This latent code is corresponding to the horizontal scaling ratio, and a larger latent code represents a larger compression ratio.

Fig. 6 shows the results of DCGAN, where, the better results are highlighted with a red boxes. If looking from a distance, most of the results look like a face. However, when taking a close look, it is easy to observe that these are not the face of real people. Most of the generated faces look like a combination from several different faces. I think it probably because that the loss function is computed in the pixel domain, which does not well capture the high level structure of a face.

TABLE I

Beta-VAE Training Hyper-Parameters

|  |  |
| --- | --- |
| Batch size | 128 |
| Training episode | 1000 |
| Experience buffer size | 5000 |
| Optimizer | Adam |
| Learning rate | 0.0005 |
| Discount Factor | 0.95 |
| Training Episode | 1000 |
| Update target network | Every 50 iterations |



**Fig. 5.** Disentanglement with beta-VAE.

However, none of them looks like real people. It seems that most of the results are finally converge to one or two specific faces, and the results are blur than those of DCGAN. I am not sure if it is because that the is computed in the feature space instead of real pixel.

1. **My Findings**

The competitiveness of the Generator and Discriminator is extremely import to train a GAN model. Otherwise, the network could be fairly unstable. That is exactly why it takes me a lot of time to implemente the VAE-GAN model. Nevertheless, it is good experience to me, and I have learn a lot from this assignment.