Lab 9: Deep Q Network

Lab Objective:

In this project, you need to design/construct and train a neural network that is able to play Breakout. The training method is deep Q-learning, which is a variation of Q-learning. The input of the neural network consists of 4 consecutive frames and the output is $Q(s,a)$ for each action.

Lab Description:

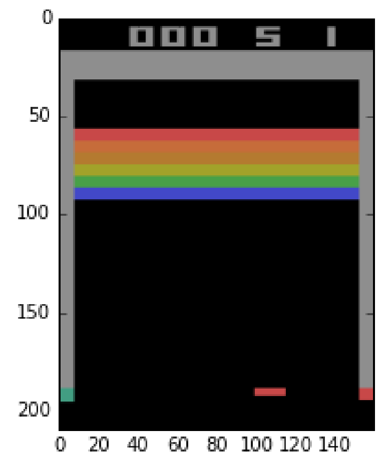
- Learn how to combine Q-learning with neural network to do reinforcement learning with raw input data
  - Using experience replay mechanism to break the correlations and reduce the variance of the updates
- Understand the training workflow:
  - Image processing (rgb to grayscale, image resize)
  - Design Structure of Neural Network
- How to apply Deep Q-learning algorithm to video games (e.g. Breakout).
- Please hand in your source code and report, and demo to TAs.

Environment Setup:

- Python 2.7 or 3
- Tensorflow $\geq$ 1.0
- Openai-gym
  - pip install gym[all]

Game Environment – Breakout:

- Introduction: Breakout is an arcade game developed and published by Atari, Inc. In the game, a layer of bricks lines the top third of the screen. A ball travels across the screen, bouncing off the top and side walls of the screen. When a brick is hit, the ball bounces away and the brick is destroyed. The player loses a turn when the ball touches the bottom of the screen. To prevent this from happening, the player has a movable paddle to bounce the ball upward, keeping it in play.
- Actions: 0 (noop), 1 (fire), 2 (left) and 3 (right)
- Reward
  - Hit a brick: +1
  - Die: -1



Breakout

Implementation Details:

Network Architecture

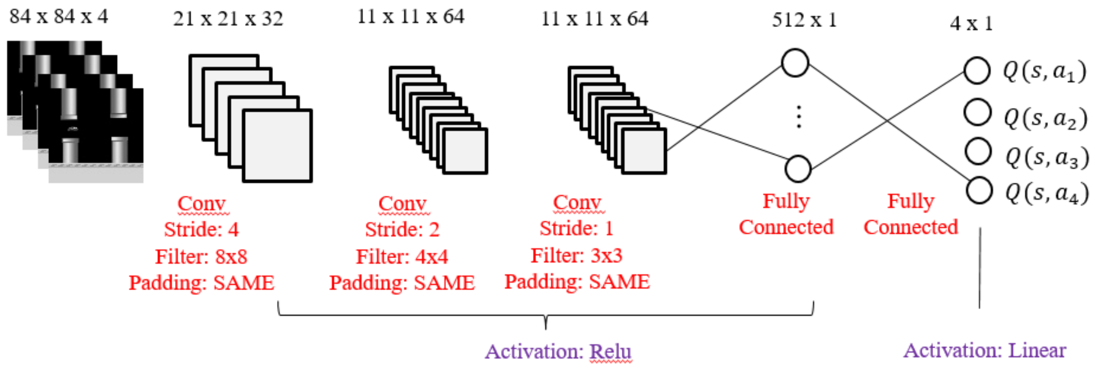
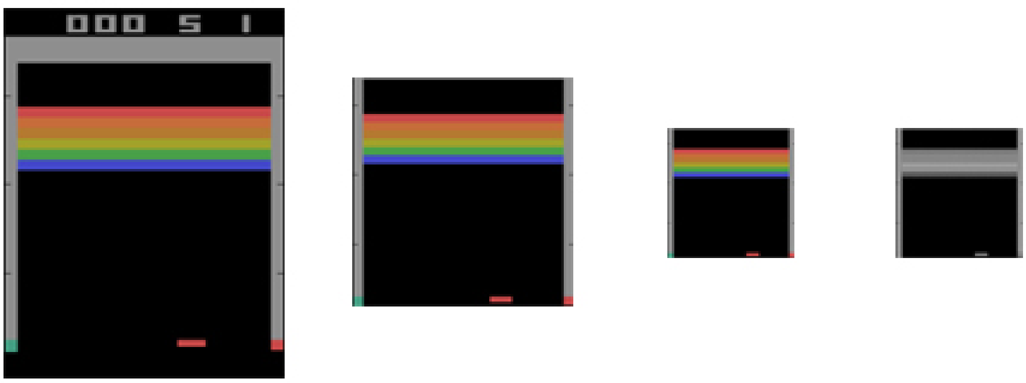


Image Processing

1. Original image size: 210x160
2. Crop the image to 160x160
3. Resize the image to 84x84
4. Convert the image from rgb to grayscale



Sample of Image Processing

Training Arguments

- Optimizer: RMSprop
  - Learning Rate: 0.00025
  - Decay: 0.99
  - Momentum: 0.0
  - Epsilon: 1e-6
- Epsilon: 1 → 0.1 (anneal linearly from 1.0 to 0.1 over first 500000 steps)
- Batch Size: 32
- Experience buffer size = 300000
- Gamma(Discount Factor): 0.99
- Training Episode: 10000
- Update target network every 10000 steps

Misc.

◆ Training Time: approx. 1 day on GTX960

<u>Requirements:</u>

1. Implement Deep Q Network
   - Construct the neural network
   - Target value, loss function
   - Action prediction with DQN
2. Implement Deep Q-learning learning algorithm
   - Replay buffer
   - Update algorithm for the network

<u>Rule of Thumb:</u>

1. The performance should greatly improve after training about 1000 episodes. (reaching 30 points)
2. Don’t set replay buffer size too big. If it costs more memory than your RAM, training process would become very slow.

Methodology:

Algorithm - Deep Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

For $episode = 1, M$ do

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\varphi_1 = \varphi(s_1)$

  For $t = 1, T$ do

    With probability ε select a random action a_t

    otherwise select $a_t = \underset{a}{\operatorname{argmax}} Q(\varphi(s_t), a; \theta)$

    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 $\varphi_{t+1} = \varphi(s_{t+1})$

    Store transition $(\varphi_t, a_t, r_t, \varphi_{t+1})$ in D

    Sample random minibatch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{for terminal } \varphi_{j+1} \\ r_j + \gamma \max_{a'} Q(\varphi_{j+1}, a'; \theta) & \text{for non terminal } \varphi_{j+1} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\varphi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\theta^- = \theta$

  End For

End For

Scoring Criteria:

- Report (70%)
  - A plot shows episode rewards of 10000 training episodes (35%)
  - Explain your deep Q network implementation
    - Network Structure (10%)
    - Loss function (15%)
  - Describe the way you implement update_target_network() (10%)
  - Explain how you implement the training process of deep Q learning
    - Populate replay memory (5%)
    - Select actions (5%)
    - Update Epsilon (5%)
    - Prepare minibatch for network update (5%)
- Performance – Highest episode reward during training (20%)
  - $\geq$ 40 points : 100%
  - $\geq$ 35 points : 90%
  - $\geq$ 30 points : 80%
  - $\geq$ 25 points : 70%
  - $\geq$ 20 points : 60%
  - $\geq$ 15 points : 50%
  - $<$ 15 points : 0%
- Upload any one video during the training process (5%)
- Upload the last one tensorflow checkpoint file (5%)

References:

[1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[3] (Github) asrivat1/DeepLearningVideoGames

[4] (Github) dennybritz/reinforcement-learning