# ADL x MLDS 2017 Fall HW3 - Game Playing

2017/11/19

adlxmlds@gmail.com

# Outline

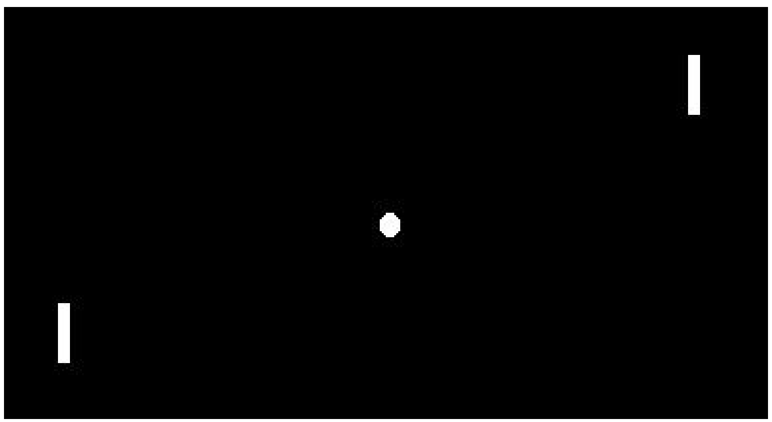
- **Introduction**
  - Game Playing
  - Game Type
- **Deep Reinforcement Learning**
  - Policy Gradient
  - Deep Q-Learning (DQN)
  - Improvements to Policy Gradient & DQN
- **Grading & Format**
  - Grading Policy
  - Code Format
  - Report
  - Submission

# Game Playing

- Implement an agent to play Atari games using Deep Reinforcement Learning
- In this homework, you are supposed to implement **Policy Gradient** and **Deep Q-Learning** (DQN)

## Environment

Pong



Breakout


https://gym.openai.com/envs/

## Policy Gradient

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

# Deep Reinforcement Learning

## REINFORCE Baseline

1. Training loop(simplest version):
    a. Play until a game is over(one player gets 21 points) with policy network $\pi_\theta$ and store (s,a,r) tuples into memory m.
    b. Discount and normalize rewards in memory into $r'$ to reduce variance
    c. Approximate gradient $\nabla_\theta J(\theta) \approx \sum_{(s_t, a_t, r_t') \in m} \nabla_\theta \log \pi_\theta(a_t | s_t) r_t'$
    d. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
    e. Clear the memory m
2. Tips:
    a. The trajectory length varies from game to game, hence sum the gradient instead of averaging it.
    b. Feed $s_t' = s_t - s_{t-1}$ into policy network, where $s_t$ comes from environment at time step t and $s_0' = s_0$
    c. When one player gets point, reset the running add of discounted reward to zero

# Deep Q-Learning (DQN)

"classic" deep Q-learning algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update $\phi'$: copy $\phi$ every $N$ steps

# Improvements to Policy Gradient (BONUS)

- Variance Reduction
- Advanced Advantage Estimation
- Off-policy learning by Importance Sampling
- Natural Policy Gradient
- Trust Region Policy Optimization
- Proximal Policy Optimization

http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_4_policy_gradient.pdf
http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_13_advanced_pg.pdf

# Improvements to DQN (BONUS)

- Double Q-Learning
- Dueling Network
- Prioritized Replay Memory
- Multi-Step Learning
- Noisy DQN
- Distributional DQN

https://arxiv.org/pdf/1710.02298.pdf

# Grading Policy

- Baseline (6%)
  - Policy Gradient (3%)
    - Getting averaging reward in 30 episodes over **7** in **Pong**
  - DQN (3%)
    - Getting averaging reward in 100 episodes over **50** in **Breakout**
- Report (10%)
- Bonus (4%)

# Grading & Format

## Baseline (6%)

- Policy Gradient (3%)
  - Getting averaging reward in 30 episodes over **7** in **Pong**
  - Without OpenAI's Atari wrapper & reward clipping
- DQN (3%)
  - Getting averaging reward in 100 episodes over **50** in **Breakout**
  - With OpenAI's Atari wrapper & reward clipping

# Grading & Format

## Code Format

- Please download the sample files from github
- Follow the instructions in README to install required packages
- **Four** functions you should implement in agent_[pg|dqn].py
  1. __init__(self, env, args)
  2. init_game_setting(self)
  3. train(self)
  4. make_action(self, state, test)
- **DO NOT** add any parameter in __init__(), init_game_setting() and make_action()
- You can add new methods in the agent_[pg|dqn].py
- You can add your arguments in argument.py

# Report (10%)

- Basic Performance (6%)
    - Describe your Policy Gradient & DQN model (1% + 1%)
    - Plot the learning curve to show the performance of your Policy Gradient on Pong (2%)
    - Plot the learning curve to show the performance of your DQN on Breakout (2%)
    - X-axis: number of time steps
    - Y-axis: average reward in last 30 episodes

# Grading & Format

## Report (10%)

- Experimenting with DQN hyperparameters (4%)
  - Choose one hyperparameter of your choice and run at least three other settings of this hyperparameter
  - You should find a hyperparameter that makes a nontrivial difference on performance
  - Plot all four learning curves in the same graph (2%)
  - Explain why you choose this hyperparameter and how it effect the results (2%)
  - Candidates: learning rate, gamma, network architecture, exploration schedule/rule, target network update frequency, etc.

# Grading & Format

## Bonus (4%)

- You can train on any environment to show your results
- Improvements to Policy Gradient (2%)
  - Implement at least two improvements to Policy Gradient (p.8) and describe why they can improve the performance (1%)
  - Plot a graph to compare and analyze the results with and without the improvements (1%)
- Improvements to DQN (2%)
  - Implement at least two improvements to DQN (p.9) and describe why they can improve the performance (1%)
  - Plot a graph to compare and analyze the results with and without the improvements (1%)
- Implement other advanced RL method, describe what it is and why it is better (2%)
  - Ex: Actor-Critic, A2C, A3C, ACKTR
- Up to 4 bonus points

# Grading & Format

## Late submission

- Please fill the late submission form first only if you will submit HW late
- Please push your code before you fill the form
- There will be 25% penalty per day for late submission, so you get 0% after four days
- You get 0% if the required files has bug.
    - If the error is due to the format issue, please come to fix the bug at the announced time, or you will get 10% penalty afterwards.

# Grading & Format

## Submission

- Deadline: **2017/12/16 23:59 (GMT+8)**
- Your github **MUST** have 5 files under directory hw3/
  - agent_dir/agent_pg.py
  - agent_dir/agent_dqn.py
  - [saved_model_file] * 2
  - report.pdf
  - argument.py (optional)
  - README (optional)
  - download.sh (optional)
  - other files you need
- If your model is too large for github, upload it to a cloud space and write download.sh to download the model
- Do not upload any file named the same with other sample codes

# Grading & Format

## Grading

- Please use Python with version >= 3.5
- The TAs will execute ‘python3 test.py --test_pg --test_dqn’to run your code
- The execution should be done within 10 minutes, excluding model download
- Allowed packages:
    - PyTorch v0.2.0
    - Tensorflow r1.3
    - Keras 2.0.7 ( Tensorflow backend only )
    - MXNet 0.11.0
    - CNTK 2.2
    - Numpy
    - Pandas
    - Python Standard Lib
- If you use other packages, please ask for permission first !!!

# Related Materials

- Course & Tutorial:
  - Berkeley Deep Reinforcement Learning, Fall 2017
  - David Silver RL course
  - Nips 2016 RL tutorial
- Blog:
  - Andrej Karpathy's blog
  - Arthur Juliani's Blog
- Text Book:
  - Reinforcement Learning: An Introduction

# TA Information

## TA hours

- 有問題請利用TA hours 、信箱或FB社團，請不要FB私訊助教！！
- If you have other questions,
    - please contact TAs via adlxmlds@gmail.com
    - post your questions on facebook group
    - go to TA office hours
        - 陳璽安 Thur 14:00-15:30 (德田536)
        - 王耀賢 Fri 16:00-17:30 (電二531) (11/30開始)
        - 葉奕廷 Mon 10:30-12:00 (德田524) (12/4請假)