

# DQN Project

**Updated** Saturday November 30

## **Changes:**

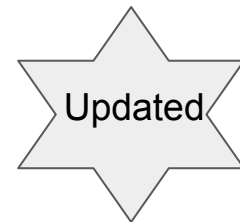
- Slide 3
- Slide 5
- Slide 12 (new)
- Slide 13

## Important Note

**Let me reiterate that all submissions will be run through a program that automatically detects copies including minor variations and duplicate parts (e.g., doesn't have to be the whole code).**

**Please do not risk copying code from anywhere. If you copy the whole or part of a code, including web sources, we will find out and immediately refer the student(s) to the university as an academic dishonesty case.**

# Objective



You are going to implement a DQN agent that plays the Atari 2600 game StarGunner (for 100% credit) or CartPole (for 90% credit) or both (105% credit):

- Playing StarGunner means showing learning even if performance is still poor
  - Current record (Saturday Nov. 30) at Lehigh: average of 1000 points per episode after 2099 episodes (took 24 hours to run in Google Colab)
- Playing CartPole means showing learning and high performance
  - Current record (Saturday Nov. 30) at Lehigh: learn to solve the problem in 403 episodes (took less than 1 hour in Google Colab.)
- 5% extra credit for doing both!
- Suggest you do CartPole first



# Required Readings

Please read carefully **both**:

- Human-level control through deep reinforcement learning:  
<https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf>
- Section 16.5 of the textbook

before attempting to solve the project.

# Environment

StarGunner is available in the OpenAI Gym:

<https://gym.openai.com/envs/StarGunner-v0/>

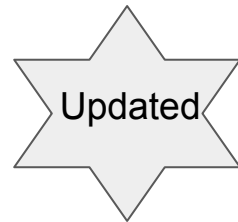
CartPole-V0 is available at: <https://gym.openai.com/envs/CartPole-v0/>

To get started with the OpenAI Gym, please read this tutorial:

<https://gym.openai.com/docs/>

To create the game environment and preprocess the game frames:

```
import gym
env = gym.make('StarGunner-v0')
env = gym.wrappers.AtariPreprocessing(env)
```



# Implementation

You can use any machine learning library of your choice to implement DQN's architecture. We recommend Keras (<https://keras.io/>) because it is easy to use.

You can use Google Colaboratory so you don't need to install anything locally.

The next several slides provide some hints on how to implement the learning algorithm.

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$  ← E.g., a python list

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

        With probability  $\varepsilon$  select a random action  $a_t$

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

        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  $D$

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

        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

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

        Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**

### Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$  ←

E.g., create a Keras model and add Conv2D layers, Dense layers, etc. Use the hyperparameters described in the paper.

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

With probability  $\varepsilon$  select a random action  $a_t$

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

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  $D$

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

Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

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

Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**



**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$  ← The same as the previous step; create another model with identical architecture.

**For** episode = 1,  $M$  **do**

Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

With probability  $\varepsilon$  select a random action  $a_t$

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

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  $D$

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

Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

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

Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**


**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$   E.g., a Python deque that maintains 4 game frames

**For**  $t = 1, T$  **do**

With probability  $\varepsilon$  select a random action  $a_t$

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

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  $D$

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

Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

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

Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**

### Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

With probability  $\varepsilon$  select a random action  $a_t$

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

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  $D$

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

Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

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

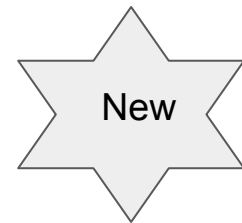
Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**

E.g., create two lists  $x$  and  $y$ ;  $x$  is the states, and  $y$  is the targets. Use the Keras function *fit* to train the model.

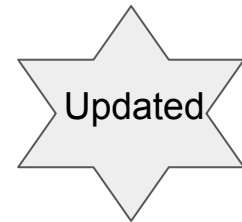
# Changes for running CartPole-v0 (Compared to StarGunner)



- No need to stack the 4 states as a single input
  - that is, each state is an input
- For CartPole-0 DQN doesn't need to use the two CNN layers
  - That is, input state should go directly to the fully-connected layer
- Suggested dimensions for hidden layers:
  - 1 fully connected with 64 units
  - 1 fully connected with 32 units

# Submission (via Course Site)

Please submit either two files or 4 files:



- Your DQN code (for StarGunner)
- A 1-page PDF file with the following information about your system (StarGunner):
  - Plotting on the x-axis (training episodes) and the y-axis (episodic reward)
  - A description of the results
- Your DQN code (for CartPole)
- A 1-page PDF file with the following information about your system (CartPole):
  - Plotting on the x-axis (training episodes) and the y-axis (episodic reward)
  - A description of the results

**Deadline:** Monday, December 16, 2019, 7:00 AM

# Meeting with Instructor - REQUIRED

- Each group is required to meet with the instructor
- In the meeting the group will demonstrate their system
- The instructor will ask questions about the code
- **The code submitted via Course Site must be identical to the one in the demonstration**
- **Registration information for these meetings will be available on Course Site (first come, first serve)**
- Meetings dates: Monday December 16, Tuesday December 17