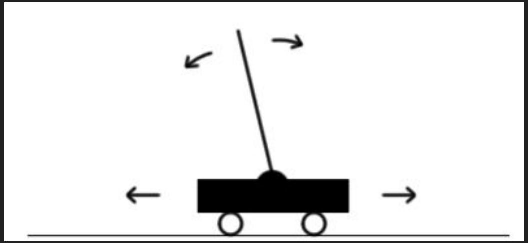
CS291T: Reinforcement Learning Project

OpenAI - Gym: Cartpole

Jaehoon Ganesh Gregory

The Cart Pole Problem

- Inverted pendulum attached to a Cart

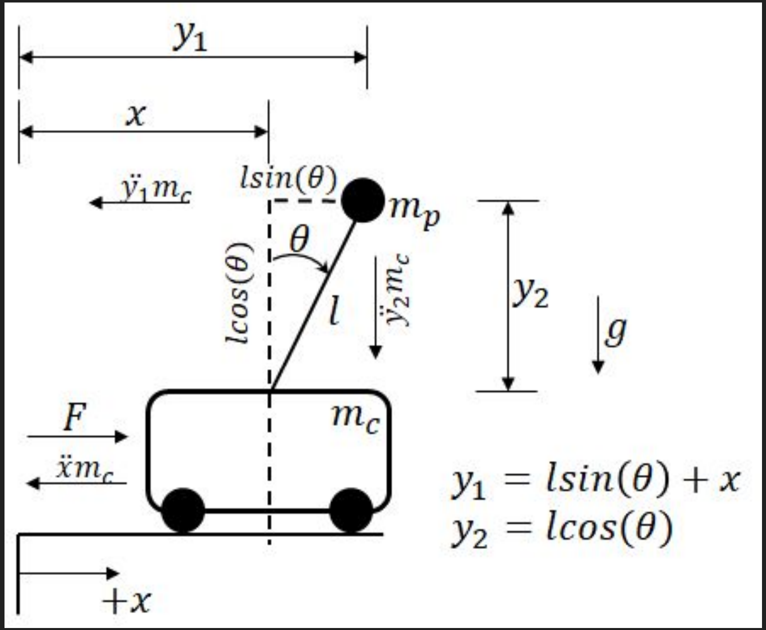


- Cart Moves Side to Side
- Goal: Move the cart such that the pendulum stays upright at all times

Classical Controls Approach

- Sum forces and torques to develop dynamics equations

- Results in highly non-linear 4th order system
  - Linearize model (Taylor series & PID control)
  - Sliding Mode Control
  - Fuzzy Logic



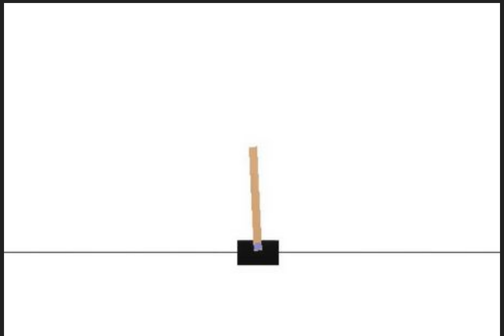
$$m_p l^2 \ddot{\theta} + m_p l \ddot{x} \cos(\theta) = m_p g l sin(\theta)$$

$$(m_c + m_p)\ddot{x} \; - m_p l \dot{\theta}^2 sin(\theta) + m_p l cos(\theta)\ddot{\theta} = 0$$

Reinforcement Learning Approach

- Model as Markov Decision Process (MDP)
    - States = (x, θ, x’, θ’)
    - Actions = { left , right }
    - Transitions Probabilities = 1 (guaranteed to take requested action)
    - Rewards = 1
        - As long as pole does not cross -12 or +12 degrees
        - And cart does not go past -2.4 or +2.4
    - Start state: uniform random value for all observations between -0.05 & +0.05

- Simulation handled by OpenAI-Gym python package



Linear Function Approximation w/ Q-Learning

- Cart-Pole has Continuous state space
  - Creating a table with Q values would be inefficient
  - Instead Q should be a function

$$Q(s,a) = Q^{\pi}(s,a) = w^T\varphi(s,a) = w_0 + w_1\varphi_1(s,a) + \cdots + w_n\varphi_n(s,a)$$

$$\varphi\,(s, left\,) = \begin{bmatrix} x \\ x' \\ \theta \\ \theta' \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \varphi\,(s, right) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ x \\ x' \\ \theta \\ \theta' \end{bmatrix}$$

- Q will have a set of weights w
  - Learn by Stochastic Gradient Descent
- Weights will be multiplied by features $\varphi_{\mathrm{i}}(\mathrm{s}, \mathrm{a})$
  - Features will simply be a vector of observations
  - Offset observations for each action

Linear Function Approximation w/ Q-Learning

- Policy

$$\pi^{\epsilon} = \begin{cases} argmax_a Q^{\pi}(s,a), & 1-\epsilon \\ UnfiormRandom(A), & \epsilon \end{cases}$$

- Update Rule (Stochastic Gradient Descent)

$$Q^{+}(s,a) = r + \gamma \max_{a'} Q(s',a')$$
$$\delta = Q^{+}(s,a) - Q(s,a)$$
$$w = w + \alpha\delta\varphi(s,a)$$

- Demo

Open AI Environment & Functions

- Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.
- env = gym.make(environment_name) <- sets up the environment.
- env.reset() <- resets the environment to starting point.
- env.step(action) <- takes action and goes to state S_t+1.
- env.render() <- renders the output.

Algorithm Implemented: Deep Q-Learning

```
Initialize replay memory D to size N
Initialize action-value function Q with random weights
for episode = 1, M do
    Initialize state s_1
    for t = 1, T do
        With probability ϵ select random action a_t
        otherwise select a_t=max_a  Q(s_t,a; θ_i)
        Execute action a_t in emulator and observe r_t and s_(t+1)
        Store transition (s_t,a_t,r_t,s_(t+1)) in D
        Sample a minibatch of transitions (s_j,a_j,r_j,s_(j+1)) from D
        Set y_j:=
            r_j for terminal s_(j+1)
            r_j+γ*max_(a^' )  Q(s_(j+1),a'; θ_i) for non-terminal s_(j+1)
        Perform a gradient step on (y_j-Q(s_j,a_j; θ_i))^2 with respect to θ
    end for
end for
```

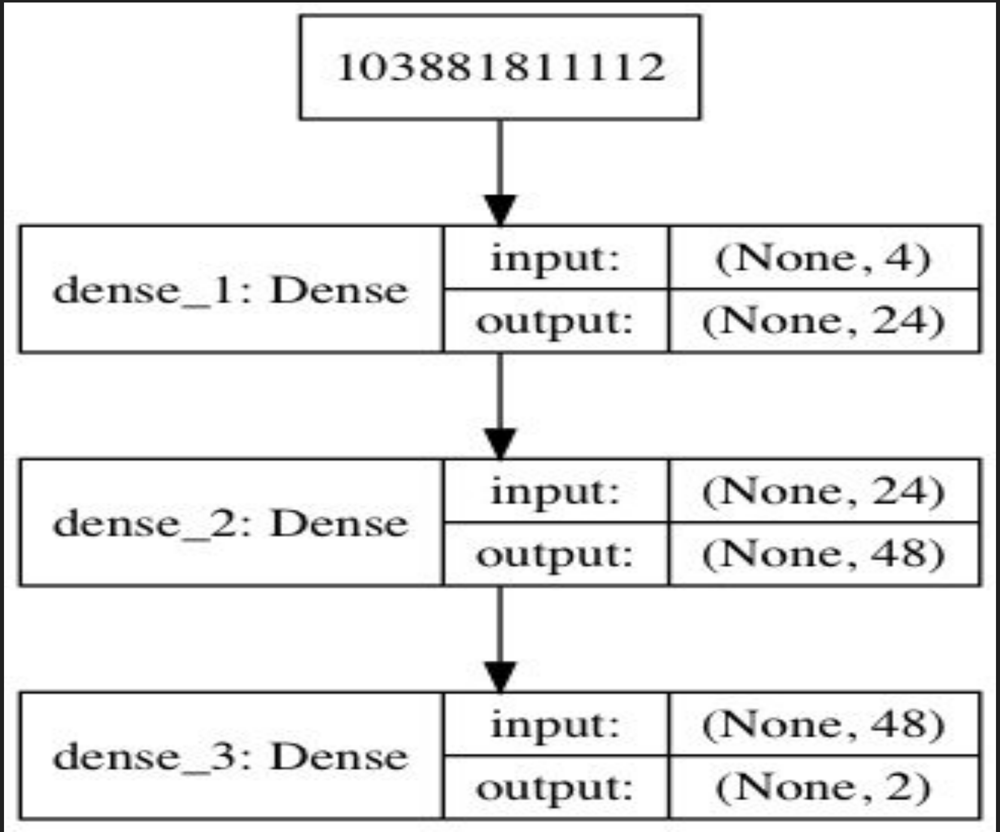
Training Parameters

- n_episodes=1000
- gamma= 1.0
- epsilon = 1.0
- epsilon_min = 0.01
- epsilon_decay = 0.995
- learning_rate = 0.01
- batch_size = 64

NN Architecture - Keras Framework

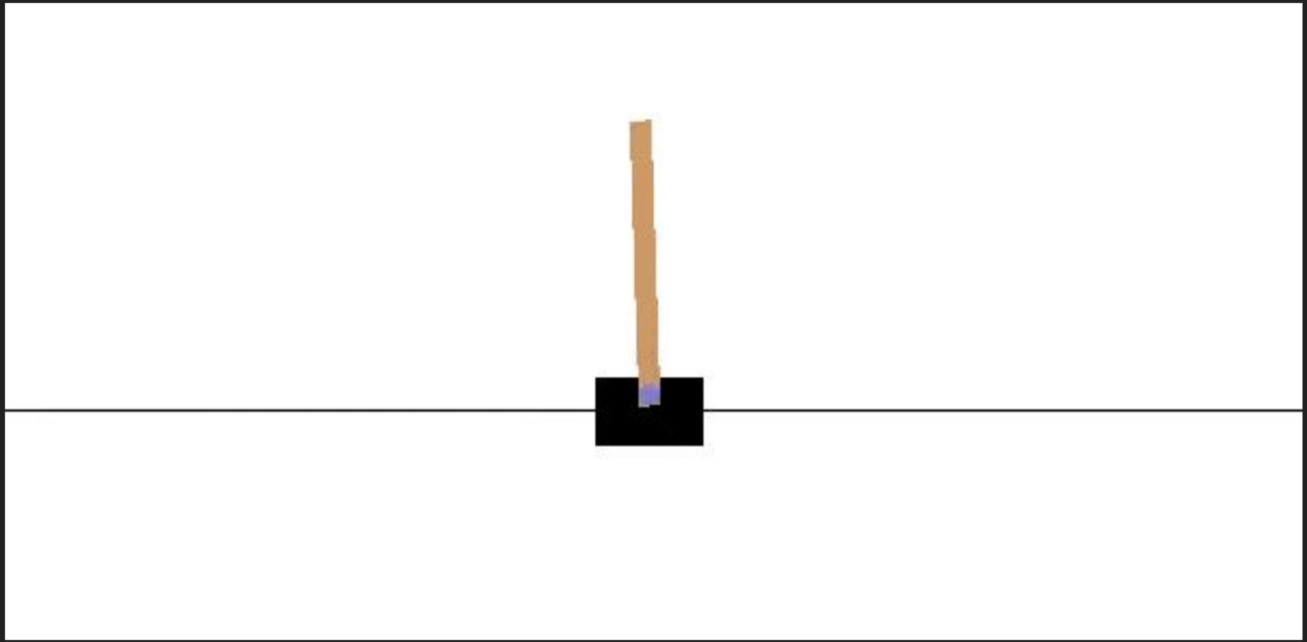
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam


#Model Definition
model = Sequential()
model.add(Dense(24,input_dim=4,activation = 'relu'))
model.add(Dense(48,activation = 'relu'))
model.add(Dense(2,activation = 'relu'))
model.compile(loss='mse',optimizer=Adam(lr=alpha,decay = alpha_decay))
```



103881811112
dense_1: Dense
input:
(None, 4)
output:
(None, 24)
dense_2: Dense
input:
(None, 24)
output:
(None, 48)
dense_3: Dense
input:
(None, 48)
output:
(None, 2)

Deep Q-Learning Output GIF



DQN Results

```
018-12-02 13:01:13.643510: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread po
r best performance.
Epsiode20] - Mean survival time over last 100 episodes as 10.714285714285714 ticks.
Epsiode40] - Mean survival time over last 100 episodes as 12.146341463414634 ticks.
Epsiode60] - Mean survival time over last 100 episodes as 14.655737704918034 ticks.
Epsiode80] - Mean survival time over last 100 episodes as 16.22222222222222 ticks.
Epsiode100] - Mean survival time over last 100 episodes as 18.79 ticks.
Epsiode120] - Mean survival time over last 100 episodes as 20.74 ticks.
Epsiode140] - Mean survival time over last 100 episodes as 27.39 ticks.
Epsiode160] - Mean survival time over last 100 episodes as 31.76 ticks.
Epsiode180] - Mean survival time over last 100 episodes as 40.02 ticks.
Epsiode200] - Mean survival time over last 100 episodes as 39.38 ticks.
Epsiode220] - Mean survival time over last 100 episodes as 49.19 ticks.
Epsiode240] - Mean survival time over last 100 episodes as 49.33 ticks.
Epsiode260] - Mean survival time over last 100 episodes as 47.6 ticks.
Epsiode280] - Mean survival time over last 100 episodes as 41.04 ticks.
Epsiode300] - Mean survival time over last 100 episodes as 48.07 ticks.
Epsiode320] - Mean survival time over last 100 episodes as 41.64 ticks.
Epsiode340] - Mean survival time over last 100 episodes as 48.08 ticks.
Epsiode360] - Mean survival time over last 100 episodes as 50.55 ticks.
Epsiode380] - Mean survival time over last 100 episodes as 55.61 ticks.
Epsiode400] - Mean survival time over last 100 episodes as 59.25 ticks.
Epsiode420] - Mean survival time over last 100 episodes as 60.12 ticks.
Epsiode440] - Mean survival time over last 100 episodes as 57.0 ticks.
Epsiode460] - Mean survival time over last 100 episodes as 74.16 ticks.
Epsiode480] - Mean survival time over last 100 episodes as 76.22 ticks.
Epsiode500] - Mean survival time over last 100 episodes as 68.5 ticks.
Epsiode520] - Mean survival time over last 100 episodes as 71.65 ticks.
Epsiode540] - Mean survival time over last 100 episodes as 70.28 ticks.
Epsiode560] - Mean survival time over last 100 episodes as 55.59 ticks.
[[Epsiode580] - Mean survival time over last 100 episodes as 69.06 ticks.
Epsiode600] - Mean survival time over last 100 episodes as 75.66 ticks.
Epsiode620] - Mean survival time over last 100 episodes as 78.86 ticks.
Epsiode640] - Mean survival time over last 100 episodes as 87.48 ticks.
Epsiode660] - Mean survival time over last 100 episodes as 89.73 ticks.
Epsiode680] - Mean survival time over last 100 episodes as 76.75 ticks.
Epsiode700] - Mean survival time over last 100 episodes as 74.1 ticks.
Epsiode720] - Mean survival time over last 100 episodes as 80.85 ticks.
Epsiode740] - Mean survival time over last 100 episodes as 90.23 ticks.
Epsiode760] - Mean survival time over last 100 episodes as 97.55 ticks.
Epsiode780] - Mean survival time over last 100 episodes as 110.02 ticks.
Epsiode800] - Mean survival time over last 100 episodes as 125.9 ticks.
Epsiode820] - Mean survival time over last 100 episodes as 130.77 ticks.
Epsiode840] - Mean survival time over last 100 episodes as 121.83 ticks.
Epsiode860] - Mean survival time over last 100 episodes as 125.63 ticks.
Epsiode880] - Mean survival time over last 100 episodes as 125.07 ticks.
Epsiode900] - Mean survival time over last 100 episodes as 120.14 ticks.
Epsiode920] - Mean survival time over last 100 episodes as 119.87 ticks.
Epsiode940] - Mean survival time over last 100 episodes as 130.64 ticks.
Epsiode960] - Mean survival time over last 100 episodes as 138.36 ticks.
Epsiode980] - Mean survival time over last 100 episodes as 139.06 ticks.
```

Experience Replay and DDQN Information

Tensorflow with Keras

	NN	Epsilon	Reward	Sample	Discount Rate	Learning Rate(NN)
DQN	3 x 24	0.9975	1 / -10	-	0.95	0.001
DQN ER	3 x 24	0.9975	1 / -10	32	0.95	0.001
DDQN	(2) 3 x 24	0.9975	1 / -10	-	0.95	0.001
DDQN ER	(2) 3 x 24	0.9975	1 / -10	32	0.95	0.001

Deep Q Learning with Experienced Replay

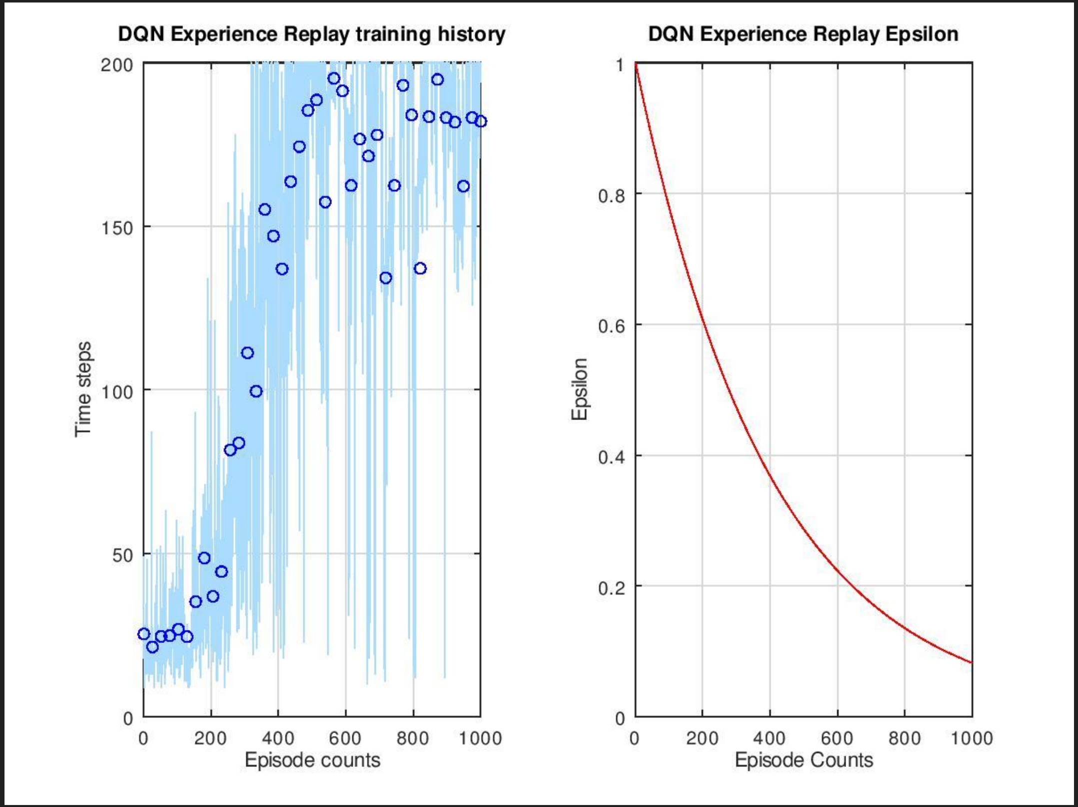
Experience replay: Storing previous experience with environment and when you train an agent, you sample n sizes to train the agent, instead of training off of only previous experiences.

Key: You store <**state**, **reward**, **action**, and **next_state**>, in order to calculate q target value.

Implemented with **deque** data structure.

Loss: Mean Square Error and Optimizer: Adam Optimizer

DQN - Experience Replay Training Graph



Deep Q Learning Equations

Q Learning Algorithm

Original Q Value Update:

$$Q_t(s,a) = Q_t(s,a) + \alpha \cdot \left[r + \gamma \cdot max_{a'}(Q_{t+1}(s_{t+1},a')) - Q_t(s,a)\right]$$

Q Value Neural Network Target:

$$Q_t(s,a) = r + \gamma \cdot max_{a'}(Q_{t+1}(s_{t+1},a'))$$

Original Double Q Learning:

$$Q_1(s,a) = Q_1(s,a) + \alpha \cdot \left[r + \gamma \cdot Q_2(s', argmax_a Q_1(s',a)) - Q_1(s,a)\right]$$
$$Q_2(s,a) = Q_2(s,a) + \alpha \cdot \left[r + \gamma \cdot Q_1(s', argmax_a Q_2(s',a)) - Q_2(s,a)\right]$$

DDQN Value Neural Network Target:

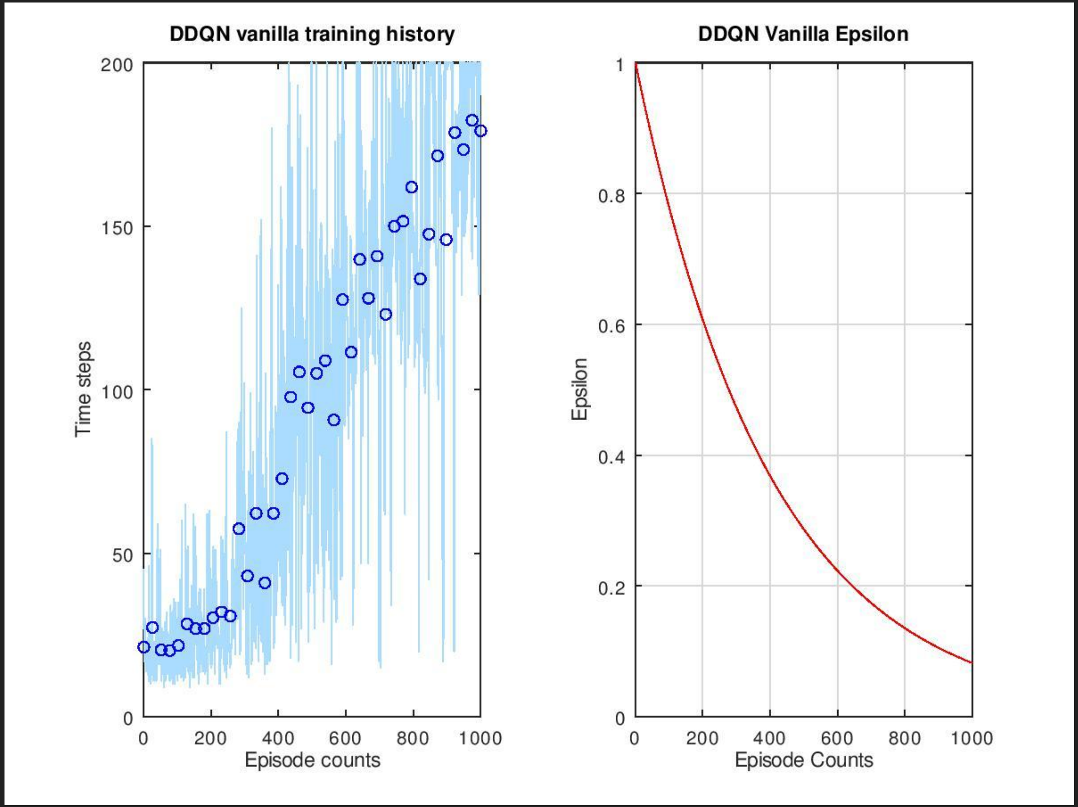
$$Q_1(s,a) = r + \gamma \cdot Q_2(s', argmax_a Q_1(s',a))$$
$$Q_2(s,a) = r + \gamma \cdot Q_1(s', argmax_a Q_2(s',a))$$

Double Q Learning with Neural Network

Double Q Learning: Two neural networks to estimate the values. Action and train with 50% probability of choosing one or the other. Training using other DQN feedforward from the action that was chosen by target NN.

Input: 4 neurons hidden layers: 3 layers of 24 neurons output layer: 2 neurons

DDQN Training Graph

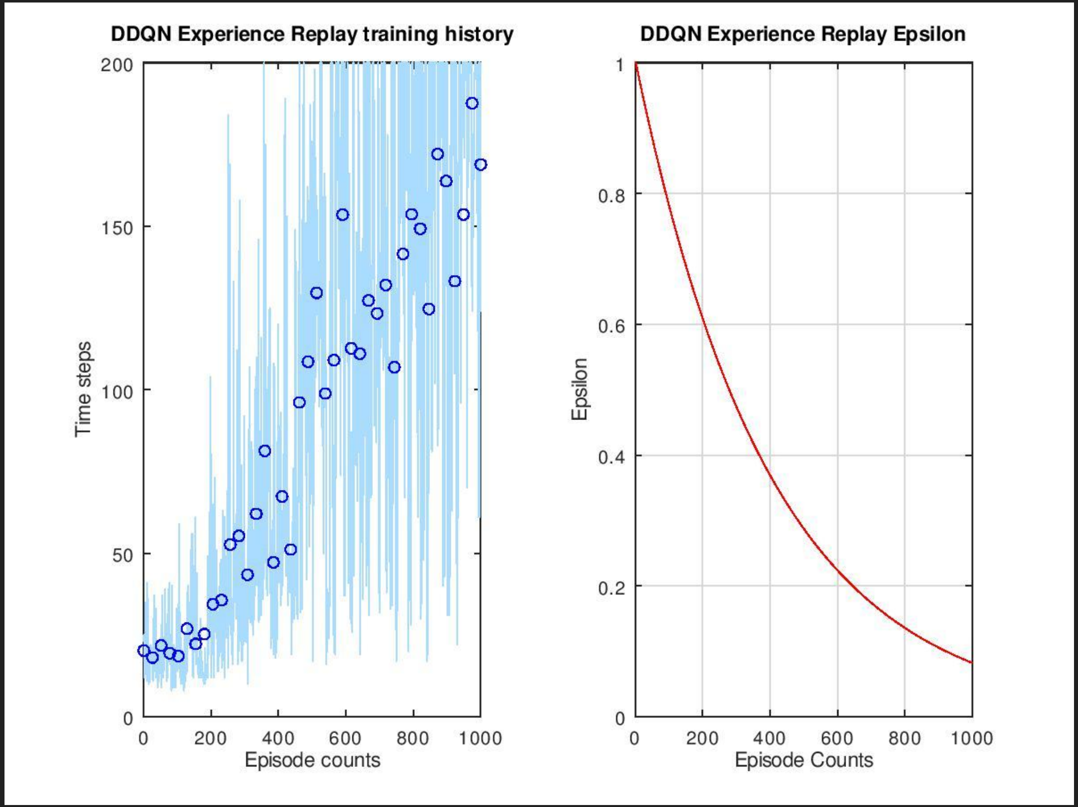


DDQN with Experience Replay

Similar to DQN with experience replay.

Key: You store <**state**, **reward**, **action**, and **next_state**>, in order to calculate q target value.

DDQN with Experience Replay Training Graph



Benchmark Test

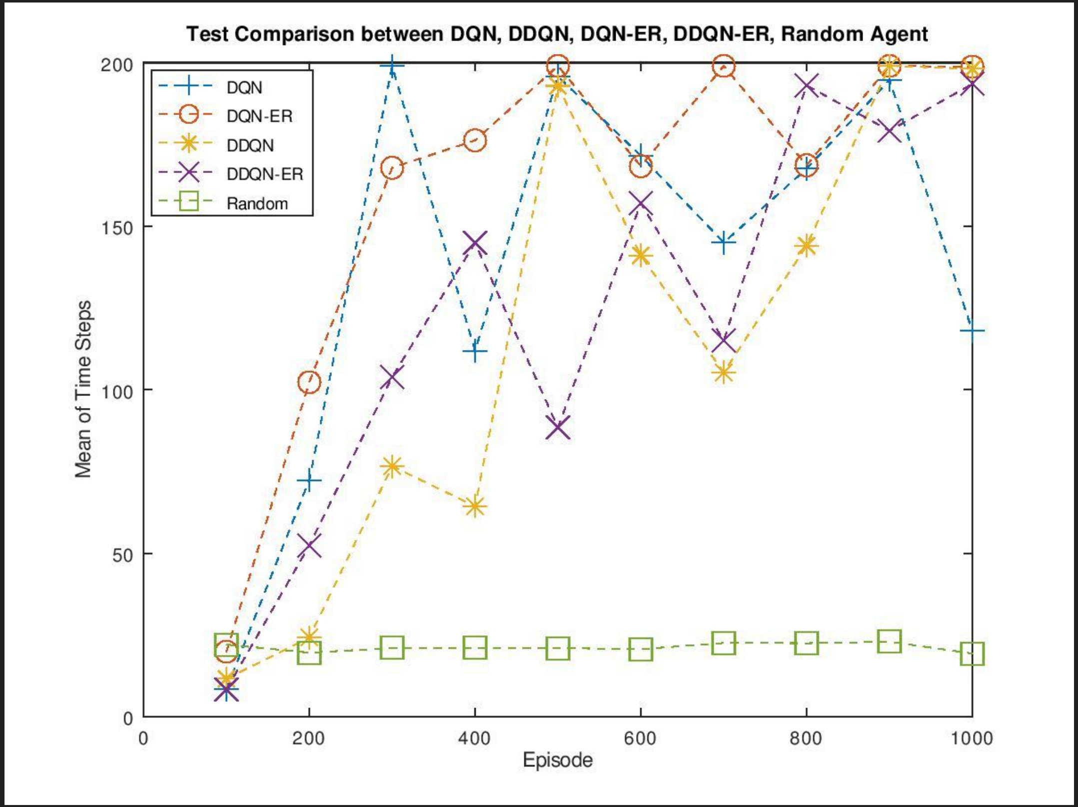
Agents: DQN, DQN-ER, DDQN, DDQN-ER, Random Agent (Base)

Running with 0.9975 decay and reward of 1/-10, running total of 1000 episodes

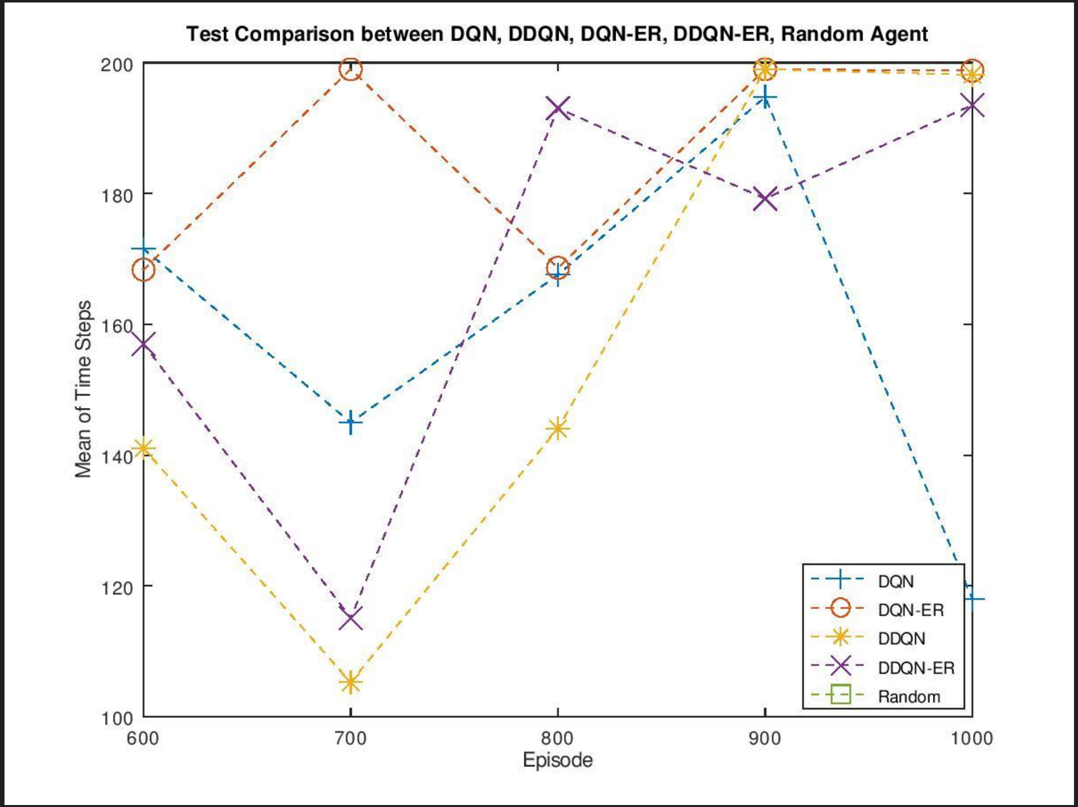
Benchmark every 100 training episodes with 200 test episodes with epsilon of 0.

- Total of 10 times and each time with 200 episodes.

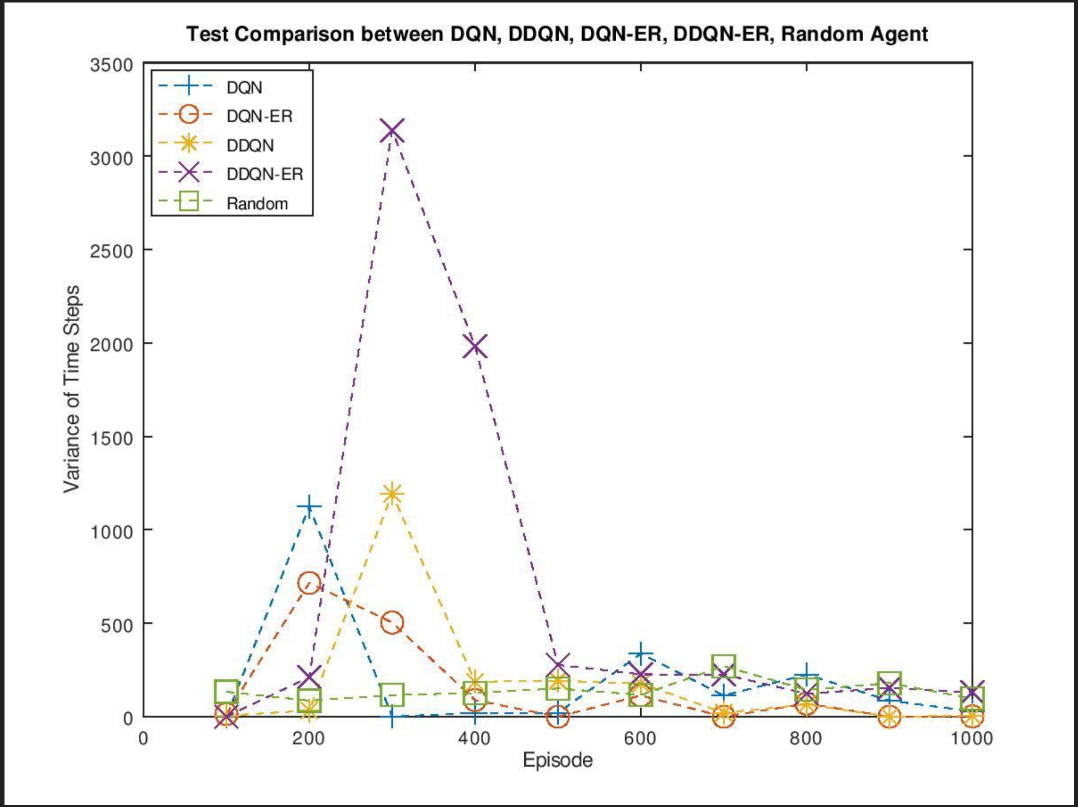
Benchmark Result: Mean



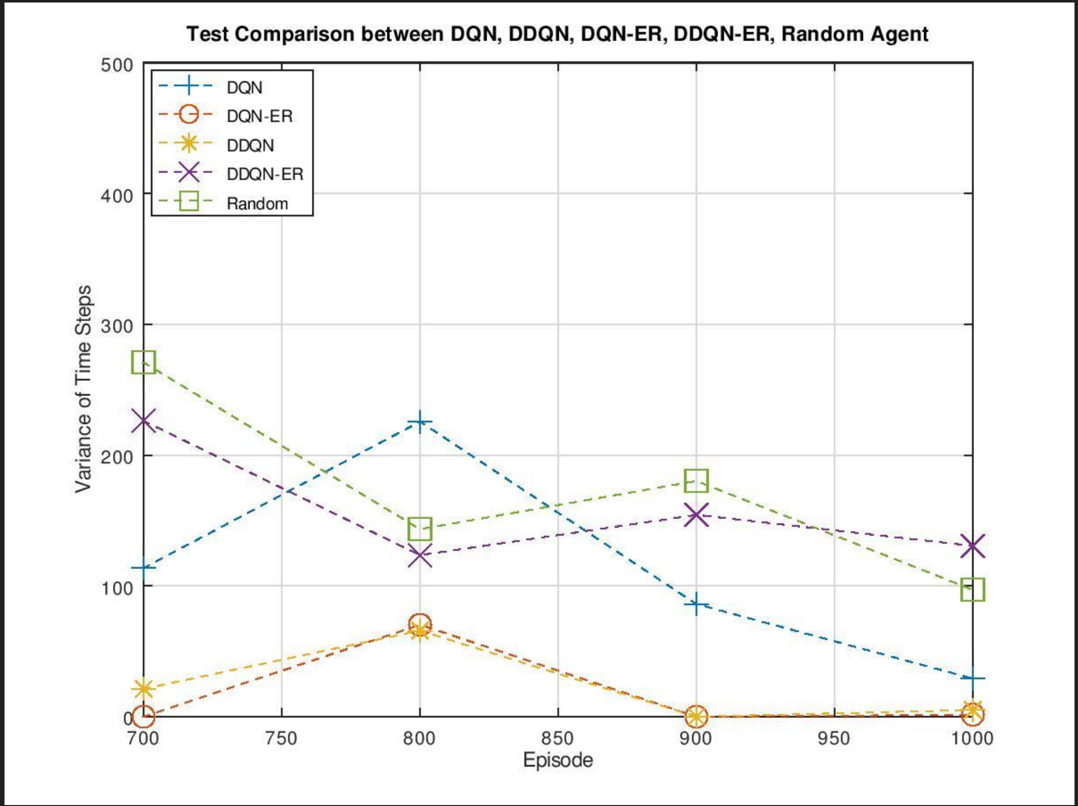
Benchmark Result: Mean Closer look



Benchmark Result: Variance



Benchmark Result: Variance Closer look



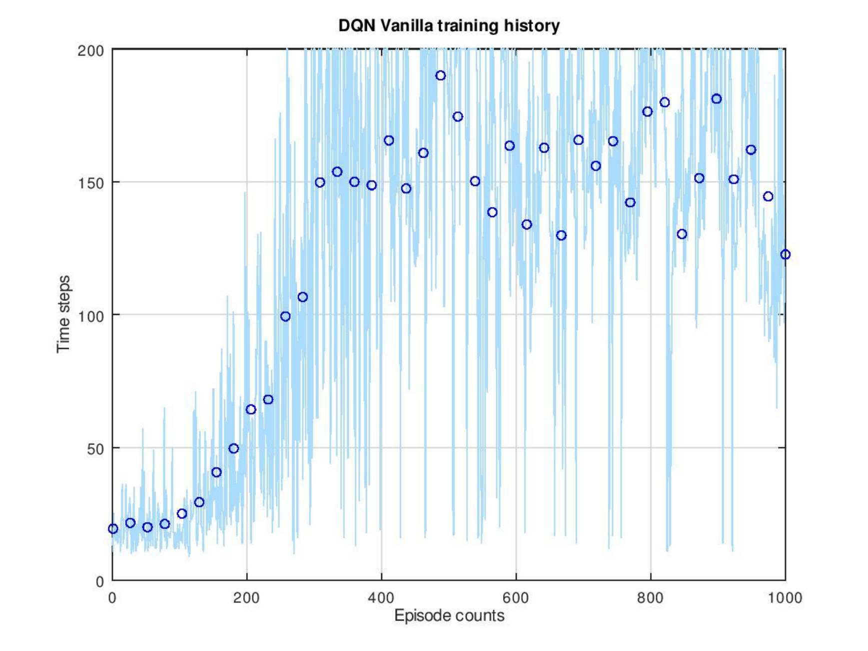
Benchmark Conclusion

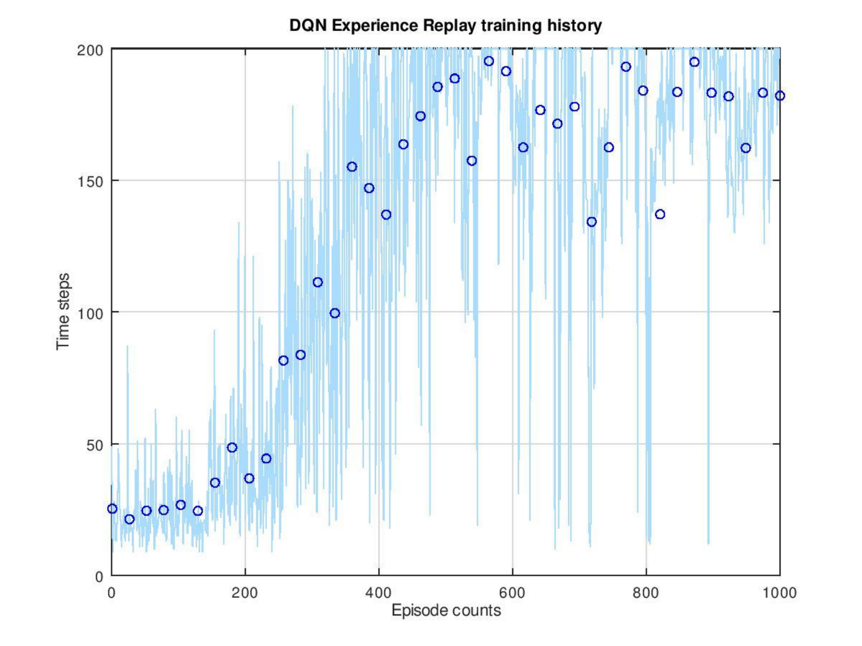
- DQN trains fastest, but doesn’t yield consistent results
- DDQN and DQN-ER yields consistent (close to 0 variance)
- DDQN-ER might need more training episodes to be more stable.

DEMO

Questions & Answers

Reference: DQN and DQN-ER Graphs





Reference: DDQN and DDQN-ER Graphs