Using Deep Reinforcement Learning for Autonomous Cars in 2D and 3D Simulated Environments

ECE 510 - Deep Learning Theory and Practice

Final Project - Spring 2019

Husnu Melih Erdogan

Mohammed Abidalrekab

Abdirahman Alasow

Portland State University

Goal

- Our main goal in this project is discovering and learning more about Deep Reinforcement Learning for autonomous vehicle navigation as we build and test a self driving car application in a simulated environment.

- We compared several different Deep Reinforcement methods and used them with several different simulations.

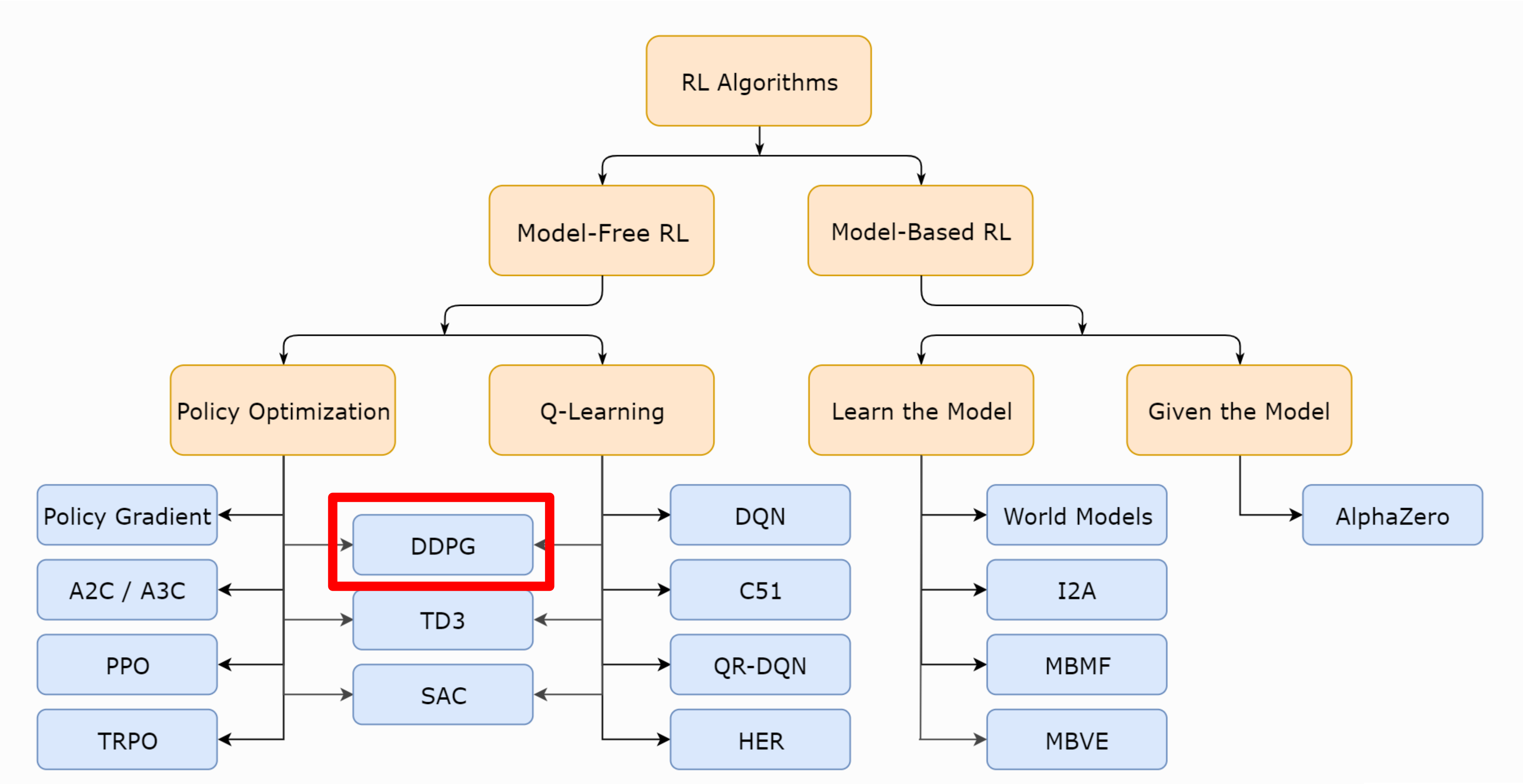
Papers

- Continuous Control With Deep Reinforcement Learning
  - Timothy P. Lillicrap∗ , Jonathan J. Hunt∗ , Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
  - Google Deepmind – 2016

- Continuous Control Automated Lane Change Behavior Based on Deep Deterministic Policy Gradient Algorithm
  - Pin Wang, Hanhan Li, Ching-Yao Chan
  - IEEE IVS, 2019

- Soft Actor-Critic – Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor
  - Thomas Haarnoja, Aurick Zhou, Pieter Abbel, Sergey Levine
  - ICML, 2018

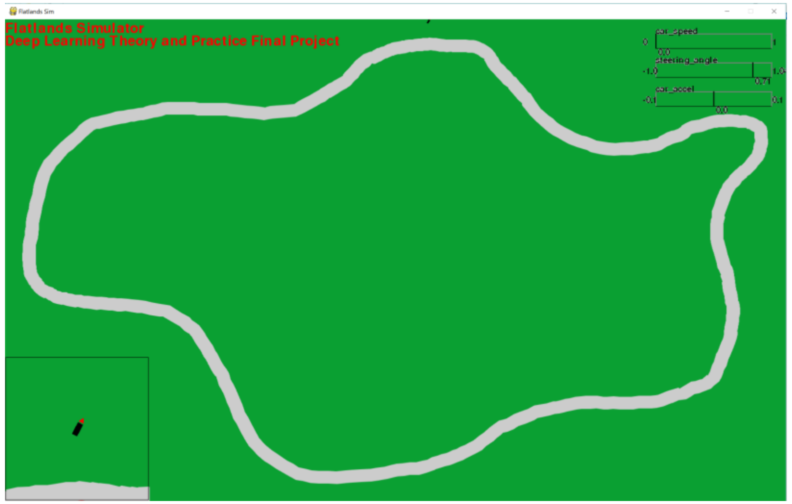
Taxonomy of Reinforcement Learning Algorithms



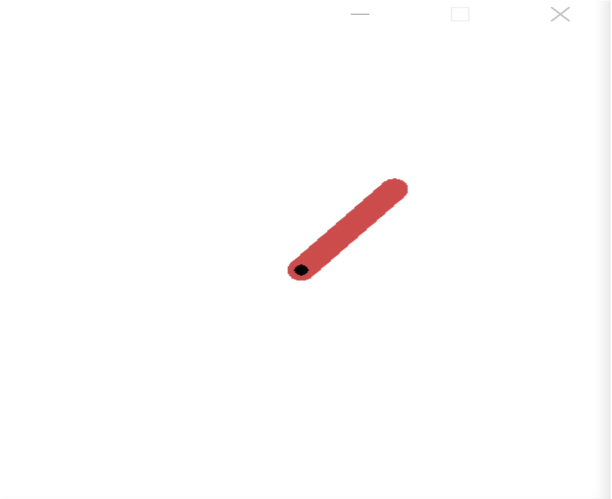
Simulations

- TORCS
  - 3D Car Simulator
  - 29 States (speed, wheelspin, rpm, track etc.)
  - 3 Actions (Steering, Acceleration, Brake)

- Flatlands
  - 2D Car Simulator
  - 3 States (Distances from the track)
  - 2 Actions (Steering, Acceleration, Brake)



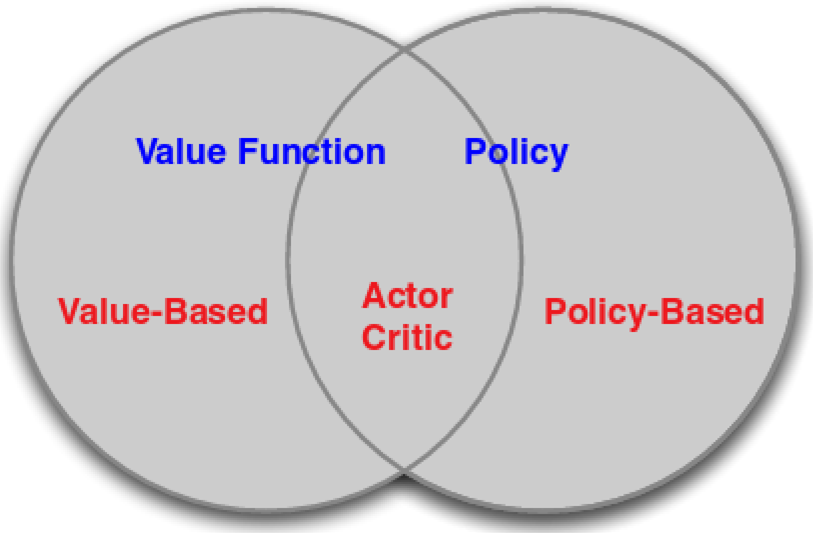
- Pendulum
  - 2D Pendulum Simulator
  - 3 States (position, velocity, angle)
  - 1 Actions (Joint effort)

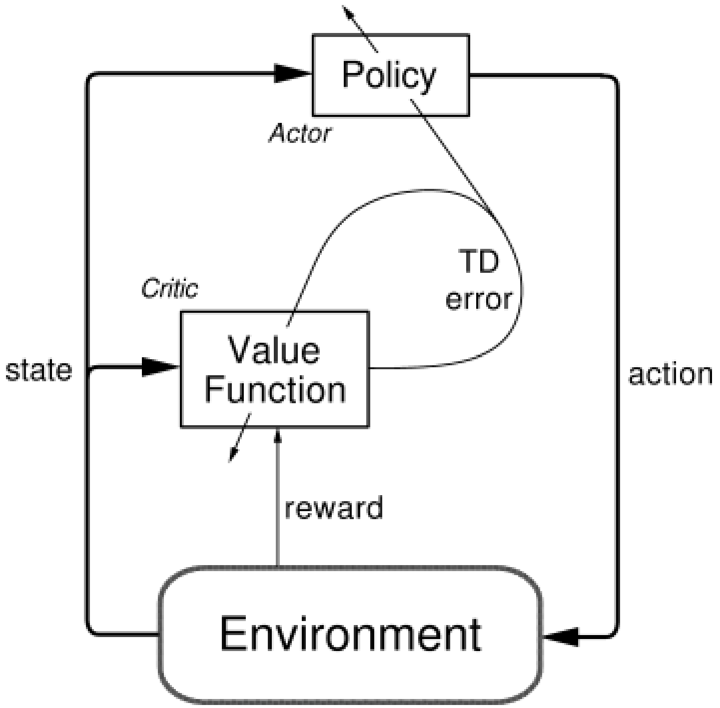


Portland State UNIVERSITY

DDPG

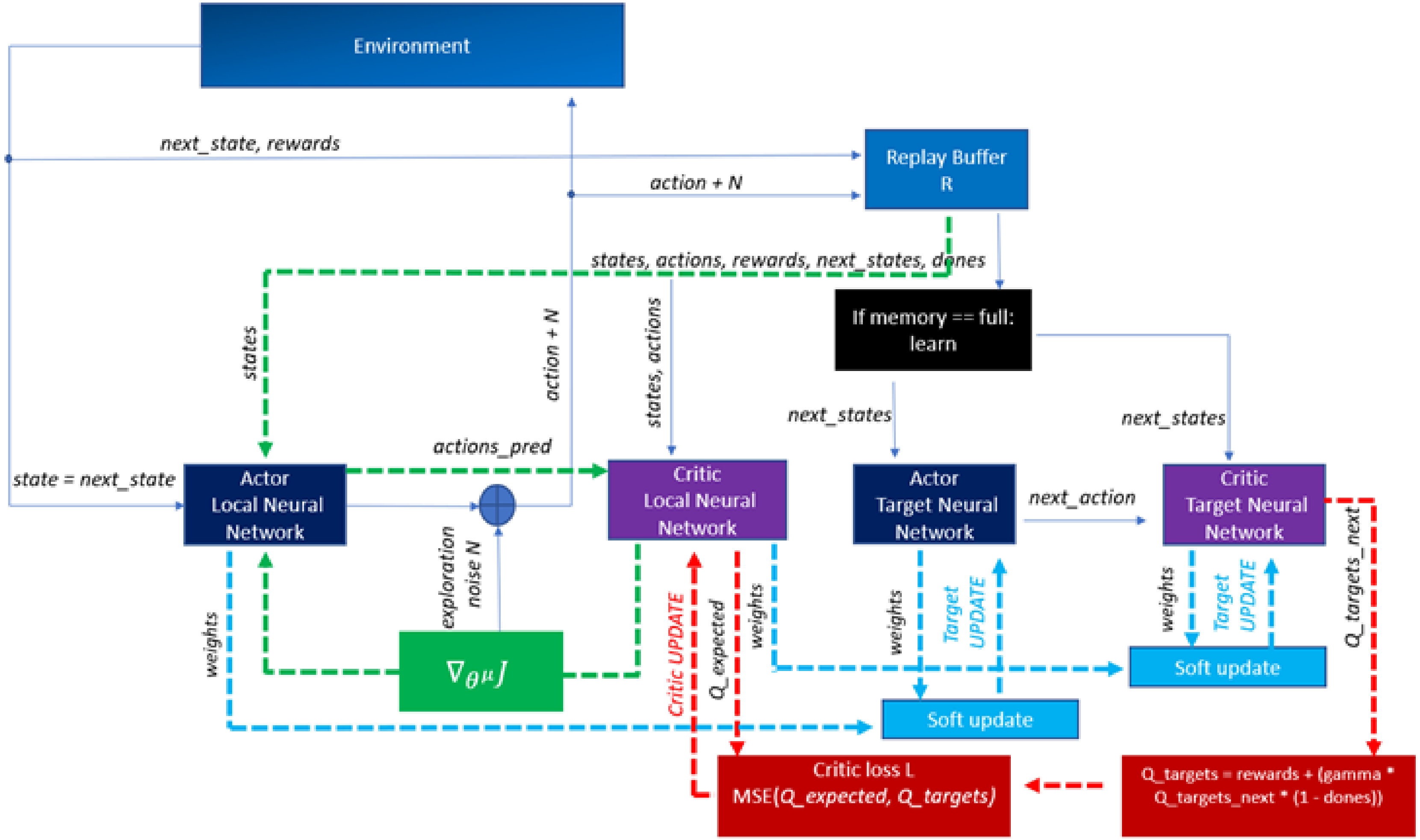
- DDPG - Deep Deterministic Policy Gradient





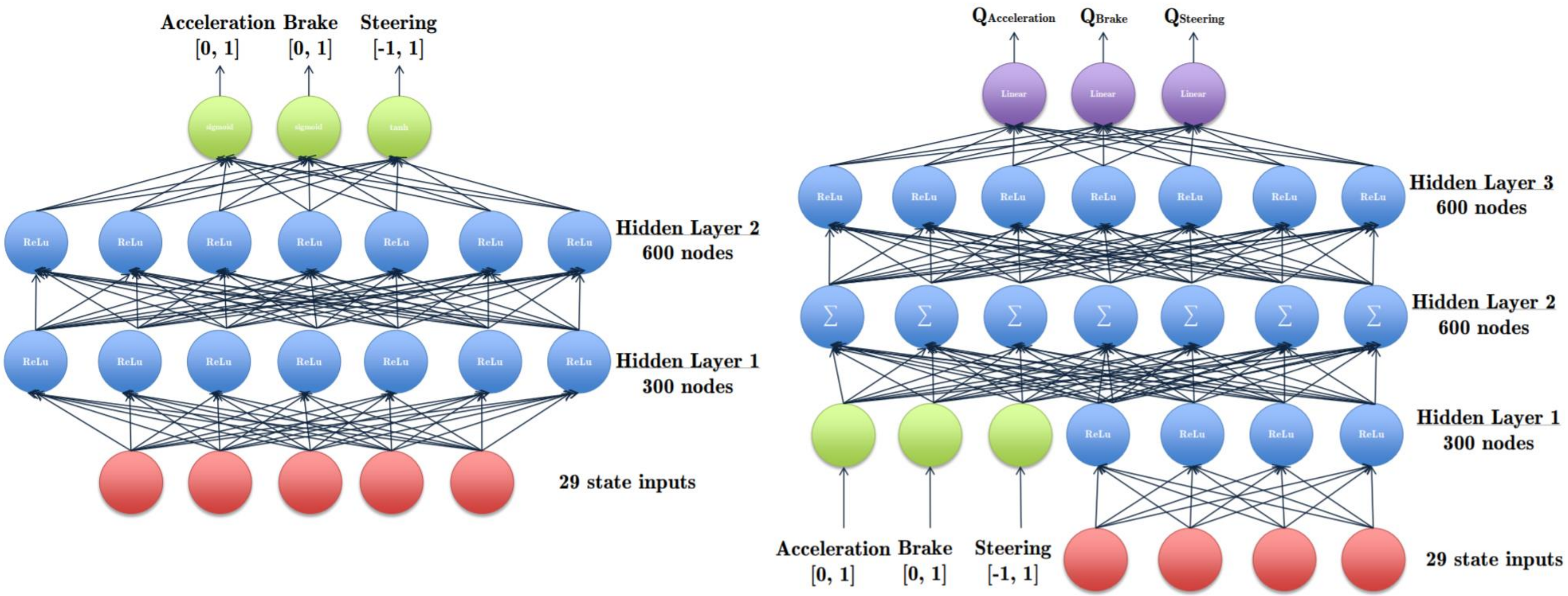
$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

DDPG



Environment
next_state, rewards
Replay Buffer
R
action + N
states, actions, rewards, next_states, dones
If memory == full:
learn
states
action + N
states, actions
next_states
next_states
actions_pred
state = next_state
Actor
Local Neural
Network
Critic
Local Neural
Network
Actor
Target Neural
Network
next_action
Critic
Target Neural
Network
exploration
noise N
weights
Critic UPDATE
Q_expected
weights
weights
Target
UPDATE
weights
Target
UPDATE
Q_targets_next
$\nabla_{\theta^\mu} J$
Soft update
Soft update
Critic loss L
MSE(Q_expected, Q_targets)
Q_targets = rewards + (gamma *
Q_targets_next * (1 - dones))

DDPG



(a) Actor network with two ReLu activated hidden layers

(b) Critic network with ReLu and linear activated hidden layers

DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

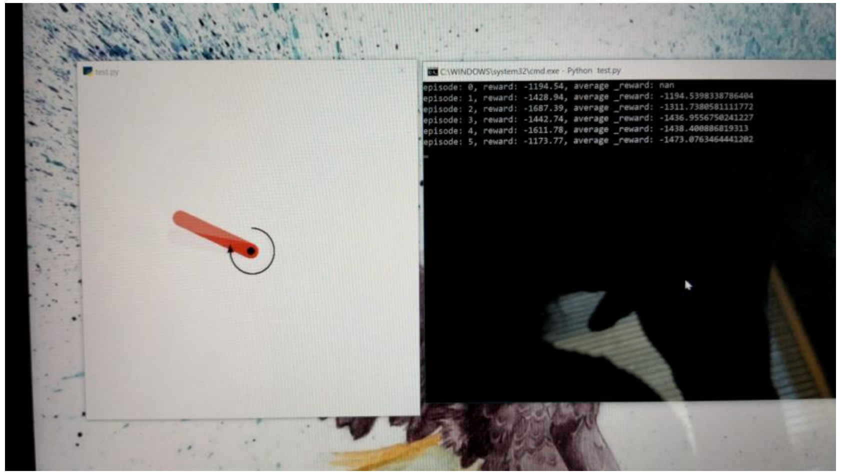
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

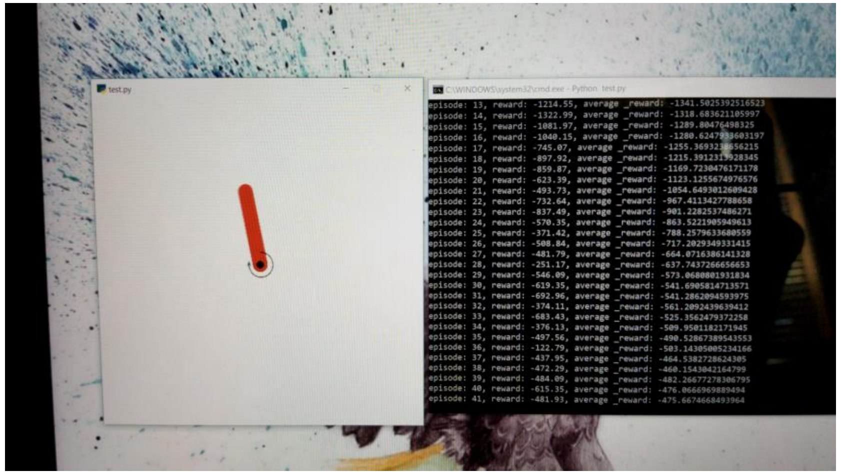
  end for
end for

Experiments - Pendulum

- Beginning

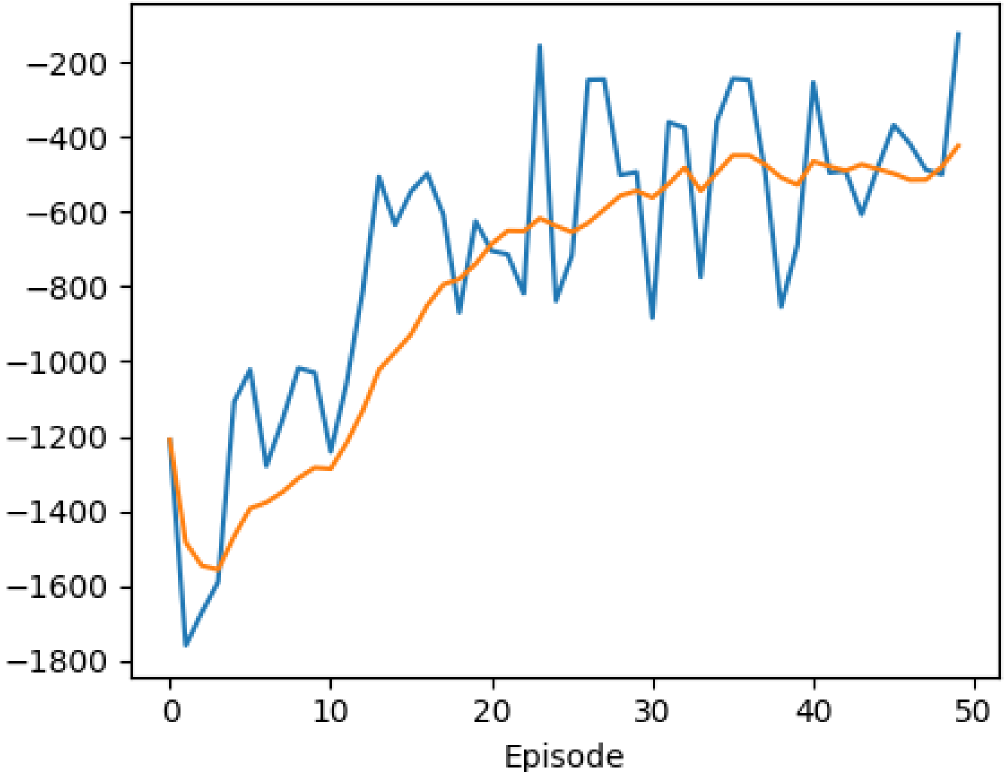


- Result



- We did this experiment to understand and prove the algorithm
- Training time:
  - 50 episodes (~5 mins)
- Result:
  - Shows that it learns the desired behavior
  - Able to keep the pendulum up

Experiment - Pendulum

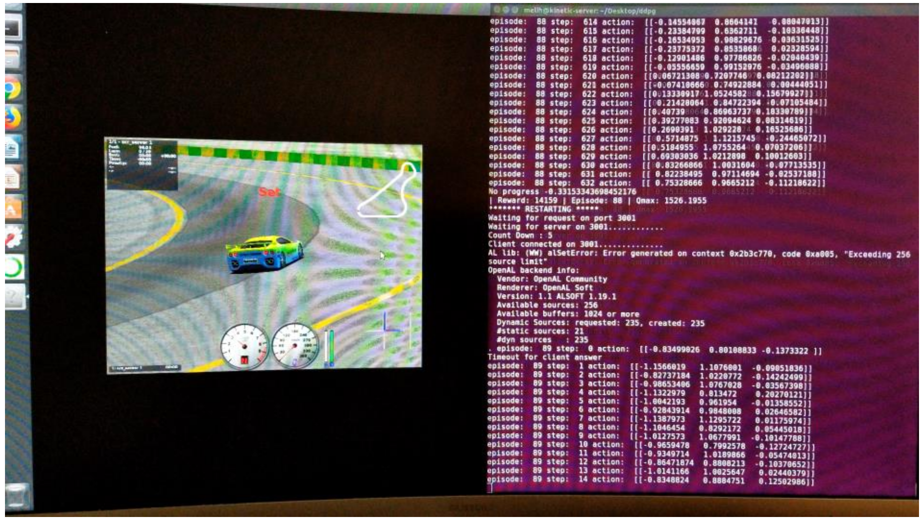


Portland State UNIVERSITY

Experiments - TORCS

- Beginning

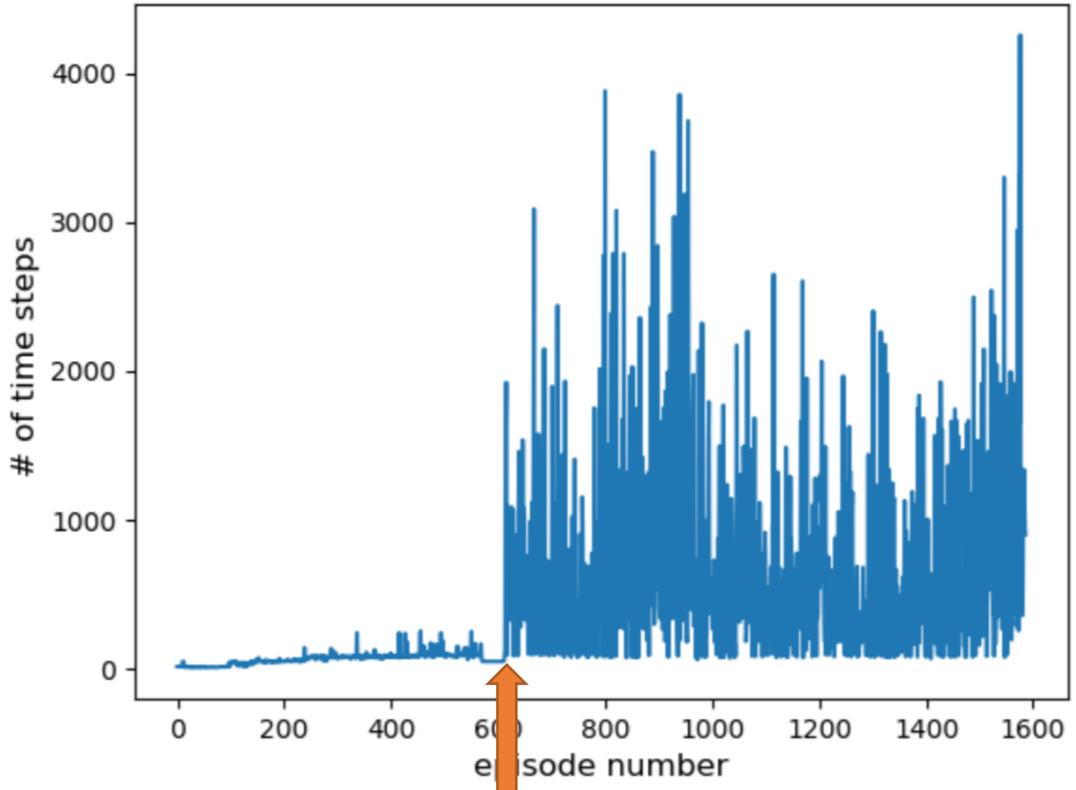
- Result



- We did this experiment to test the DDPG method for our application idea
- Training time:
  - 100 episodes (~30 mins)
  - Training this car simulation using CPU only requires ~2-3 days
- Results:
  - Shows that it is on the way of learning the desired behavior
  - Able to make turns and keep the car straight at a certain speed.

Portland State UNIVERSITY

Experiment - TORCS



>10 hours

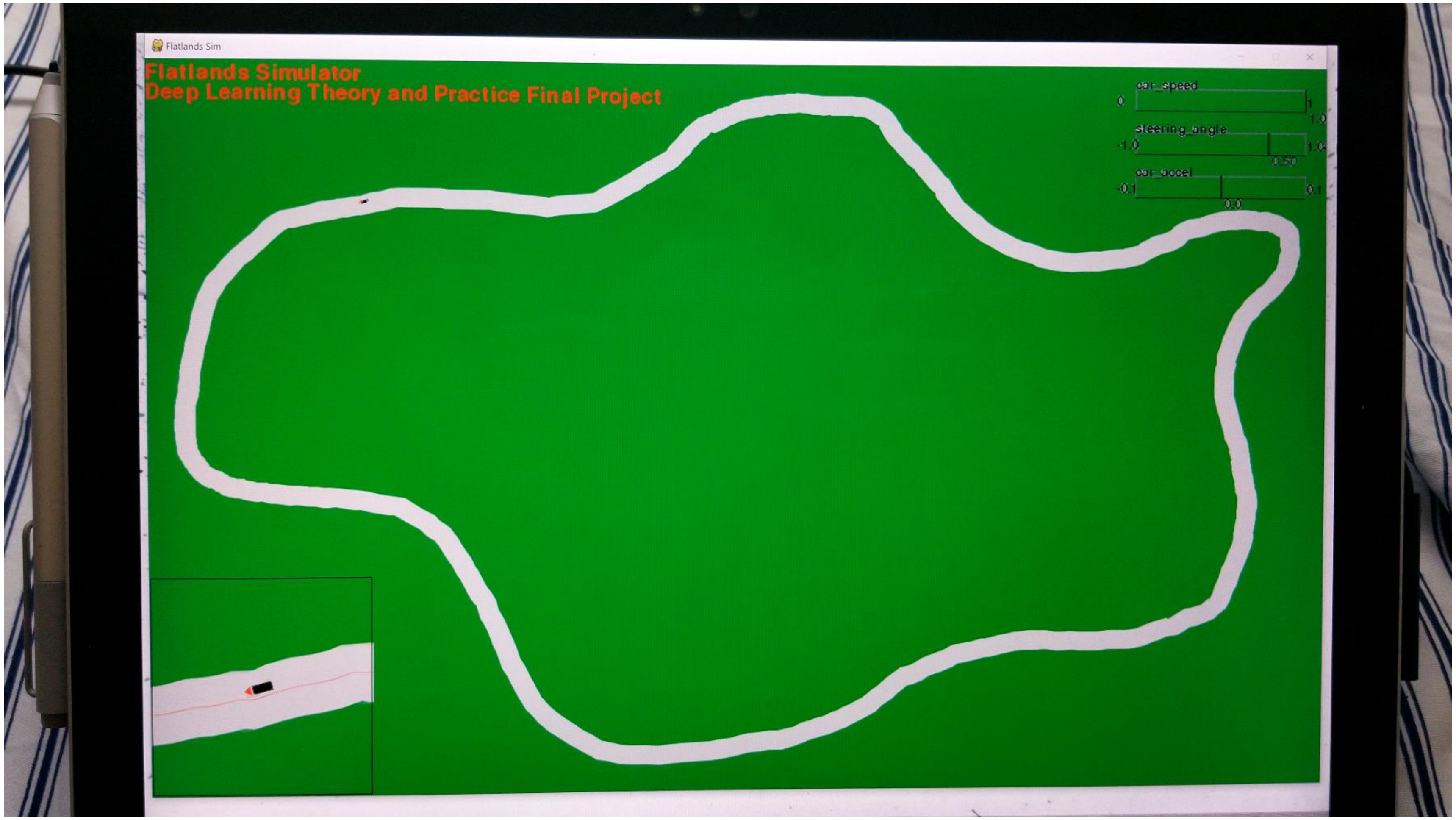
https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789533583/8/ch08lvl1sec70/training-a-ddpg-agent-to-learn-to-drive

Experiment - Flatlands

- We did this experiment to test our work
- Training time:
  - 10 episodes (~15 mins)
  - 20 episodes (~30 mins)
- Tests:
  - Tested over 20 different reward functions
  - Tested different noise levels
  - Different buffer sizes

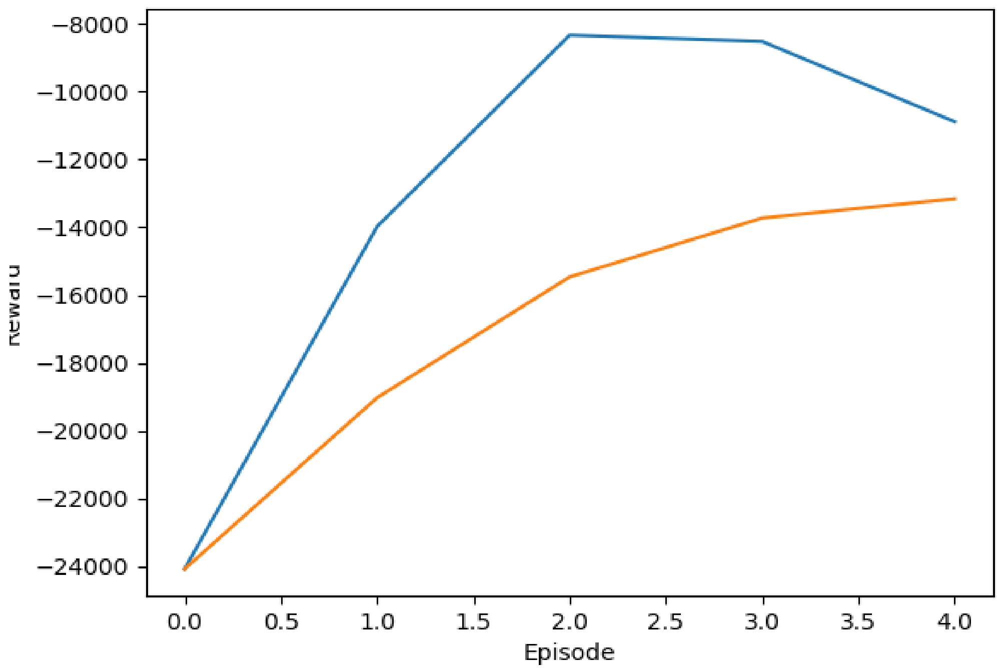
Final Result

Reward_funtions = 30 * abs(c_speed) - 10 * abs(r_speed) - 5 * (abs(math.sqrt(distance_x ** 2 + distance_y ** 2))) - 2 * head_x



Portland State
UNIVERSITY

Final Result



Future Work

- Testing different reward functions
- Testing the effects of different step size
- Using different RL algorithms to solve the same problems
  - PPO
  - A2C
  - A3C
- Testing our work on different benchmarks

Conclusion

- We were able train our agent (self-driving car) to learn how to stay closer to the track and move
- We had an opportunity to use the tools and knowledge that we learned in this course such as Pytorch, Deep Reinforcement Learning, MLP.
- Finding a good simulator that satisfies our needs for this project was a challenge.
- We have tested many different simulations build for OpenAI gym and Unity

Thanks for listening!