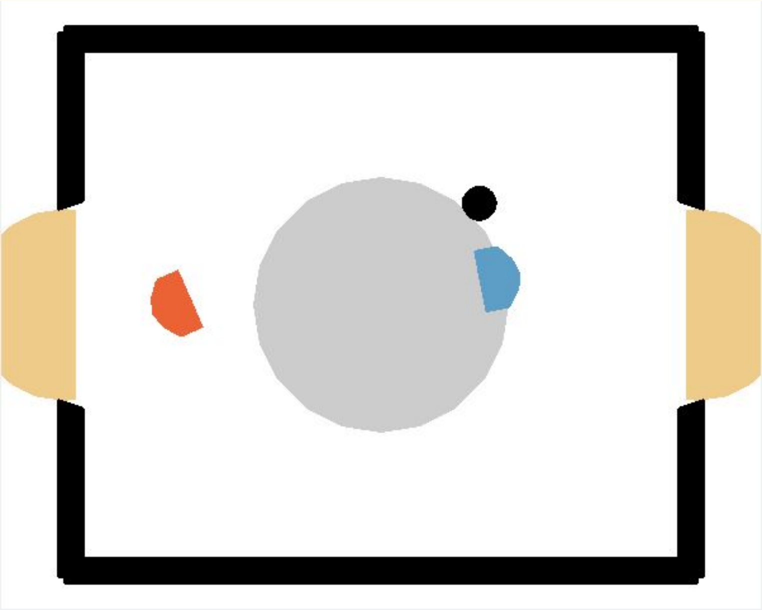
Reinforcement Learning 2024/25 WS Final Project Presentation

Emre Güçer

Task Description

- Create a Reinforcement Learning agent to play air hockey
- Create an algorithm from scratch or make modifications to an existing one.



Chosen Base Algorithm: TD3

- An improved version of DDPG
  - DDPG uses **one** Q-Function => overestimation bias
  - Solution: Use **two** Q-Function

- My code and TD3 algorithm are based on "Addressing Function Approximation Error in Actor-Critic Methods" research paper written by Scott Fujimoto, Herke van Hoof and David Meger

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Changes I made to TD3 Algorithm

1) LeakyReLU Instead of ReLU
   - ➢ Prevents dead gradients for negative inputs
2) Normalization of Layers
   - ➢ Enhances stability of training
3) He Initialization for Better Weight Distribution
   - ➢ Minimizes vanishing and exploding gradients
4) Weight Decay in Adam Optimizer
   - ➢ Prevents overfitting
5) Gradient Clipping
   - ➢ Prevents exploding gradients
6) Usage of Torch Tensors for GPU Optimization
   - ➢ Increases speed of processing
7) Not Having Explicit Save and Load Methods
   - ➢ These were moved to training file; therefore, no impact

Methodology of Training and Evaluation

1) Get parameters from user
2) If exists, load existing best agent and observed transitions
3) Start training the agent using given parameter settings
4) On every promising episode (or check point), evaluate quality of episode
    a) If results would be better than older one, replace the best agent with the new one
5) At every 5 iterations save memory so that script can refer to saved episodes

Customizability of the Training Pipeline

```python
if __name__ == '__main__':
    # all possible changes are editable by updating user_variables
    # user should be able to comment out any of these variables and it should still work since I made all variables optional
    # tried my best with adapting my Automation Engineering internship experience here.
    user_variables = {
        # train and run at the end
        "whether_to_train": False, # Boolean to train
        "whether_to_run_best_model": True, #Boolean to test the current best model
        "whether_to_render": True, # Boolean to render test
        # training settings
        "total_episodes": 80000, # Number of episodes to run
        "batch_size": 32,
        "episode_length": 251, # Depth of an episode (Hockey env has max depth 251)
        "exploration_noise": 0.2,
        "min_exploration_noise": 0.0001,
        "exploration_decay": 0.999999,
        "train_against_weak_n_number_of_times": 1000,
        # running settings
        "number_of_games": 100, # Number of games to test an episode's quality or to render - a high number is recommended since it effected my training a lot
        "opponents": [h_env.BasicOpponent(weak=True), h_env.BasicOpponent(weak=False)], # list of opponents
        # TD3 settings
        "discount": 0.99,
        "tau": 0.005,
        "policy_noise": 0.2,
        "noise_clip": 0.5,
        "policy_freq": 2,
        "hidden_dim_1": 256,
        "hidden_dim_2": 256,    You, 2 days ago • current version update.
        "learning_rate": 1e-5,
        "weight_decay": 1e-5,
        "grad_clip": 1.0,
        "leaky_relu_grad": 0.01,
        "memory_limit": 100000 # memory limit to allocate for the task
    }
    main(user_variables)
```

Performance Against the Basic Opponents

Played 100 games against both agents:

1) Test against weak opponent
2) Test against strong opponent

```
user_variables = {
    # train and run at the end
    "whether_to_train": False, # Boolean to train
    "whether_to_run_best_model": True, #Boolean to test the current best model
    "whether_to_render": False, # Boolean to render test
    # training settings
    "total_episodes": 80000, # Number of episodes to run
    "batch_size": 32,
    "episode_length": 251, # Depth of an episode (Hockey env has max depth 251)
    "exploration_noise": 0.2,
    "min_exploration_noise": 0.0001,
    "exploration_decay": 0.999999,
    "train_against_weak_n_number_of_times": 1000,
    # running settings
    "number_of_games": 100, # Number of games to test an episode's quality or to render - a high number i
    "opponents": [h_env.BasicOpponent(weak=True), h_env.BasicOpponent(weak=False)], # list of opponents
    # TD3 settings
    "discount": 0.99,
    "tau": 0.005,
    "policy_noise": 0.2,
    "noise_clip": 0.5,
    "policy_freq": 2,
    "hidden_dim_1": 256,
    "hidden_dim_2": 256,
    "learning_rate": 1e-5,
    "weight_decay": 1e-5,
    "grad_clip": 1.0,
    "leaky_relu_grad": 0.01,
    "memory_limit": 100000 # memory limit to allocate for the task
}
```

Testing Against Weak Opponent

- Agent success (Number of games with positive reward) count: 29
- Average reward: -5.43
- Success rate (Percentage of games with positive reward): 29.0%

Testing Against Strong Opponent

- Agent success (Number of games with positive reward) count: 23
- Average reward: -6.95
- Success rate (Percentage of games with positive reward): 23.0%

Overall Performance

- Average reward across all opponents: -6.19
- Average success rate (Percentage of games with positive reward): 26.0%

```
PS C:\Users\emreg\Downloads\RL> & C:/Users/emreg/AppData/Local/Programs/Python/Python310/python.exe
ckey-env-master/train.py
Loaded memory buffer with 12066 transitions (Limited to 100000)
Loaded agent from models/best_actor.pth and models/best_critic.pth

Agent will play 100 games against both Basic Opponents.

Testing against weak opponent:
Against weak opponent:
Agent success (number of games with positive reward) count:  29
Average reward: -5.43
Success rate (Percentage of games with positive reward): 29.0%

Testing against strong opponent:
Against strong opponent:
Agent success (number of games with positive reward) count:  23
Average reward: -6.95
Success rate (Percentage of games with positive reward): 23.0%

Overall Performance:
Average reward across all opponents: -6.192202752788684
Average success rate (Percentage of games with positive reward): 26.0%
```

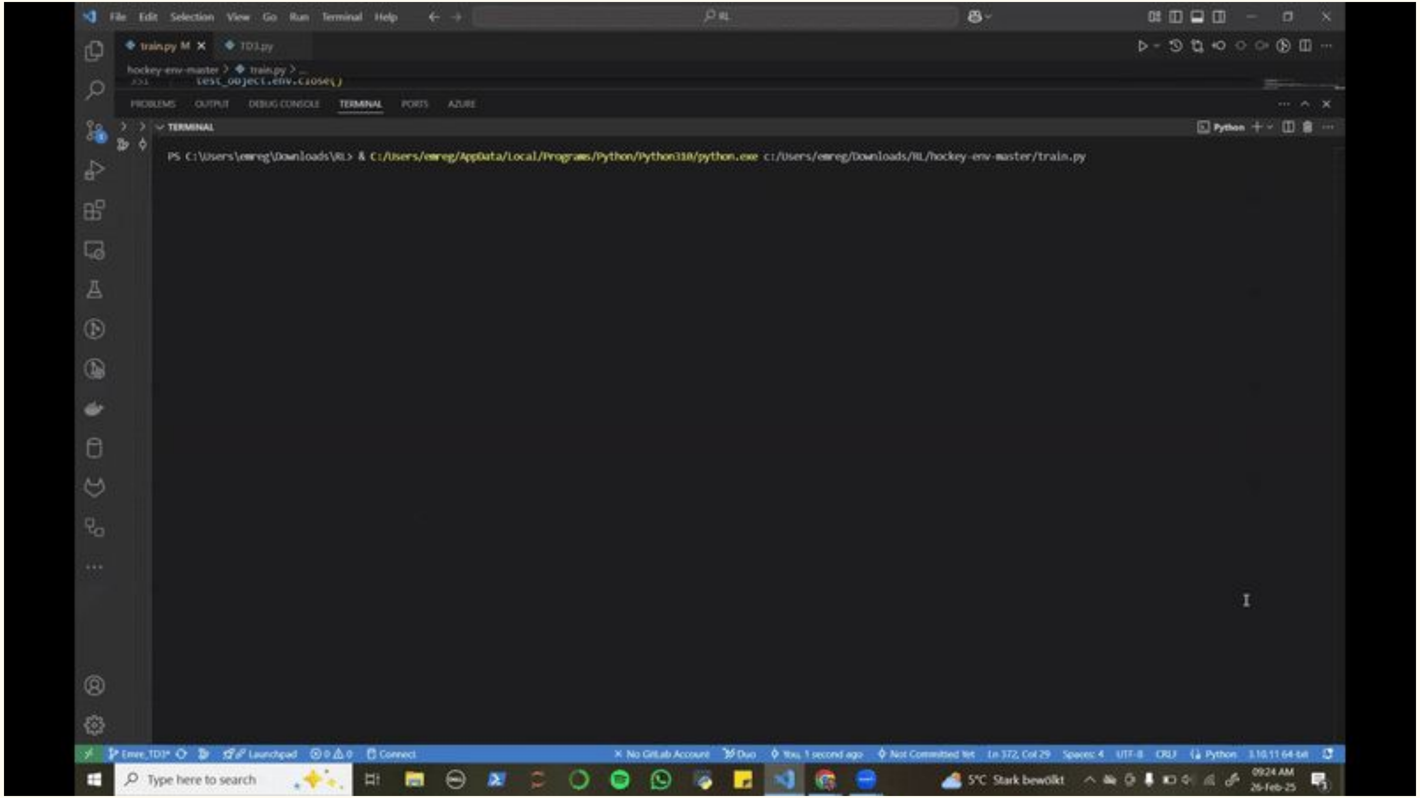
How to Train

```
user_variables = {
    # train and run at the end
    "whether_to_train": True, # Boolean to train
    "whether_to_run_best_model": False, #Boolean to test the current best model
    "whether_to_render": False, # Boolean to render test      You, yesterday • Latest version
    # training settings
    "total_episodes": 80000, # Number of episodes to run
    "batch_size": 32,
    "episode_length": 251, # Depth of an episode (Hockey env has max depth 251)
    "exploration_noise": 0.2,
    "min_exploration_noise": 0.0001,
    "exploration_decay": 0.999999,
    "train_against_weak_n_number_of_times": 1000,
    # running settings
    "number_of_games": 100, # Number of games to test an episode's quality or to render - a high number is recommended since it effected my training a lot
    "opponents": [h_env.BasicOpponent(weak=True), h_env.BasicOpponent(weak=False)], # list of opponents
    # TD3 settings
    "discount": 0.99,
    "tau": 0.005,
    "policy_noise": 0.2,
    "noise_clip": 0.5,
    "policy_freq": 2,
    "hidden_dim_1": 256,
    "hidden_dim_2": 256,
    "learning_rate": 1e-5,
    "weight_decay": 1e-5,
    "grad_clip": 1.0,
    "leaky_relu_grad": 0.01,
    "memory_limit": 100000 # memory limit to allocate for the task
}
```

How to Watch Games

```
user_variables = {
    # train and run at the end
    "whether_to_train": False, # Boolean to train
    "whether_to_run_best_model": True, #Boolean to test the current best model
    "whether_to_render": True, # Boolean to render test
    # training settings
    "total_episodes": 80000, # Number of episodes to run
    "batch_size": 32,
    "episode_length": 251, # Depth of an episode (Hockey env has max depth 251)
    "exploration_noise": 0.2,
    "min_exploration_noise": 0.0001,
    "exploration_decay": 0.999999,
    "train_against_weak_n_number_of_times": 1000,
    # running settings
    "number_of_games": 100, # Number of games to test an episode's quality or to render - a high number is recommended since it effected my training a lot
    "opponents": [h_env.BasicOpponent(weak=True), h_env.BasicOpponent(weak=False)], # list of opponents
    # TD3 settings
    "discount": 0.99,
    "tau": 0.005,
    "policy_noise": 0.2,
    "noise_clip": 0.5,
    "policy_freq": 2,
    "hidden_dim_1": 256,
    "hidden_dim_2": 256,
    "learning_rate": 1e-5,
    "weight_decay": 1e-5,
    "grad_clip": 1.0,
    "leaky_relu_grad": 0.01,
    "memory_limit": 100000 # memory limit to allocate for the task
}
```

Demo: Recorded Games



Future Improvements

- Forcing some parameters (especially number times an episode gets evaluated for this project) to have certain bounds.

Repo Link

https://github.com/gucere/RL

Thank You for Listening