04/18/2025

Northeastern University

CS 5180:
Reinforcement Learning and Sequential Decision Making

Sec 01 Spring 2025

Project: Humanoid Locomotion using Reinforcement Learning

Instructor: **Prof. Robert Platt**

Akilan Baskar

I. Motivation / Problem Statement

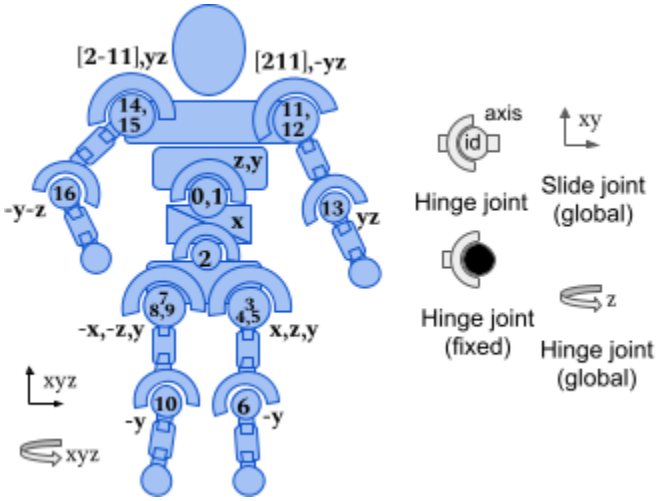
Humanoid locomotion is the ability of the robots to move like humans, typically walking or running. This remains a challenging problem in robotics and reinforcement learning due to the complexities involved such as – multi joint coordination like head, arms, torso, legs. It is not just making the robot move but maintaining balance and ensuring natural and stable human like movements. Humans are very skilled at maintaining balance while walking, whereas a robot needs to adjust its posture dynamically and stay balanced while walking, especially on uneven surfaces. Moreover, the more degree of freedom a robot has, the harder it is to manage and coordinate the movement of each joint to ensure smooth motion.

Solving humanoid locomotion problems will allow robots to assist humans with everyday tasks like carrying objects, helping elderly, etc. They will be able to work in environments such as homes, offices, and hospitals as well as outdoor environments (e.g., rescues missions) providing assistance.

II. Simulator / Environment Details

OpenAI Gymnasium is a toolkit for developing and comparing reinforcement learning (RL) algorithms — where RL models can be trained and tested. It provides a wide variety of environments like games, robot simulations, and control tasks. For this project, I use **OpenAI Gymnasium Humanoid** environment. The Humanoid-v4 environment simulates a humanoid robot using the **MuJoCo simulator**. MuJoCo (Multi-joint Dynamics with Control) is a physics engine designed to simulate complex robotic systems. The 3D bipedal robot is designed to simulate a human. It has a torso with a pair of legs and arms, and a pair of tendons connecting the hips to the knees. The legs each consist of three body parts — thigh, shin, foot, and the arms consist of two body parts — upper arm and forearm. The robot has a high number of degrees of freedom, making it one of the complex and advanced environments in the Gym suite.

Action Space:



Action	Control Min	Control Max	Name (in corresponding XML file)	Joint	Type (Unit)
Torque applied on the hinge in the y-coordinate of the abdomen	-0.4	0.4	abdomen_y	hinge	torque (N m)
Torque applied on the hinge in the z-coordinate of the abdomen	-0.4	0.4	abdomen_z	hinge	torque (N m)
Torque applied on the hinge in the x-coordinate of the abdomen	-0.4	0.4	abdomen_x	hinge	torque (N m)
Torque applied on the rotor between torso/abdomen and the right hip (x-coordinate)	-0.4	0.4	right_hip_x (right_thigh)	hinge	torque (N m)

Fig. 1. Robot joints and a subset of the action space.

Action space is the set of all possible actions an agent can take at any given time in the environment. The humanoid robot has **17** continuous action spaces which include hinge joints, and the joint torques. The range of joint torques that can be applied to each joint is [-0.4, 0.4] Nm.

Observation Space:

Observation	Min	Max	Name (in corresponding XML file)	Joint	Type (Unit)
z-angle of the abdomen (in lower_waist)	-Inf	Inf	abdomen_z	hinge	angle (rad)
y-angle of the abdomen (in lower_waist)	-Inf	Inf	abdomen_y	hinge	angle (rad)
x-angle of the abdomen (in pelvis)	-Inf	Inf	abdomen_x	hinge	angle (rad)

Fig. 2. A subset of the observation space of humanoid robot.

Observation space is the set of all possible information the agent can sense or receive from the environment at each time step. The observation space consists of the following parts:

- The position values of the robot's body parts: *22*
- The velocities of these individual body parts: *23*
- Mass and inertia of the rigid body parts: *130*
- Center of mass based velocity: *78*
- Constraint force as the actuator force at each joint: *17*
- Center of mass based external force on the body parts: *78*

➢ Total observation space: **348**

Rewards:

Reward is the feedback from the environment that tells the agent how good or bad its action was in a given state. The agent's goal is to maximize the total reward over time.

- *Healthy reward*: Every timestep that the robot is alive, it gets a +5 reward.
- *Forward reward*: A reward for moving forward.
- *Control cost*: A negative reward to penalize the robot for taking actions that are too large.
- *Contact cost*: A negative reward if the external contact forces are too large.

➢ *Total reward = Healthy reward + Forward reward – Control cost – Contact cost*

Termination:

Termination or episode ends mark the end of an interaction cycle between the agent and the environment. This could be due to success, failure or time limit.

- The episode ends when the robot is unhealthy. (z-coordinate of the torso is not in [1.0, 2.0])
- Default duration of an episode is 1000 timesteps.

III. Methodology / Approach

The objective of this project is to:

- Make a humanoid robot walk as fast as possible without falling and maintaining balance.
- Apply and compare two closely related RL algorithms.

In order to accomplish this objective, I used the following method / approach:

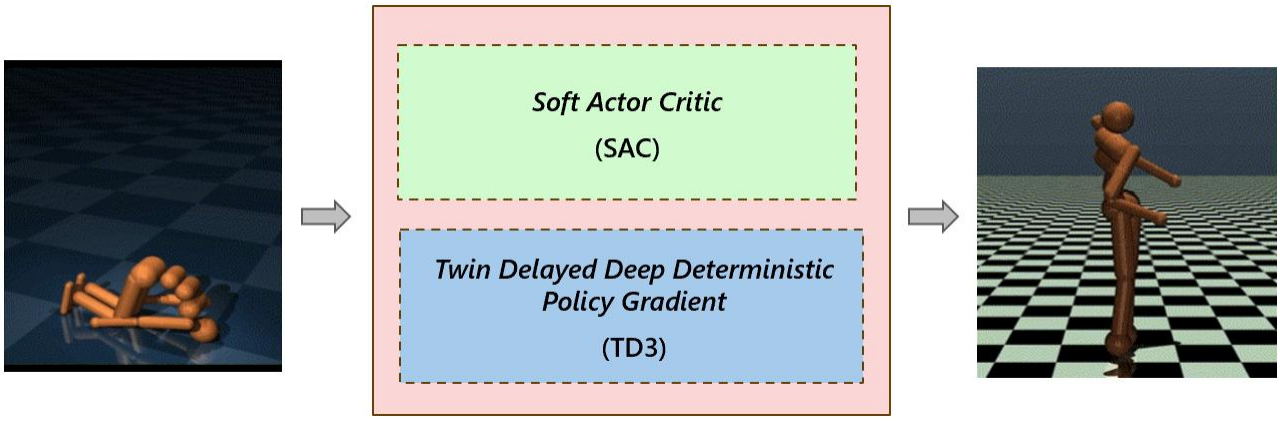


Fig. 3. System Architecture.

Experimental Setup

Input: Action (17-D) [Actor: state → action; Critic: state, action → Q-value]
Output: Next state (348-D), reward.
RL Algorithms Used: Soft Actor Critic & Twin Delayed Deep Deterministic Policy Gradient.

Soft Actor Critic (SAC):

Soft Actor Critic is an off-policy RL algorithm that is designed to work with environments that have continuous action spaces. It learns efficient policies by maximizing the reward and maximizing the entropy. This encourages exploration by making the agent choose more diverse actions rather than always sticking to the greedy ones. The algorithm uses Actor-Critic architecture — Actor (policy network) that learns which action to take based on the current state; Critic (Q-network) estimates the value of actions for state-action pairs. SAC uses two Q-networks to reduce overestimation bias and learns from previously collected experiences (replay buffer), making it sample efficient. It also uses Entropy Regularization for stable learning. By adding entropy to the reward function, SAC encourages the agent to maintain exploration by keeping the policy more stochastic. This makes it less likely to get stuck in suboptimal deterministic strategies, which is particularly useful in complex environments where exploration is important.

Twin Delayed Deep Deterministic Policy Gradient (TD3):

TD3 is an off-policy RL algorithm that is designed to improve the stability and performance of DDPG especially in continuous action spaces. TD3 uses twin Q-networks for estimating the Q-value, which helps reduce overestimation bias. The algorithm updates the policy network less frequently than Q-networks to improve stability. TD3 adds action noise to encourage exploration and performs target value smoothing to ensure the agent doesn't get stuck in suboptimal policies or overfit. It is sample-efficient and makes good use of experience replay.

	SAC	TD3
Policy Type	Off-policy (stochastic)	Off-policy (deterministic)
Architecture	Actor-critic	Actor-critic
Q-networks	Two Q-networks	Two Q-networks
Experience Replay	Yes	Yes
Exploration	Exploration via Entropy	Exploration via action noise
Stability	Entropy maximization/regularization	Delayed updates, target smoothing

Thus, SAC and TD3 are the two closely related algorithms that are used in this experiment.

IV. Results

SAC and TD3 algorithms are trained and tested on the Humanoid-v4 Gymnasium environment. The following are the models hyperparameters:

```
GAMMA = 0.99                # Discount factor
LR = 3e-4                   # Learning rate for all optimizers
TAU = 0.01                  # Soft update rate for target networks
ALPHA = 0.2                 # Entropy regularization coefficient for SAC
POLICY_NOISE = 0.1          # Noise added to target policy during critic update (TD3)
NOISE_CLIP = 0.2            # Range to clip target policy noise (TD3)
POLICY_DELAY = 4            # Frequency of policy updates (TD3)
BUFFER_SIZE = 1000000       # Maximum size of replay buffer
BATCH_SIZE = 256            # Mini-batch size for updates
SAVE_INTERVAL = 25000       # Steps between saving model checkpoints
```

Fig. 4. Hyperparameters of SAC and TD3.

I created SAC and TD3 actor networks and critic Q-network using neural networks. I also created replay buffers to store experiences. I trained both the algorithms for 5M timesteps and saved the logs and models. I visualized the plots on TensorBoard, and the following are the results:

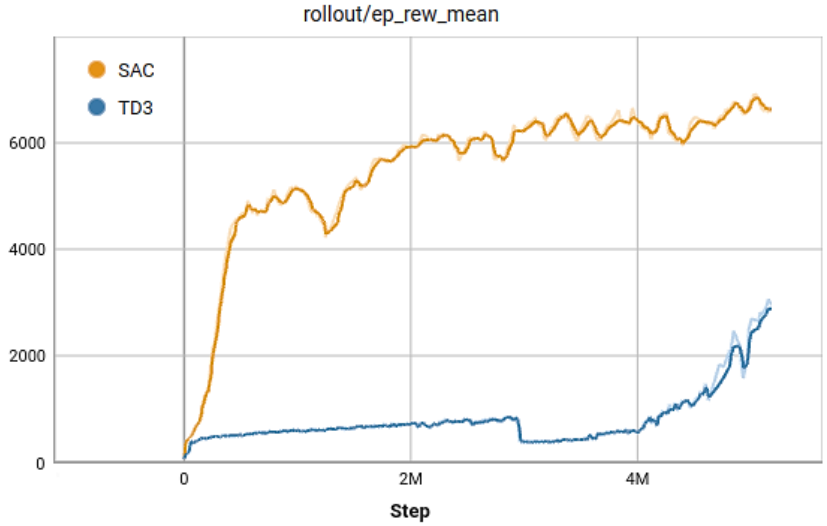


Fig. 4. Average rewards of SAC and TD3 algorithms.

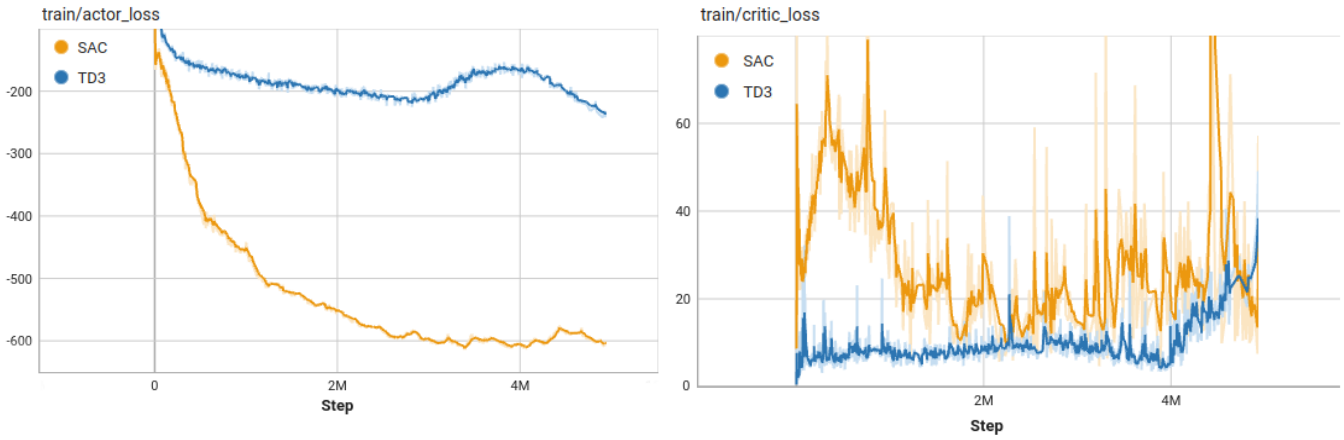


Fig. 5. Actor and critic loss of SAC and TD3 algorithms.

- From Fig.4, we could see that SAC quickly climbs to ~4500 reward by ~500k timesteps and stabilized around 6000+ at the end of training. Whereas, TD3 learns more slowly, staying under 1000 reward. Notable improvements only after 4M timesteps, finishing at 2500.
- SAC's actor loss decreases steadily and saturates around -600, indicating increasing confidence in policy improvement. Whereas TD3's actor loss hovers around -200, with mild fluctuations but no steep convergence like SAC.
- SAC's critic loss is highly volatile during training, whereas TD3 exhibits lower and more stable critic loss.

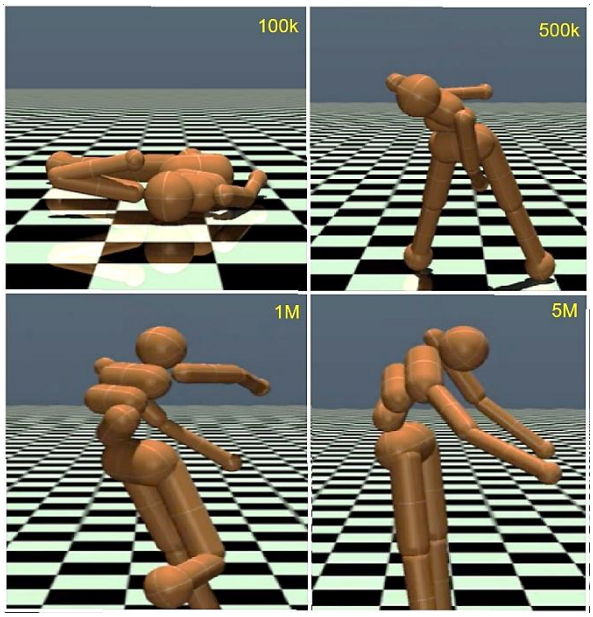


Fig. 6. Snapshots of Humanoid trained on SAC.

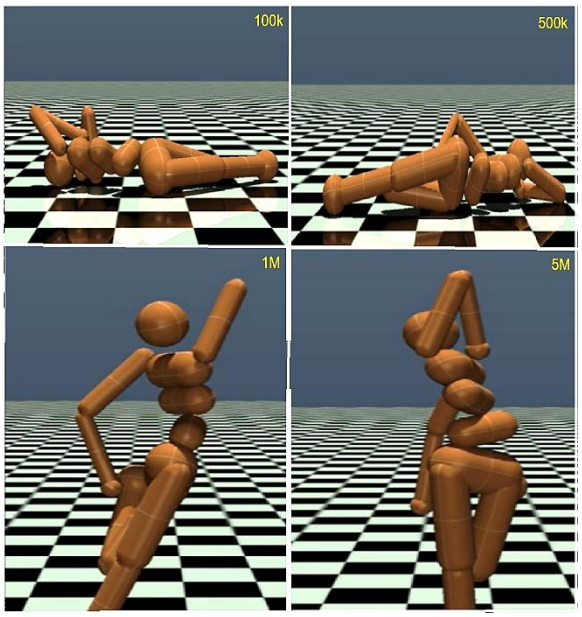


Fig. 7. Snapshots of Humanoid trained on TD3.

- In case of SAC, by 1M timesteps, the humanoid robot started to walk upright but slowly. At 5M timesteps, the humanoid started to walk almost like a human, really fast without falling.
- In case of TD3, at 1M timesteps the humanoid was able to barely stand up and at 5M, the humanoid didn't walk but was hopping around without falling. The posture was nothing like human.

V. Conclusion

Humanoid locomotion involves complex multi joint coordination and control. RL allows robots to learn them by exploring and adapting movements, without needing to explicitly program every detail. It also allows robots to learn robust locomotion policies that generalize better across variable conditions and terrain without falling. From the experiment, we could see that RL algorithms can indeed make humanoid robots walk without falling. Upon training, we could see from the results that SAC algorithm learns faster and converges to a better performing policy compared to TD3. On the other hand, TD3 is more conservative and stable but struggles with early performance and slower convergence. In conclusion, SAC is better suited for complex, continuous high dimensional environments like Humanoid.

VI. Future Work

Some of the challenges that I faced during the project include — high dimensional environment and hyperparameter tuning. Humanoid-v4 environment's high observation and action space, it makes learning harder and slower due to curse of high dimensionality. Both algorithms are sensitive to hyperparameters, and I had a hard time finding the right parameters. Also even with GPUs, training time took several hours for running 5M steps. Given more time, I would try to optimize hyperparameters and improve training efficiency. I would also try to run on other newer algorithms to see how they perform.

VII. Miscellaneous

Link to GitHub:

- https://github.com/baskar-ak/ReinforcementLearningFinalProject

Links to output video:

- SAC: Humanoid on SAC
- TD3: Humanoid on TD3