Making a Humanoid Robot learn how to walk using Reinforcement Learning

M Faraz Khalil – 33050322

MSc Artificial Intelligence

Sheffield Hallam University, Department of Computing

Abstract

With the advancement in the field of Artificial Intelligence, intelligent algorithms to solve real-world problems and aid human life are becoming more and more relevant and this drive has brought us to making an actual humanoid machine that can mimic a real person but can perform highly advanced tasks. This research is one step towards that goal. The algorithm focuses on making the robot try to learn to take as many steps as possible using an AI (Artificial Intelligence) technique called reinforcement learning. Reinforcement learning works without any pre-collected data, it's a technique that makes the algorithm learn in real-time by continuously observing its state and environment.

Introduction

1. Problem:

The algorithm focuses on making a robot learn how to walk using a simulated environment, initially robot starts standing and the algorithm continuously moves its limbs i.e. legs to try to make it walk. The goal of the project is to make it walk as fast as possible without falling over.

2. Dataset:

The environment used to run the algorithm is Humanoid-v4, it's a 3D bipedal robot that is designed to simulate humans, it has a torso with a pair of legs and arms. The pair of legs and arms both have two links each to move. The environment is part of the Mujoco project currently managed by Gymnasium API for reinforcement learning. It's a maintained fork of OpenAI's Gym library which is widely used for reinforcement learning problems.

AI Technique:

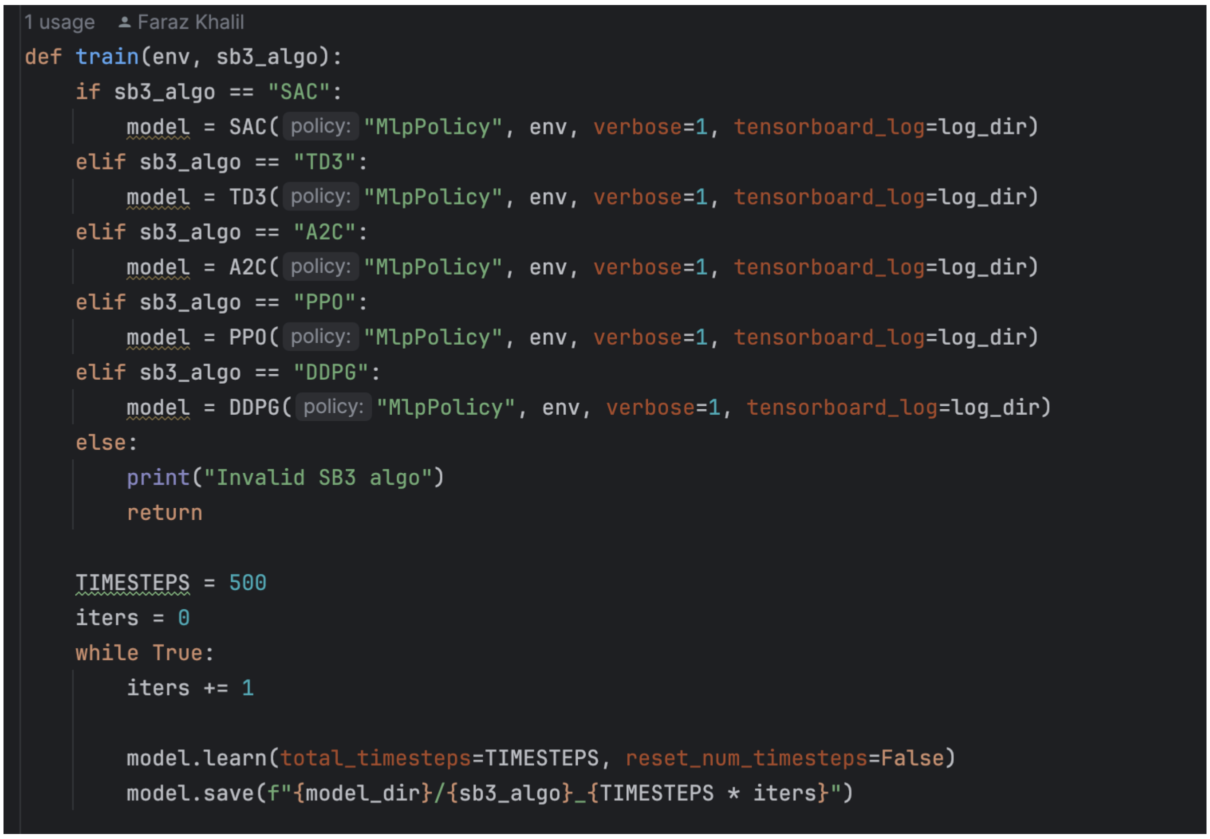
The AI technique used to solve our problem is called Reinforcement Learning. In RL, we do not provide or have a set of pre-collected data from any source to train our model against as in traditional supervised learning. The agent is made to perform random actions initially in each state and the effect of those actions in the environment is observed, if the action proves helpful towards our goal, then the likelihood of that action is increased, and a reward is given to the agent but if the action is not proven good then a negative reward is given to discard that action hence a policy is created using the agent's action and the policy is updated after every action taken by the agent in any given state so the algorithm is continuously creating its data for training in real-time as it interacts with its environment and is learning from it with a specialized reward setting. The algorithm used for this research to train the humanoid robot is Proximal Policy Optimization (PPO).

1. Reason

Due to the model-free environment problem, the agent was trained against multiple suitable algorithms like:

- Advantage Actor Critic (A2C)
- Deep Deterministic Policy Gradient (DDPG)
- Proximal Policy Optimization (PPO)
- Soft Actor Critic (SAC)
- Twin Delayed DDPG (TD3)

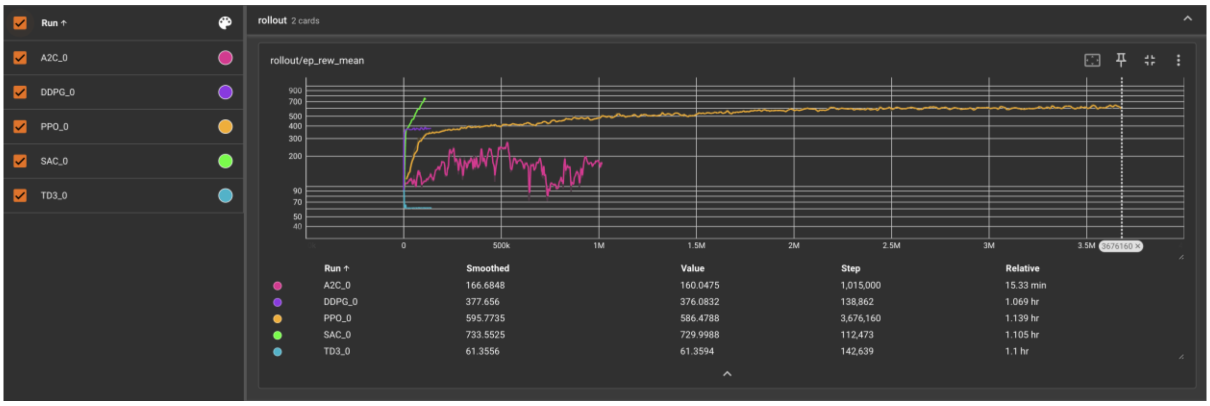
Here a piece of code shows running the training comparison between all of the above algorithms.



```
1 usage   Faraz Khalil
def train(env, sb3_algo):
    if sb3_algo == "SAC":
        model = SAC( policy: "MlpPolicy", env, verbose=1, tensorboard_log=log_dir)
    elif sb3_algo == "TD3":
        model = TD3( policy: "MlpPolicy", env, verbose=1, tensorboard_log=log_dir)
    elif sb3_algo == "A2C":
        model = A2C( policy: "MlpPolicy", env, verbose=1, tensorboard_log=log_dir)
    elif sb3_algo == "PPO":
        model = PPO( policy: "MlpPolicy", env, verbose=1, tensorboard_log=log_dir)
    elif sb3_algo == "DDPG":
        model = DDPG( policy: "MlpPolicy", env, verbose=1, tensorboard_log=log_dir)
    else:
        print("Invalid SB3 algo")
        return

    TIMESTEPS = 500
    iters = 0
    while True:
        iters += 1

        model.learn(total_timesteps=TIMESTEPS, reset_num_timesteps=False)
        model.save(f"{model_dir}/{sb3_algo}_{TIMESTEPS * iters}")
```

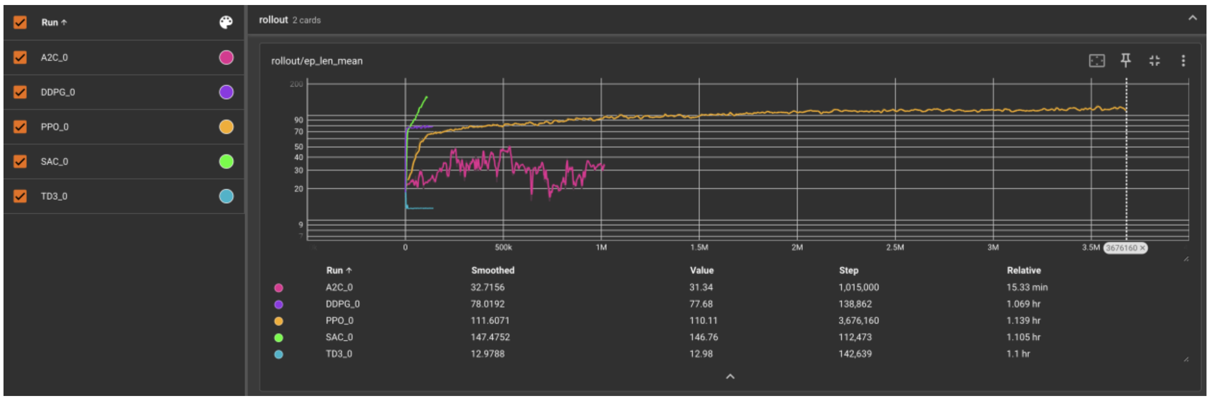
Fig 1: Comparison code for training

After training our agent with all the algorithms with default settings for more than 3.5 million timesteps the output was observed as:



	Run ↑	Smoothed	Value	Step	Relative
●	A2C_0	166.6848	160.0475	1,015,000	15.33 min
●	DDPG_0	377.656	376.0832	138,862	1.069 hr
●	PPO_0	595.7735	586.4788	3,676,160	1.139 hr
●	SAC_0	733.5525	729.9988	112,473	1.105 hr
●	TD3_0	61.3556	61.3594	142,639	1.1 hr

Fig 2: Mean Episode Reward



	Run ↑	Smoothed	Value	Step	Relative
●	A2C_0	32.7156	31.34	1,015,000	15.33 min
●	DDPG_0	78.0192	77.68	138,862	1.069 hr
●	PPO_0	111.6071	110.11	3,676,160	1.139 hr
●	SAC_0	147.4752	146.76	112,473	1.105 hr
●	TD3_0	12.9788	12.98	142,639	1.1 hr

Fig 3: Mean Episode Length

As presented in Fig 3 and Fig 4 graphs only 2 out of 5 algorithms i.e. PPO and SAC show promising results with increasing rewards received with time. Other algorithms either performed very poorly or even stopped learning altogether so were discarded right away.

In the case of SAC even though it shows a good sum of rewards with time, its training process was significantly slower than PPO which hints that the problem might be a bit complex for SAC to solve and further modifications to it could even make it slower which is not good in the long run. Hence, PPO was selected as an obvious option to move forward with since its learning time was significantly better than the others with a very consistent sum of rewards.

2. About PPO

it's an on-policy mode-free algorithm that is one of the most widely used techniques for reinforcement learning. In 2018, PPO received a wide variety of success in its performance in robotics and for excelling in games like Atari and Dota 2. As compared to other algorithms PPO is known for its simplicity, stability, and sample efficiency.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta))\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Fig 4: PPO Objective

PPO was initially introduced to eliminate the flaws of Trust Region Policy Optimization (TRPO) which is less sample-efficient. The main idea of PPO is that after an update in the state, the new policy should not be too far from the old policy, and for that, it uses a clipping method to avoid a large policy update. It also uses an advantage function (A^t) which is the difference between the expected sum of the reward and the actual some of the reward of an agent. If the advantage function gets a positive value, then that action's future probability increases and vice versa.

Implementation

1. Pre-processing on data:

OpenAI gyms are very widely set of environments for any reinforcement algorithm, and they work right of the box without the need for any kind of pre-processing for the dataset.

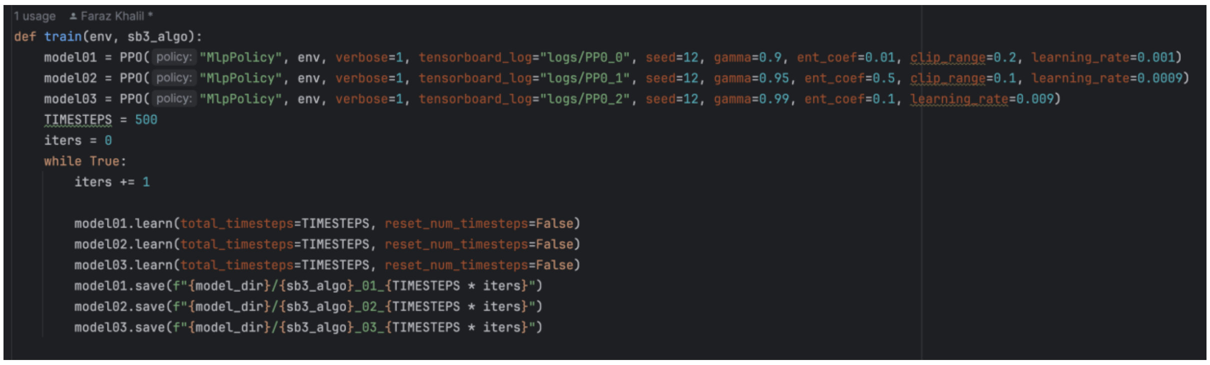
2. Hyperparameters Initialization:

PPO uses multiple hyperoperators which are as follows with default initializations "learning_rate=0.0003" (an update step in gradient descent), "n_steps=2048" (number of steps to run for environment per episode), "batch_size=64" (minibatch size), "n_epochs=10" (number of epoch while optimizing surrogate loss), "gamma=0.99" (discount factor), "clip_range=0.2" (clipping parameter for policy change) "enf_corf=0.0" (Entropy coefficient used for clipping).

Fine Tuning

The fine-tuning introduced into the default implementation of the algorithm to try to make it get better rewards is tweaking in a couple of its hyperparameters, which includes:

- learning_rate: An update step in gradient descent
- gamma: Discount Factor
- ent_coef: Entropy coefficient
- clip_range: Clipping parameter for policy change



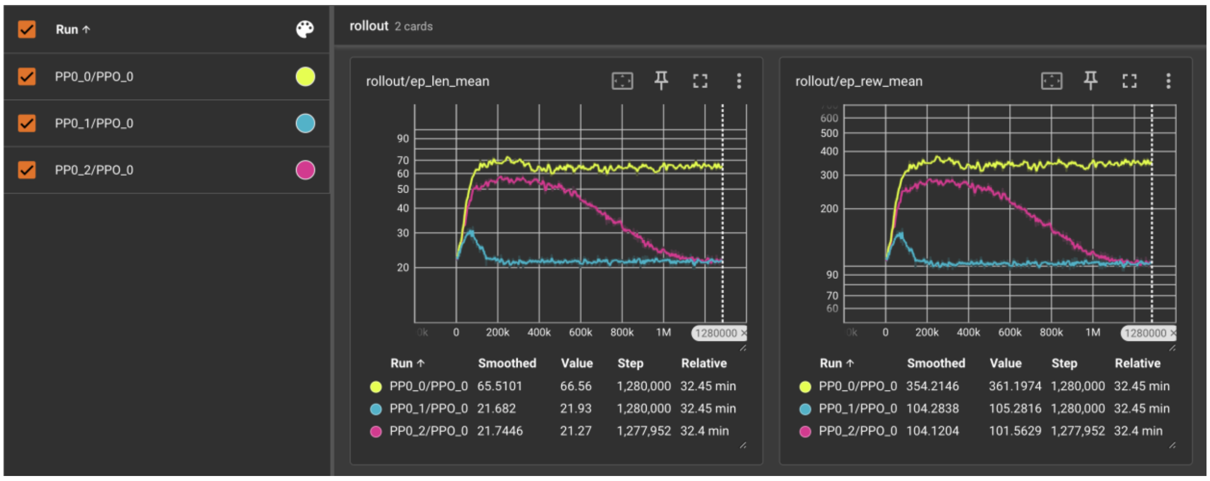
```
def train(env, sb3_algo):
    model01 = PPO( policy: "MlpPolicy", env, verbose=1, tensorboard_log="logs/PPO_0", seed=12, gamma=0.9, ent_coef=0.01, clip_range=0.2, learning_rate=0.001)
    model02 = PPO( policy: "MlpPolicy", env, verbose=1, tensorboard_log="logs/PPO_1", seed=12, gamma=0.95, ent_coef=0.5, clip_range=0.1, learning_rate=0.0009)
    model03 = PPO( policy: "MlpPolicy", env, verbose=1, tensorboard_log="logs/PPO_2", seed=12, gamma=0.99, ent_coef=0.1, learning_rate=0.009)
    TIMESTEPS = 500
    iters = 0
    while True:
        iters += 1

        model01.learn(total_timesteps=TIMESTEPS, reset_num_timesteps=False)
        model02.learn(total_timesteps=TIMESTEPS, reset_num_timesteps=False)
        model03.learn(total_timesteps=TIMESTEPS, reset_num_timesteps=False)
        model01.save(f"{model_dir}/{sb3_algo}_01_{TIMESTEPS * iters}")
        model02.save(f"{model_dir}/{sb3_algo}_02_{TIMESTEPS * iters}")
        model03.save(f"{model_dir}/{sb3_algo}_03_{TIMESTEPS * iters}")
```

Fig 5: Hyperparameters tuning in PPO

These are some hyperparameters that are key in PPO which are key parameters in meeting parameter objective goal.

Evaluation/Result Analysis

After training multiple agents in parallel with a different set of hyperparameters shows that indeed the PPO doesn't need lot of hyperparameters tuning as suggested by its developers that this algorithm was created for ease of use and implementation. Fig 5 shows the code to used to compare the PPO algorithm with different set of algorithms and the result graph shows something like this:



rollout/ep_len_mean

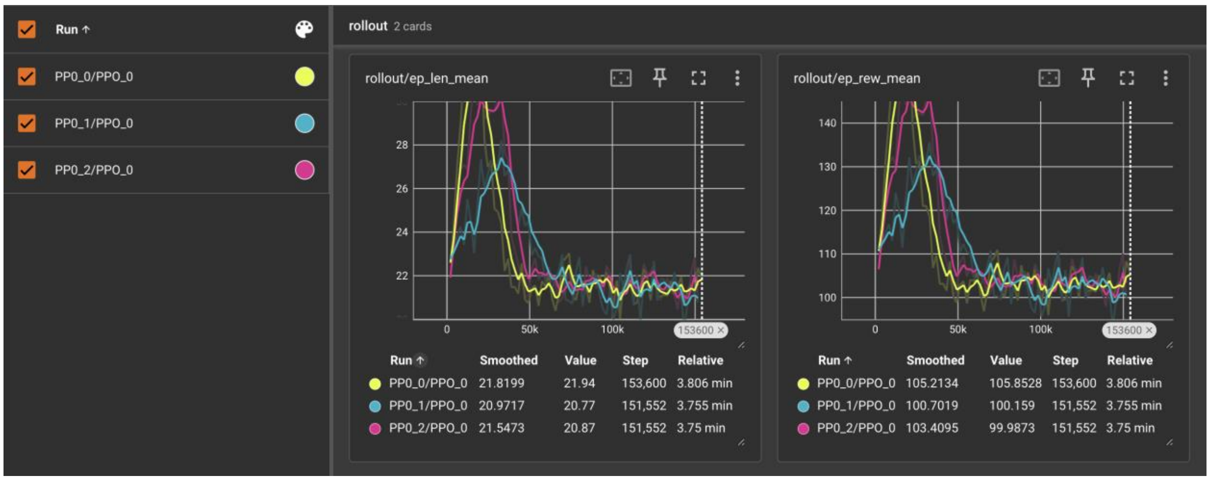
Run ↑	Smoothed	Value	Step	Relative
PPO_0/PPO_0	65.5101	66.56	1,280,000	32.45 min
PPO_1/PPO_0	21.682	21.93	1,280,000	32.45 min
PPO_2/PPO_0	21.7446	21.27	1,277,952	32.4 min

rollout/ep_rew_mean

Run ↑	Smoothed	Value	Step	Relative
PPO_0/PPO_0	354.2146	361.1974	1,280,000	32.45 min
PPO_1/PPO_0	104.2838	105.2816	1,280,000	32.45 min
PPO_2/PPO_0	104.1204	101.5629	1,277,952	32.4 min

Fig 6: Hyperparameters tuning output graphs using PPO

In Fig 6, the PPO_0 the shows the logs of the agent with least amount of hyperparameters changing and the result graph is shown in yellow line, the agents with more changes just stops working eventually.



rollout/ep_len_mean

Run ↑	Smoothed	Value	Step	Relative
PPO_0/PPO_0	21.8199	21.94	153,600	3.806 min
PPO_1/PPO_0	20.9717	20.77	151,552	3.755 min
PPO_2/PPO_0	21.5473	20.87	151,552	3.75 min

rollout/ep_rew_mean

Run ↑	Smoothed	Value	Step	Relative
PPO_0/PPO_0	105.2134	105.8528	153,600	3.806 min
PPO_1/PPO_0	100.7019	100.159	151,552	3.755 min
PPO_2/PPO_0	103.4095	99.9873	151,552	3.75 min

Fig 7: Hyperparameters tuning output graphs using PPO

More changes in hyperparameters results in complete loss of reward in the agents.

Ethical Implications

There are quite opposing views on both sides of the argument when it comes to ethical implications of any kind of robot. Some of the positive aspects are:

- More productivity in menial tasks which will ultimately result in more production.
- Tasks done by robots will be less prone to error.

- Human life can be preserved by robots replacing them in danger environments.
- Humans can focus more on research then any physical jobs.

While on the other hand, the negative said argues:

- Robots will result in unemployment for human
- AI bias (which can be called as creator bias)
- Being surpassed by robots
- Risk of Artificial error in new scenarios

Conclusions

In this paper we have tried coming up with the most optimal way/algorithm to train an agent to learn to walk using reinforcement learning, we explored a widely used dataset/environment for machine learning and compared multiple reinforcement learning algorithms by training them in parallel on the same environment and comparing their learning progress. After training all we shortlisted PPO for our final task and ran the model with multiple tuned versions of PPO to check whether we could get better results or not. Overall, this was a great learning experience and an even better insight into the workings of reinforcement learning which will prove to be a stepping stone to even better and complex projects.

References

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. ArXiv.org. https://arxiv.org/abs/1707.06347

Todorov, E., Erez, T., & Tassa, Y. (2012, October 1). *MuJoCo: A physics engine for model-based control*. IEEE Xplore. https://doi.org/10.1109/IROS.2012.6386109

Tassa, Y., Erez, T., & Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. https://www.semanticscholar.org/paper/Synthesis-and-stabilization-of-complex-behaviors-Tassa-Erez/71b552b2e058d5a6a760ba203f10f13be759edd3

johnnycode8. (2024, March 2). *johnnycode8/gym_solutions*. GitHub. https://github.com/johnnycode8/gym_solutions

Papers with Code - Reinforcement Learning (RL). (n.d.). Paperswithcode.com. Retrieved April 23, 2024, from https://paperswithcode.com/task/reinforcement-learning-1

Papers with Code - OpenAI Gym. (n.d.). Paperswithcode.com. Retrieved April 23, 2024, from https://paperswithcode.com/paper/openai-gym