Teaching a Simulated Bipedal Robot to Navigate an Uneven Terrain

Anas Rezk, Chris Hounslow, Greg Toth, Marco D'Amato
Department of Computer Science
University of Bath
Bath, BA2 7AY
`ar2601@bath.ac.uk, ch2522@bath.ac.uk, gt566@bath.ac.uk, mda63@bath.ac.uk`

1 Problem Definition

The target is to solve the so-called bipedal walker, included in OpenAI's 'Gym' (Brockman et al., 2016). Oleg Klimov (2022) describes it as "a simple 4-joint walker robot environment". There are two versions:

- Normal, with slightly uneven terrain.
- Hardcore, with ladders, stumps, pitfalls.

To solve an episode of the normal version, a reward of 300 or greater is required. An agent is considered to have completed this problem by getting an average reward of at least 300 over 100 consecutive trials (BipedalWalker v2, 2020).

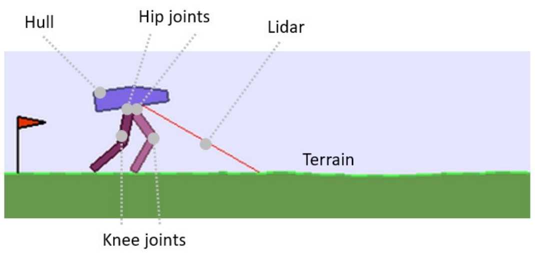
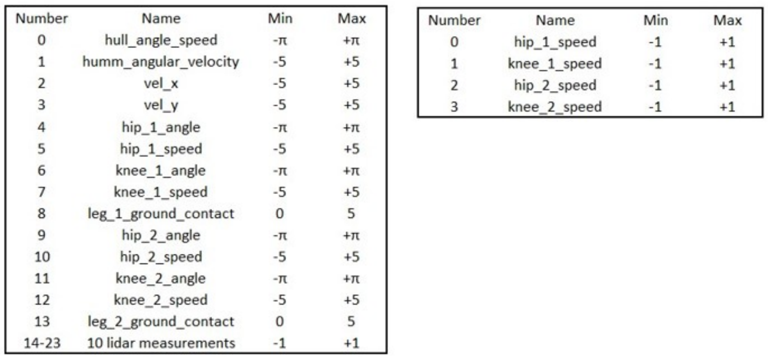


Figure 1: Bipedal walker

The bipedal walker has 4 joints (2 knees and 2 hips), hence a total of 4 motors. The action space is defined by the motor speed as continuous values in the [-1, 1] range.
Looking at the observation space, it consists of a 24-dimensional continuous vector (24 parameters per state), ground contact points, and 10 lidar rangefinder measurements values (Figure 2). Therefore, the sequential decision problem here is qualified as a Partially Observed Markov Decision Process (MDP) because the agent does not have complete information about the environment such as the terrain it has walked on as well as satisfying the Markov property, where future states depend only on the current state.



Number	Name	Min	Max
0	hull_angle_speed	-π	+π
1	humm_angular_velocity	-5	+5
2	vel_x	-5	+5
3	vel_y	-5	+5
4	hip_1_angle	-π	+π
5	hip_1_speed	-5	+5
6	knee_1_angle	-π	+π
7	knee_1_speed	-5	+5
8	leg_1_ground_contact	0	5
9	hip_2_angle	-π	+π
10	hip_2_speed	-5	+5
11	knee_2_angle	-π	+π
12	knee_2_speed	-5	+5
13	leg_2_ground_contact	0	5
14-23	10 lidar measurements	-1	+1

Number	Name	Min	Max
0	hip_1_speed	-1	+1
1	knee_1_speed	-1	+1
2	hip_2_speed	-1	+1
3	knee_2_speed	-1	+1

Figure 2: Bipedal walker action and observation space

The action space is a 4-dimensional continuous vector, and each dimension has a value between [-1, 1]. These four values control the speed of the robot's four joints (2 hips and 2 knees) respectively (Figure 2). Talking about the transition dynamics, the agent directly interacts with the environment, and it is used to train a neural-network dynamic prediction model. In turn, the dynamic prediction model is used to improve the learning efficiency of reinforcement learning. The transition dynamics are defined as a probability mapping from state-action pairs to states.
The reward is directly proportional to the distance travelled forward by the bipedal walker, with a maximum reward of 300 if it reaches the end point. A fall will cost -100 points and the episode will end. Reward is also slightly reduced based on torque applied. In short, this problem consists of teaching the agent to control the bipedal walker so that it learns to walk forward naturally, penalising stumbling, falling and overexertion.

2 Background

There are many RL algorithms, such as dynamic programming, Monte Carlo methods, and temporal-difference methods including Sarsa and Q-learning. These are also called tabular methods, since the learned value function (the value of a state and action under the policy) learned is stored in a table of size equal to all possible states by all possible actions. On top of those concepts, there is also the fact that many advancements in reinforcement learning have come from the use of "planning", which is when the agent employs a model to improve an agent's policy (temporal difference TD). However, storing information about every state-action pair separately in memory is not a viable solution when dealing with continuous state and or action domains. The reason is because it requires an unmanageable memory size, as the size of the continuous state and/or action domain would be too large to gain enough knowledge only by direct experience with the environment.
The task of the assignment, the bipedal walker, has both continuous state and action spaces, which has led the group to focus on function approximation methods. This problem calls for a solution, which is to approximate the action-value function through a parameterized function form, with weights instead of a tabular form ($\widehat{v}(s, \theta) \approx\ v_{\pi}\ (s)$) with number of weights much smaller than the number of states and were changing one weight changes the estimated value of many states. Consequently, when a single parameter is updated, the change generalizes to affect the values of many states. Employing function approximation could help RL to be applicable to partially observable problems, in which the full state is not available to the agent.
We have conducted a literature review on papers that attempt to solve this problem, a summary is provided in the appendix.
The aim of this review is to find candidates that can be implemented with reasonable effort and will yield an acceptable result in terms of rewards achieved and computational time. See the appendices for the full list of reviewed papers.
The research paper written by F. Hu (2020) was a good reference point since it compared the tabular algorithms such as Q-learning with approaches more suitable to continuous state and/or action spaces like Deep Q-Network and Twin Delayed DDPG (TD3). As per F. Hu (2020), Q-learning could not perform due to the discretization of the action space, which is necessary for a tabular method. This problem led the writer to add a deep neural network model which created a DQN algorithm. However, as per T. P. etal. (2015), while DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces.

Consequently, as stated by Lillicrap (2016), “Directly implementing Q learning with neural networks proved to be unstable in many Environments”. One possible solution to improve the performance of the algorithm could be to reduce the continuous action space by discretising it. However, on one hand a coarse discretization would probably fail since the agent would not be able to precisely select the necessary actions and a fine discretisation of the action space would lead to an explosion in the number of discrete actions and result in the curse of dimensionality (Bellman, 1957).
The methods described till this point were “*Value-Based Methods*”, where they first learn value function estimates, and then from those estimates they determine the policies (using –greedy epsilon action selection).
After many policy iteration steps, where an update of the value function then updates the policy, the value function estimates, and the policy converge to the target optimal value function and optimal policy respectively. However, there is another class of RL called “*Policy-Based Methods*", where instead of learning the value function they learn a policy function directly (rather than deriving it from the estimated value function). Consequently, when learning a policy function directly there is implicit flexibility during the exploration phase, where it could be modulated to even reach a deterministic policy. There is also another reinforcement learning approach, called “*Actor-Critric Methods*”, where the actor takes as input the state and outputs the best action. It combines ideas from the previous methods, and the actor learns the optimal policy (policy-based), and the critic, on the other hand, evaluates the state–action pair choices i.e. the policy by computing the value function (value based).

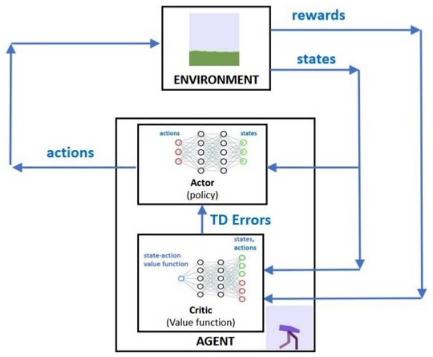


Figure 3: Reinforcement Learning Actor-Critic Method

As stated by Nicosia (2020), the current state-of-the art to build controllers for robots moving in environments with continuous action spaces utilises model-free off-policy algorithm Deep Deterministic Policy Gradient (DDPG) as well as its improved versions, twin-delayed DDPG (TD3). Similarly to Nicosia’s work, we have decided to select DDPG as the baseline method for our bipedal walker control problem. DDPG is the natural next step from DQN, since it combines Deterministic Policy Gradient (DPG) while using Q-Learning and adopting experience replay, target value and policy networks. Moreover, DDPG is an actor-critic algorithm (figure 3) with 4 different neural networks in total: the actor, the critic, target actor and target critic. In addition to the known performance of DDPG, it is also the fact that it produced the most human-like walking policy (Fussell 2018).
Looking at TD3, this algorithm is built on top of double DQN to fix the overestimation issue problem typical of DDPG. TD3 is made of 6 distinct networks, an actor, two critics, a target actor, and two target critics. Actor networks aim to approximate the policy using parameters while critic networks approximate the Q-Value function using parameters.
TD3 presents three prominent improvements on DDPG; the addition of a second critic network, reduced policy and target updates and the addition of noise to the target action.
Target actor and critic networks, involve delaying updates to the weights of these networks during training. This allows for increased learning stability, as the agent is not learning against a continually changing target. In the original TD3 paper (Fujimoto, van Hoof and Meger, 2018) these networks are updated every two time steps, and are only partially updated, containing elements of the previous state, governed by a hyper-parameter tau.

3 Method

DDPG, a model-free off-policy algorithm for solving complex continuous control problems. Since its inception in 2016, Lillicrap claims that his method resolves the walker problem after training an agent on it for 2.5 million steps. We adopted a similar architecture choice to Lillicrap's but also experimented with different improvements on underlying architecture. DDPG use an actor-critic model that combines both value-based and policy-based learnings. Regarding actor network, a current state is feed an input to the actor network which consists of 3 dense layer (400, 300, 200, 4) that have "*Relu*" (for first two layers) and "*tanh*" (for the last layer) as its activation functions. On the other hand, the critic network takes as input the current state in two MLP (400, 300, 200, 1) before concatenating them with action (the output of actor network) and output expected future reward after passing the resulted concatenation in 200 MLP. The two networks are trained using a baseline target network each to reduce the update variance hence stabilizing the learning process.
The algorithm uses experience replay and target networks to prevent instability during training. Target network parameters are updated slowly with a tau rate of 0.005, reducing the effect of sudden shifts in the learning process. The algorithm uses a replay buffer of size 100'000 to store experiences and draw 512 samples from it to decorrelate the data.
The DDPG algorithm optimizes the actor and critic networks by minimizing the difference between the predicted and actual reward using Adam optimizer with a learning rate of 0.0004, 0.0004 respectively. The actor network is optimized by maximizing the expected future reward for a given state, while the critic network is optimized by minimizing the mean-squared error between the predicted and actual reward.
In addition, several iterations of TD3 implementations were tested to identify the most performant. The baseline architecture is composed of a replay buffer of size 200,000 being used to train actor and critic networks with 400 and 300 neurons for the 1st and 2nd hidden layer, respectively. For actor models, a "*Relu*" activation function is used for intermediate layers, and "*tanh*" for the output layer, to align with action values in the inclusive range [-1, 1]. The two critic networks follow this same architecture, except for the output function where a weighted sum (linear default output) is used to calculate the expected value of a given state-action pair. An MSE loss function was used for all networks.
The agent applies randomised normal noise to each action clipped to +-0.5, to push for further exploration as suggested in the original paper. The training process starts with 50,000 exploration steps, where the environment is stepped forward randomly without updating any networks. This is to fill the replay buffer and provide a range of states to sample for training. After this, actions are decided by the model, which is updated on a randomly selected minibatch of 100 examples from the replay buffer of the 200,000 most recent experiences. One critic model is trained every timestep. Every 2 timesteps actor models are trained, and target actor and target critic models' weights are adjusted closer (using a parameter referred to as tau set at 0.005) to the respective updated actor and critic models. Actor and critic learning rates were reduced over time as per the popular Adam optimisation (Kingma and Ba, 2017), starting at 4e-4. Many of these parameters are replicated from Fujimoto, van Hoof and Meger's approach (2018). In fact, variations on learning rate, actor model update rate and tau all led to an agent that did not converge in our baseline, hence these hyperparameters were used across all agents builds. All environments were created with the same seeded value to allow for reproducibility across the starting state of all experienced episodes for each tested agent. Several variations of this TD3 baseline were also tested, each varying one aspect of the design based on related research:

- Huber Loss was used instead of MSE for all networks, reduce the impact of outliers that would update networks drastically and therefore reduce model training instability.
- A prioritised experience replay buffer was used to select experiences with high temporal difference error, instead of random selection.
- A much smaller random buffer, of size 20,000 (vs 200,000), was used.
- A baseline model was tested against the hardcore version of this game, which includes obstacles and pitfalls.

DDPG TD3 implementations follow pseudocode presented by Lillicrap et al. (2019) and Fujimoto, van Hoof and Meger, (2018), respectively.

4 Results

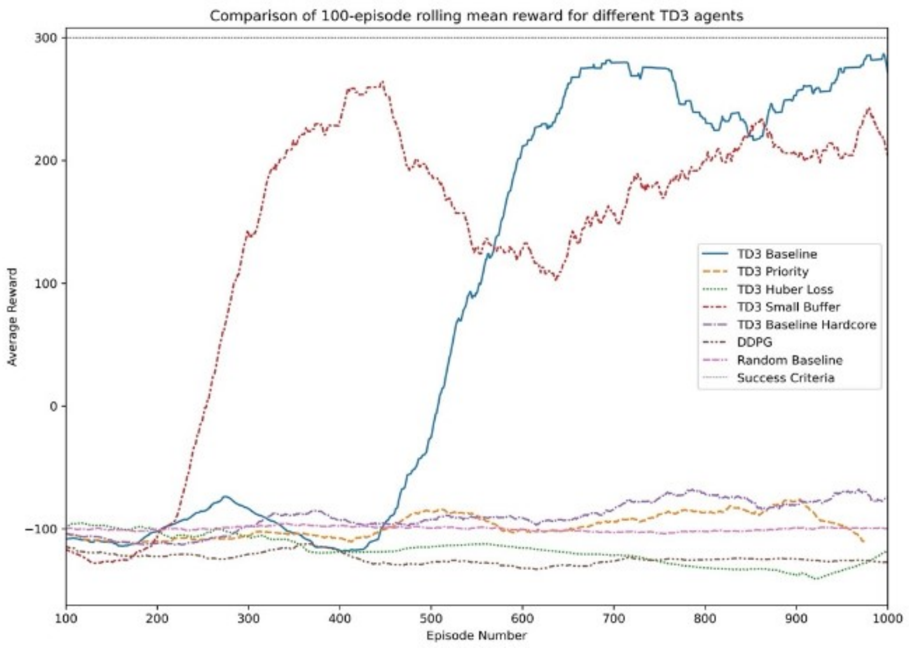


Figure 4: 100-episode rolling mean reward for agents tested, as well as the success criteria of 300.

Agent Name	**TD3 Baseline**	TD3 Priority	TD3 Huber	TD3 Small	TD3 Baseline	DDPG	Random
Highest Reward	**286**	-76	-95	264	-67	-112	-95
Episode No.	**996**	904	116	447	786	366	341

Table 1: highest reward achieved in any single episode by each agent, and which episode number this was reached at.

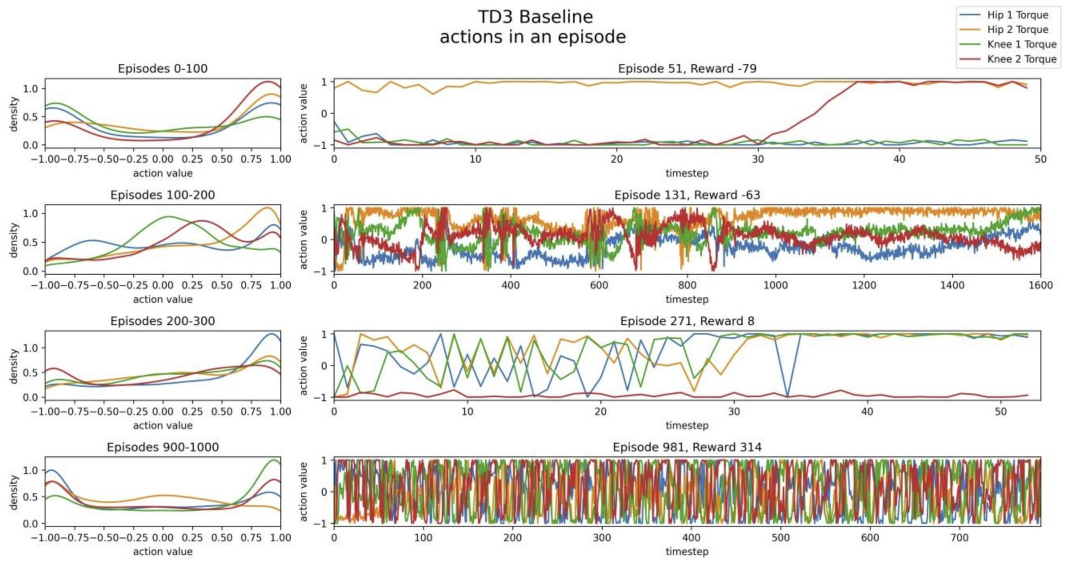


Figure 5: TD3 baseline agent, left shows the density of actions within each 100-episode window and right shows actions over time of the episode within each 100-episode window with the highest reward, full figure in appendix.

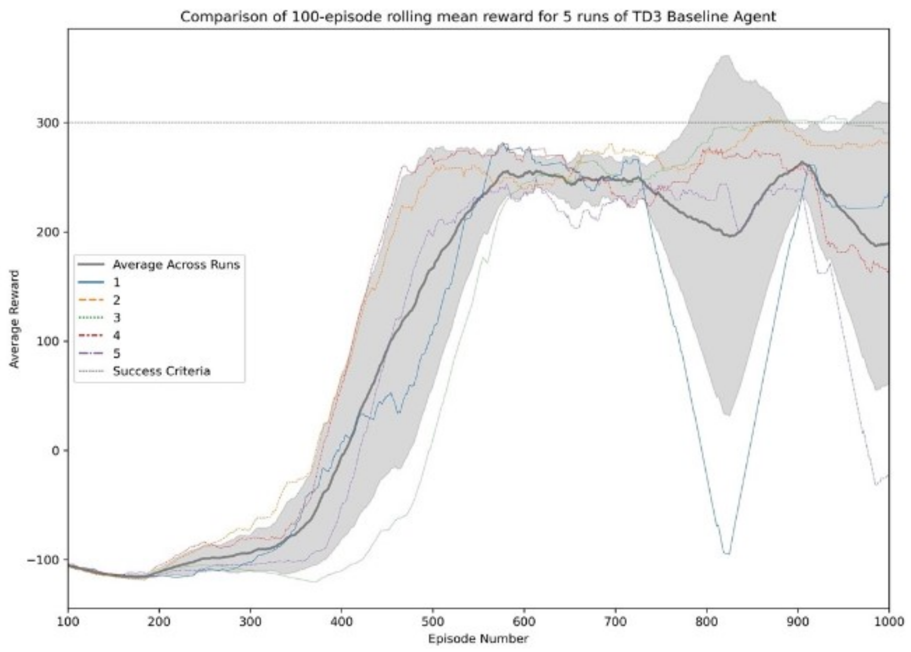


Figure 6: 100-episode rolling mean for 5 TD3 baseline runs, shaded region indicates standard deviation.

Agent	TD3 Baseline
Success Rate (*% episodes with reward >= 300*)	16.5%
Sample Efficiency (*Time taken to reach a reward of 300 or more*)	555 Episodes

Table 2: success rate and sample efficiency for TD3.

5 Discussion

Lillicrap's (2016) assessment about DDPG low sample efficiency was accurate. Our DDPG agent ran 2000+ episodes with no consistent sign of learning. Going through this task, our group underestimated the computational power this task requires and the low utility of multi-threaded computation. Different sets of hyperparameters (including Lillicrap's) and underlying architecture choice (noise, replay method) were tested however the agent reward never reached a positive value, results displayed show the best agent from these tests, which did not out-perform random actions.
Comparing across all agent types tested, several implementations did not manage to achieve scores significantly greater than the random process. This could be due to implementation error, however many of the adjustments are minor. As part of the development process, results varied run-by-run despite environment seeding, and hence it is likely that due to the stochastic nature of the agents replay buffer, action decision or a combination of these, multiple runs would be required to understand performance differences further. A small test *(Figure 4)* using 5 copies of the TD3 baseline in the same environment, saw minor differences in performance for this agent type, although one version did show symptoms of catastrophic forgetting.
The baseline and small buffer implementations showed strong performance, reaching top 100-episode average rewards of 286 and 264, respectively. The small buffer agent is the first to reach positive rewards, before dropping down back, potentially the result of losing useful experiences causing the agent to degrade. We see a later increase of performance, although it is unclear whether this is likely to continue or become hampered by the small buffer again. Here, it is likely that the small buffer removes early, random experiences and hence the agent can focus on the more recent rewarding experiences. Nevertheless, the initial performance of such a small buffer is intriguing as this is much lower than the 200k used in our main implementation, and 1 million or unlimited experiences used by Lillicrap et al. (2019) and Fujimoto (2018), respectively. The approach may be improved by reducing impact of earlier experiences, as is the case for the small buffer, this could help the agent

reach positive rewards in less episodes.
The Prioritised Replay TD3 agent did not converge at all, suggesting this led to worse experiences being selected for training than at random. The Huber Loss agent also did not converge, potentially because of reducing the impact of heavy reward/loss state-action pairs. A natural way to evaluate the walk is to use an evaluation involving the reward, such as the mean return. However, the reward function does not necessarily associate with an effective gait.
By looking at the videos of the agent supplied with this report, a qualitative analysis can be performed. In later episodes, the walker appears to have gained a walking behaviour akin to humans. Looking at the distribution plots *(Figure 5)* of the joints, suggests there may be symmetry in this motion as is the case for humans (Laroche 2012), that could be used to create an additional reward point based on action symmetry.
Overall, our baseline TD3 agent achieved promising results, which could be improved by running this agent for longer or further optimisation of the hyperparameters. Comparison with related techniques in this problem space shows several reaching the benchmark in less than 1000 episodes (Leaderboard, 2023), highlighting the art of the possible.

6 Future Work

(Elliott 2023) Suggest the implementation of the wisdom of the crowd concept for RL by implementing an ensemble method. The paper experimented with a naïve combination of Q-functions on tasks such as the bipedal walker to improve performance over the basic architecture.
This approach trains any chosen number of critics, only sharing the experience replay buffer while ensemble members can be updated independently in parallel. So that the ensemble can recommend an average action-value estimate, that is especially useful to navigate "regions of high decision space volatility" that "are the culprit behind instability". The task becomes computationally more expensive but requires significantly less interaction with environment and due to the critic updates parallelisation only a "negligible increase in wall-clock time" is reported.
While tending towards the mean is assumed to lead to suboptimal solutions the paper demonstrates that the stability in learning allows the ensemble to "build upon its knowledge" (Elliott 2023) and smoothly converge to a more efficient solutions without catastrophic forgetting. The paper proves to yield a more efficient solution with ensemble method. This solution could help smooth the variance over training TD3 and help navigate DDPG to a solution within a shorter time frame.
During the learning, it was observed that the agent randomly wiggles until it learns to use one leg and then the other. So, it is hypothesised that adjusting the reward function in a way that would promote symmetrical walking could optimise the time to converge.

7 Personal Experience

Development of a deep learning RL agent presented new difficulties compared to previous units. Utilizing Google Colab, for its collaborative workflow and easy access to GPUs for training we quickly hit the free limit on this product. Switching to developing locally raised hardware limitations for storing the replay buffer. Finally, we obtained results using massively distributed cloud computing systems requiring redeveloping the codebase. Many referenced papers highlight the long training times required for agents, which points to a technical barrier limiting development in this field.
As Hendersen suggests, we also experienced that our choice of the policy and value function greatly affect performance and it is a determining factor for learning convergence.
Collaboration was great, responsibilities were distributed to all project members and multiple meetings were held to align and decide the following steps. The challenge was to balance between individual's circumstance: allocated time to study, progress in the course material and general availability. The main difficulty was offered by this technology's leading-edge nature, namely TensorFlow. There was an unexpected learning curve to comprehend some of the ambiguous outputs and error messages of TensorFlow that lead to a significant delay in model training and iteration for DDPG.
Specifically, the ambiguity of the error *'No Gradients Provided'* (Debugging Hugging Face Course, n.d.) and processing deadlocks (parallelised computations waiting for non-existing tasks) caused issues, which in this case was driven by hardware limitations. This certainly was the first time where the architecture and hyperparameters of a learning model could raise such difficulties which are exacerbated by the computational run time of the task.

References

Francisco Hu, Eddie Xu, Syed Affan (2020). Teaching BipedalWalker to Walk with Various Reinforcement Learning Algorithms. Stony Brook University

Doo Re Song, Chuanyu Yang, Christopher McGreavy, and Zhibin Li (2018). Recurrent Deterministic Policy Gradient Method for Bipedal Locomotion on Rough Terrain Challenge. School of Informatics, University of Edinburgh, UK.

Ugurcan ozalp (2021), bipedal robot walking by reinforcement learning in partially observed environment, a thesis to school of applied mathematics of middle east technical university.

Oleg Klimov. (2022). Bipedal Walker. Retrieved on 5/04/2022 from
$https : //gymnasium.farama.org/environments/box2d/bipedal_walker/starting - state$

Stephen Dankwa, Wenfeng Zheng (August 2019).Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent. ICVISP 2019: Proceedings of the 3rd International Conference on Vision, Image and Signal ProcessingAugust 2019 Article No.: 66.

Stephen Dankwa, Wenfeng Zheng (2019).Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent. ICVISP 2019: Proceedings of the 3rd International Conference on Vision, Image and Signal ProcessingAugust 2019 Article No.: 66.

Song, D.R., Yang, C., McGreavy, C. and Li, Z., 2018. recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge. International Conference on Control, Automation, Robotics and Vision [Online]. Available from: https://doi.org/10.1109/icarcv.2018.8581309.

G. Nicosia, V Ojha, E. La Malfa, G. Jansen, V. Sciacca, P Pardalos, G. Giuffrida, R. Umeton.Siena, Italy, July 19–23, 2020.Machine Learning, Optimization, and Data Science. 6th International Conference, LOD 2020 Siena, Italy, July 19–23, 2020 Revised Selected Papers, Part II.

Levi Fussell, Taku Komura. 2018. Exploring Bipedal Walking Through Machine Learning Techniques. The University of Edinburg.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra (2016). Continuous control with deep reinforcement learning. Google Deepmind, London UK.

T. Schaul, J. Quan, I. Antonoglou, D. Silver (2016). Prioritized experience replay. Published as a conference paper at ICLR.

R. Galljamov (2020). Sample-Efficient Learning-Based Controller For Bipedal Walking In Robotic Systems.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D., 2019. Deep Reinforcement Learning that Matters [Online]. [Online]. Available from: http://arxiv.org/abs/1709.06560 [Accessed 23 April 2023].

Kingma, D.P. and Ba, J., 2017. Adam: A Method for Stochastic Optimization [Online]. [Online]. Available from: https://doi.org/10.48550/arXiv.1412.6980 [Accessed 23 April 2023]

D. L. Elliott and C. Anderson, "The Wisdom of the Crowd: Reliable Deep Reinforcement Learning Through Ensembles of Q-Functions," in IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 1, pp. 43-51, Jan. 2023, doi: 10.1109/TNNLS.2021.3089425.

Fujimoto, S., van Hoof, H. and Meger, D., 2018. Addressing Function Approximation Error in Actor-Critic Methods [Online]. [Online]. Available from: https://doi.org/10.48550/arXiv.1802.09477 [Accessed 23 April 2023].

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2019. Continuous control with deep reinforcement learning [Online]. [Online]. Available from: https://doi.org/10.48550/arXiv.1509.02971 [Accessed 23 April 2023].

Leaderboard, 2023. GitHub [Online]. Available from: https://github.com/openai/gym/wiki/Leaderboard [Accessed 23 April 2023].

BipedalWalker v2, 2020. GitHub [Online]. Available from: https://github.com/openai/gym/wiki/BipedalWalker-v2 [Accessed 23 April 2023].

Bellman, R., 1957. Dynamic Programming. Princeton University Press.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. OpenAI Gym [Online]. [Online]. Available from: https://doi.org/10.48550/arXiv.1606.01540 [Accessed 23 April 2023].

Debugging the training pipeline - Hugging Face Course [Online], n.d. Available from: https://huggingface.co/learn/nlp-course/chapter8/4 [Accessed 23 April 2023].

Laroche, D.P., Cook, S.B. and Mackala, K., 2012. Strength Asymmetry Increases Gait Asymmetry and Variability in Older Women. Medicine Science in Sports Exercise [Online], 44(11), p.2172. Available from: https://doi.org/10.1249/MSS.0b013e31825e1d31.

Appendices

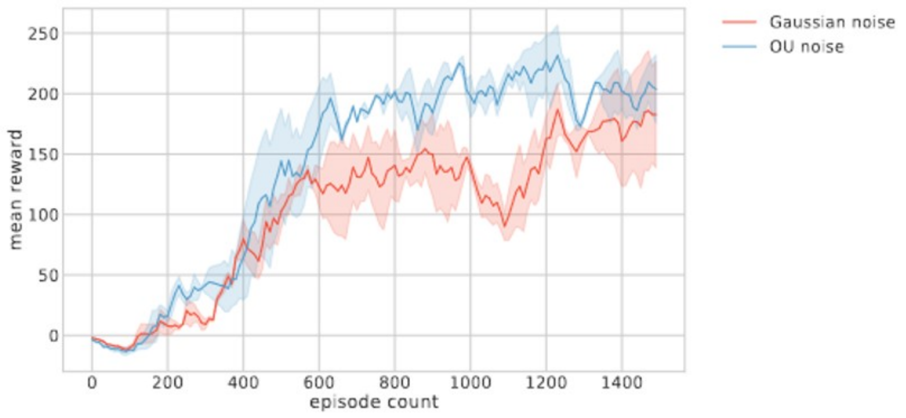
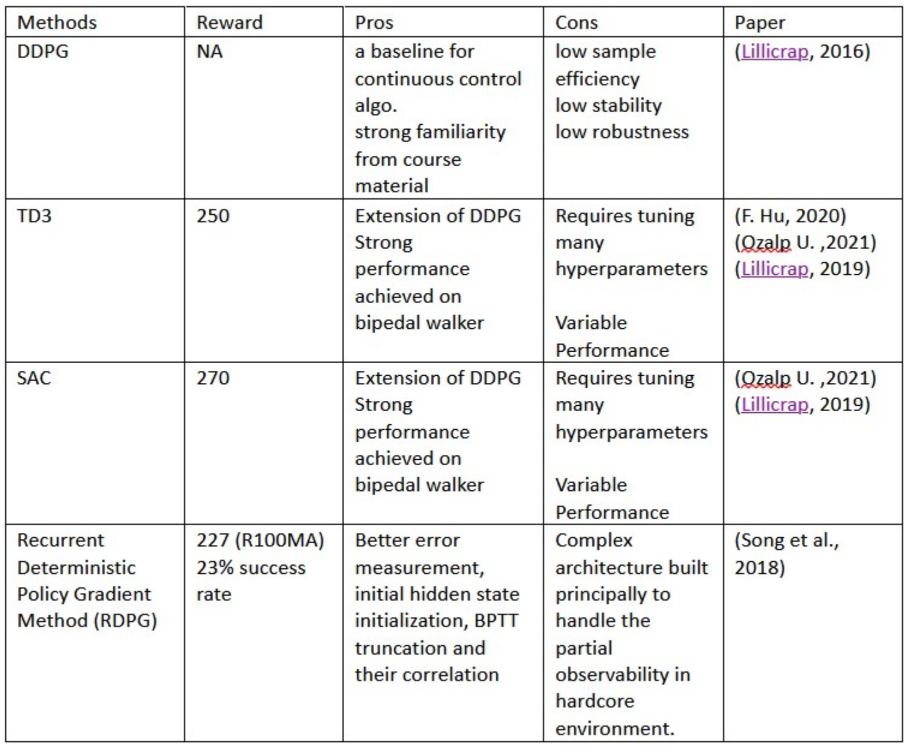


Figure 7: Results of using Ornstein-Uhlenbeck noise versus Gaussian noise. Graph from Fussell (2018) Exploring Bipedal Walking Through Machine Learning Techniques



Methods	Reward	Pros	Cons	Paper
DDPG	NA	a baseline for continuous control algo. strong familiarity from course material	low sample efficiency low stability low robustness	(Lillicrap, 2016)
TD3	250	Extension of DDPG Strong performance achieved on bipedal walker	Requires tuning many hyperparameters Variable Performance	(F. Hu, 2020) (Ozalp U. ,2021) (Lillicrap, 2019)
SAC	270	Extension of DDPG Strong performance achieved on bipedal walker	Requires tuning many hyperparameters Variable Performance	(Ozalp U. ,2021) (Lillicrap, 2019)
Recurrent Deterministic Policy Gradient Method (RDPG)	227 (R100MA) 23% success rate	Better error measurement, initial hidden state initialization, BPTT truncation and their correlation	Complex architecture built principally to handle the partial observability in hardcore environment.	(Song et al., 2018)

Table 3: full list of reviewed papers

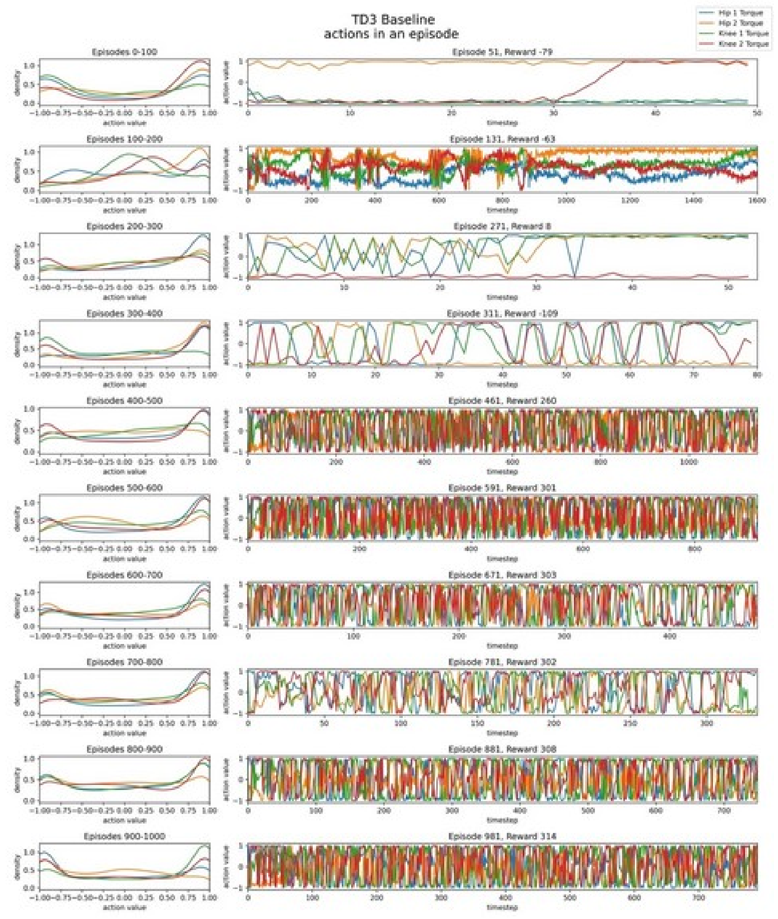


Figure 8: Full picture for TD3 baseline agent, left shows the density of actions within each 100-episode window and right shows actions over time of the episode within each 100-episode window with the highest reward