Simulating a 3D-Humanoid Walk through Reinforcement Learning and Imitation Learning

Osama Yousuf
Department of Computer Science
Dhanani School of Science and Engineering
Habib University – Karachi, Pakistan
oy02945@st.habib.edu.pk

Reeba Aslam
Department of Computer Science
Dhanani School of Science and Engineering
Habib University – Karachi, Pakistan
ra02528@st.habib.edu.pk

***Abstract* – The report includes findings of the implementation of two well-known approaches in machine learning, namely, behavioral cloning or imitation learning, and reinforcement learning, on the Humanoid-v2 environment provided by the MuJoCo-200 binding for LinuxOS, which comes with OpenAI's GYM package for Python3. The goal of the specified environment is to train the humanoid to perform a balanced walk from scratch – one that doesn't deviate from its initial straight trajectory. Thus, the report presents an analysis of the pros and cons of both the approaches on this model in terms of their overall performance and efficiency.**

Index Terms* – *Machine Learning, Artificial Intelligence, Reinforcement Learning, Imitation Learning, Behavioral Cloning.

I. Introduction

MuJoCo (abbreviation for multi-joint dynamics with contact) is a publicly available physics engine – originally developed in 2012 for simulating model-based control systems [1]. *Humanoid-v2* is one of its elaborate multi-joint systems – whose dynamics are represented in generalized coordinates and computed via recursive algorithms.

Broadly speaking, the humanoid model has two states: passive – which is the untrained state, and active – which is the trained state. In the passive state (refer to Figure 1), the humanoid behaves simply as a ragdoll, performing no actions in its environment; in the active state, it behaves as an active agent, attempting to utilize its state-space policy to perform a balanced walk in its environment. We aim to obtain and observe different versions of this policy through different algorithms and optimization techniques, mainly imitation learning or behavioral cloning (both with and without dataset aggregation) as well as reinforcement learning.

The humanoid model has a total of 25 degrees of freedom (DOFs), as summarized in Table I, as well as a total of 16 control dimensions (joints). The environment constrains the policy such that it takes in an observation in $\mathbb{R}^{376}$ and returns an action in $\mathbb{R}^{17}$. The pre-defined cost function attempts to minimize the actuation (i.e. forces on humanoid muscles) as well as horizontal torso velocity over the center of mass [2].

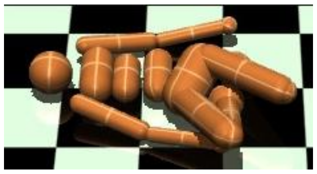


Fig. 1: Humanoid in Passive Mode

TABLE I
HUMANOID DOFS

Joint Name	**# of DOFS**
Spatial	6
Abdomen	2
Shoulders	2-2
Elbows	2-1
Hips	2-2
Knees	2-1
Ankles	2-1

II. Background

A. Reinforcement Learning

Reinforcement learning is an unsupervised machine learning technique that employs an agent to learn through trial and error in an interactive environment. The agent, given the knowledge of its current state s at time t in the environment, performs actions a which are judged and rewarded or punished by a factor r accordingly, as captured in Figure 2.

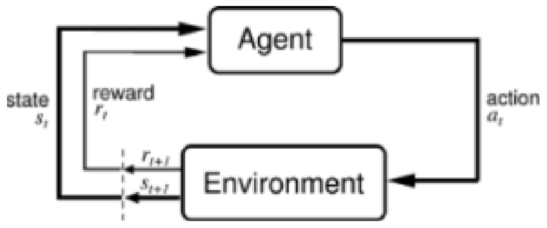


Fig. 2: Functional Flow of Reinforcement Learning.

Apart from the agent and the environment, following are the main elements of a reinforcement learning framework which are relevant to this project.

- **Policy (π):** A policy is the mapping of states of the environment to the actions that should be performed when in those states. Policies can be deterministic or stochastic.
- **Rewards (R)**: Rewards are the feedback value that the agent receives when it performs an action. Positive rewards indicate a good action while negative rewards are punishments which label the action as unfavorable.
- **Value Function (V)**: Value functions take as an input the current state and return the expected cumulative reward that could be collected over the future starting from the input state. In contrast to rewards which is an

immediate feedback of an action, value of a state can be considered as a long-term feedback.

In reinforcement learning (RL), the goal of the agent is to maximize its value function V, but the value function isn't a simple sum of all future rewards. A discount factor γ is incorporated as in reality, the probability of collecting a future reward is lesser due to uncertainties. The value function is given below:

$$V_\pi(s_t) = E\left(\sum_{k=0}^{\infty} \gamma^k + R_{t+k+1}\right)$$

An example of such a value based learning algorithm is Q-learning [3], which takes an additional input of action a_t along with the state s_t; this value function is called the Q-function and similar to the original, it also outputs the long term reward (q-value) of performing an action a_t at in state s_t. A state-action table is maintained which stores the q-value. Next time the state is encountered, the algorithm performs the action which has the maximum q-value in the table.

Several variants of Q-learning exist, but efficient performance of such algorithms is limited to simpler environments, as for environments with large action and state spaces, maintaining the Q-table becomes increasingly difficult. In order to overcome limitations of traditional reinforcement learning algorithms, reinforcement-learning is combined with neural networks. This combination, known as deep RL, has allowed researchers to mimic some human problem-solving capabilities, even in high-dimensional spaces. Some notable work includes Google's DeepMind AI and its deep Q-Learning algorithm that was able to successfully learn control policies for different Atari-2600 games [4], AlphaGo by DeepMind [5], Bipedal robot learning to walk [6], among many more.

B. *Imitation Learning*

In comparison to RL, where we maximize the rewards for our actions while focusing on the model's system dynamics to optimize our decisions, imitation learning (also called behavioral cloning) focuses simply on imitating expert demonstrations. Imitation learning is thus a supervised machine learning approach where we have a set of observation and action pairs from experts stored in the following form:

$$\{(o_i, a_i)\}_{i=1}^{N}$$

Here o_i is observation that we collect from environment (sensory outputs), and a_i is the best action that the agent can take (comes from expert). It must be noted here that in majority of its applications, the human plays the role of the expert. Similar to any other supervised learning algorithm, the goal in imitation learning is to estimate a function $f(o)$ so that given an observation o, the agent performs the best action $a' = f(o)$, so as to minimize the loss (cost) as follows:

$$\min_i \sum \left(dist(a', a)\right)^2$$

In our implementation, this expression translates to the Euclidean mean-squared difference between the two action vectors a and a', both of which are in $\mathbb{R}^{\mathbf{17}}$.

The underlying terminologies in imitation learning overlap with the same as presented in reinforcement learning, but the biggest comparative trade-off and the biggest challenge in imitation learning is collecting expert demonstrations. Unless it has a huge business potential, the attached cost can be prohibitive. The second major issue in imitation learning is in terms of imitation itself; an agent can never duplicate observation-based actions exactly due to the possibility of non-markovian and/or multi-modal behavior, and in a continuous environment such as the Humanoid-v2, error can accumulate rapidly. This issue is known as distributional drift, and it introduces the possibility of encountering a state that the expert has never previously seen, which in turn brings the risk of the agent having no choice but to take a bad action. Figure 3 illustrates this concept aptly.

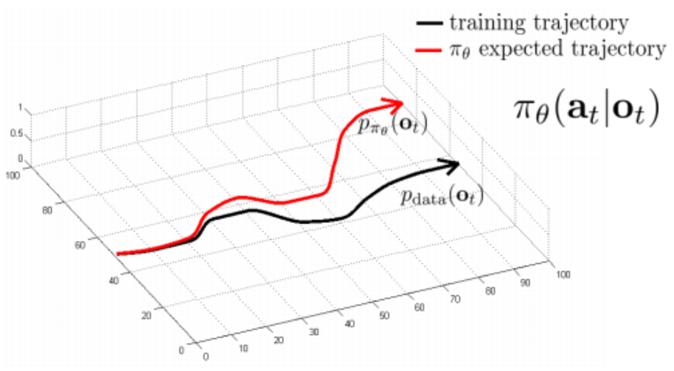


Fig. 3: Visualization of Imitation Drift-off

Here the black line indicates the expert trajectory, while the red-line indicates the imitated trajectory based on the trained policy distribution p_{π_θ}. It can be seen that small drift-offs can add up and lead to considerably different expected results. In practical applications, establishing corrective counter-actions to these drift-offs is almost always necessary. In this project, we have implemented the well-known dataset aggregation (DAgger) technique to counter this issue (explained in the next section).

Regardless of its drawbacks, imitation learning has led to ground-breaking results in the field of machine learning. With the great success of supervised learning, it is comparatively easier to train a policy based on collected expert demonstrations. Because of this fact, imitation learning has better stability than many RL methods. Most notable mentions in imitation learning include NVIDIA's self-driving car [7], the psychological study of human altruistic helping [8], and Finn's work on inverse reinforcement learning [9].

III. Implementation

A. *Reinforcement: Deep Deterministic Policy Gradient*

The deep RL algorithm we implemented in the project is Deep Deterministic Policy Gradient (DDPG) [10]. It applies an actor critic approach to solve continuous action space problems.

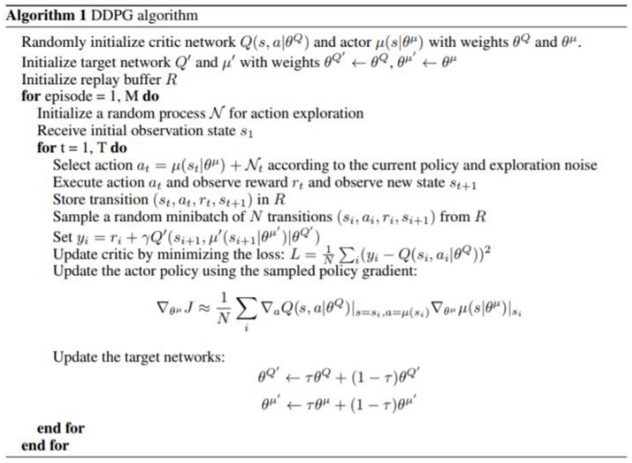
In DDPG, the actor is a neural network with the job to learn the deterministic policy function, while the critic is a neural network which, in contrast to Q-learning where a table of Q-values is stored, approximates the Q-function. The network takes a state s as an input and produces Q-value for all possible actions.

Neural networks in traditional machine learning algorithms require a data set. Here, this data set is acquired through the means of a replay buffer. A replay buffer is a queue of transition tuples $(s_t, a_t, r_t, s_{t+1}, d)$, where d indicates whether state s_{t+1} is terminal. Periodically, a sample from the replay buffer is drawn to train the network. A mean squared bellman error (MSBE) function is used to find out how close the neural network is to satisfying the Q-function.

An issue with value-based functions when dealing with continuous action spaces is calculating the maximum Q-value at every time step. DDPG solves this problem by assuming the Q-function to be differentiable with respect to the action argument as the action space is continuous. This allows us to set up an efficient, gradient-based learning rule for a policy $\mu(s)$ which exploits this fact, as given below:

$$\max_{\theta} E_{s\sim D}\left[Q\varphi(s, \mu_\theta(s))\right]$$

Figure 4 summarizes the algorithm pseudocode, as presented in the original paper. The author suggests to introduce noise into the actions while training, so as to increase the overall policy exploration.



```
Algorithm 1 DDPG algorithm
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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
```

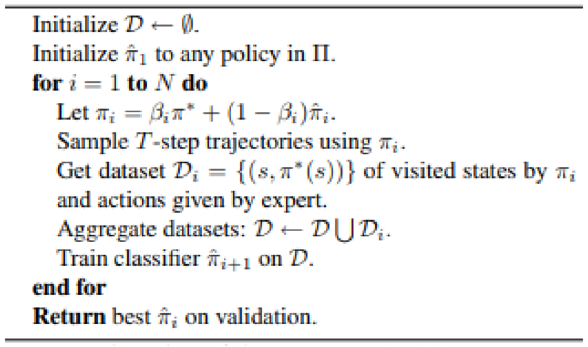
Fig. 4: Pseudocode for DDPG

B. *Imitation: Dataset Aggregation*

The imitation learning algorithm that we implemented for the project is Dataset Aggregation (DAgger) [11]. In this case, our agent policy is a simple fully-connected 3-layer deep neural network, which according to the humanoid environment restrictions, takes an observation o_t in $\mathbb{R}^{376}$ as its input and produces an action a_t in $\mathbb{R}^{17}$ based on its imitation policy. Here, instead of a human expert, we used an expert humanoid policy which was pre-trained to walk efficiently.

As previously mentioned, the DAgger algorithm addresses the distributional drift problem. Referring to Figure 3, it attempts to set $p_{\pi_\theta} = p_{data}$ by the following iterated approach. At the first iteration, it uses the expert's policy to gather a dataset of trajectories D and trains a policy π that best mimics the expert on those trajectories. Then at iteration i, it uses π_i to collect more trajectories and adds those trajectories to the dataset D, forming an aggregated dataset D'. The next policy π_{i+1} is the policy that best mimics the expert on the whole dataset D'.

Figure 5 summarizes the algorithmic pseudocode. In can be seen that DAgger proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets.



```
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
  Let $\pi_i = \beta_i\pi^* + (1-\beta_i)\hat{\pi}_i$.
  Sample $T$-step trajectories using $\pi_i$.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$
  and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.
```

Fig. 5: Pseudocode for DAgger

IV. Results

A. *Reinforcement Learning*

According to our research and pre-existing work, a typical reinforcement learning algorithm takes approximately 100,000 epochs of training for having an efficiently walking humanoid. Due to a severe limitation and lack of computation power, resources, and time, the DDPG actor-critic model could only be trained for a total of 7,000 epochs. The results were, therefore, not appealing – but they do unveil a trend upon inspection of their generated rewards.

Figure 6 depicts the average and best rewards obtained against intervals of 1,000 training epochs for analyzing the performance of the trained model throughout the training phase. Even though the humanoid couldn't properly walk after 7,000 epochs, the figure shows an overall positive trend in the generated reward, thus hinting to the fact that perhaps the humanoid would've started walking, had it been trained for a larger number of epochs, as the award can be seen to increase with the increase in training epochs.

Figure 7 displays the performance of the trained humanoid against 25 different rollouts. The graph shows that the reward rate is not steady and keeps fluctuating with each rollout; this shows that the trained model is not consistent and requires further training for improvement.

In comparison to Figure 8, it can be seen that our trained reinforcement learning model, even after 7.000 epochs, performs worse than our trained regular policy imitation learning model, which is the worst in its competition. There is however a rollout case where both perform equally bad in terms of their reward, but this can be accounted to the scenario where the humanoid tips into its passive mode.

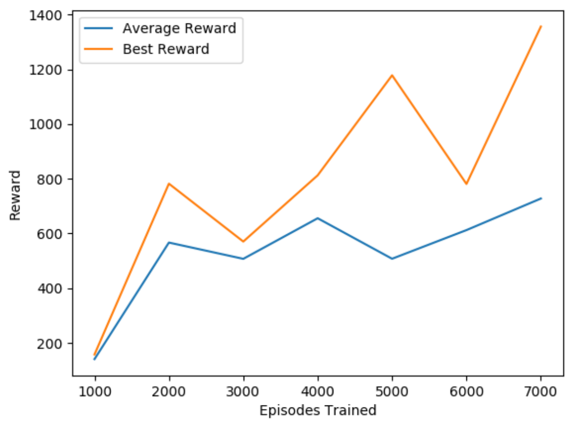


Fig. 6: Reward against trained episode/epoch no.

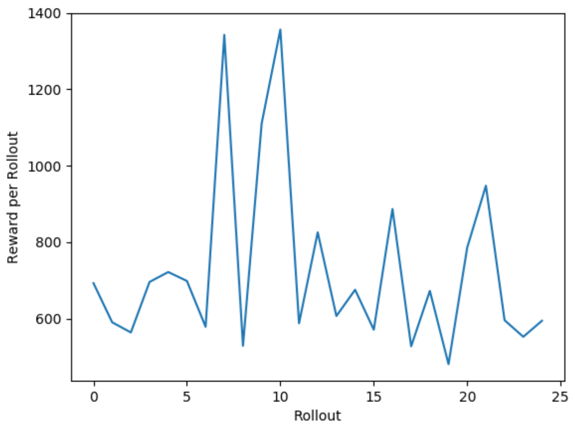


Fig. 7: Reward against trained episode/epoch no.

Due to the incompleteness in the training phase, we are here forced to conclude that the continuous and high-dimensional DOFs of the humanoid environment require a higher degree of computation power and time; only then can the performance of DDPG on it be properly analyzed and studied.

B. *Imitation Learning*

Policy parameters (i.e. network parameters) were manually chosen to reflect a balanced convergence rate, as summarized in Table II. The optimizer used for adjusting policy weights and biases post the network feed-forward operation is Adam [12], originally proposed as an adaptive learning rate optimization algorithm having several advantages over stochastic gradient descent (SGD) algorithms in terms of training time.

TABLE II
POLICY PARAMETERS

Hidden network layers	3
Neurons in hidden layers	100
Regular Epochs	400
DAgger Epochs	400
No. of rollouts	25
Initial learning rate	1e-24

To observe the enhancement of DAgger, imitation learning was implemented twice; first without dataset aggregation and second with it. In both cases, the resulting humanoid walk was quite efficient and managed to mimic the expert in a comparatively low amount of time. However, it was noted that without DAgger, even though the humanoid managed to complete several rollouts, it lacked consistency. There were often rollouts in which an initial tumble in the humanoid's spawn point led it to enter a passive mode altogether. There were also cases where the humanoid walked in a path which was significantly deviated from what the expert demonstrations showed. Referring back to the background, both these cases can be classified as issues arising due to distributional drift. In comparison, the issue of consistency did not arise in case of the humanoid trained via DAgger, and it performed almost as good as the expert in all cases.

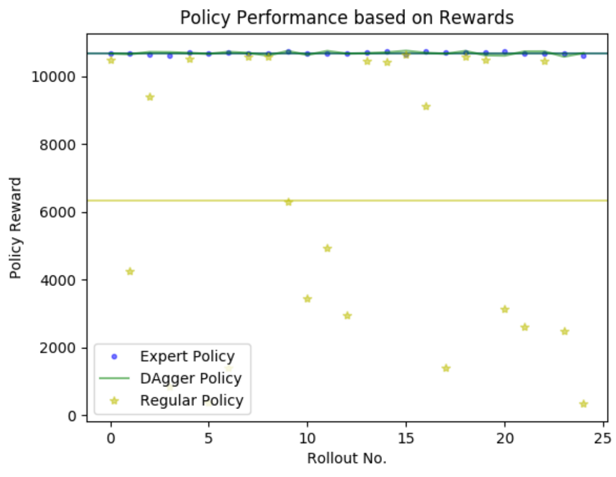


Fig. 8: Imitation Learning – Reward in different rollouts

Figure 8 shows that the regular imitation policy (yellow) i.e. the policy without dataset aggregation obtained the least rewards on average (straight line), while at the same time maintained the highest standard deviation from its mean (scatter points). There are a number of cases in which regular rollouts perform comparably and as well as the expert demonstrations (blue). The high standard deviation points to the inconsistency of the trained regular policy, as pointed out above.

Figure 9 displays a zoomed-in picture of the graph in Figure 8, so as to compare the performance of the DAgger policy against the expert. Firstly, it is evident that only 4 out of 25 rollouts of the regular policy perform on a scale comparable to the expert and the DAgger policies. This means that almost 84% of the time, the regular policy failed to keep up with the expert demonstrations and terminated due to distribution drift. Secondly, it can also be seen that on average, the DAgger policy differs from the expert by only a small amount in terms of performance, so much so that it can be argued to be performing overall as good as the expert. Table III summarizes these results concisely (taken over 25 repeated rollouts).

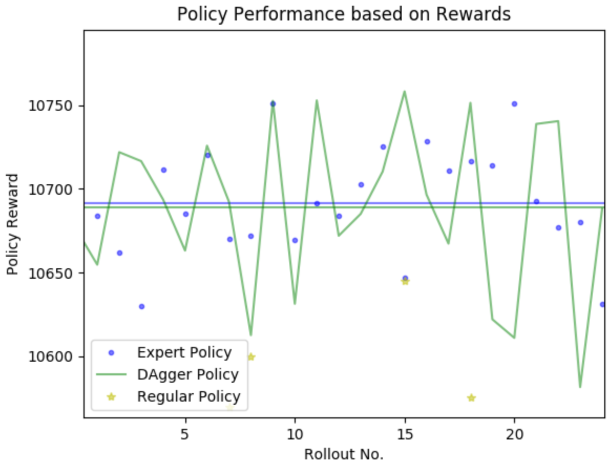


Fig. 9: Imitation Learning – Zoomed-in view of Fig. 8

TABLE III
POLICY PERFORMANCE – MEAN AND STANDARD DEVIATION

Policy Reward					
Regular		**DAgger**		**Expert**	
Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
6334.41	4037.50	10688.71	48.91	10691.69	31.45

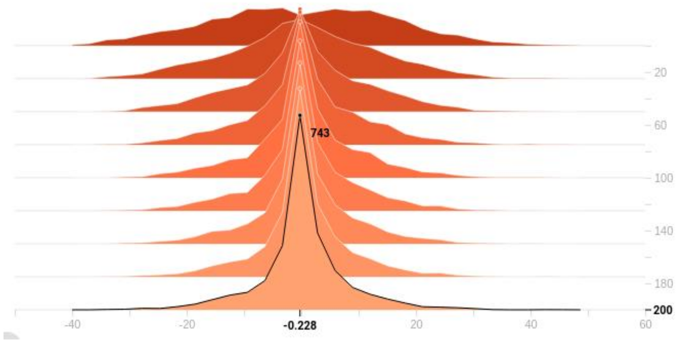


Fig. 10: Layer 1 Pre-Activation Scores – Regular Policy

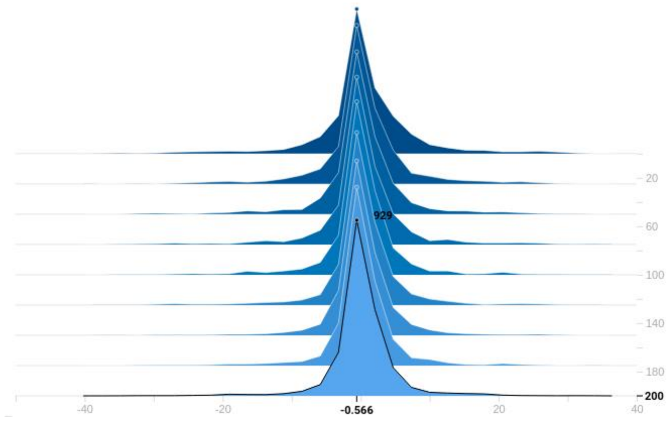


Fig. 11: Layer 1 Pre-Activation Scores – DAgger Policy

Due to low computation power, the training was carried out in two portions of 200 epochs each. In terms of time, the DAgger policy took much longer to train over the same number of epochs in comparison to the regular policy. This difference can be attributed to the extra step of generating an aggregated dataset at each training epoch. This trade-off in time however produced significantly better results and managed to address the issue of distributional drift, as previously described. This boost in performance can be further visualized through the histograms in Figures 10 & 11.

Figures 10 & 11 depict the pre-activation score during the training phase for the last 200 epochs slice of the network. The orange histogram is for the regular policy, while the blue histogram is for the DAgger policy. It can be observed that not only does the DAgger policy produce a higher score at the final epoch (929 against 743), it outperforms the regular policy consistently at all epoch stages, especially in the early ones.

Having put forward this analysis, we can thus conclude that at the expense of longer training time, dataset aggregation performs as good as the expert policy, and does so consistently and also addresses the distributional mismatch problem, described earlier, with high precision and accuracy.

C. Comparison

It would be unfair to draw a generalized conclusion here between the two approaches based on how they performed, because of the very fact that we lacked computation power. Clearly, imitation learning proved to be much more efficient than reinforcement learning under our restrictions and constraints, but both methods have their pros and cons.

The problem in imitation learning is that we need to provide expert data, which typically comes from a human, thus there's always a limit of human finiteness. Another problem is that the trained model can only be as good as the expert at best. In comparison, reinforcement learning isn't bounded by any of these issues, so it has an edge above imitation learning in this regard.

The obvious drawback is in terms of training time. As we experienced in this project, if we do have expert demonstrations in good quantities, imitation learning can achieve better results than reinforcement learning under computation restrictions in significantly lesser amounts of time.

V. Conclusion

We implemented two different algorithms for simulating a 3D-humanoid walk; the first was based on reinforcement learning, while the other was based on imitation learning. The latter approach outperformed the former in terms of rollout reward as well as training time since we had access to expert demonstrations. Furthermore, we also implemented an optimized version of imitation learning through dataset aggregation, which led to the best results – almost as good as the expert which we knew to be our limit beforehand. The final verdict on the two algorithms is that the better approach really depends on the problem description, problem model and application, and also on the availability of resources. It might also be possible to re-produce better results with a hybrid approach – one that encapsulates demonstrations as well as direct rewards. However, for purposes of this report, we will mark this as a future trajectory for this project and conclude here.

Acknowledgment

We would like to acknowledge University of California Berkeley for their publicly available course-work for CS 294: Deep Reinforcement Learning.

References

[1] Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. doi:10.1109/iros.2012.6386109

[2] Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. doi:10.1109/iros.2012.6386025

[3] C. J. C. H. Watkins and P. Dayan, Q-learning, Machine Learning, vol.8, no. 3-4, pp. 279-292, 1992.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G.Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S.Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D.Wierstra, S. Legg, and D. Hassabis, Human-level control through deep reinforcement learning, Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D.Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanc-tot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever,T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[6] C. Liu, A. Lonsberry, M. Nandor, M. Audu, A. Lonsberry, and R. Quinn, Implementation of Deep Deterministic Policy Gradients for Controlling Dynamic Bipedal Walking, Biomimetics, vol. 4, no. 1, p. 28, 2019.

[7] Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, and Zhang X. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316. 2016.

[8] Warneken F, and Tomasello M. Altruistic helping in human infants and young chimpanzees. Science. 2006.

[9] Finn C, Christiano P, and Abbeel P, Levine S. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. arXiv preprint arXiv:1611.03852. 2016.

[10] C. Liu, A. Lonsberry, M. Nandor, M. Audu, A. Lonsberry, and R. Quinn, Implementation of Deep Deterministic Policy Gradients for Controlling Dynamic Bipedal Walking, Biomimetics, vol. 4, no. 1, p. 28, 2019.

[11] Ross S, Gordon G, and Bagnell D. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 627-635, 2011.

[12] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. arXiv:1412.6980v9. 2014.