MODELING HUMAN GAIT WITH IMITATION LEARNING

Justin Brantley
CIS522: Deep Learning
University of Pennsylvania

Aydin Imranov
CIS522: Deep Learning
University of Pennsylvania

Felipe Parodi
CIS522: Deep Learning
University of Pennsylvania

1 Introduction

In reinforcement learning (RL), an agent learns by trial and error to achieve an objective based on the rewards and state in a given environment [1]. In other words, the reward reinforces the desired behavior to "teach" an agent (also referred to as "policy" or "model" or "AI") to reach a particular goal. Imitation learning (IL) is a powerful and practical alternative to reinforcement learning for learning sequential decision-making policies from a supervisor [2]. While an RL agent learns from a responsive environment, an IL agent learns from data, or demonstrations. Imitation learning has benefited from recent progress in core learning techniques, increased accessibility to GPUs, and fidelity of demonstration data. In general, imitation learning is useful when it is easier for an expert to demonstrate the desired behavior to an agent, rather than relying on an agent to directly learn the policy.

Although existing RL algorithms can learn a "walking" policy well, the produced gaits express unnatural movements. It also requires significant compute and training time to fully train these agents to succeed in such environments. Behavioral cloning (BC), a form of imitation learning, can be leveraged to train an agent by learning directly from human "expert" demonstrations [3]. By giving an agent supervised data, the agent can better learn to reproduce the skilled behavior. Behavior cloning can also be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. Here, we applied behavioral cloning to teach a Humanoid agent how to walk with ecological validity. We explore whether an agent, given expert data, can learn to walk better and more human-like than RL algorithms trained from scratch.

Our project builds upon two seminal works: DeepMind's Deep Q-Learning from Demonstrations (DQfD) and DeepMimic. In DeepMimic, the authors show that existing RL algorithms can be adapted to learn robust control policies capable of imitating a broad range of example motion clips [4]. By combining a motion-imitation objective with a task objective, they were able to train characters that react intelligently in interactive settings. This approach thus combines the convenience and motion quality of using motion clips to define the desired style and appearance, with the flexibility and generalizability afforded by RL methods and physics-based animation. In DQfD, the authors demonstrate that the agent's learning process can benefit from demonstration data, while boosting its generalizability to real-world tasks [5, 6, 7]. We aim to leverage advances in imitation learning and motion imitation to demonstrate that i) using imitation learning can significantly reduce an agent's training time, and, by extension, compute; and ii) supervising an agent with human "expert" demonstrations will yield movements more naturalistic than RL algorithms trained from scratch.

2 Methods

2.1 Implementation

Environment The goal of this project was to train a humanoid to learn to walk using expert data from humans. While developing the implementations, we quickly realized that we faced several technical challenges related to the humanoid environment. The first option for a humanoid agent was the popular *Humanoid* environment in the OpenAI Gym [8]. However, this implementation requires a license to the paid physics engine MuJoCo (www.mujoco.org) [9]. Although a free trial is available, this limits the ability for study replication and future improvement in the absence of the paid software. Thus, we chose to use the humanoid environments available through the open-source physics engine *PyBullet* [10]. Despite our efforts to perform all of the analyses with a single environment, we were ultimately forced to use two humanoid models with slightly varying action and state space dimension (although both inherit from bipedal locomotor

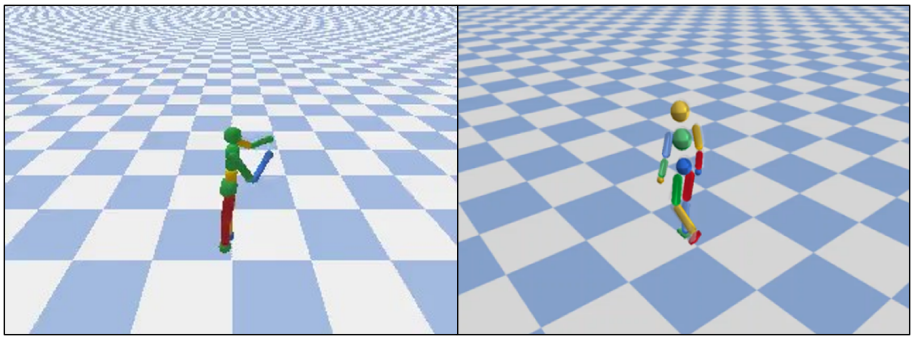


Figure 1: Two versions of the PyBullet Humanoid: HumanoidBulletEnv-v0 (left) and PyBulletDeepMimicEnv (right).

robot classes and thus have similar or identical low-level action and state spaces). This is due to the fact that the one used for training the baseline RL algorithms (Sections 2.3 and 2.4) can be readily built using the standard OpenAI gym environment framework (`env = gym.make("humanoid")`). The humanoid used for generating expert data is intertwined with the motion capture data at multiple levels within the environment class inheritance tree. Thus, the standard gym framework could be as easily applied to this humanoid implementation. Due to time limitations we were unable to reconcile the differences and forced to use two environments.

HumanoidBulletEnv-v0 This humanoid has a state space of 44 corresponding to rotations at primary joints within the body (e.g., hip, knee, ankle, shoulders). The agent has an action space of 17, corresponding to torques at joints in trunk and lower body. Incomplete documentation of the state and action space are provided here. This environment was used for learning from scratch, as in Sections 2.3 and 2.4.

PyBulletDeepMimicEnv The environment has a state space of 197 and an action space of 36. The reason for the significantly expanded dimensions is that the full body is represented in the state and action vectors (e.g., 4-dimensional quaternion representation for joint rotations, both state position and state velocity). This is because the expert and agent action and state domains must be aligned, or a projection must exist to map the expert domain to the agent domain in an imitation learning context. In this humanoid environment, expert data are collected using motion capture software, which provide physical data, such as position, velocity, acceleration, and sometimes reaction forces. Thus, these physical data must be translated into the agent domain and then converted to the action space. In the expert data, the only known information are the states of the joints. The action, or force, must be estimated using the transition between states at times t and $t-1$. The domain translation for this humanoid is approximately as follows:

Data: Motion capture (mocap) data during walking
for *each time step, t, in Data* **do**
    get pose from mocap data at time t;
    get pose from mocap data at time $t+1$;
    quaternion spherical line interpolation (slerp) for $t \rightarrow t+1$;
    compute joint torques using proportional-derivative (PD) controller;
    apply joint torques;
end

Initially, this approach was implemented for generating expert data. The motion capture data was used to compute joint torques for each state transition. However, no correction was done at each step thus resulting in error propagation and overall joint torque decay. The resulting PD controller was insufficient to adequately mimic the observed human gait alone. Thus, expert data were generated using the the pre-trained DeepMimic model available through PyBullet [10] (available here). This advanced implementation utilizes proximal policy optimization [11] for policy estimation combined with a reward function that evaluates the humanoid pose and replication of the reference pose to generate robust human-like behavior. Since the reference data (mocap) and humanoid behavior are intimately intertwined at the lowest levels of the robot architecture, we opted to use this pre-trained network to generate the $(state,\ action)$ pairs required for behavioral cloning. This process was as follows (see next page):

Data: Motion capture (mocap) data during walking
initialize humanoid; **for** *each time step, t, in Data* **do**
  get humanoid state at time t;
  compute action from trained policy;
  get reference pose from mocap data at time t , $t+1$;
  get path: quaternion spherical line interpolation (slerp) for $t \rightarrow t+1$;
  compute joint torques using proportional-derivative (PD) controller;
  apply joint torques;
  compute pose-based reward: joint position, joint velocity, end effector position, center of mass;
  save {state(t) , action(t), state(t+1)} ;
end

The result of this process was a set of $(state,\ action)$ pairs that could be used for training a policy network for behavioral cloning.

Dataset We used walking data from the CMU Graphics Lab Motion Capture Database (available from `http://mocap.cs.cmu.edu/`) for expert demonstrations.

2.2 Non-Deep Baseline: Ridge Regression

Model Priors As a non-deep learning baseline, we implemented Ridge Regression [12]. Ridge Regression extends ordinary linear regression by adding a regularization penalty to the loss function to provide more robust results since the dimensionality of the state and action vectors were both high. Although a regression model may be naïve, it is a useful baseline because it makes the simple assumption that the state and action data are linearly related, and thus a fixed weight vector will be sufficient to map a state to an action.

Training details The generated $(state,\ action)$ pairs were divided into training and test data (15%; `sklearn.model_selection.train_test_split`) and regressed using `sklearn.linear_model.Ridge` in Python ($\lambda = 1$).

2.3 Deep Learning Baseline: Soft Actor-Critic (SAC)

Model Priors To establish a deep learning baseline, we used Soft Actor-Critic (SAC) [13], which is an off-policy algorithm which follows the Soft Q-Learning SQL and incorporates the double Q-learning trick from Twin Delayed DDPG (TD3) (discussed in the following section). In particular, we used the SAC implementation from Stable Baselines, a "set of improved implementations of RL algorithms based on OpenAI Baselines." [8] Notably, we chose SAC because it is trained to maximize the trade off between expected return and entropy in an off-policy way [reference spinning up blog post]. Stable Baselines was chosen based on its simplicity of use and consistency. For example, training the SAC agent can be done is less than 25 lines of code. Finally, this implementation of Soft Actor-Critic through Stable Baselines, originally derived from the implementation of OpenAI Spinning Up, is particularly suited for environments with continuous action spaces, like PyBullet Humanoid-v0. The SAC *MlpPolicy* uses a 2-layer multilayer perceptron (MLP) of 64 neurons. We discuss the notable similarities and differences of SAC and TD3 in the next section.

Training details The Soft Actor-Critic agent was trained and evaluated in Google Colaboratory [14]. Tuned hyperparameters:

- Batch Size = 256
- Buffer Size = 1000000
- Gamma = 0.90
- Entropy Coefficient = 0.01
- Learning Rate = 0.0003
- Tau = 0.005

It is important to note that it is typically required to train this agent for approximately 20 million iterations. Due to compute and time constraints, we could only train each agent for 3.5 million iterations.

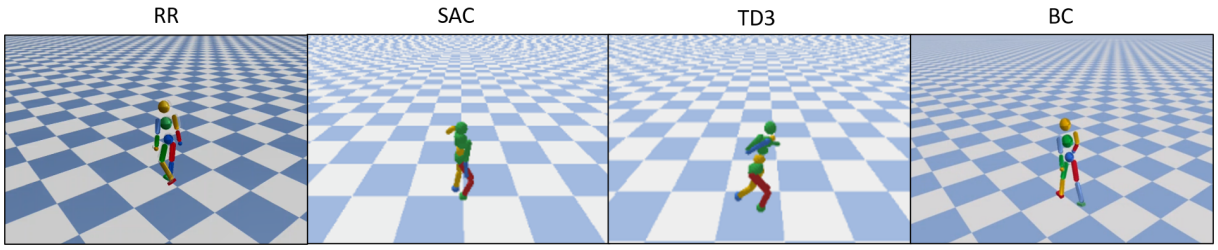


Figure 2: Each model developed a unique gait during the learning process.

2.4 Advanced DRL Approach: Twin Delayed DDPG (TD3)

Model Priors To test an advanced deep reinforcement learning (DRL) algorithm, we implemented Twin Delayed DDPG (TD3), an off-policy algorithm that builds upon DDPG by introducing three notable "tricks." [15] First, TD3 uses two critic networks by learning two separate Q-functions, which then uses clipped double Q learning to take the smaller of the two Q-values to form the targets in its loss functions. Second, TD3 uses a delayed update of the policy, only updating it every 2 time steps, resulting in more stable and efficient training. Finally, TD3 uses a regularization technique known as target policy smoothing by adding a small amount of random noise to the target and averaging over mini-batches. By doing so, TD3 makes it harder for the policy to exploit Q-function errors, placing a preference on actions that are more robust.

Although these "tricks" explain why TD3 was chosen over DDPG, what makes it different from SAC? Both SAC and TD3 Q-functions regress to a single target. This shared target is computed using target Q-networks, whose parameters are averaged throughout training. In addition, both the shared target of SAC and TD3 makes use of the clipped double Q "trick". However, in TD3, the next-state actions used in the target come from the target policy, rather than the current policy. We also found it preferable to train a deterministic policy with TD3 and then add noise to the next-state actions for target policy smoothing, rather than relying on the inherent noise of a stochastic policy. Like for SAC, we used a 2-layed MLP with 64 units, only because no other policy was available. We used the Stable Baselines implementation of TD3 for the same reasons listed in the **SAC Model Architecture** section.

Training details The Twin Delayed DDPG agent was trained and evaluated in Google Colaboratory. Tuned hyperparameters:

- Batch Size = 100
- Buffer Size = 200000
- Gamma = 0.98
- Noise = Normal, 0.1
- Learning Rate = 0.001

Like SAC, this agent is typically trained for 20 million iterations. Once again, given compute and time constraints, we could only train each agent for 3.5 million iterations.

Code to train and visualize the SAC and TD3 agents can be found here.

2.5 Behavioral Cloning

Model Priors Behavioral cloning is a supervised learning approach to decision making in which we directly imitate the actions of an expert. In this project, we utilize the generated $(state,\ action)$ data from humans (expert bipedal ambulators) to train an agent to directly imitate the expert behavior [3]. While this approach is known to have issues in practice (such as mismatch between the training trajectory and the policy trajectory), it is an interesting foundation in imitation learning. In our model, we assume that a deep neural network can be used to learn a policy for selecting actions given a current state.

Training details We formulate the observed $(state,\ action)$ pairs into a replay buffer for sampling during training. We use a neural network with two hidden layers and rectified linear units for activation to learn a policy for obtaining an action given a state. We used mean-squared error loss and Adam optimization for gradient descent.

- 2 hidden layers with 128 units
- Batch Size = 10
- Epochs = not limited
- Number of iterations = 200
- Learning Rate = 0.0003

The training procedure for the BC agent was as follows:

Data: Replay buffer containing $(state,\ action)$ pairs
initialize humanoid;
for *each epoch* **do**
  initialize loss $\leftarrow 0$ **for** *number of iterations* **do**
    sample buffer $\leftarrow (state,\ action)$;
    get estimated action from state: forward pass through network;
    compute loss $\leftarrow$ MSELoss(action, estimated action);
    back-propagate error and step optimizer;
    add loss to total;
  end
  get humanoid initial state;
  initialize reward $\leftarrow 0$ **while** *not done* **do**
    get action from policy given state;
    compute humanoid state from action;
    add reward to total;
  end
  reset humanoid;
end

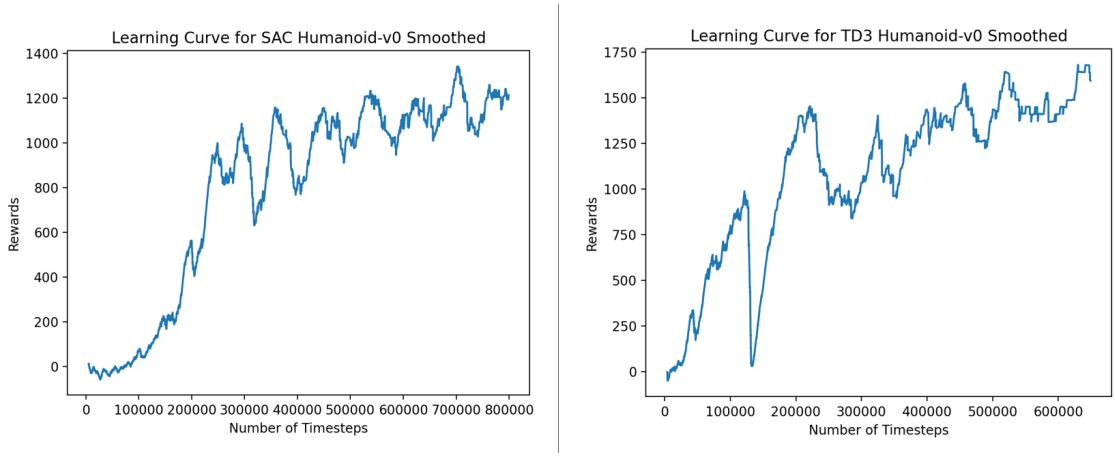
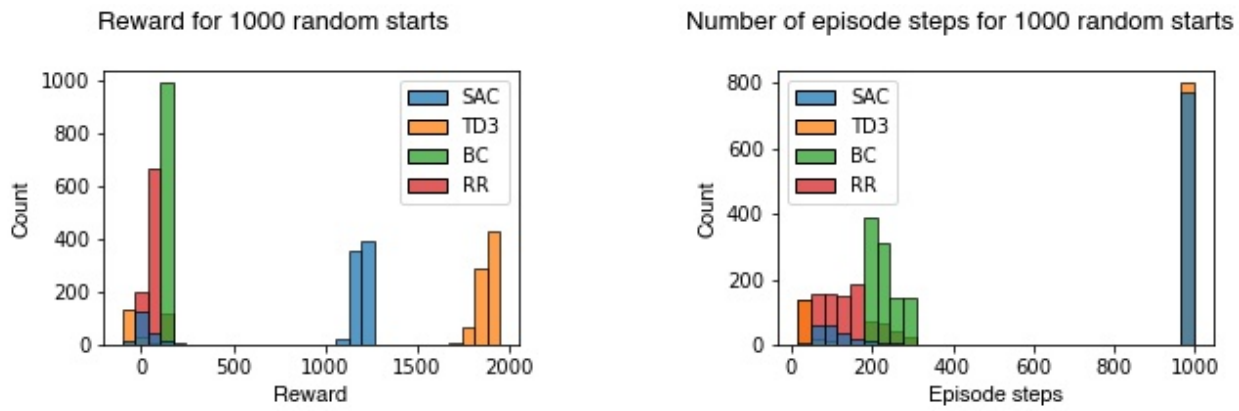


Figure 3: Rewards earned in a sample training session of SAC and TD3

3 Analysis

Ridge Regression The resulting cross-validated model had a mean-squared error of 0.002 and an R^2 value of 0.95, indicating that the action data had significant predictive power for decoding the state. However, the humanoid was never able to achieve a complete step due to an accumulation of error that resulted in it falling to the ground.

Soft Actor-Critic and Twin Delayed DDPG Using 3 different Gmail accounts, the agent was trained for 600,000 iterations at a time for a total of 3.5 million iterations. It is worth reiterating that these agents are normally trained for 20 million iterations, over 5 times as much as we have trained. We were not able to train for longer given compute constraints as Google Colab logged us off the GPUs after a handful of hours. Regardless, both the SAC and TD3 agents



(a) Distribution of the total reward per episode.

(b) Distribution of the total number of episode steps.

Figure 4: The trained agents were initialized 1000 times. Note that SAC and TD3 were upper bound at 1000 episodes steps.

are able to walk through the environment. See figure 3 for a comparison of rewards across training for SAC and TD3. The TD3 agent accumulated more rewards than SAC, further supporting our choice to consider it the "advanced" model. Regardless, both agents required significant resources as we approximate it took nearly 35 hours of time on Colab GPU support for the agents to "walk" in a non-realistic manner in the environment.

Behavior Cloning The BC agent resulted in an agent that appeared to learn to start to walk with a natural gait pattern. However, due to technical challenges faced in developing the code and humanoid environment, the network was only trained for 50,000 epochs. Thus, it is difficult to comment on the overall performance of the BC agent for gait imitation.

Model Comparison Unfortunately, due to the technical issues faced with training the BC agent and differences in humanoid environments we are unable to directly compare the performance of learning from scratch and imitation learning. Furthermore, since the BC agent did not complete a sufficient number of training epochs, the values are low and likely do not reflect the potential that the agent can achieve. Nonetheless, to estimate their current state, we simulated 1000 random starts with the trained agents to obtain a distribution of the reward values and total number of episodes steps taken (shown in Figure **??**).

Animated GIFs of the most up-to-date trained agents performing the task may be found here.

The results shown in Figure 4 reveal that RR and BC were limited to low rewards, around 100 or less. Qualitatively, this

SAC, the base deep network in this st

4 Discussion

Conclusions In this project, we attempted to make the case that leveraging human "expert" demonstrations for behavior cloning is more efficient and effective in training a reinforcement learning agent in a humanoid walking task. In this regard, our project is ultimately inconclusive as we were unable to carry out training to comparable levels. Our goal was to teach an agent to mimic naturalistic human gait with ridge regression–the non-deep model, Soft Actor-Critic–the base deep model, Twin Delayed DDPG–the advanced deep model, and behavior cloning–the experimental model. We wanted to demonstrate that, given minimal "expert" data, time, and computation, an agent could learn a policy more efficiently than RL agents trained on no data and "unlimited" computational resources. Most of our time was consumed by setting up the environments and tuning our hyperparameters that we were unable to train the Behavior Cloning (BC) agent for *at least* the same number of iterations as the Soft Actor-Critic (SAC) and the Twin Delayed DDPG (TD3).

During the learning process, each model developed a particular gait pattern. The SAC agent tended to take small steps while leaning backwards, stepping with its right foot forward, left arm outwards, and right arm to the side. The TD3 agent tended to take large steps while leaning forwards, stepping with its right foot first, both arms flailing up and forward, with a slight "hop" in its step. Currently, the BC agent stood upright, with its arms to the side, and took

one step forward before collapsing. The authors would like to highlight that despite its quick "failure," the BC agent expressed the most *natural* gait. Like BC, the RR agent stood upright and took a step forward like an average human does, and this is due to the high predictive nature between the state and action data. When initialized, the actions predicted from the current state using the RR model appeared to replicate natural human gait. However, as the errors accumulated, the action predictions led to torque values that quickly decayed, resulting in the humanoid collapsing and falling to the ground. We define "walking" as taking more than two steps without collapsing. Given that our SAC and TD3 agents were trained long enough to meet this criteria, and that the BC agent was not, we are unable to compare these three models. In light of these limitations, the next immediate step is to continue training all three of these models, ideally until all of them walk. We predict that given sufficient training time and computational resources, the BC agent will walk with more natural human gait and yield greater rewards in the environments than the SAC and TD3 agents.

Future work on this project would:

- Develop a unified PyBullet humanoid model for evaluating learning from scratch and imitation learning techniques. This provides comparable rewards, loss, and training time between models.
- Consider both reward types implemented by each model. The PyBulletHumanoid-v0 model considers only the humanoid pose when computing reward. On the other hand, the DeepMimic humanoid considers a weighted sum of the humanoid pose and the ability to replicate the expert.
- Expand training time, particularly for the behavioral cloning implementation to determine if walking can be achieved.
- Introduce perturbations to see how well these humanoids can continue walking in the face of perturbations, such as external forces and varying terrains.

Research Impact The advancement of RL has led to human-like performance in a number of tasks, including video games and board games. However, another use for these models is in the understanding of behavior and cognition. The simultaneous development of advanced learning methods, such as reinforcement learning and inverse reinforcement learning can help us model observed human behavior during cognitive and motor tasks. Some example problems include sequential decision making tasks and credit-assignment. The continued advancement of these techniques helps to expand our tool set for modeling human behavior, which may help in future work for the diagnosis and treatment of cognitive and motor disorders.

Environmental Impact The authors of this work acknowledge that the carbon footprint of these models is of sizable concern. Experiments were conducted using Google Cloud Platform in region us-east1, which has a carbon efficiency of 0.37 kgCO_2eq/kWh. A cumulative of 35 hours of computation was performed on hardware of type GTX 1080 Ti (TDP of 250W). Total emissions are estimated to be 3.24 kgCO_2eq. Estimations were conducted using the MachineLearning Impact calculator presented in [16]. With this point we hope to emphasize that the training and inference of deep learning models requires significant resources, which can have potentially harmful long-term effects on the environment [17].

Code Availability

All code can be found here.

References

[1] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[2] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, USA, 2010. AAI3438449.

[3] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *CoRR*, abs/1805.01954, 2018.

[4] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *CoRR*, abs/1804.02717, 2018.

[5] Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[6] Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. Integrating behavior cloning and reinforcement learning for improved performance in sparse reward environments. *CoRR*, abs/1910.04281, 2019.

[7] Yunda Liu, Sheng Bi, Min Dong, Yingjie Zhang, Jialing Huang, and Jiawei Zhang. A reinforcement learning method for humanoid robot walking. In *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 623–628, 2018.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[9] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[10] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[12] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[14] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[15] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[16] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[17] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery.