Robo-Daycare

Saurabh Ravindranath Anubhab Sen Geethika Rasineni

Anmol Khanna Pooja Anjee

Contents

1 Introduction

Humans are capable of learning new tasks very quickly by leveraging their experience. Our innate intelligence allows us to recognize objects from very few examples and learn to complete a new foreign task with very little experience. Ideally, we want our intelligent agent to be able to do the same thing, i.e., use prior experience to pick up new skills more quickly.

Our initial goal is to teach a robot how to walk using the Proximal Policy Optimization algorithm and then use it as a baseline for various other transfer learning tasks like playing soccer or walking upright in an environment with external forces applied from different directions. With hyperparameters and architecture kept similar across different environments, we plan to see if a PPO agent trained to walk in ideal conditions, will be able to walk in a customized, new OpenAI Gym [1] environment with little or no additional training.

The applications of our work are to optimize the time taken (and therefore the cost involved) to train robots in tasks like Robo-soccer in the real world, as a result of transferring behaviours learned within virtual environments to real-world scenarios. The trained movement data of Agents performing these tasks, being physically accurate, could also be used as a starting point for animations in games and other animated media, allowing for parallelization of tasks and thus leading to a more efficient workflow within the 3D animation pipeline.

We tested several modes of training Agents of varying complexities (the Hopper, Walker, and Humanoid Agents) to perform related tasks of increasing difficulty, with the end-goal of evaluating the efficacy of a Transfer Learning-based approach on the time taken to train a high-dimensional Agent. It was observed that Transfer Learning from less complex tasks and Agents resulted in significantly improved training time for more complex tasks and Agents at the same level of hardware computing power. This was verified by the Humanoid Agent successfully learning to locate, navigate towards, dribble and kick a football in a fraction of the time it would have taken for it to learn the same behaviors from scratch. It was thus seen that the salient elements of navigation and interacting with the football transfer well even to Agents of much greater complexity than the Agents used during the pre-training phase.

2 Background

The environments we use to train our bot is implemented in **PyBullet**. It is an easy-to-use Python module for physics simulation, robotics, and deep reinforcement learning based on the Bullet Physics SDK. The bodies of the robots can be specified in various file formats, such as URDF and SDF, which can be loaded into PyBullet. It also supports rendering,

which helps to visually observe the performance of our agents in the environment.

PyBullet ships with twelve environments. We will be using three built-in environments for this project namely Hopper, walker, and humanoid.

2.1 The Hopper

The Hopper agent has 4 joints/actions and 31 observations. The reward function has a positive reward signal for the pelvic velocity and pelvis uprightness. Similarly, there is a negative reward signal for the effort of the current action state and a penalty if the height is below 1.1m. The termination function triggers if a non-foot labeled body part collides with the terrain, if the height falls below 3m, or if the head tilts more than 0.4 units

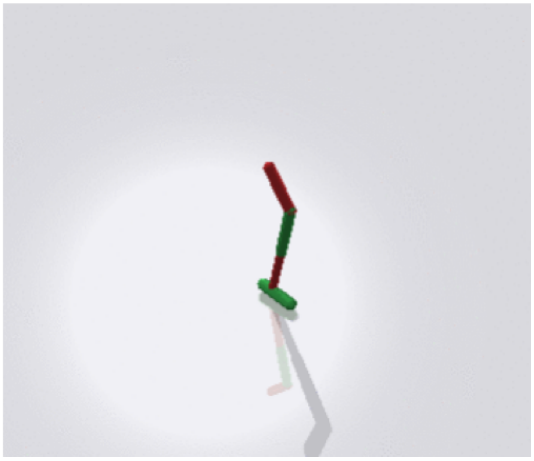

Figure 1: Hopper

2.2 The Walker

The Walker agent has 6 joints/actions and 41 observations. The reward function has a positive reward signal for the pelvic velocity and pelvis uprightness. There is a negative reward signal for the effort of the current action state and a penalty if the height is below 1.1m. The termination function triggers if a non-foot body part collides with the terrain.

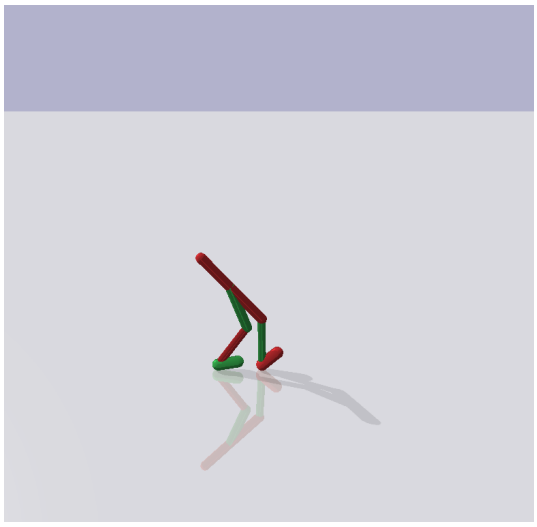

Figure 2: Walker

2.3 The Humanoid

The Humanoid agent has 21 joints/actions and 88 observations. The reward function has a positive reward signal for the pelvic velocity and pelvic uprightness, a negative reward signal for the effort of the current action state, and a penalty if the height is below 1.2m. It also adds additional rewards based on the phase cycle of its legs. The termination function triggers if a non-foot labeled body part collides with the terrain. As you can see humanoid is a very high dimensional control task and training in such high dimensional continuous spaces is quite challenging.

Figure 3: Humanoid

2.4 Modified Environments

A gym environment is a class with 4 functions. The first function is the initialization function of the class, which will take no additional parameters and initialize a class. It also

sets the initial state of our problem. The second function is the step function, which will take an action variable and will return a list of four things — the next state, the reward for the current state, a boolean representing whether the current episode of our model is done and some additional info on our problem. The other functions are reset, which resets the state and other variables of the environment to the start state and render, which gives out relevant information about the behavior of our environment so far. So, when we create a custom environment, we need these four functions in the environment.

Wind Environments We have created three environments -**'Walker2DWind'**, **'Hopper-Wind'** and **'HumanoidWind'** that contain a wind force, to show the feasibility of transfer learning. We transfer learn from vanilla environments, to quicken the training process in these new environments.

Soccer Environments We created three environments - **Walker2DDribble**, **Walker2DKick**, and **HumanoidDribble** suited for soccer tasks. The goal of these environments is to build a bot that can perform soccer tasks using transfer learning to build on behaviors that were previously learned.

The algorithm that we decided to use is Proximal Policy Optimization (PPO) which is the current state of the art, with its main advantage being its simplicity and efficiency.

3 Prior Research

Some of the most viable and promising processes through which robots can be trained to perform specific tasks are elaborated on below.

3.1 Deep Q Learning

Q Learning constructs a matrix for the agent which it uses to maximize its reward gained over time. This isn't feasible for environments with a large number of states and actions as it's not scalable. Deep Q Learning, in contrast, uses a neural network to estimate values. This doesn't affect performance when compared to Q learning when relative importance is preserved. This method was used to train an Agent to play Atari 2600 games at nearly professional human levels, with a high-dimensional pixel representation of the game screen and the game score as inputs. [4]

3.2 Policy Iteration

The agent only cares about finding the optimal policy in an environment. And we can obtain convergence of the policy before the values sometimes in an environment. Policy iteration redefines the policy at each step and finds the values according to the new policy until convergence. It is guaranteed to converge into the optimal policy and in practice, often takes fewer iterations than value iteration. Deep Conservative Policy Iteration [10] was

applied to the Cartpole Problem and Atari games, and showed clear improvements over DQN in terms of score achieved, low sensitivity to the random seed as well as low variance. This method provided an alternative to the regularization of greediness performed by the use of the KL divergence in TRPO and the clipping of policies in PPO, that being a stochastic mixture enabled by the decision to consider a discrete action space, though it also introduced a tradeoff between increased stability at the cost of speed.

3.3 OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It gives access to a plethora of "Environments", that are essentially playgrounds with physics engines for physically accurate simulations that interact with Agents made by the user. The Agent sends actions to the Environment and the Environment responds with Observations and Rewards. It is compatible with several Agents and has garnered much interest and attention in the field of Reinforcement Learning.

The work involved in OpenAI Gym deals with the creation of various Environments, each with it's own unique objective/s, within which Agents, either user-defined or vanilla, may be trained.

The Agents we have created for our Environments all work on the principle of having a Deep Neural Network as the policy network. This network takes the current state of the Environments as input and has a continuous output space, whose dimensions are based on the Environment.

To train this policy network, we use Reinforcement Learning (RL). The RL algorithm we use is called Proximal Policy Optimization (PPO).

After the Agent was trained on the Environment, we used Transfer Learning to train it in a different Environment, to observe whether its experience with the previous environment allowed it to train quicker, on the whole, within this new Environment.

3.4 Proximal Policy Optimization

PPO is a Policy Gradient algorithm. The objective of a Policy Gradient algorithm is to maximize the "expected" reward when following a parameterized (by the weights of the Neural Network, θ) policy, π_θ. One of the defining characteristics of this algorithm, when compared to other algorithms is the steps it takes to ensure the stability of the policy during training:

Advantage: Advantage ($\hat{A}_t$) is a term that denotes how good the steps taken by the policy are, from a given state. This is measured by the difference in the cumulative rewards obtained by the policy from the state and the expected *value* of that state. However, since the

expected value of a state is not known, the agent uses another Neural Network, namely the CRITIC network, that estimates the value of each state, as the policy network is being trained. Using this Advantage term instead of the cumulative reward gives more stability during training.

When training using Reinforcement Learning, it is important for the policy being trained to not be changed too drastically, as it could result in the agent having a policy that repeatedly yields sub-optimal results. This will further lead to bad policy choices in the future, resulting in unsuccessful training. To ensure that such drastic changes are not taken, a "trust region" is established [11], which is a region in the policy optimization space inside which it is safe to change the policy. PPO ensures that the new policy trained will remain inside this trust region by using a loss function with **cut-offs**.

$$L_\theta = E\left[\sum_{t=0}^{T} \min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right]$$

Here, $r_t(\theta)$ denotes the ratio between the old and new policies.

3.5 Proximal Policy Optimization with Covariance Matrix Adaptation

In a continuous action space, PPO is observed to prematurely shrink the exploration variance. This leads to slow progress, or worse, cause the agent to be stuck in a local minimum. The PPO-CMA is an adaptation of the PPO algorithm that aims to fix this [5]. It adaptively expands the exploration variance to speed up progress. It has specifically been observed to have improved performance in Roboschool continuous control benchmarks.

3.6 NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies also known as the NEAT algorithm [8] is a blend of genetic algorithms and neural networks. This algorithm evolves neural net architectures using a genetic algorithm and has shown great promise in reinforcement learning tasks. It uses all the steps of a genetic algorithm including a principled method of crossover of different topologies, speciation, and mutation. It shows how evolution can optimize as well as make more complex solutions simultaneously by evolving increasingly complex solutions over generations.

4 What We Built

We started with pybullet-gym [2], which is an open-source implementation of OpenAI Gym's MuJoCo [9] environments that could be used in tandem with OpenAI Gym's reinforcement learning research platform. The MuJoCo environments were appropriate for the task as it contains humanoid robots that bear a likeness to robots that are present in the real world and can be trained to perform certain tasks. There are also robots like the

walker, and the hopper, that are simpler versions of the humanoid. The walker is capable of playing soccer.

4.1 Environments

We first had to build environments that the robots are supposed to perform their tasks in, along with defining the task itself. We started with an environment where the robot (walker and hopper) needs to walk towards a target set of coordinates.

The next environment we developed was for the same task but with the presence of directional forces present to simulate a windy environment to make the walking more robust and as a proof of concept, to identify the feasibility of transfer learning in this domain. We found that there was considerable computational and time saving when transfer learning was used.

After successfully testing the usefulness of transfer learning, we started building environments that are appropriate for a robot learning to play soccer.
We built an environment for the robot to dribble the ball, that is, move forward along with the ball without kicking the ball too far ahead. Then we built an environment for the robot to shoot the ball. Finally, we attempted the dribbling problem with a humanoid robot, as that is several magnitudes more complex of a learning problem than the walker.

It was important to visualize the robot performing the tasks too, as often, we'd get good reward returns, which was obtained because of some flaw in the reward function that the robot was exploiting. This causes the robot to get high rewards without actually performing the task that was required.

4.2 Reward Functions

The trickiest part of the project was to develop good reward functions to make the robots perform the tasks we want them to. It involved a lot of reasoning, explaining the behavior of the robot towards a given reward function, and plain trial and error too. Broadly, we focused on four different rewards that add up to give the total reward. The rewards are:

- Alive Bonus: A small reward that is given to the robot for every time step that it remains alive. This motivates the robot to survive for longer periods rather than dying (falling over). This lets the robot complete drawn out tasks.
- Progress Reward: This gives a reward to the robot depending on how close it is to a target location.
- Ball Bonus: This gives a reward to the robot depending on how far the ball is moved from its initial position.

- Electricity Cost: A negative reward to ensure that the robot achieves its goals with the minimum movement possible.

We tried to make these rewards cumulative, to prevent one reward dominate every other and for mathematical elegance too.

We designed different reward functions with an amalgamation of the above rewards with different scales, sometimes even negative, to teach the robot to perform the tasks. The reward functions for each of the tasks are discussed in further detail in the Results and Analysis section.

4.3 Proximal Policy Optimization and Visualization

We decided to use the PPO algorithm to train the robot as it has shown promise in Gym environment tasks because of its ease of use and faster performance.

It was also crucial to monitor the performance of the robot (changes in the average reward obtained) during its training, as it is easy for the robot to get stuck in bad policy, which without monitoring the trends of the reward function, wouldn't be possible to identify. That would lead to wastage of computation, as training often takes several days. So we logged the rewards during the training and monitored them, visualized as graphs.

5 Results and Analysis

5.1 The Agent

As mentioned above, our agent uses a Deep Neural Network as the policy function approximator. Thus, the output of this network is the probability distribution of the actions the agent would take in the environment. The idea of Actor-Critic is to have two neural networks [3]. We estimate both: ACTOR: A policy function, controls how our agent acts. CRITIC: A value function, measures how good these actions are. The actor neural network has four hidden layers while the critic neural network has three hidden layers.

5.2 Transfer Learning

Transfer learning [6] is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task [7]. In our project, we trained the hopper and walker to walk from scratch. As proof of concept, we used transfer learning to train the hopper and walker in a windy environment. We also trained the agent from scratch in the wind environment, to compare the effects of transfer learning.

5.3 Training The Hopper

In the Hopper environment, the reward scores indicate the speed at which the Hopper agent moves, with higher rewards, when the agent moves quicker.

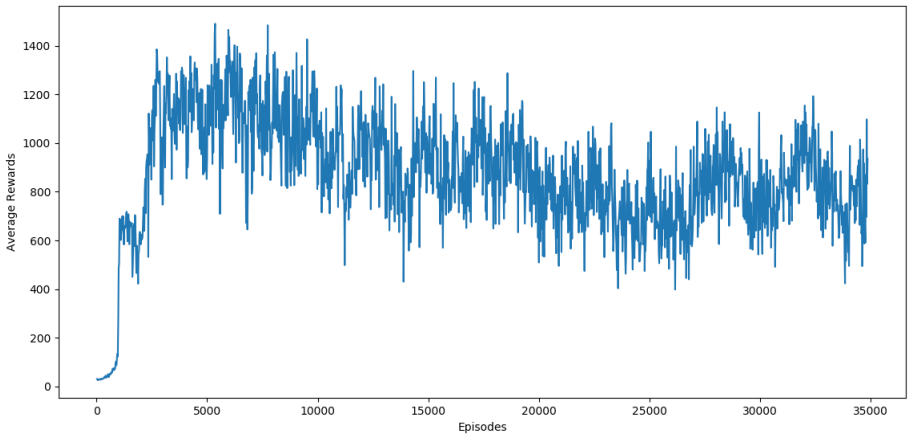

Figure 4: The training graph for the Vanilla Hopper environment.

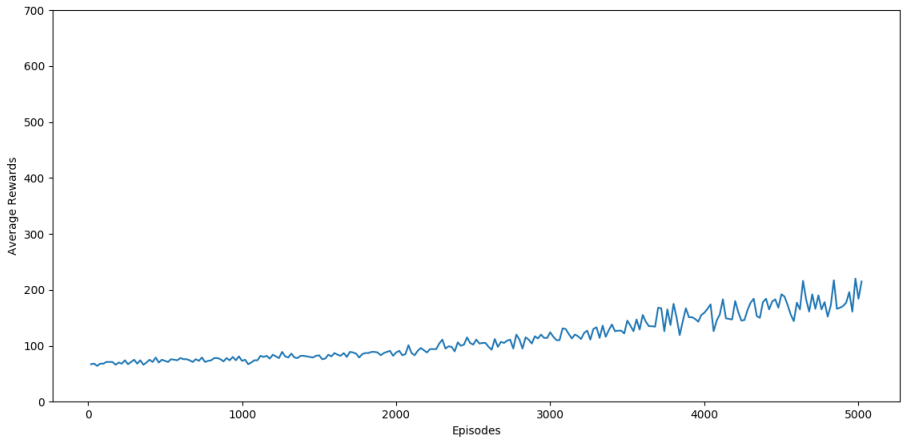

Figure 5: The training graph for the Hopper-Wind environment, when the agent is trained from scratch.

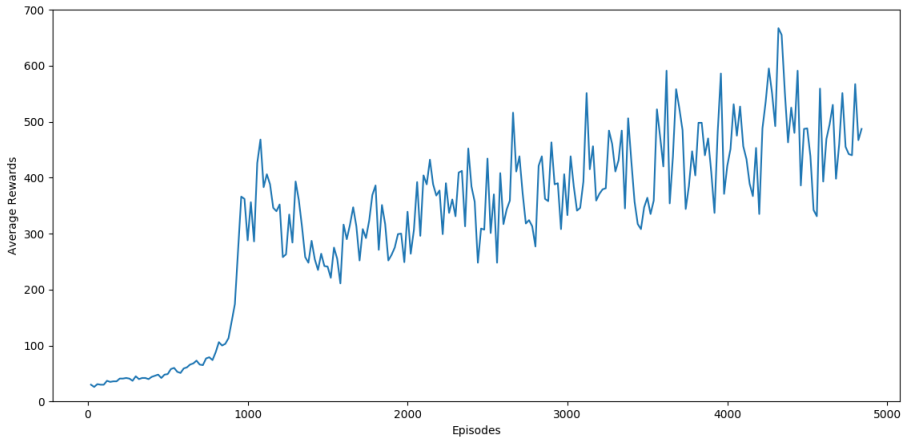

Figure 6: The training graph for the Hopper-Wind environment, when the agent from Figure 4 is trained using transfer learning.

From the above figures, we can see the effects of transfer learning in hastening the training of the agent in the Hopper-Wind environment.

5.4 Training The Walker

In the Walker environment, the reward scores indicate the speed at which the Walker agent moves, with higher rewards, when the agent moves quicker.

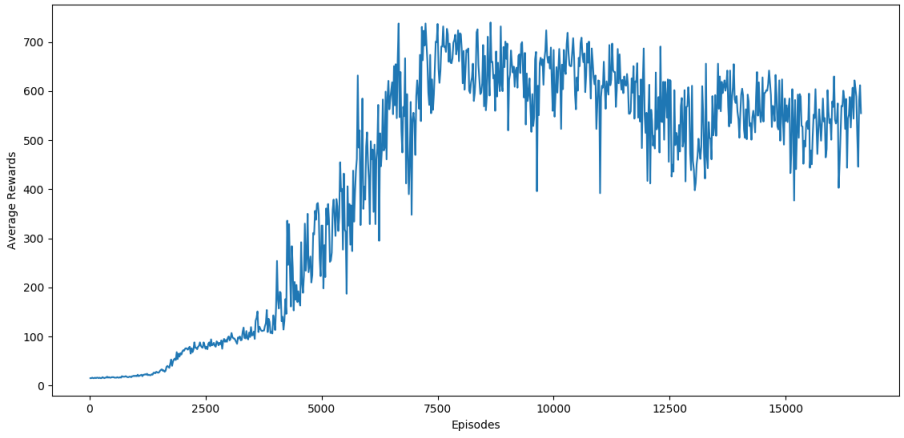

Figure 7: The training graph for the Vanilla Walker environment.

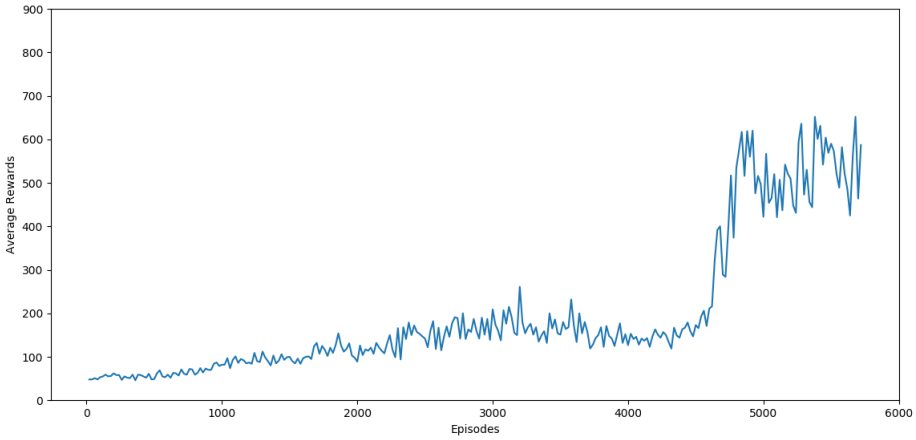

Figure 8: The training graph for the Walker-Wind environment, when the agent is trained from scratch.

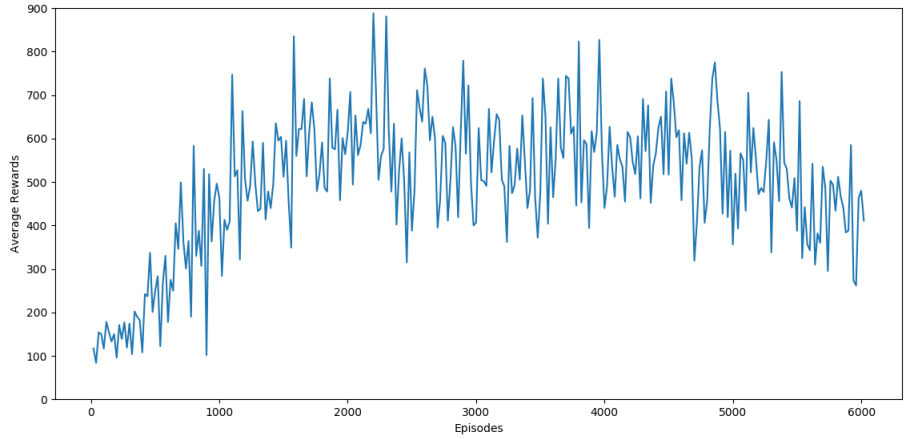

Figure 9: The training graph for the Walker-Wind environment, when the agent from Figure 7 is trained using transfer learning.

From the above figures, we can see the effects of transfer learning in hastening the training of the agent in the Walker-Wind environment.

5.5 Training The Walker to Dribble a Ball

To train the walker, we initially added the ”ball bonus” reward function to the environment, keeping the reward functions unchanged.
However, we observed that when training from scratch, this causes the walker to balance on the ball, instead of dribbling it, as the alive bonus dominates the reward function.

Figure 10: This is a frame in which the walker robot simply balances on the ball. The video link

To resolve this issue, we decided to transfer learn this task from the walker task. We observed that it trained successfully once the transfer learning was introduced, as the dribbling task requires knowledge of walking.

Figure 11: The walker learnt to dribble the ball successfully. The video link

5.6 Training The Walker to Shoot a Ball

Training the Walker to shoot the ball was a more difficult task, as we were unable to achieve decent results when transfer learning from walking.

This was because the Walker is supposed to kick the ball, then stop moving. To this end, we initially tried a few reward functions, such as only rewarding it for being alive and moving the ball, which resulted in the agent shown in Figure 12:

Figure 12: The walker only stays still as it can receive the alive bonus this way. The video link

Next, we tried scaling up the reward for moving the ball. This caused the Walker to simply touch the ball and then lose balance. As we wanted it to stay standing at the end of the episode, we had to change our training method.

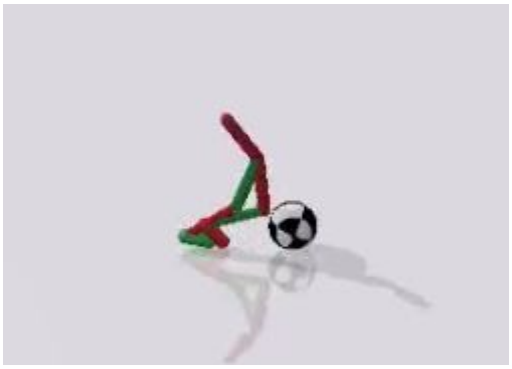

Figure 13: The walker manages to touch the ball, but loses balance soon after The video link

Finally, the training method that ended up working was to use transfer learning, but when using a different base environment.

We first created a Walker agent that simply balances itself by only having the alive bonus as the reward function. We then introduced that agent to our Walker Shooting environment, with the reward function only as the bonus obtained from kicking the ball. This ended up working well, and we created an agent that learned to kick the ball such that it traveled a large distance.

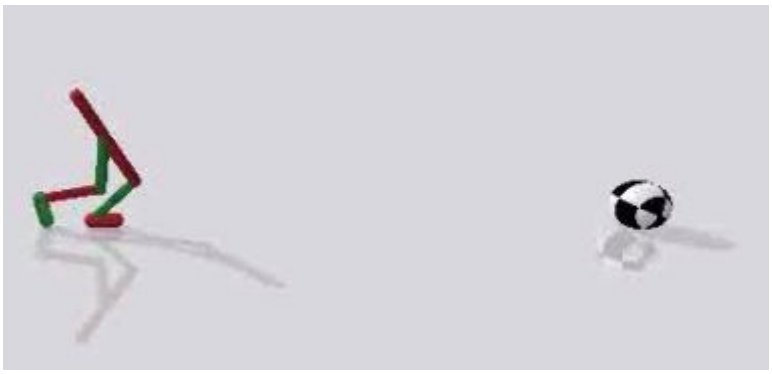

Figure 14: The walker kicks the ball quite far. The video link

5.7 Training the Humanoid to Dribble a Ball

The Humanoid is an agent with high input space. Hence training the Humanoid to walk proved to be difficult. Even after about a week of training, the results were not satisfactory.

However, transfer learning using a pre-trained Humanoid agent that walks yielded good results. The robot was quick to learn to first walk to, then dribble a ball that is spawned in a random location.

6 Limitations, Conclusions, and Future Work

6.1 Limitations

The limitations of this work are primarily related to the degree of correlation between the pre-training task and the fine-tuned task. For instance, the vanilla Walker was a good base Agent for the dribbling task, however, it proved to be an unsuitable start point for the shooting Environment.

6.2 Conclusions

It was observed that Transfer Learning has the potential to make real strides in this use case, since behaviors trained on simpler models provide a good starting point from which to begin training more complex tasks, exemplified by the vanilla Walker learning to shoot the ball successfully. This behaviour also transferred well to the Humanoid Agent, and Transfer Learning can thus be seen to yield excellent results in the training of even high-dimensional Agents with continuous action spaces.

It was also observed that the pre-training task must be selected carefully, keeping in mind the intended behavior of the fine-tuned task- for example the decision to use an Agent we trained only to stand upright, instead of the trained Walker Agent, as the pre-trained model for Transfer Learning to the Shooter Environment.

6.3 Future Scope

This work presents concrete evidence of the myriad advantages of applying Transfer Learning to the field of physics-based simulated Agents. Future work in this area might therefore include-

- Investigating Transfer Learning between different Environments within the same class of robot, such as two roughly humanoid robots with different part lists and/or weight distributions- such a study would be useful in the pursuit of extracting the general salient elements behind complex, multi-faceted behaviors that human beings perform on a day-to-day basis.
- Layering together objectives in increasing order of importance, a task that will likely benefit from the boost in training performance afforded by Transfer Learning. An example of this that follows from this work is to layer on the ability for a Humanoid Agent to navigate itself and the ball towards a goalpost in the presence of opponent Agents, and, once within a certain distance or when an advantageous opportunity presents itself, take a shot and attempt to score a goal.

Bibliography

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[2] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[3] S. Karagiannakos. Actor-Critic Networks. (1), 2018.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[5] X. M. J. L. Perttu Hamalainen, Amin Babadi. PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation. (1), 2018.

[6] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, page 213–226, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] D. Sarkar. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. (1), 2018.

[8] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[9] E. Todorov, T. Erez, and Y. 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] N. Vieillard, O. Pietquin, and M. Geist. Deep conservative policy iteration, 2019.

[11] X. T. Y. G. Yuhui Wang, Hao He. Trust Region-Guided Proximal Policy Optimization. 33(1), 2019.