The Art of Human Movement

Haodong Ma (dryice@stanford.edu), Renke Cai (renke.cai@stanford.edu), Klee Tang (kleetang@stanford.edu)
Advisor: Łukasz Kidziński (lukasz.kidzinski@stanford.edu)

Abstract

The goal of our project is to solve a physics-based reinforcement learning challenge "Learning to Run" by training a physiologically-based human model to navigate a course (with obstacles) as quickly as possible. We experimented with a list of state-of-art RL algorithms such as DDPG, PPO, and Continuous DQN. Due to the nature of the high-dimensional and continuous state and action space, we discovered that training is computationally intensive, and require long training cycles. It was challenging to debug and fine tune parameters. After some state space transformation, we can successfully train a walkable model, and also found that DDPG is the most promising and produce better results given the slowness of the simulator.

1 Introduction

The human musculoskeletal system is an organ system that gives humans the ability to move using their muscular and skeletal systems. This system provides form, support, stability, and movement of the body. Based on a framework supplied by Stanford Neuromuscular Biomechanics Laboratory (NIPS 2017: Learning to Run), our goal is to develop a motion controller to enable a physiologically-based human model to navigate a complicated obstacle course as quickly as possible. Our agent includes body segments for each leg, a pelvis segment, and a single segment to represent the upper half of the body. The environment setup is as follow: at each iteration, the motion controller is provided with an observation of the overall environment and the agent body. The motion controller needs to return an action, in which, the 18 muscles are actuated (9 per leg), so the action space is a list of length 18 of continuous values in [0, 1] corresponding to excitation of muscles. The Simbody physics engine computes activations of muscles, calculates torques generated and effects due to muscle activations, and eventually produces a new state based on forces, velocities, and positions of joints.

The reward is computed as a change in position of the pelvis along the x-axis minus the penalty for the use of ligaments. The termination is determined if either 1000 iterations are reached, or the pelvis height is below 0.65 meters.

The problem we are trying to solve is to build a controller (a function) that takes the current state observation and returns the muscle excitations action (18-dimension vector) in a way that maximizes the reward.

2 Related work

Reinforcement learning is becoming a common method to overcome modeling inaccuracies and improve performance in the robotics locomotion community. There has been research in this area including using reinforcement learning on bipedal robots [10] [11] , learning control for single legged robots [12] [13] , teaching robot to walk by demonstration [14] [15] , learning complex behavior for interaction with the environment [16] [17] , improving dynamic stability by using reinforcement learning [18] [19] .

In a related work by training robot learning to walk [20], it using a statistical actor-actor algorithm which updates the control parameters with each step and uses correlations between changes in control parameters and changes in the return map error to climb the performance gradient. The major limitation of the algorithm is convergence rate. In high dimensions, it requires many trials to effectively sample the entire state space. To solve the problem, it added a manual dimensionality reduction step which decomposes the control problem in the frontal and sagittal planes and exploits the dynamic coupling between the joint variables. The research also increases the number of degrees of freedom by using the formulation of the learning method on the discrete dynamics of the return map. The stochastic policy gradient algorithm may have problems with scaling as the increase of degrees of freedom, it requires more learning trails to obtain a good estimate of the correlation.

In the continuous action domain, the classic reinforcement learning methods do not perform well,

that is it can only handle low-dimensional action spaces. Recent research [21] presents progress in high-dimensional continuous action spaces by using deep deterministic policy gradient (DDPG) algorithm. It results demonstrate stable learning and requires fewer steps of experience to find solution. The research suggests that DDPG could solve even more difficult problems by more simulation time. Also, there are limitations using DDPG, that it requires a large number of training. And the research suggest using a robust model-free approach such as an important component of larger system to attach the limitations. We aim at achieving similar results for skeleton legs with DDPG training.

3 Dataset and Features

The model is implemented in OpenSim[1], which relies on the Simbody physics engine. The agent is a musculoskeletal model that include body segments for each leg, a pelvis segment, and a single segment to represent the upper half of the body (trunk, head, arms). The segments are connected with joints (e.g., knee and hip), and the motion of these joints is controlled by the excitation of muscles. The muscles in the model have complex paths (e.g., muscles can cross more than one joint, and there are redundant muscles). The muscle actuators themselves are also highly nonlinear. Given the musculoskeletal structure of bones, joints, and muscles, at each step of the simulation (corresponding to 0.01 seconds), the engine calculates activations of muscles from the excitations vector provided to the step() function to generates a new state based on forces, velocities, and positions of joints.

The initial set of observation contains 41 values:

- position of the pelvis (rotation, x, y)
- velocity of the pelvis (rotation, x, y)
- rotation of each ankle, knee and hip
- angular velocity of each ankle, knee and hip
- position of the center of mass
- velocity of the center of mass
- positions of head, pelvis, torso, left and right toes, left and right talus
- strength of left and right psoas
- next obstacle: x distance from the pelvis, y position of the center relative to the the ground, radius.

We started our baseline using these observations as the state for the Markov Decision Process. However, we discovered that even though we had some initial success quickly (the skeletal model was able to stand and take steps after training using 500K steps), a more extended period of training didn't produce any improvements over the test results. After analyzing the list of observations, we noticed that some information is missing, for example, we only have position information of the head, torso, etc. but no velocity information is provided, which need to be calculated based on two consecutive observations. We also further normalize the x position information across all parts, using relative coordinates regarding pelvis bone, while setting the pelvis position to zero.

Since only information about the obstacle right in front of the agent is provided, we modified the logic to obtain the three closest obstacle information. The final state contains 57-dimensions

4 Methods

We experimented with a list of state-of-art RL algorithms that work in this continuous high-dimensional action and state space environment.

Q-learning is a model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. It works in a way that we create a table to store each state and each action’s Q value in this state.

Deep Q Network (DQN) algorithm uses a Neural Network instead of a table described above and this therefore solves problems with high-dimensional observation spaces, but it can only handle discrete and low-dimensional action spaces. It is not possible to straightforwardly apply Q-learning to continuous action spaces, because in continuous spaces finding the greedy policy requires an optimization of at every time step, which is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Recently researchers have developed two ideas for extending the success of Deep Q-Learning to the continuous action domain: **Deep Deterministic Policy Gradient** (DDPG [3]) and **Continuous DQN**[6].

DDPG is an off-policy method and it consists of actor and critic neural networks. Critic is trained using Bellman equation and off-policy data and provide the actor with the loss function:

$$Q(S_t, a_t) = r(S_t, a_t) + \gamma Q(S_{t+1}, \pi\theta(S_{t+1}))$$

and the actor is trained to maximize the critic's estimated Q-values through back propagation:

$$\partial \mathcal{L}(\theta) \partial \theta = \mathrm{E}[\partial Q(S_t, a_t) \partial a \partial a \partial \theta]$$

Continuous DQN [6] is a continuous-action-space variant of the Q-learning algorithm which we refer to normalized advantage functions (NAF), as an alternative to the more commonly used policy gradient and actor-critic methods. The idea behind normalized advantage functions is to represent the Q-function Q(xt,ut) in Q-learning in such a way that its maximum arg maxu Q(xt,ut) , can be determined easily and analytically during the Q-learning update.

This method combines the NAF representations with deep neural networks into an algorithm that can be used to learn policies for a range of challenging continuous control tasks with experience replay, and substantially improves performance on robotic control tasks.

Proximal Policy Optimization (PPO [4]) is an on-policy method and a variant of Trust Region Policy Optimization (TRPO) that adds a KL penalty term (or an alternative is clipped surrogate objective) to training loss function, and train the policy using gradient descent like normal neural network. While Policy Gradient is fundamental in using deep neural networks, it suffers from a problem that they are very sensitive to learning rate (step size). The progress will be extremely slow if the choice of step size is too small, and becomes too noisy if the step size is too large. PPO's penalty term utilizes the comparison between new policy and old policy to control the update of each step, reducing the sensitivity to step size.

5 Experiments/Results/Discussion

Initially, we trained models on laptops; it became apparent quickly that we need to find more compute power to parallelize various configuration tests. We decided to utilize AWS EC2 instances, in particular, compute-intensive C5 instances. These instances are designed for compute-heavy applications like batch processing, distributed analytics, and high-performance computing. However, they are quite expensive @ $0.085 per hour for the cheapest version with 2 vCPUs, and we have set a hard limit with a budget of $250 for this project, and completed the project with $23 over budget.

Our test results using 500 steps

Training Steps	**DDPG w/o state transformation**	**DDPG with state transformation**	**PPO**	**Continuous DQN**
500,000	1.24m, 158 steps	0.37m, 500 steps	<0m, 104 steps	0.421m, 74 steps
1,000,000	2.57m, 350 steps	2.84m, 468 steps	<0m, 125 steps	0.561m, 112 steps
2,000,000	1.95m, 182 steps	4.93m, 370 steps	0.99m, 121 steps	not attempted
4,000,000	0.32m, 63 steps	7.34m, 405 steps	1.29m, 127 steps	not attempted
8,000,000	1.48m, 160 steps	not attempted	1.07m, 123 steps	not attempted

For DDPG, we used both 3 hidden layer NN for Actor and Critic component, with hidden layers containing 32 and 64 nodes for Actor and Critic correspondingly. We chose sigmoid as the activation function for the Actor component, as the action space for the muscle activation fall into the range of 0-1. and we used the following hyper parameters

Actor NN	[32,32,32] sigmoid activation
Critic NN	[64,64,64] linear activation
gamma	0.995
replay buffer	100k
Batch Size	100
learning rate	0.001
OU exploration	theta=.15, mu=0., sigma=.2

With DDPG, after the state transformation and normalization, we start seeing improvements and stabilization of our agent.

For Continuous DQN, we used NAFAgent from Keras-rl (a RL package) with both 3 hidden layer NN for Q Value function and Action, and a 2 hidden layer NN for Loss. Specifically, the hyper parameters were:

- Q value NN [32,32,16] Relu activation for hidden layers and Linear for output layer
- Action NN [32,32,16] Relu activation for hidden layers and Sigmoid for output layer
- Loss NN [64,64] Relu activation for hidden layers and Linear for output layer

Other parameters are the same as those of DDPG we used above. Due to the limitation of computing resources, we only ran up to 1m iterations and the results did not significantly outperformed that of DDPG.

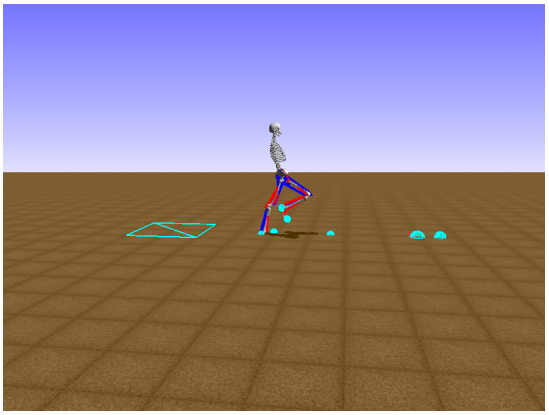

Figure 1: walking model

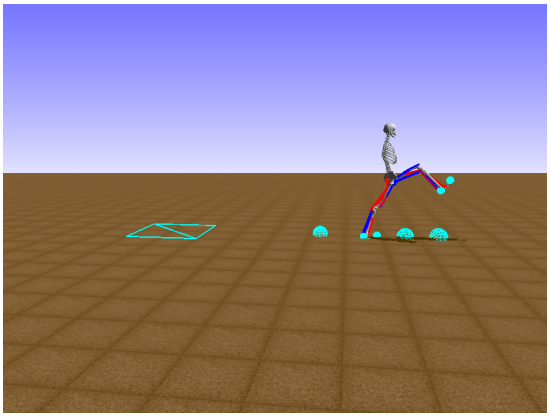

Figure 2: jump over obstacles

We ran into some issues while trying to integrate the PPO algorithm. Namely, the PPO implementation from OpenAI's baselines package is based on Python 3.5, which is not compatible with the OpenSim models (which is based on Python 2.7). We first attempted uplifting OpenSim with Python 3.5, as we hope it will also create a faster simulator. After some research, we managed to rebuild OpenSim for Python 3.5 on Linux environment, however, since we only do training on Linux, but carry out the most of the development and validation on Mac OS X, we also have to rebuild OpenSim for Mac OS X as well, this turned out to be a time-consuming process, and at the end, we had to abandon the Python 3.5 uplift, and went the other way instead. Porting OpenAI's PPO implementation back to Python 2.7 was straightforward.

We used the default parameters from OpenAI Hopper Mojuco during our evaluation

2 hidden layers, each with 64 nodes	
gamma	0.99
lam	0.95
batch size	64

MPI (Message Passing Interface) is used to parallelize compute to run multiple cores and trained our models with up-to 8 million steps.

Since PPO is a on-policy based algorithm, which updates agent's behavior based on the current agent's policy, our results from PPO based agent is not optimal. One possible theory is our simulator speed. From early on, we noticed that OpenSim based simulator is extremely slow, ~5 time steps per second. From OpenSim discussion board, Qin Yongliang posted that there is a way to recompile OpenSim with less accuracy in terms of physics simulation, but could double the simulator speed, unfortunately we ran out of AWS EC2 budget, and was not able to explore this option and validate its impact on PPO based agent.

6 Conclusion/Future Work

Continue focusing on DDPG, and dive deeper to fine-tune its parameters. As described by Pavlov et al. [7], we may try parameter noise (Plappert et al. 2017), Layer normalization (Ba, Kiros, and Hinton 2016) and Actions and states reflection symmetry to see how they may improve our DDPG method.

Since RL is computationally intensive, we need to find better ways to distribute DDPG optimization over cheaper AWS EC2 instances and reduce the training time required. We would also like to explore ways to retrain models without having to start from scratch each time. We aim to try out different options with replay buffer to improve learning speed and understand the impact on the learning speed related to exploration noise.

Contributions

Haodong - Data processing, DDPG, and PPO integration and testing, research, poster, and report preparation.
Renke - research, Continuous DQN implementing, poster and report preparation.
Klee - Research on relative works and different algorithms, run testing with DDPG and PPO, poster and report preparation.

Acknowledgements

We would like to thank Prof. Andrew Ng, Prof. Dan Boneh, all the TAs of CS229, and Łukasz Kidziński for offering us this great opportunity to stretch our abilities and imaginations to work on such an interesting and fascinating project. When we first saw

the human model started walking after long period of training, like Prof. Ng mentioned during one of his lectures, it felt like "magic", and it was a priceless and joyful feeling.

References/Bibliography (No page limit)

[1] Stanford Neuromuscular Biomechanics Laboratory NIPS 2017: Learning to Run https://www.crowdai.org/challenges/nips-2017-learning-to-run

[2] Reinforcement learning environments with musculoskeletal models 2017 GitHub https://github.com/stanfordnmbl/osim-rl

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra 2016. Continuous control with deep reinforcement learning arXiv:1509.02971

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov 2017. Proximal Policy Optimization Algorithms arXiv:1707.06347

[5] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, David Silver 2017. Emergence of Locomotion Behaviours in Rich Environments arXiv:1707.02286

[6] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, Sergey Levine, 2016. Continuous Deep Q-Learning with Model-based Acceleration arXiv:1603.00748

[7] Mikhail Pavlov, Sergey Kolesnikov, Sergey M. Plis, 2017. Run, skeleton, run: skeletal model in a physics-based simulation arXiv:1711.06922

[8] Dhariwal, Prafulla and Hesse, Christopher and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, 2017 OpenAI Baselines GitHub https://github.com/openai/baselines

[9] Qin Yongliang https://ctmakro.github.io/site/index.html

[10] Miller, W.T., 1994. Real-time neural network control of a biped walking robot. IEEE Control Systems, 14(1), pp.41-48.

[11] Benbrahim, H. and Franklin, J.A., 1997. Biped dynamic walking using reinforcement learning. Robotics and Autonomous Systems, 22(3-4), pp.283-302.

[12] Fankhauser, P., Hutter, M., Gehring, C., Bloesch, M., Hoepflinger, M.A. and Siegwart, R., 2013, November. Reinforcement learning of single legged locomotion. In Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on (pp. 188-193). IEEE.

[13] Doerschuk, P.I., Simon, W.E., Nguyen, V. and Li, A., 1998. A modular approach to intelligent control of a simulated jointed leg. IEEE Robotics & Automation Magazine, 5(2), pp.12-21.

[14] Meriçli, Ç. and Veloso, M.M., 2010, July. Biped Walk Learning Through Playback and Corrective Demonstration. In AAAI (pp. 1594-1599).

[15] Argall, B., Browning, B. and Veloso, M., 2007, March. Learning by demonstration with critique from a human teacher. In Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on (pp. 57-64). IEEE.

[16] Buchli, J., Stulp, F., Theodorou, E. and Schaal, S., 2011. Learning variable impedance control. The International Journal of Robotics Research, 30(7), pp.820-833.

[17] Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M. and Silver, D., 2017. Emergence of locomotion behaviours in rich environments. arXiv preprint arXiv:1707.02286.

[18] Tedrake, R. and Seung, H.S., 2002, November. Improved dynamic stability using reinforcement learning. In International Conference on Climbing and Walking Robots (CLAWAR) (pp. 341-348).

[19] Goswami, A., 1999. Postural stability of biped robots and the foot-rotation indicator (FRI) point. The International Journal of Robotics Research, 18(6), pp.523-533.

[20] Tedrake, R., Zhang, T.W. and Seung, H.S., 2005, June. Learning to walk in 20 minutes. In Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems(Vol. 95585, pp. 1939-1412). New Haven (CT): Yale University.

[21] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[22] Gu, S., Holly, E., Lillicrap, T. and Levine, S., 2016. Deep reinforcement learning for robotic manipulation. ArXiv e-prints.