Run, Reward, Repeat: Training Musculoskeletal Models with Deep Deterministic Policy Gradients

Amy Chou
Stanford University
amyachou@stanford.edu

Rory Lipkis
Stanford University
rlipkis@stanford.edu

Victoria Tsai
Stanford University
vitsai@stanford.edu

Abstract

Deep reinforcement learning techniques have shown to be promising in high-dimensional state and continuous action spaces. We apply deep deterministic policy gradients (DDPG) in a high-dimensional state and action space to train a skeletomuscular human model to run, as simulated by the Stanford NMBL running environment based on the OpenSim simulation and OpenAI Gym environment. We test a variety of different configurations on the sequential actor and critic neural networks of the off-policy model-free DDPG, which learn a policy giving 18-dimensional action vectors from an input of 41-dimensional observation vectors representing the human frame. Our results indicate that the learning configuration best suited toward achieving long-term rapid locomotion uses three-layer fully connected sequential neural networks with leaky reLU activation in both the actor and critic networks. This project has further implications in the fields of robotics and prosthetics, in which robust bipedal locomotion is a longstanding objective.

1 Introduction

The machine learning community has seen a surge of reinforcement learning techniques in recent years, largely spurred by the advent of deep reinforcement learning, which combines deep learning with traditional Q-learning techniques. In light of the success of such techniques as applied to human-level control problems as variegated and challenging as car driving, the continuous bandit problem [3], or playing Atari [4], it is of interest to explore the benefits of applying deep RL to a high dimensional control problem.

Applying reinforcement learning efficiently to high dimensional state-action systems of human movement remains an area of active research; however, its implications for humanoid robotics are monumental. Recent advances include OpenAI's studies on robot controllers trained entirely in simulation using reinforcement learning in October 2017 [5, 1].

We address the "Learning to Run" challenge featured in the NIPS 2017 Competition Track with specifications laid out by the Stanford Neuromuscular Biomechanics Laboratory (NMBL) group as follows:

> Given a human musculoskeletal model and a sandbox physics simulation environment, train an agent to navigate an obstacle course optimizing for distance travelled over a set time interval. Potential obstacles may be both external and internal, including but not limited to steps, slippery floors, muscle weakness, and motor noise [2].

The objective of this project is twofold: to train a simulated human to run and to evaluate various parameter configurations on their likelihood of quickly achieving long-term locomotion. Rather than optimizing the computational infrastructure to generate a model that results in the human simulation to navigate an obstacle course as quickly as possible, we are interested achieving this objective most efficiently with limited computational resources.

As a secondary problem, we train the human model to balance while standing still. The environment for standing requires a lower-dimensional observation vector; thus, it is less complex than the running problem and informs the results for the primary objective of running.

2 Background

The core of reinforcement learning revolves around learning to act in a dynamic system in such a way as to accomplish a given objective. Most reinforcement learning approaches follow a general paradigm of learning a particular policy with respect to some reward or loss by exploring choices and picking actions that maximize value in the long run.

Many such problems can be reformulated as Markov decision processes, wherein the Markov property holds approximately true. Specifically, given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, if an agent at time t is in state $s_t \in \mathcal{S}$ and takes action $a_t \in \mathcal{A}$, the probability of ending up in state s'_{t+1} can be fully described by a transition function $\tau : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ which is dependent only the current state and action s_t and a_t as opposed to, as one might expect, the entire history of states and actions $s_{1:t}$ and $a_{1:t}$ or the time t.

A policy function π maps actions to states so that an agent in state s knows to choose action $\pi(a)$. Policy functions may be deterministic or stochastic in nature, but the goal of reinforcement learning is to iteratively improve an estimate of the optimal policy over time.

Traditional reinforcement learning operates in discretized state and action spaces, and estimates the transition function τ by exploring the environment, choosing actions over some policy π_t and observing rewards. Then, as the transition function improves, a better estimate of the value function

$$Q(s, a) = R(s, a) + \sum_{s' \in \mathcal{S}} \tau(s, a, s') + R(s', a)$$

where the reward function R and transition function τ are approximated through maximum likelihood based on all observations to date, and $\tau(s, a, s') = p(s'|s, a)$. The class of techniques which follow the above general heuristic is termed Q-learning, and has been shown to be effective in a variety of contexts.

When the action space $\mathcal{A}$ is continuous, it becomes much more difficult to apply Q-learning as outlined. Instead, policy gradient methods provide an effective approach to such problems. Traditional policy gradients represent the stochastic policy $\pi_\theta(a|s)$ of taking action a in state s as a probability parameterized by θ, then take gradients with respect to the parameter θ to maximize the value function Q. In such a case, the performance objective, which is the expected reward of a given policy, can be written

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s, a) r(s, a) \, da \, ds$$

where $\rho^\pi(s')$ gives the discounted state distribution. The policy gradient theorem proved by Sutton in [7] demonstrates that

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) \, da \, ds$$

which is equivalent to

$$\mathbb{E}_{s \sim \rho^\pi, \, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a) \right]$$

Because the target policy is stochastic, policy gradients must be integrated over the set of all states and actions, making them extremely computationally expensive even to estimate.

Silver et. al in [6] present a method of taking deterministic policy gradients; i.e., learning a deterministic rather than stochastic policy, so that π is a function only of the state. While this method was previously considered unviable without a specific model, the deterministic policy gradient theorem gives an analog to Sutton's policy gradient theorem, so that for a deterministic policy $\mu(s)$ (to differentiate from the stochastic π),

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s)\nabla_\theta(s)\nabla_a Q^\mu(s,a)|_{q=\mu_\theta(s)}\,ds \\ &= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta\mu_\theta(s)\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]\end{aligned}$$

While on-policy and off-policy exploration strategies are both possible, it is far more effective to explore the entire state and action space using a stochastic policy of action choice but still estimating a deterministic policy [6], which can only be done using an off-policy learning algorithm. A common framework is the actor-critic architecture, wherein the actor estimates the policy function $\pi_\theta(s)$ for a particular parameter, and the critic calculates the value function $Q^\pi(s,a)$. Over time, the two actor and critic networks calculating the given functions iteratively optimize each other and themselves with input from the environment. The DDPG approach given is model-free, and thus highly generalizable.

3 Methodology

We use the open-source OpenSim simulation libraries backed by the Simbody physics engine to dynamically simulate motion and the Keras-RL framework backed by Tensorflow to build the model. The Stanford NMBL Group provides a modifiable running environment built atop the OpenSim libraries and OpenAI Gym. From this starting point, we tune a DDPG model as outlined in the previous section to fit simulated observations and actions. Deep Q-networks were considered as an alternative approach, but have demonstrated poor performance in continuous action spaces despite widespread success in discrete action spaces [3].

The inputs to the learning model are successive 41-dimensional observation vectors representing the state of the agent. Simulated by the OpenSim running environment, the observation vector includes features such as positions and velocities of relevant joints. The action space is an 18-dimensional vector representing excitation levels of 9 different muscles on each leg, while the reward function is given by the horizontal distance traveled by the pelvis minus a penalty for use of ligament forces. Applying the off-policy, model-free actor-critic DDPG algorithm given in the section above, we model both the actor and critic functions with fully-connected sequential neural networks.

In our application of DDPG, we vary the number of hidden layers in the actor and critic networks from 1 to 3, each time with fully-connected layers and the same activation function. For the actor network, we test hidden layers with both 16 and 32 parameters each, while for the critic network we test layers with 32 and 64 parameters, the rationale being that Q-learning is expected to be more complex than policy estimation. In any given run, we keep the number of parameters and activation function identical for every hidden layer in a network, but also test both reLU and leaky reLU activation functions in the hopes that eliminating the vanishing gradient problem would yield a better model.

4 Results

As a baseline, it is apparent that without any training, the agent immediately falls backwards. The skeleton shown in Figure 1 trained for 0 steps and incurred a negative reward of -0.809.

We experimented with the number of hidden layers for the actor and critic networks, maintaining 32 hidden nodes per layer in the actor network and 64 per layer in the critic network. With a series of experiments in which each model was trained for 200,000 to 500,000 simulation iterations, neither one nor two hidden layers adequately captured the complexity of the problem. Only with three hidden layers in each network was the agent able to take a definitive step forward after 200,000 simulations with both the reLU and leaky reLU activation functions. However, with only two hidden layers for each network, the agent was unable to take a step forward, even after 500,000 simulation steps with either activation function, and with one hidden layer, the agent fell forward without taking a step when using leaky reLU and fell backwards when using reLU. These results show that in the early stages of training, a model with three hidden layers for each neural network was necessary. We did not increase the number of hidden layers beyond three for reasons of training time.

We also experimented with the activation function used in the neural network with the option of a reLU activation function or leaky reLU with a leakiness parameter of 0.2. Both qualitatively and

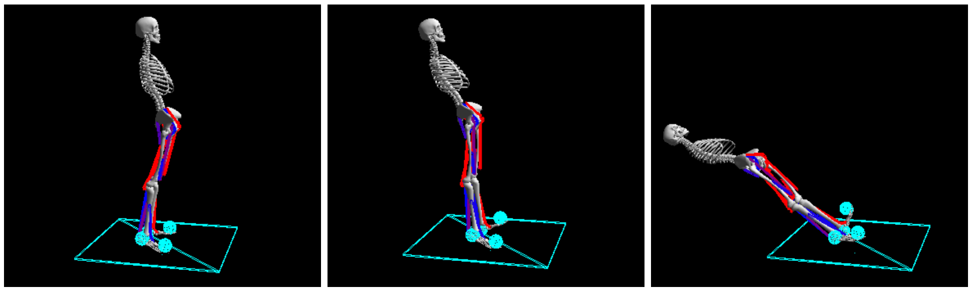

Figure 1: The baseline simulation on the default random policy when no training is applied.

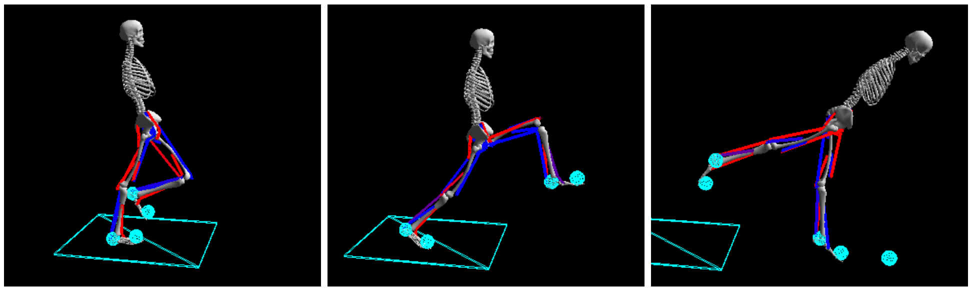

Figure 2: The agent trained with leaky reLU activation functions an 3-layer fully-connected actor and critic networks for 200,000 steps.

quantitatively, the leaky reLU model outperformed the reLU one. With a reward of 2.219 when testing after 200,000 simulation training iterations, the agent trained by the leaky reLU model took a definitive running step forward and resumed a running position for the second step, as demonstrated in Figure 2.

The reLU-trained agent in Figure 3 was also able to take one step forward, but it incurred a smaller reward while testing (1.566 in this particular iteration), and qualitatively, it appeared to use motions less similar to that of running before falling over.

Controlling for the number of hidden layers and activation function and using a smaller number of hidden nodes — 16 and 32 for the actor and critic layers, respectively — the test run after training for 200,000 steps using the leaky reLU activation function and three hidden layers achieved a reward of 1.252 and was not able to take a definitive step.

Thus, the parameters that best capture the complexity of the problem use actor and critic neural networks with three hidden layers with 32 and 64 nodes, respectively, and an activation function of leaky reLU. Using these parameters to train for upwards of 500,000 simulation steps, we compute statistics every iteration of 10,000 training steps as well as the reward incurred at a particular step. Shown in Figure 4 are the resulting graphs of reward per iteration and per step.

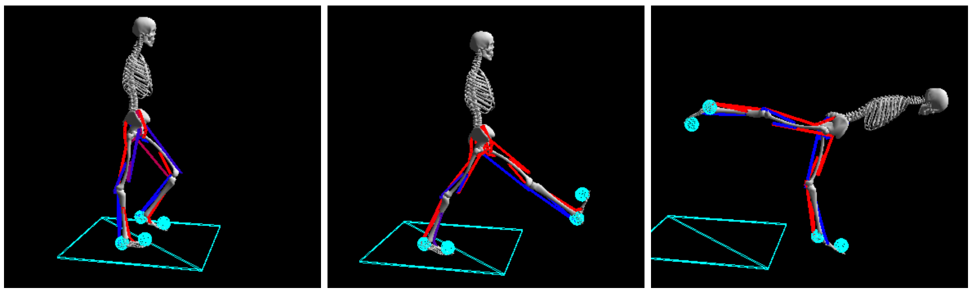

Figure 3: The agent trained with reLU activation functions on 3-layer fully-connected actor and critic networks for 200,000 steps.

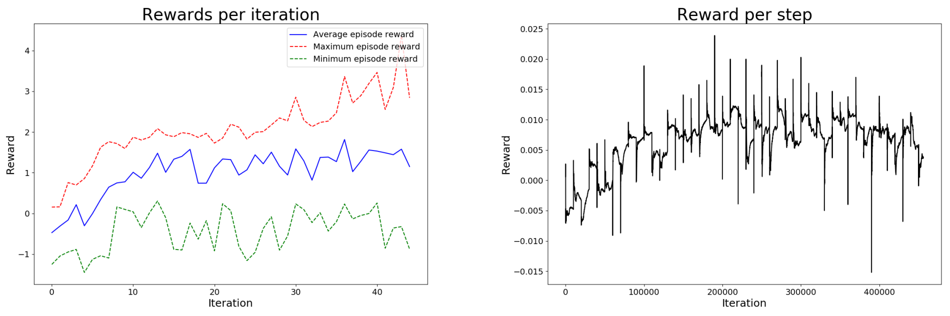

Figure 4: Reward per iteration on the optimal actor-critic configuration after 500,000 simulation steps.

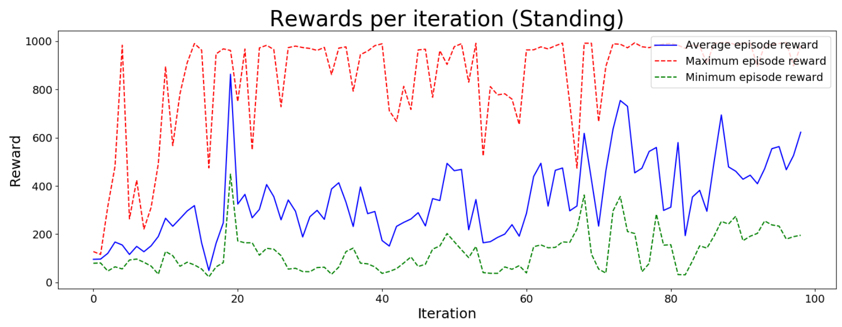

Figure 5: Reward per iteration after 1 million simulation steps for the standing objective. Capped at a reward of 1000 for 1000 training steps.

We see that the average and maximum reward per episode increased with more iterations, as does the reward per step. At the end of 500,000 training steps, the maximum reward per episode reached a peak of 4.0, which is about four steps taken by the agent.

The secondary objective was training the agent to stand in order, which offered a 10 fewer degrees of freedom in the action space. The lower complexity of the standing problem allowed a 1 million step simulation, after which the agent was able to stay balanced for at least 3000 simulation steps, at which point it was assumed that the agent will stay balanced for an arbitrarily long amount of time. Figure 5 gives reward over training, which peaks quickly.

5 Conclusion

After around 500,000 training steps, the agent successfully learned to take steps but did not achieve long-term rapid locomotion in the allocated time. The simulation was very computationally heavy, with 500,000 training steps taking over 36 hours to train on Stanford's high performance computing clusters, and difficult to parallelize as the physical simulation was the performance bottleneck and took up over 90% of runtime. Optimizing for both performance and training time, we discovered three hidden layers in the actor and critic networks with leaky reLU activations performed best. The reward function continues to increase both on average and at maximum at around 500,000 training steps. In the case of the standing problem, long term stability was achieved.

While tuning the training parameters do indeed yield better results, we expect great improvements with infrastructure rather than parameter optimizations. The project would be best extended by gaining access to a high-performance cluster for a long period of time or by parallelizing computations.

Contributions

Model design, algorithms, and approach mainly due to Victoria Tsai; hyperparameter tweaking, infrastructure, and other configuration largely attributed to Rory Lipkis, and analysis credits and model design experimentation go to Amy Chou. Coding and paper-writing was a joint collaborative effort among all group members, with provided configuration code at `https://github.com/stanfordnmbl/osim-rl`.

References

[1] Maruan Al-Schedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Meta-learning for wrestling, 2017.

[2] Lukasz Kidzinski and Sharada Mohandy. Nips 2017: Learning to run, 2017.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] Xue Bin Peng, Lerrel Pinto, Alex Ray, Bob Mcrew, Jonas Schneider, Josh Tobin, Marcin Andrychowicz, Peter Welinder, Pieter Abbeel, and Wojciech Zaremba. Generalizing from simulation, 2017.

[6] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

[7] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.