Deep learning based motor control unit

Viktor Makoviichuk (06086584)
viktorm@stanford.edu

Peter Lapko(06167608)
plapko@stanford.edu

Advisor: Lukasz Kidzinski
lukasz.kidzinski@stanford.edu

Abstract

We present a deep reinforcement learning approach to control musculoskeletal biomechanical models. Our work consists of 2 different parts: experimental, with simple Arm model to explore hyperparameters space of DDPG algorithm and practical - training simplified Human model to stand and to walk. After a number of experiments we found optimal parameters values that greatly improved learning rates and stability of training Arm and Human models In the second stage we achieved different of eternal standing - on one leg, in semi-split, and robust standing on 2 legs. For the walking environment we successfully trained our model to do the first steps.

1 Introduction

The development of physics-based locomotion controllers, independent from captured motion data, has been a long-standing objective in computer graphics and robotics research and recently in biomechanical communities. In robotics with the help of reinforcement learning instead of expensive and complex explicit programming robots can teach themselves how to move in virtual spaces and real environments like in [1].

In biomechnics research a lot of models were developed, to fit the clinical data to understand underlying causes of injuries using inverse kinematics and inverse dynamics. For many of these models there are controllers designed for forward simulations of movement, however they are often finely tuned for the model and data. Advancements in reinforcement learning may allow building more robust controllers for large number of tasks.

For our project we used two musculoskeletal models: ARM with 6 muscles and 2 degrees of freedom and HUMAN with 18 muscles and 9 degrees of freedom. For ARM 1 environment was used, where the ARM's joints were supposed to reach certain randomly generated angles. This environment was used for hyperparameters space exploration. For HUMAN we used 2 variants: standing still and walking - gait.

2 Related Work

Methods for physics-based character animation that use forward dynamic simulations have been a research focus for over two decades, most often with human locomotion as the motion of interest. A survey of the work in this area can be found in [2]. Impressive results were achieved in classical works by Coros et al.[3], Wang et al.[4] and Geijtenbeek et al.[5]. But most of these works focus on controlling physics-based locomotion over flat terrain. Recently there were some very interesting attempts to use Deep RL for locomotion on complex terrain, for example [6], but only in 2D at the moment. Another interesting approach using biologically inspired idea that motor systems are hierarchical was suggested in [1].

3 Methods

In our work we use Deep Deterministic Policy Gradient (DDPG1) algorithm and based on the observations received the actor sends activation signals to the muscles. First we do a brief overview of reinforcement learning and it's current state.

3.1 Overview

We consider a standard reinforcement learning setup consisting of an agent interacting with an environment E in discrete timesteps. At each timestep t the agent receives an observation x_t, takes an action a_t and receives a scalar reward r_t. In all the environments considered here the actions are real-valued $a_t \in \mathbb{R}^N$. In general, the environment may be partially observed so that the entire history of the observation, action pairs $s_t = (x_1, a_1, ..., a_{t-1}, x_t)$ may be required to describe the state. Here, we assumed the environment is fully-observed so $s_t = x_t$.

An agent's behavior is defined by a policy, π, which maps states to a probability distribution over the actions $\pi\colon \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The environment, E, may also be stochastic. We model it as a Markov decision process with a state space $\mathcal{S}$, action space $\mathcal{A} = \mathbb{R}^N$, an initial state distribution $p(s_1)$, transition dynamics $p(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$.

The return from a state is defined as the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$ with a discounting factor $\gamma \in [0, 1]$. Note that the return depends on the actions chosen, and therefore on the policy π, and may be stochastic. The goal in reinforcement learning is to learn a policy which maximizes the expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}\left[R_1\right]$. We denote the discounted state visitation distribution for a policy π as ρ^π.

The action-value function is used in many reinforcement learning algorithms. It describes the expected return after taking an action a_t in state s_t and thereafter following policy π: $Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi}\left[R_t | s_t, a_t\right]$. Many approaches in reinforcement learning make use of the recursive relationship known as the Bellman equation: $Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma\, \mathbb{E}_{a_{t+1} \sim \pi}\left[Q^\pi(s_{t+1}, a_{t+1})\right]\right]$

If the target policy is deterministic we can describe it as a function $\mu : \mathcal{S} \leftarrow \mathcal{A}$ and avoid the inner expectation:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))\right] \tag{1}$$

The expectation depends only on the environment. This means that it is possible to learn Q^μ off-policy, using transitions which are generated from a different stochastic behavior policy β.

Q-learning [7], a commonly used off-policy algorithm, uses the greedy policy $\mu(s) = \arg\max_a Q(s, a)$. We consider function approximators parameterized by θ^Q, which we optimize by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right] \tag{2}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q). \tag{3}$$

While y_t is also dependent on θ^Q, this is typically ignored.

The use of large, non-linear function approximators for learning value or action-value functions has often been avoided in the past since theoretical performance guarantees are impossible, and practically learning tends to be unstable. Recently, [8, 9] adapted the Q-learning algorithm in order to make effective use of large neural networks as function approximators. Their algorithm was able to learn to play Atari games from pixels. In order to scale Q-learning they introduced two major changes: the use of a *replay buffer*, and a separate *target network* for calculating y_t. These are employed in the context of DDPG and will be described in the next section.

3.2 Algorithm

It is impossible to straightforwardly apply Q-learning to continuous action spaces: it will require an optimization of a_t at every timestep; this optimization is too slow with large, unconstrained function

Algorithm 1 DDPG

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$
        Update the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}; \qquad \theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
end for
```

approximators and nontrivial action spaces. Instead, an actor-critic approach based on the DPG algorithm[10] is used.

The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s, a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t} \right] \end{aligned} \tag{4}$$

In [10] it is proved that this is the *policy gradient*, the gradient of the policy's performance (In practice we ignore the discount in the state-visitation distribution ρ^β.).

Introducing non-linear function approximators means that convergence is no longer guaranteed. However, such approximators appear essential in order to learn and generalize on large state spaces. Also most optimization algorithms assume that the samples are independently and identically distributed. Obviously, when the samples are generated from exploring sequentially in an environment this assumption no longer holds. Additionally, to make efficient use of hardware optimizations, it is essential to learn in minibatches, rather than online.

A replay buffer is used to address these issues, which is a finite sized cache $\mathcal{R}$. Transitions were sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s_{t+1}) was stored in the replay buffer. When the replay buffer was full the oldest samples were discarded. At each timestep the actor and critic are updated by sampling a minibatch uniformly from the buffer. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

Directly implementing Q learning (equation 2) with neural networks proved to be unstable in many environments. Since the network $Q(s, a|\theta^Q)$ being updated is also used in calculating the target value (equation 3), the Q update is prone to divergence. The solution is similar to the target network used in [8] but modified for actor-critic and using "soft" target updates, rather than directly copying the weights. We create a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, that are used for calculating the target values. The weights of these target networks are then updated by having them slowly track the learned networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. So the target values are constrained to change slowly, greatly improving the stability of learning. This simple change moves the relatively unstable problem of learning the action-value function closer to the case of supervised learning, a problem for which robust solutions exist. Having both a target μ' and Q' was required to have stable targets y_i in order to consistently train the critic without divergence.This may slow learning, however, in practice it was found this greatly improves the stability of learning.

When learning from low dimensional feature vector observations, the different components of the observation may have different physical units (for example, positions versus velocities) and the ranges may vary across environments. This can make it difficult for the network to learn effectively and may make it difficult to find hyper-parameters which generalise across environments with different scales of state values. In DDPG a recent technique from deep learning called *batch normalization*[11] is adapted. This technique normalizes each dimension across the samples in a minibatch to have unit mean and variance. In addition, it maintains a running average of the mean and variance to use for normalization during testing (in our case, during exploration or evaluation). In deep networks, it is used to minimize covariance shift during training, by ensuring that each layer receives whitened input.

To improve continuous action space exploration for this off-policies algorithm we can treat the problem of exploration independently from the learning algorithm. Exploration policy μ' is constructed by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy: $\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$. $\mathcal{N}$ can be chosen to chosen to suit the environment. Ornstein—Uhlenbeck process [12] is used to generate temporally correlated exploration for exploration efficiency in physical control problems with inertia (similar use of autocorrelated noise was introduced in [13]).

4 Results/Analysis

Most of our hyperparameters search experiments were done in Arm environment due to it's simplicity and low number degrees of freedom number. It results in faster convergence to the good solution and shorter simulation time for each iteration. We tested different variants of $\gamma = 0.5, 0.9, 0.99, 0.999$; actors and critics of different NN architectures, different θ and σ values for the Ornstein–Uhlenbeck process: Figures 1, 2, 3 and 4.

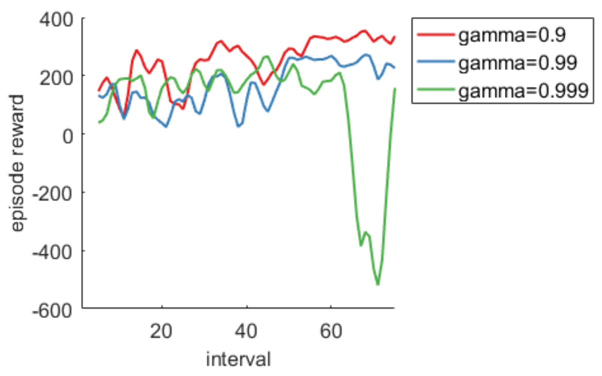

Figure 1: Reward dependance on gamma

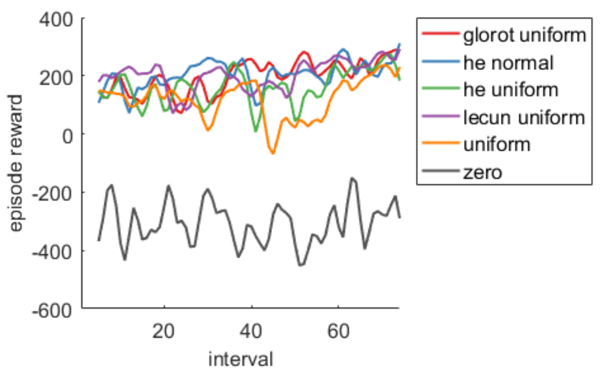

Figure 2: Reward dependance on initial weight distribution

Best parameters found: we used Nadam[14] and [15] for learning the neural network parameters with a learning rate of $1.5 \cdot 10^{-3}$ and 10^{-3} for the actor and critic respectively. For Q we used a discount factor of $\gamma = 0.99$. For the soft target updates - $\tau = 0.001$. The neural networks used the rectified non-linearity [16] for all hidden layers. The final output layer of the actor was a *sigmoid* layer, to bound the actions. The actor networks had 3 hidden layers with 24 units for Arm and 32 for Human respectively. Actions were not included until the 2nd hidden layer of Q.The critic networks had 3 hidden layers with 48 units for Arm and 64 for Human. Layers weights of both the actor and critic were initialized from a uniform LeCun distribution. We trained with minibatch size of 32.

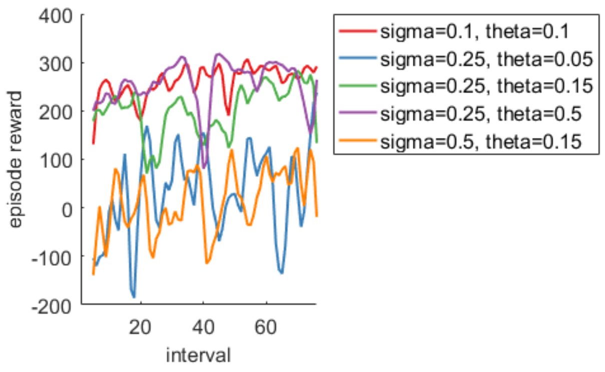

Figure 3: Reward dependance on sigma and theta

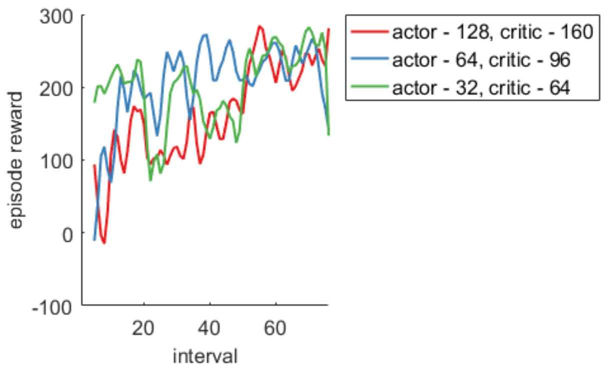

Figure 4: Reward dependance on actor's and critic's network sizes

For the exploration noise process we used temporally correlated noise in order to explore well in physical environments that have momentum. We used an Ornstein-Uhlenbeck process [12] with $\theta = 0.1$ and $\sigma = 0.2$ for Arm and Human standing and $\theta = 0.15$ and $\sigma = 0.25$ for Human walking environments. The Ornstein–Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around 0. At that stage we tried to made rewards functions as simple as possible, for example:

$$R_{arm}(s) = 1 - \text{angular\_dist}(\alpha_{cur}, \alpha_{target}) - \text{angular\_dist}(\beta_{cur}, \beta_{target}) \tag{5}$$

$$R_{stand}(s) = 100 - ||a_{cm}||_2^2 - 2||V_{cm}||_2^2 - \frac{100}{N_{out}} \sum_i \left(||a_i||_2^2 + ||b_i||_2^2\right) \tag{6}$$

$$R_{walk}(s) = pos_x \tag{7}$$

, where α_{cur} and α_{target} are current and target angles of shoulder joint; β_{cur} and β_{target} are current and target angles of elbow joint; a_{cm} and V_{cm} are current acceleration and speed of the mass center of the model; N_{out} is the number of muscles we control; a_i and b_i are current angle and angle acceleration of i-th joint; pos_x is the x coordinate of the mass center of the model (positive direction of the X-axis is the direction of movement).

The result of the training of walking model can be seen in videos [17, 18]. Video [19] shows our best trained arm model. Videos [20, 21] show the results of the training of standing model using reward (5) and video [22] with $R^*_{stand}(s) = R_{stand}(s) - 50\alpha$.

5 Future work

We used DDPG algorithm in our current work. It requires fine-tuning and hyperparameters search to get good stability and performance but we achieved good results with Arm and Human environments on standing and walking tasks using surprisingly small neural networks for actor and critic. Our future plans can be divided into two large categories:

- Algorithmic part: use suggested in "Benchmarking Deep Reinforcement Learning for Continuous Control" [23] approach to test and compare different algorithms using standing or walking environments. And:
  - Improve DDPG - implement "Prioritized Experience Replay"[24].
  - Test one of the state of the art on-policy algorithms for example "Trust Region Policy Optimisation" (TRPO[25]) and Continuous DQN (NAF[26])
- Action part:
  - Train models on more complex tasks - walking in the given direction, running, jumping, and switching between different kinds of activities: standing $\leftrightarrow$ walking, standing $\leftrightarrow$ jumping, etc.
  - Add regularization to the reward functions to minimize for example spent energy and make muscle activations more smooth and biologically plausible.

References

[1] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and Transfer of Modulated Locomotor Controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[2] Thomas Geijtenbeek and Nicolas Pronost. Interactive character animation using simulated physics: A state-of-the-art review. In *Computer Graphics Forum*, volume 31, pages 2492–2515. Wiley Online Library, 2012.

[3] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel Van De Panne. Locomotion skills for simulated quadrupeds. In *ACM Transactions on Graphics (TOG)*, volume 30, page 59. ACM, 2011.

[4] Jack M Wang, Samuel R Hamner, Scott L Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM transactions on graphics*, 31(4), 2012.

[5] Thomas Geijtenbeek, Michiel van de Panne, and A Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):206, 2013.

[6] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[7] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] 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.

[10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *ICML*, 2014.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] George E Uhlenbeck and Leonard S Ornstein. On the theory of the Brownian motion. *Physical review*, 36(5):823, 1930.

[13] Pawel Wawrzynski. Control policy with autocorrelated noise in reinforcement learning for robotics. *International Journal of Machine Learning and Computing*, 5(2):91, 2015.

[14] Timothy Dozat. Incorporating Nesterov Momentum into Adam. *http://cs229.stanford.edu/proj2015/054*$_r$*eport.pdf*, 2015.

[15] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hintor. On the importance of initialization and momentum in deep learning. *Proceedings of the 30 th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: WCP volume 28 http://www.jmlr.org/proceedings/papers/v28/sutskever13.pdf*, 2013.

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[17] Viktor Makoviichuk and Peter Lapko. Model learning how to walk, 1st funny step. `https://youtu.be/GOwuLMpj8DY`.

[18] Viktor Makoviichuk and Peter Lapko. Model learning how to walk, 1st large step. `https://youtu.be/nNM7QhQ2mhs`.

[19] Viktor Makoviichuk and Peter Lapko. Arm video. `https://youtu.be/1R6UjBZPzBE`.

[20] Viktor Makoviichuk and Peter Lapko. Standing model after 500k iterations. `https://youtu.be/7e-0aRX0cM0`.

[21] Viktor Makoviichuk and Peter Lapko. Standing model after 750k iterations, balancing on 1 leg. `https://youtu.be/eHXvRjbh1vY`.

[22] Viktor Makoviichuk and Peter Lapko. Robust standing model. `https://youtu.be/eNIC8Jgnt6k`.

[23] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *arXiv preprint arXiv:1604.06778*, 2016.

[24] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952*, 2015.

[25] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

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