Muscle Excitation Estimation in Biomechanical Simulation Using NAF Reinforcement Learning

Amir H. Abdi, Pranit Saha, Venkata Praneeth Srunagarapu, and Sidney Fels

Electrical and Computer Engineering Department, University of British Columbia, Vancouver, Canada
amirabdi@ece.ubc.ca

Abstract. Motor control is a set of time-varying muscle excitations which generate desired motions for a biomechanical system. Muscle excitations cannot be directly measured from live subjects. An alternative approach is to estimate muscle activations using inverse motion-driven simulation. In this article, we propose a deep reinforcement learning method to estimate the muscle excitations in simulated biomechanical systems. Here, we introduce a custom-made reward function which incentivizes faster point-to-point tracking of target motion. Moreover, we deploy two new techniques, namely, episode-based hard update and dual buffer experience replay, to avoid feedback training loops. The proposed method is tested in four simulated 2D and 3D environments with 6 to 24 axial muscles. The results show that the models were able to learn muscle excitations for given motions after nearly 100,000 simulated steps. Moreover, the root mean square error in point-to-point reaching of the target across experiments was less than 1% of the length of the domain of motion. Our reinforcement learning method is far from the conventional dynamic approaches as the muscle control is derived functionally by a set of distributed neurons. This can open paths for neural activity interpretation of this phenomenon.

Keywords: Deep reinforcement learning, Muscle excitation, Inverse motion-driven simulation, Normalized advantage function, Deep Q-network

1 Introduction

Biomechanical modeling provides a powerful tool for analyzing structure and function of human anatomy, thereby establishing a scientific basis for treatment planning. Modeling is particularly indispensable in cases when mechanical variables are hard or impossible to measure with the available technologies [10]. One of the unknown variables in understanding a musculoskeletal system is the muscle excitation trajectory. Muscle excitations reflect underlying neural control processes; they form a connection between the causal neural activities and the resultant observed motion. Researchers have proposed various techniques, including forward-dynamics tracking simulation, inverse dynamics-based static optimization, and optimal control strategies, to predict muscle excitations and forces [4].
However, the muscle redundancy problem makes the task far more challenging. To address this problem, many approaches have been investigated including minimization of motion tracking errors, squared muscle forces and combined muscle stress [4]. Unfortunately, the performance errors, high computational costs, and their sensitivity to optimal criteria and extensive regularizations have directed many researchers to seek alternative strategies [7,12].

Deep Reinforcement Learning (RL) is a popular area of machine learning that combines RL with deep neural networks to achieve higher levels of performance on decision-making problems including games, robotics and health-care [9]. In these approaches, an agent interacts with the environment and makes intelligent decisions (actions) based on value functions $V(s)$, action-value functions $Q(s,a)$, policies $\pi(s,a)$, or learned dynamics models.

Recent studies in deep RL have pushed the boundaries from discrete to continuous action spaces [5], extending its possibilities for complex biomechanical control applications. Although most researchers encode the locomotion in terms of joint angles for robotics applications, some progress has been made in the muscle-driven RL-based motion synthesis. Izawa et al. introduced an actor-critic RL algorithm with subsequent prioritization of control action space to estimate the motor command of a biological arm model [6]. Broad et al. explored a receding horizon differential dynamic programming algorithm for arm dynamics optimization through muscle control policy to achieve desired trajectories in OpenSim [2]. These approaches require an a priori information such as the relative action preferences or ranking systems which are rarely known in biomechanical systems. Quite recently, in the non-RL domain, deep learning approaches are also explored to predict muscle excitations for point reaching movements, which rely on training data provided by inverse dynamics methods [1,7].

This work is aimed at developing an improved understanding of the effective coordination of muscle excitations to generate movements in musculoskeletal systems. It borrows ideas from and extends on a continuous variant of the deep Q-network (DQN) algorithm known as Normalized Advantage Function (NAF-DQN). This research introduces three contributions: a customized reward function, the episode-based hard update of the target model in double Q-learning, and the dual buffer experience replay. It also takes advantage of a reduced-slope logistic function to estimate muscle excitations. The proposed approach is agnostic to the dynamics of the environment and is tested for systems with up to 24 degrees of freedom. Due to its independence from training data, the trained models work as expected in unseen scenarios.

## 2 Background

The goal of reinforcement learning is to find a policy $\pi(a|s)$ which maximizes the expected sum of returns based on a reward function $r(s,a)$, where $a$ is the action taken in state $s$. During training, at each time step $t$, the agent takes the
action $a_t$ and arrives at state $s_{t+1}$ and is rewarded with $R_t$, formulated as

$$R_t = \sum_{i=t} \gamma^{i-t} r(s_i, a_i),$$

(1)

where $\gamma < 1$ is a discount factor that reduces the value of future rewards.

In physical and biomechanical environments, system dynamics are not known and a model of the environment, $p(s_{t+1}|s_t, a_t)$, cannot be directly learned. Therefore, a less direct model-free off-policy learning approach is more beneficial. Q-learning is an off-policy algorithm that learns a greedy deterministic policy based on the action-value function, $Q^\pi(s, a)$, referred to as the Q-function. Q-function determines how valuable the $(s, a)$ tuple is under the policy $\pi$. Q-learning is originally designed for discrete action spaces and chooses the best action ($\mu$) as

$$\mu(s_t) = \operatorname{argmax}_a Q(s_t, a_t).$$

(2)

Deep Q-network (DQN) is an extension of Q-learning that uses a parameterized value-action function, $\theta^Q$, determined by a deep neural network. In DQN, the objective is to minimize the Bellman error function

$$L(\theta^Q) = \mathbb{E}[(Q(s_t, a_t|\theta^Q) - y_t)^2]$$

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})),$$

(3)

where $y_t$ is the observed discounted reward provided by the environment [13].

To enable DQN in continuous action spaces, normalized advantage function (NAF) was introduced by Gu et al. [5], where the Q-function is represented as the sum of two parameterized functions, namely the value function, $V$, and the advantage function, $A$, as

$$Q(s, a|\theta^Q) = V(s|\theta^V) + A(s, a).$$

(4)

Value function describes a state and is simply the expected total future rewards of a state. The Q function, on the other hand, describes an action in a state and explains how good it is to choose action $a$ at state $s$. Therefore, the advantage function, defined as $Q(.) - V(.)$, is a notion of the relative importance of each action. NAF is defined as follows

$$A(s, a) = -\frac{1}{2} (a - \mu(s|\theta^\mu))^T P(s|\theta^P) (a - \mu(s|\theta^\mu)),$$

$$P(s|\theta^P) = L(s|\theta^P) L(s|\theta^P)^T,$$

(5)

where $\mu$ is a parametric function determined by the neural network $\theta^\mu$; and $L$ is a lower-triangular matrix filled by the neural network $\theta^L$ with the diagonal terms squared; consequently, $P$ is a positive-definite square matrix [5]. While there are numerous ways to define an advantage function, the restricted parametric formulation of NAF ensures that the action that maximizes $Q^\pi$ is always given by $\mu(s|\theta^\mu)$. 
Fig. 1. The Normalized Advantage Function algorithm and architectures of the three neural networks, namely $\theta^\mu$, $\theta^V$, and $\theta^L$ (Eq. 3-5). The arrows demonstrate the data flow in the feedforward path.

### 3 Materials and Method

#### 3.1 Model Architecture

The deep dueling architecture used in this study is depicted in Fig. 1. The $\theta^V$ and $\theta^V'$ neural networks receive the system state as input and estimate the value functions $V(s)$ and $V'(s)$. This design follows the double Q-learning approach [14] to separate the action selector and evaluator operators. During training, $\theta^V'$ receives the next state ($s_{t+1}$) to calculate the observed discounted total reward ($y_t$), while $\theta^V$ receives the current state ($s_t$) to predict the reward.

Here, we propose the episode-based hard update technique, where the weights of the $\theta^V'$ network are only updated at the end of each episode by the weights
Fig. 2. The four simulation environments: (a) 6-muscle 2D, (b) 14-muscle 2D, (c) 24-muscle 2D, (d) 8-muscle 3D. Notice the green and blue particles which visualize the position of the target (T) and the point mass (P).

of $\theta^V$. This technique helps separate the exploration and training processes and mitigates the risk of getting stuck in a positive feedback loop.

The $\theta^P$ network receives the current state and generates $([a]^2+|a|)/2$ values to fill the lower triangular matrix of $L$, which, in turn, is squared to create matrix $P$ (Eq. 5). The $\theta^\mu$ network receives the current state, and estimates the excitations of the $|a|$ muscles as a value between 0 and 1. We deploy a reduced-slope logistic function, $f(x) = 1/(1 + e^{-mx})$, in the final layer of the action selector $\theta^\mu$, where $0 < m < 1$ defines the slope. This reduction in steepness mitigates the variance of muscle activations and results in a smoother muscle control.

### 3.2 Methods

Training is composed of disjoint episodes. Each episode starts with a target point, $T$, randomly positioned in the motion space of the point mass, $P$. Motion space (domain) is defined as the entire area (volume) that the point mass can traverse in the simulated environment with muscle activations. An episode ends either by the agent reaching the terminal state, i.e., the point mass reaching the target, or going over a maximum number of steps per episode.

The size of the action space, $|a|$, is equal to the number of muscles. The state space is the spatial position of the target $(T_x, T_y, T_z)$, where $T_y$ is constantly zero in the 2D simulations. The position of the point mass (P) is excluded from the state formulation to imitate a real-world biomechanical system where the exact position of each joint is not known.

**Reward Function.** The goal of the agent is to set the muscle excitations so that $P$ reaches $T$. To incentivize this, two factors were encoded in the reward function: distance and time,

$$r(s_t, a_t) = \begin{cases} -1 & |P_{t+1} - T| \geq |P_t - T| \\ 1/t & |P_{t+1} - T| < |P_t - T|, \\ \omega & |P_{t+1} - T| \leq d_{\text{thres}} \end{cases}$$

(6)
where $\omega > 0$ is a constant value rewarded in a successful terminal state, and $|P - T|$ is the Euclidean distance between $P$ and $T$. Terminal state is defined as a distance of less than $d_{\text{thres}}$ between $P$ and $T$. The agent is penalized if $|P - T|$ is not lessened as a result of the action. Moreover, the rewards for correct decisions are reduced by a time factor to incentivize fast direct reach of the target, as opposed to curved trajectories.

**Action Exploration.** To enable action exploration in the continuous action domain, outputs of the $\theta^\mu$ network, i.e. the muscle excitations, were augmented with a zero-mean stationary Gaussian-Markov stochastic process, known as the Ornstein Uhlenbeck. The stochastic process was initialized with a variance of 0.35, which was annealed as a function of $t$ down to 0.05.

**Dual Buffer Experience Replay.** We borrow the idea of experience replay from the works of Mnih et al. for higher data efficiency and training on non-consecutive samples [9]. In this technique, a replay buffer stores seen samples as $(s_t, a_t, r_t, s_{t+1})$ tuples. Samples are then randomly selected for training from the buffer. In order to avoid feedback loops during an episode of training, we propose the dual buffer experience replay strategy, where the episode buffer stores samples of the current episode while the training samples are chosen from the back buffer. At the end of each episode, contents of the episode buffer are copied into the back buffer. This strategy is mostly important in the beginning of the training when the replay buffer is fairly sparse.

**Training Hyper-parameters.** If the success threshold radius, $d_{\text{thres}}$, is set too high, the agent is given the success reward too early and effortless; thus, will not learn the exact activation patterns. On the contrary, a small $d_{\text{thres}}$ value makes it impossible for the point mass to reach the success state which delays training. Based on the above intuition, $d_{\text{thres}}$ was set so that less than 1% of the motion domain of the point mass is defined as the success state. The learning rate ($\alpha$) and the reward discount factor ($\gamma$) were set to 0.01 and 0.99, respectively. Each episode of training was constrained to 200 steps at the end of which the episode would restart and the state would reset to a new random position.

### 4 Experiments and Results

An open-source biomechanical simulator, ArtiSynth, was used to design the simulation environments and run the experiments [8]. The keras-rl library with the TensorFlow backend was used as the basis for the implementations of the methods [3,11]. A network interface was used to send the target positions from the simulated mechanical environment in ArtiSynth to the deep learning model and receive the new muscle activations from the model. Our implementation, consisting of the ArtiSynth model in Java and the deep RL in Python, has been made available at [https://github.com/amir-abdi/artisynth_point2point](https://github.com/amir-abdi/artisynth_point2point). The forked keras-rl repository with added functionalities proposed in this paper can be accessed from [https://github.com/amir-abdi/keras-rl](https://github.com/amir-abdi/keras-rl).
Four biomechanical environments were designed to test the feasibility and accuracy of the proposed method, including 2D environments with 6, 14, and 24 muscles, and a 3D environment with 8 muscles. Muscles were set to have a maximum active isometric force of 1 N, optimal length of 1 cm, and maximum passive force of 0.1 N, with 50% flexibility for lengthening and shortening of fibers. The damping coefficient was set to 0.1. The hyper-parameters of learning were not altered in between experiments for the results to be comparable.

In 2D environments, one end of each muscle was attached to the point mass, while the other stationary end was positioned on the circumference of a circle of radius 10 cm, as shown in Fig. 2. In the 3D environment, the stationary ends of muscles were positioned at the 8 corners of a cuboid of length 20 cm. When the muscle excitations are zero, implying that the axial muscle fibers are at rest, the point mass will move to the very center of the circle. As the muscles get activated, the point mass moves towards the direction of the net force. No other external force was applied to the point mass.

4.1 Results

Fig. 3 demonstrates the weighted exponential smoothed curves for the total reward value per episode and the distance between the point mass, $P$, and the target, $T$, at the end of each episode as a function of the number of steps. The agents were tested with 500 episodes of random point-to-point reaching tasks, and the Euclidean distance between the final position of the point mass and the target was evaluated. The root mean squared error (RMSE) of the trained agents were 1.8 mm, 1.5 mm, 1.7 mm, and 2.4 mm for the 6-muscle 2D, 14-muscle 2D, 24 muscle 2D, and 8-muscle 3D environments, respectively. The average RMSE across all environments was 1.8 mm.

The learned models were also tested in unseen scenarios where the target point was moved out of the motion domain of the point mass. The surprising result was that the point mass followed the target, to the extent allowed by the model constraints, to minimize the distance between the two.
5 Discussion and Conclusion

In this article, a general reinforcement learning method was introduced to estimate muscle excitations for a given trajectory in a muscle-driven biomechanical simulation. The results assert that the approach is applicable to various degrees of freedom and muscle arrangements. To make the method environment invariant, the agent was kept uninformed of the position of its associated point mass. Moreover, the agent is unaware of the distribution of the muscles, their arrangement, their mechanical properties, and their states. Therefore, it only receives the location of the target point and the rewards in response to its actions.

Deep reinforcement learning models are quite sensitive to hyper-parameters and smaller neural networks have a higher chance of convergence. In our experiments, neural networks with more than 2 hidden layers for $\theta^\mu$ and $\theta^V$ did not converge.

As depicted in Fig. 3, the agent learned to reach the target in all environments irrespective of the number of muscles and their configurations. However, a positive correlation was observed between the training time and the degrees of freedom of the biomechanical system, i.e., the number of muscles. The RMSE values indicate that trained agents managed to reach their target locations with a distance of less than half the designated $d_{thres}$. In other words, the point mass has reached its target with less than 1% distance with respect to its length of the domain of motion. Interestingly, with no further fine tuning of the parameters, the performance of the method remained intact in higher degrees of freedom.

The results show that there exists a positive correlation between the $d_{thres}$ value and the final RMSE of the trained model. However, setting $d_{thres}$ to smaller values increases the chance of unsuccessful episodes which delays convergence. Therefore, the authors suspect that gradually decaying this value, upon network convergence, can reduce the final RMSE of the model.

The proposed reinforcement learning method does not require any labeled data for training, as opposed to other approaches where known optimal control trajectories were used as training data [1,7]. This highlights another important finding of the current study that the trained models were functional when tested in unseen scenarios where the target point was moved out of the motion domain of the point mass. Since the model had learned the optimal muscle control independent of any training data, it was able to estimate the correct muscle excitations and move the point mass to minimize its distance to the target.

The proposed reinforcement learning approach is different from the conventional inverse dynamics methods in the sense that the muscle controls are derived parametrically from a set of distributed neurons of the neural network, i.e., $\theta^\mu$. Such approach opens the path for neural activity interpretation of the muscle control.

References

1. Berniker, M., Kording, K.P.: Deep networks for motor control functions. Frontiers in Computational Neuroscience 9 (2015)
2. Broad, A.: Generating Muscle Driven Arm Movements Using Reinforcement Learning. Master’s thesis, Washington University in St. Louis (2011)
3. Chollet, F., et al.: Keras. https://github.com/keras-team/keras (2015)
4. Erdemir, A., et al.: Model-based estimation of muscle forces exerted during movements. Clinical biomechanics 22(2), 131–154 (2007)
5. Gu, S., et al.: Continuous Deep Q-Learning with Model-based Acceleration. Cognitive Processing 12(4), 319–340 (2016)
6. Izawa, J., et al.: Biological arm motion through reinforcement learning. Biological cybernetics 91(1), 10–22 (2004)
7. Khan, N., Stavness, I.: Prediction of muscle activations for reaching movements using deep neural networks. 41st Annual Meeting of the American Society of Biomechanics, Boulder (2017)
8. Lloyd, J., et al.: ArtiSynth: A Fast Interactive Biomechanical Modeling Toolkit Combining Multibody and Finite Element Simulation. In: Payan, Y. (ed.) Soft Tissue Biomechanical Modeling for Computer Assisted Surgery, vol. 11, chap. 126, pp. 355–394. Springer Berlin Heidelberg (2012)
9. Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
10. Pileicikiene, G., et al.: A three-dimensional model of the human masticatory system, including the mandible, the dentition and the temporomandibular joints. Stomatologija, Baltic Dental and Maxillofacial Journal 9(1), 27–32 (2007)
11. Plappert, M.: keras-rl. https://github.com/matthiasmplappert/keras-rl (2016)
12. Ravera, E.P., et al.: Estimation of muscle forces in gait using a simulation of the electromyographic activity and numerical optimization. Computer methods in biomechanics and biomedical engineering 19(1), 1–12 (2016)
13. Sutton, R.S., et al.: Reinforcement Learning: An Introduction. MIT Press (1998)
14. Van Hasselt, H., et al.: Deep reinforcement learning with double $q$-learning. In: Thirtieth AAAI Conference on Artificial Intelligence. vol. 16, pp. 2094–2100 (2016)