NIPS 2017 CHALLENGE
LEARNING TO RUN

Arka Sadhu (140070011), Siddhant Garg(14D070027)

Abstract

The aim of the project is to train a physics based simulator of the human legs to run on a simulated environment with obstacles. We have explored how successful various reinforcement learning methods work on the this problem (NIPS 2017 Learning to Run Challenge). We have been able to train a model which allows the simulator to achieve a reward very close to that obtained by the top performers in the challenge, but is able to walk further than some of them.

Introduction

The goal is to develop a controller to enable a physiologically-based human model to navigate a complex obstacle course (obstacles include external obstacles like steps, or a slippery floor, along with internal obstacles like muscle weakness or motor noise) as quickly as possible. The performance score is based on the distance traveled through the obstacle course in a set amount of time. The task is to build a function f which takes the current state observation (a 41 dimensional vector) and returns the muscle excitations action (18 dimensional vector) in a way that maximizes the reward (EPFL) (Mohanty 2017).

Environment

The agent is a musculoskeletal model that includes 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. For example, there is a first order differential equation that relates electrical signal the nervous system sends to a muscle (the excitation) to the activation of a muscle (which describes how much force a muscle will actually generate given the muscle's current force-generating capacity). Given the musculoskeletal structure of bones, joint, and muscles, at each step of the simulation (corresponding to 0.01 seconds), the engine:

- computes activations of muscles from the excitations vector provided to the step() function,
- actuates muscles according to these activations,
- computes torques generated due to muscle activations,
- computes forces caused by contacting the ground,
- computes velocities and positions of joints and bodies,
- generates a new state based on forces, velocities, and positions of joints.

Reward

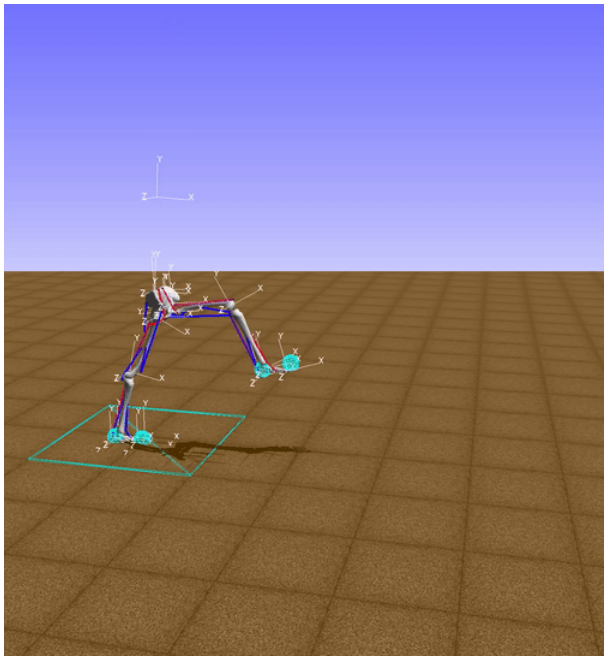
The trial ends either if the pelvis of the model goes below 0.65 meters or if the model reaches 1000 iterations (corresponding to 10 seconds in the virtual environment). The total reward is the position of the pelvis on the x axis after the last iteration minus a penalty for using ligament forces (Ligaments are tissues which prevent your joints from bending too much - overusing these tissues leads to injuries, so we want to avoid it). The penalty in the total reward is equal to the sum of forces generated by ligaments over the trial, divided by 10,000,000.

Observations and Actions

The 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 (6 values)
- angular velocity of each ankle, knee and hip (6 values)
- position of the center of mass (2 values)
- velocity of the center of mass (2 values)
- positions (x, y) of head, pelvis, torso, left and right toes, left and right talus (14 values)
- strength of left and right psoas: 1 for difficulty < 2, otherwise a random normal variable with mean 1 and standard deviation 0.1 fixed for the entire simulation
- next obstacle: x distance from the pelvis, y position of the center w.r.t the the ground, radius.

In each action, there are 18 muscles which are actuated (9 per leg).



Environment Difficulty

A parameter controls the obstacle difficulty in the environment and can be one of the following:

- 0 : corresponding to no obstacles
- 1 : corresponding to 3 randomly positioned obstacles fixed on ground
- 2 : corresponding to 3 randomly positioned obstacles fixed on ground and strength of the psoas muscles (the muscles that help bend the hip joint in the model) varies as $z * 100\%$, where z is a normal variable with the mean 1 and the standard deviation 0.1

The evaluation of the challenge was done on difficulty level 2 of the environment.

Approach

Many existing methods of reinforcement learning have treated tasks in a discrete low dimensional state space. However, controlling humanoid smooth simulators requires a continuous high-dimensional state space. To treat this issue of having an infinitely big state space, we ruled out using value function learning methods like Q-learning, etc. Instead we focused on policy search methods and intended to try out various different approaches. Further to learning in class about policy gradients, we explored utilizing a framework similar to REINFORCE for the case of continuous action spaces and we took into consideration Deterministic Policy Gradient based methods after having read several papers like Continuous control with deep reinforcement learning . We have also explored various other techniques like Trust Region Policy Optimization and Evolutionary strategies.

Deep Deterministic Policy Gradient

A parametrized policy is advantageous for control because it allows for learning in continuous actions spaces (Ramstedt 2016). DDPG (Lillicrap et al. 2015) is an actor-critic policy gradient algorithm involving a policy network $\pi(s|\xi)$ with parameters ξ in addition to the action-value network $Q(s, a|\theta)$ with parameters θ. The training targets for Q are computed as $y_j = r_j + \gamma Q(s_{j+1}, \pi(s_{j+1}|\xi^{'})|\theta^{'})$. Using the mean squared error, we derive the cost function for Q:

$$C_{\theta^{'}\xi^{'}}(\theta) = \frac{1}{m}\sum_{j}(r_j + \gamma Q(s_{j+1}, \pi(s_{j+1}|\xi^{'})|\theta^{'}) - Q(s_j, \pi(s_j|\xi)|\theta))^2$$

As the targets depend on the explicit policy network, we also need the target policy parameters $\xi^{'} = LP(\xi)$. Here, LP will be an exponential moving average with update rule $\xi^{'} \leftarrow \tau\xi + (1\tau)\xi^{'}$ and $\theta^{'} \leftarrow \tau\theta + (1\tau)\theta^{'}$. The policy is trained via policy gradient:$\nabla_{\xi}Q(s_j, \pi(s_j|\xi)|\theta) = \nabla_a Q(s_j, \pi(s_j|\xi)|\theta).\nabla_{\xi}\pi(s_j|\xi)$. That is, Q is the cost function for $\pi : C(\xi|\theta) = -Q(s_j, \pi(s_j|\xi)|\theta)$

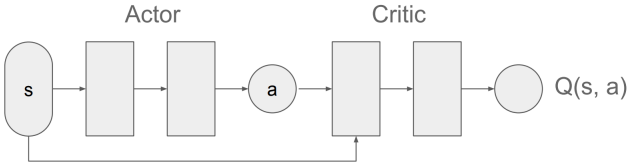


Figure 1: Block diagram actor-critic for DDPG (Hafner)

As actions are continuous, correlated Gaussian noise M_t is added to the actions to ensure exploration. More specifically,$M_{t+1} \leftarrow vM_t + N(0, \sigma))$, where $N(0, \sigma)$ a normal distributed random variable and v a hyper parameter controlling the frequency of the noise. This off-policy approach is possible because the algorithm does not learn on trajectories but only on isolated transitions.

Algorithm for DDPG:

Initialize replay memory D
Initialize π with random weights ξ and target weights $\xi' \leftarrow \xi$
Initialize Q with random weights θ and target weights $\theta' \leftarrow \theta$
for $t = 1$ *to* T **do**
  Select action $a_t = \pi(s_t) + \mathscr{M}_t$
  Execute a_t and observe reward r_t and next state s_{t+1}
  Store transition (s_t, a_t, r_t, s_{t+1}) in D
  Sample random minibatch of m transitions from D
  Set Q targets $y_j = r_j + \gamma Q(s_{j+1}, \pi(s_{j+1}|\xi')\,|\theta')$
  Perform gradient descent on cost $C = \frac{1}{m}\sum_j (y_j - Q(s_j, a_j|\theta))^2$
  Perform gradient ascent on $Q(s_j, \pi(s_j|\xi)\,|\theta)$ with respect to ξ
  Update $\theta' \leftarrow \mathrm{LP}(\theta)$
  Update $\xi' \leftarrow \mathrm{LP}(\xi)$
end

(Plappert et al. 2017) shows that adding noise to the weights of the actor-critic leads to faster convergence. Hence we

have used parameter noise in the model we have implemented.

Manual muscle activation tuning

Our starting attempt at the challenge was to try to make the body move 2 or 3 steps and try to stand in a stable manner. We had initially planned to achieve this and then build up from these muscle activations using evolutionary methods. However, even on repeated trail-error by varying the muscle activations we were unable to make the body move 3 steps. Since muscle activations need to be provided to all 18 muscles on every iteration, to ensure a 3 step movement, we were not able to manually tune the weights and get any significant results.

Evolutionary Strategies

On manually tuning the parameters of the muscle activations to have a base model which walks atleast 2 steps or can stand in a stable manner, we observed that the muscle activations values for each muscle were very close to either 0 or 1 (intermediate values were not helping much). So we decided to try out an evolutionary policy search strategy. We took muscle activations for continuous 10 iterations (i.e, $18 * 10 = 180$ values) and initialized them randomly with 0's and 1's. This weight vector now corresponds to a run on the environment for 10 iterations. We then performed evolutionary search for these vectors using concepts of mutation and crossing over. Initially several random vectors were created and evaluated: a few of the good scoring ones were chosen to cross-over (a new vector made using alternate bits from both). Occasionally, a few vectors were mutated (a random variable for each bit which which causes bits being flipped from 0 to 1 and vice versa). This strategy did not provide very encouraging results, even when we tried to move away from the binary assumption on values. Since the state space is very large and there is a strong dependence on the muscle activations in adjacent iterations if the simulator is walking, evolutionary genetic strategies did not work very well here.

Reward Shaping

The original reward provided by the simulator is the position of the pelvis on the x axis after the last iteration minus a penalty for using ligament forces. We observed that solely using this reward with DDPG does not give any significant improvements. We tried incorporating a new reward into our implementation and we also tested with several rewards available from reference implementations.

Potential Function and Reward Shaping

By definition a potential function $F : S \times A \times S \rightarrow \mathcal{R}$ is such that $\forall s, s' \in S, a \in A$ we have

$$F(s, a, s') = \gamma\phi(s') - \phi(s)$$

We note that the actual reward function (given by the environment) depends on state and action. The two new reward functions that we make are independent of the action and depend only the state and hence the sum of the two new rewards is a potential function. Therefore adding the two reward functions to the original reward (R_0) functions gives a new reward function which we call (R_{new}). Explicitly $R_{new} = R_0 + F$ where F is a potential function. By sufficiency theorem of potential function we claim that if $\pi*$ is an optimal policy of the original MDP then the new MDP which has R_{new} as the reward function also shares $\pi*$ as an optimal policy. Hence it makes sense to change the rewards suitably to get faster convergence as needed.

- To incorporate some observations we made, we modified the reward several times and observed that using the following reward with DDPG gave the best result:
  $A = |x_{pelvis} - x_{head}|$
  $B = \mathbb{1}(y_{head} > 1.2)$
  $C = \frac{x_{t-left} + x_{t-right}}{2}$
  $D = \mathbb{1}(y_{pelvis} > 0.75)$
  $E = \mathbb{1}(y_{t-left} > 0.2) * y_{t-left}$
  $F = \mathbb{1}(y_{t-right} > 0.2) * y_{t-right}$

  $$Reward_2 = A + B + C + D - E - F$$

  x_{pelvis} : x coordinate of pelvis
  y_{pelvis} : y coordinate of pelvis
  x_{head} : x coordinate of head
  x_{head} : y coordinate of head
  x_{t-left} : x coordinate of left talus
  y_{t-left} : y coordinate of left talus
  $x_{t-right}$: x coordinate of right talus
  $y_{t-right}$: y coordinate of right talus

  Here term A accounts for how far the lower body moves with respect to the head of the body and this movement must be rewarded otherwise the body will keep standing vertically. Term B accounts for the head of the body staying upright and not falling below a threshold height, failing which the body becomes unstable and hence a penalty should be imposed. Term C is to ensure that on an average the feet of the body are moving forward and hence the average x-coordinate of the two talus (lower feet) is added as reward. Term D accounts for the pelvis(hip joint) height and not falling below a threshold height, failing which the body becomes unstable and hence a penalty should be imposed. To ensure steps taken are stable, we needed to ensure that once the feet was lifted off ground, some term penalizes the height of the foot and forces it to go back to the ground. Term E and F account for the fact that they penalize the height of the left and right talus above the ground only when they have been lifted off-ground (i.e, more than a certain threshold) respectively.

- The obstacle is in the shape of a sphere with its x-coordinate, y-coordinate and radius being observed in the observations. To incorporate obstacle specific reward into the model from some references, we used the following reward.
  $x_{obs-start} = x_{obstacle} - r_{ostacle}$
  $x_{obs-end} = x_{obstacle} + r_{ostacle}$
  $y_{obs-end} = y_{obstacle} + r_{ostacle}$

$A = \mathbb{1}(x_{t-left} > x_{obs-start} - \frac{r_{ostacle}}{2})$
$B = \mathbb{1}(x_{t-left} < x_{obs-end} + \frac{r_{ostacle}}{2})$
$C = \mathbb{1}(y_{t-left} < y_{obs-end})$

$$Reward = \begin{cases} -0.5 & A \cap B \cap C \\ 0 & otherwise \end{cases}$$

Above is repeated with right talus, left toe and right toe and all these 4 rewards add together to constitute $Reward_3$. The idea behind this reward is that when the body is very close to the obstacle i.e, the x co-ordinate of the talus or the toe is withing a radius distance of the start and end of the obstacle and the y coordinate of the talus or toe is within the height of the obstacle, then we need to penalize this configuration and hence have used a penalty of -0.5 for this. Accumulating this reward for left and right talus and toes, we get a good penalty when the lower body becomes very close to the obstacle.

The final reward that was used to train the DDPG model was $Reward = Reward_1 + Reward_2 + Reward_3$ where $Reward_1$ is the defaut reward for the challenge

Layer Normalization

With layer normalization (Lei Ba, Kiros, and Hinton 2016) (Mohandas), the mean and variance is computed using all of the summed inputs to the neurons in a layer for every single training case. This removes the dependency on a mini-batch size. Unlike batch normalization, the normalization operation for layer norm is same for training and inference. Layer norm acts on a per layer per sample basis, where the mean and variance are calculated for a specific layer for a specific training point. We use layer norm because we do not want the inputs to saturate the non-linearities at the extremes. Layer norm is given by the operation below, where ϵ is a small random noise (for stability). When we apply layer norm on a layer, we are restricting the inputs to follow a normal distribution, which ultimately will restrict the nets ability to learn. In order to fix this, we multiply by a scale parameter (α) and add a shift parameter (β). Both of these parameters are trainable.

$$LN = \alpha \otimes \frac{(x_i - \mu_L)}{\sqrt{\sigma_L^2 + \epsilon}} + \beta$$

where μ_L and σ_L are the mean and variance calculated for a specific layer for a specific training point.

Trust Region Policy Optimization (TRPO)

We tried to learn using TRPO (Schulman et al. 2015) but unfortunately perhaps due to lesser iterations or some other bug in the code, or its non-applicability to the new reward function, we were unable to get the agent being able to move more than two steps. In the literature, many others have been able to get quite large rewards especially using ctmarko's Distributed TRPO (Yongliang 2017)

Implementation Details

- Manual Parameter Tuning: Implemented in python: several attempts at manually controlling the body by explicitly providing the 18 muscle activation weights. We tried several approaches where for first few iterations, a particular activations are provided, then after that they are changed to some other value in order to lift one of the legs and move forwards.
- Genetic Evolutionary Strategy: Implemented a class in python importing the opensim-rl simulator environment which has a data structure to store muscle activation weights of all muscles for 10 iterations (i.e, 180 values) and 2 functions for mutate and cross-over as described in the theoretical section above in the report. Several different populations(with different initializations of the data structure are run) and the best ones are propagated to the next iteration with mutations and cross-overs.
- Trust Region Policy Optimization : Taking reference from the code available at (Coady 2017) , we ran the implementation on the environment. However, it did not give significant results and due to lack of time were not able to perform reward shaping and other techniques to improve the score with this algorithm.
- Deep Deterministic Policy Gradient: We have used implementation of DDPG algorithm from DDPG keras-RL (Plappert 2016) implementation as the starting point and then modified the code as per the simulator environment by adding parameter weight noise, performing reward shaping, etc. We also referred to an implementation of the challenge available in Theano (Pavlov 2017) where we experimented with the parameter and policy noise and doing reward reshaping according to our problem. Both the codes have been provided in the submission.
  The actor critic network that we have used is as follows: We concatenate the states and actions and feed them to the actor module. The actor consists of 2 fully connected dense layers each of size 64 units and each layer is followed by performing layer normalization on it and then adding some non-linearity by using a non-linearity layer imported from lasagne. The outputs from the actor module are fed to the critic module which consists of 2 fully connected dense layers first one of size 64 and second one of size 32 units and each layer is followed by performing layer normalization on it and then adding some non-linearity by using a non-linearity layer imported from lasagne. The ouputs from actor to critic are also fed by a full dense layer between them.

Results

On training the model by the DDPG algorithm we observed that the score of the simulator increased on an average across learning time, unlike the case of TRPO and evolutionary strategies. As a result we ran the DDPG with layer normalization and random noise weight perturbation to train for 400,000 iterations to train. We did not have access to any GPU, and so the above training took around 4 days on a 8GB RAM CPU. The trend from the training of the model was observed and is plotted below in Figure:2 which shows the variation of the rewards obtained with respect to training time. As can be observed in an average the original reward obtained is between 6-8 with some rewards going as high as 30. The reason of this large variations and spikes in the

graph is the fact that the training is being done with difficulty level=2 where on each iteration, the muscle strength of the psoas is set to a normal random variable (to account for difference in muscle strengths for different individuals) and hence depending on what value this variable takes, the model can sometimes perform very well or worse. However some of the best scores attained by the model have been in the range of those of the leader board. One observation to notice here is that, even for a score of 6 or 7, the body takes a large number of steps (The original score contains a penalty on account of high activations of ligaments). The task we were aiming to achieve was to ensure that the body is able to run continuously and keep on taking steps (and hence we added our own rewards to the default reward) rather than aiming to maximize the default reward. Thus we have been able to accomplish the task of making the body simulator run. We can work in the future to ensure less ligament wear and tear happens by now trying to maximize the original reward starting from this trained model and the default reward.

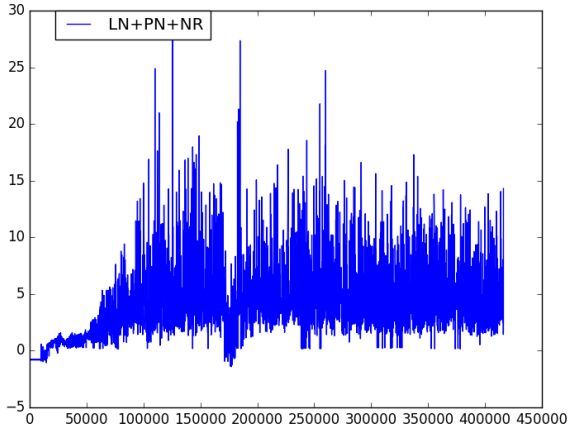


Figure 2: Plot of original reward function vs training iterations for DDPG with layer normalization, noise perturbed weights and modified reward function

Now to evaluate the significance of using the layer normalization and a modified reward function, we tried modifying the model and training under the same conditions. We compare the original reward vs training iterations graphs for 3 models in Figure:3. The blue line represents the best working model with layer normalization, noise perturbed weights and a modified reward function. The green line is for a DDPG model similar to the model of the blue line but with no layer normalization. The red line is for a DDPG model which has layer normalization and noise perturbed weights but only the default reward.
As can be observed from the graph, using layer normalization has proven to be very useful and the lack of it is the primary reason our agents were not working well initially. Also using a modified reward helps the agent improve the default reward as well as can be observed by comparing the

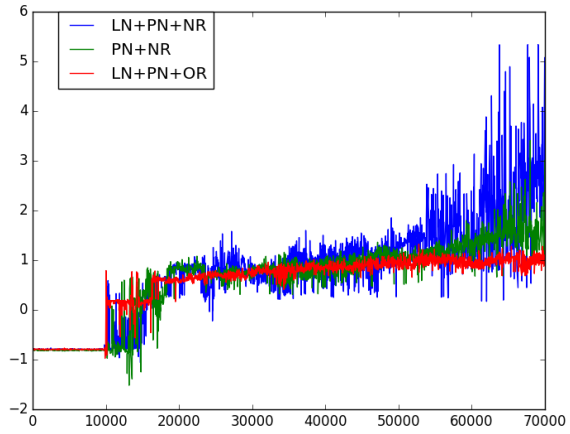


Figure 3: Plot of original reward function vs training iterations for 3 DDPG models: one with layer normalization, noise perturbed weights and modified reward function, one with noise perturbed weights and modified reward function, and one with layer normalization, noise perturbed weights and default reward function

blue and red lines of the graphs. Because of these trends in the graph, we decided to use a DDPG model with layer normalization, noise perturbed weights and a modified reward function as stated in theory above.

The below Figure:4 also plots the modified reward function vs training iterations to compare the 2 DDPG models with respect to importance of layer normalization. As can be observed from the graphs, the model with layer normalization has better modified reward function learning with the training iterations.

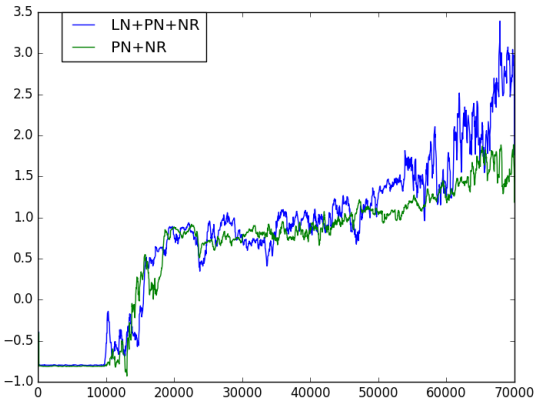


Figure 4: Plot of modified reward function vs training iterations for 2 DDPG models: one with layer normalization and the other with no layer normalization

The above is a series of snapshots of our agent running on the simulator. As can be observed easily the agent is able to walk and cover a fairly large distance during simulation. This simulation is for different time instances of the same episode of running the agent using the weights learned after 400,000 iterations

Conclusion

We have been able to make a decent successful attempt at the NIPS 2017 Challenge as we have been able to train the body to learn to walk on the ground even with obstacles for a fairly large distance (on an average a reward of 20-25) which compares significantly with the final leader board of the competition. There are several things we could have done but were unable to due to paucity of time. Future work can involve effect of different types of parameter noise on the weight of the DDPG algorithm, trying out policy optimization algorithms (TRPO and PPO), exploring evolutionary strategies once the above policy has been learn using DDPG, etc. Changing the reward function by adding a better characterized obstacle penalty reward may lead to further improvements on the score. On a broader level, the progress from here can be used as a starting point for bringing Deep Reinforcement Learning to solve problems in medicine and RL research in other computationally complex environments, with stochasticity and highly-dimensional action spaces.

Acknowledgements

We acknowledge the help we received from 2 of our batch mates who are also taking this course : Govind Lahoti (140050020) and Aviral Kumar (140070031) who were working on the NIPS challenge with us before the midsem exams. Govind worked on evolutionary genetic algorithm and Aviral helped with understanding and experimenting with TRPO (Trust Region Policy Optimization). We also acknowledge the help that we received from discussion forum at (EPFL) gitter channel.

References

Coady, P. 2017. TRPO code. https://github.com/pat-coady/trpo.

EPFL. Learning to Run. https://www.crowdai.org/challenges/nips-2017-learning-to-run.

Hafner, D. Deep Reinforcement Learning for Continuous Control. http://www.ausy.tu-darmstadt.de/uploads/Site/EditPublication/Ramstedt_BscThesis_2016.pdf.

Lei Ba, J.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *ArXiv e-prints*.

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. *CoRR* abs/1509.02971.

Mohandas, G. Gradients, Batch Normalization and Layer Normalization. https://theneuralperspective.com/2016/10/27/gradient-topics/.

Mohanty, S. 2017. Learning to Run. https://github.com/stanfordnmbl/osim-rl.

Pavlov, M. 2017. Nips RL. https://github.com/fgvbrt/nips_rl.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *CoRR* abs/1706.01905.

Plappert, M. 2016. Keras RL. https://github.com/matthiasplappert/keras-rl.

Ramstedt. 2016. Deep Reinforcement Learning for Continuous Control. http://www.ausy.tu-darmstadt.de/uploads/Site/EditPublication/Ramstedt_BscThesis_2016.pdf.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2015. Trust Region Policy Optimization. *ArXiv e-prints*.

Yongliang, Q. 2017. Stanford OSRL. https://github.com/ctmakro/stanford-osrl.

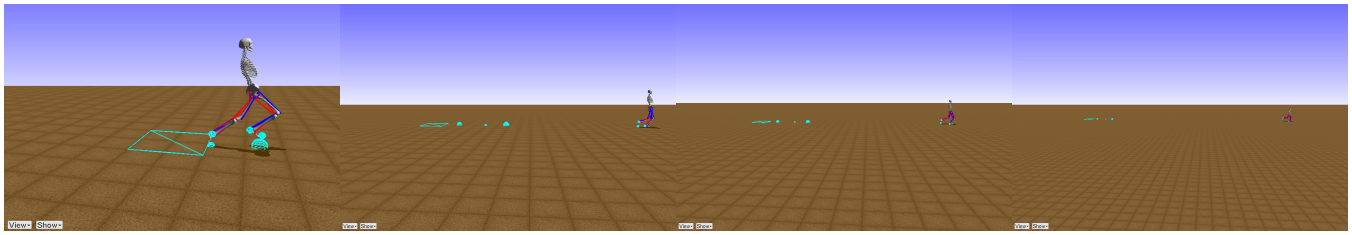


Figure 5: Simulation of the model on the environment