**Project Report on Making a human walk using reinforcement learning**

**Team**: Supriya Bhide & Sushrut Tadwalkar

**Github Repository:**

<https://github.com/sushrutt12/CognitiveComputing_FinalProject>

<https://github.com/supriyab2g/CognitiveComputing_FinalProject>

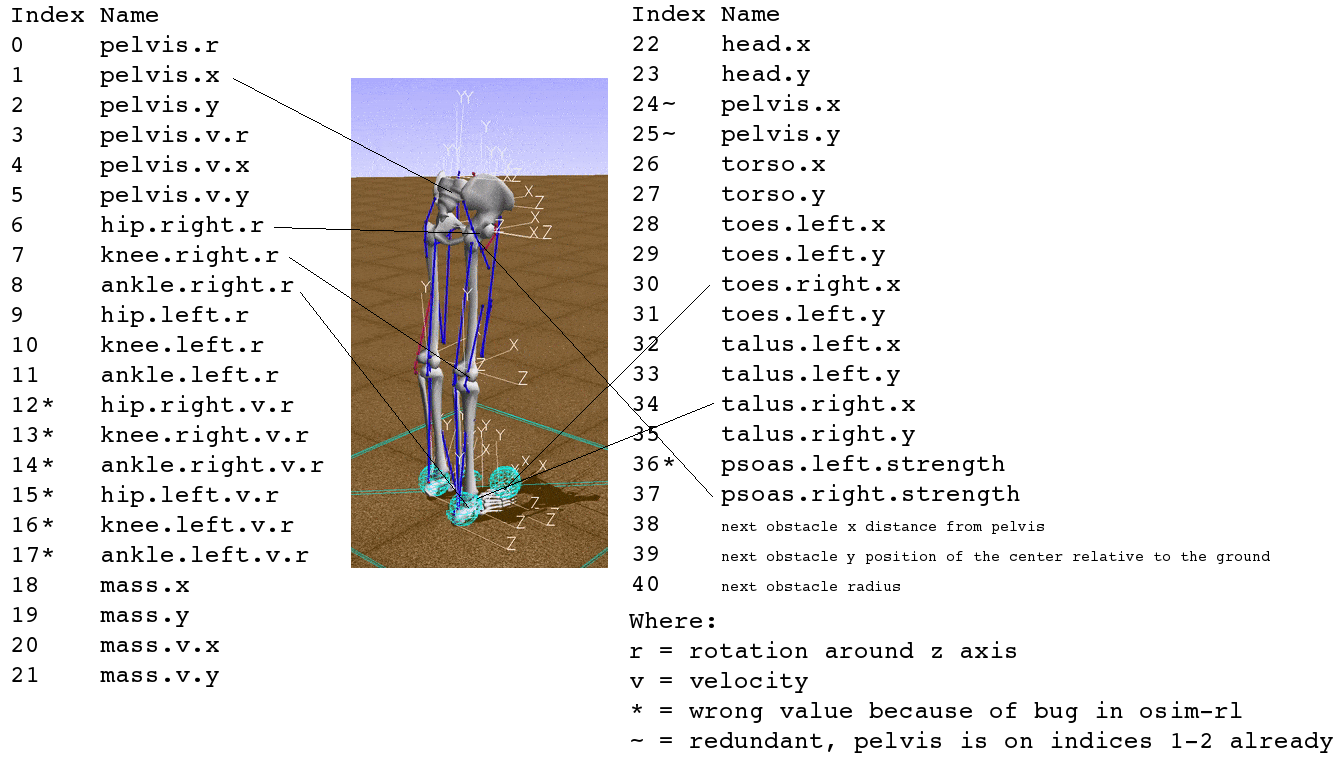
**Project Overview:**

<https://github.com/stanfordnmbl/osim-rl>

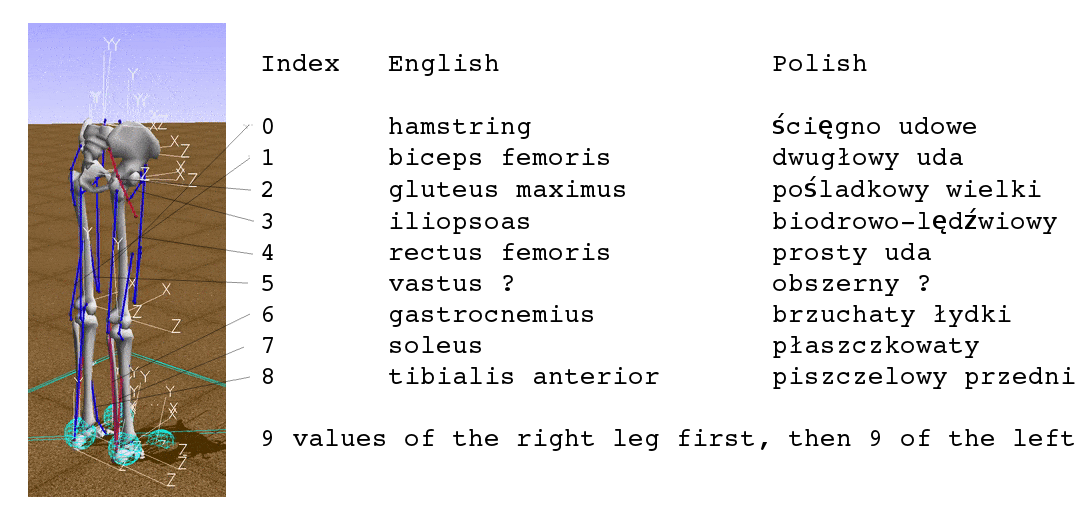
We followed the steps in the link above to create the environment where opensim provides a visualization for making a human walk.

Our Agent interacts with the environment to maximize the reward it obtains

The sample space consists of a vector of size 41 illustrated by the diagram below which is the input to our model.



The output consists of a vector of 18 (9 muscles x 2 legs)



**Approach 1 :**

Using DQN (Deep Q Network):

DQN uses a quality function (Q) to predict future rewards and instead of learning by trial and error all the time it predicts and maintains a memory to refer to.

Following is the mathematical function that depicts DQN

Q(s, a) \xrightarrow{} r + \gamma max_a Q(s', a)

S: Sample (41)

A: Action (18)

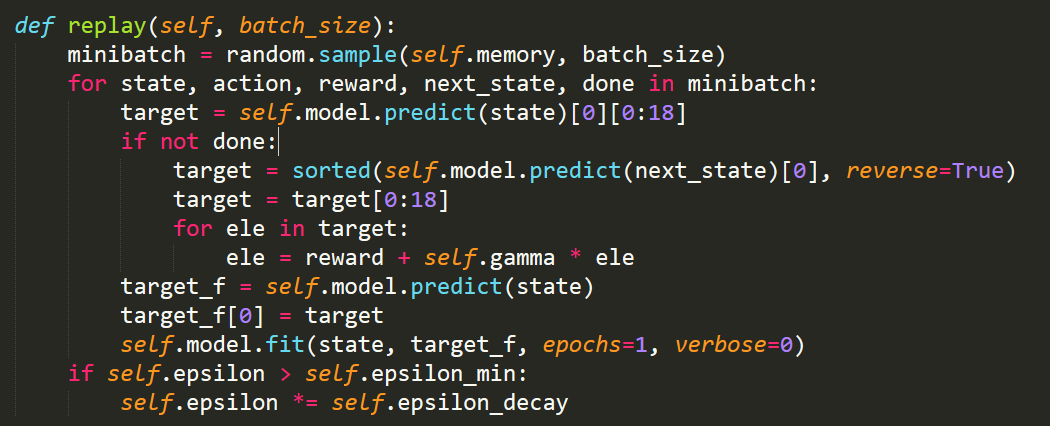
R: Reward

ɣ: Gamma (Discount)

Epsilon : Exploration

Epsilon\_decay: Exploration Rate

Max/Argmax: this function returns the highest value in the provided array



**Approach 2:**

Using DDQN:

One problem in the DQN algorithm is that the agent tends to overestimate the Q function value, due to the max in the formula used to set targets. As a solution to this problem DDQN came into existence. In this new algorithm, two Q functions – Q_1 and Q_2 – are independently learned. One function is then used to determine the maximizing action and second to estimate its value. Either Q_1 or Q_2 is updated randomly with a formula:

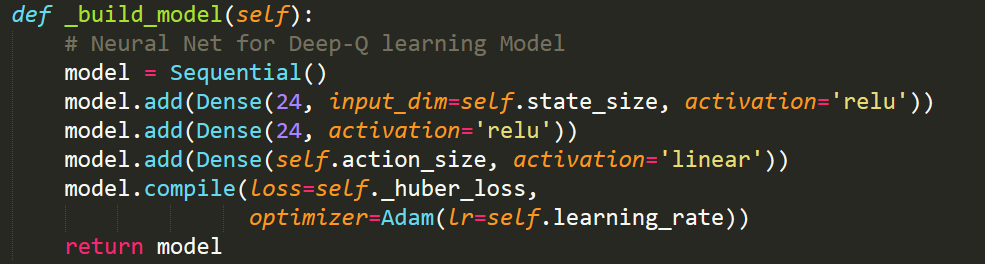
Q_1(s, a) \xrightarrow{} r + \gamma Q_2(s', argmax_a Q_1(s', a)) 

Q_2(s, a) \xrightarrow{} r + \gamma Q_1(s', argmax_a Q_2(s', a)) 

Writing it into code it becomes:

t[a] = r + GAMMA \* pTarget\_[i][ numpy.argmax(p\_[i]) ]

Following is the model used for building both these networks:



**Conclusion:**

If trained over a lot of iterations the network manages to start taking a step.

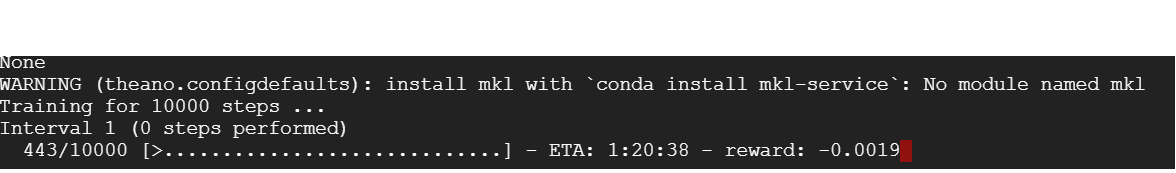
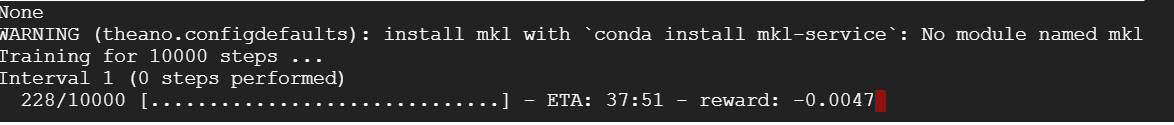
In the beginning the agent shows no sign of learning this is due to the Epsilon value (Exploration rate), the agent tries to explore all possible outcomes and as the rate starts to lower. The agent refers to memory to take actions where he knows what gives the best reward.

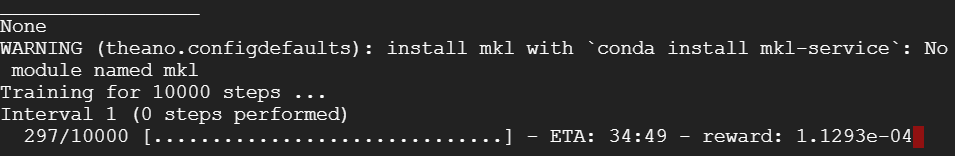
|  |  |  |
| --- | --- | --- |
|  | DQN | DDQN |
| First Step | 350 + Iterations | 127+ Iterations |

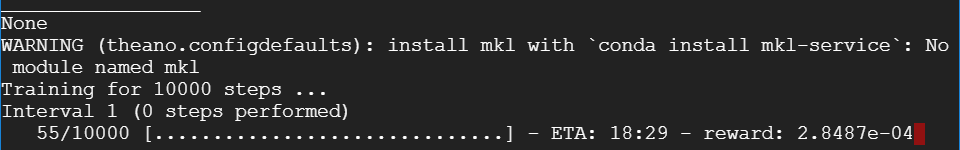
**Results on cloud:**

When run on cloud there was no visualization therefore only the console was available.

We ran the model multiple times and it differs everytime.







**Results on local:**

