Reinforcement Learning with OpenAI Gym

Yaniv zimmer
zimmer.yaniv@gmail.com

Shimon Cohen
shimonchoen655@gmail.com

March 2022

Abstract

In the following paper we will explain our attempted on solving Luna-lander and Bipedal-walker continuous environments, by using several deep RL models, including: DQN, DDQN, Dueling DQN, ACTOR-CRITIC and a variation of TD3. These models were written mostly with the Keras library and pytorch.
https://colab.research.google.com/drive/17oJuF58f5JscfLQDzbXukoQGLsaY-P25scrollTo=C5xu2QQ9X7tD

1 Introduction

Luna-lander-Continuous and Bipedal-walker are both continuous environments. In order to solve them, we first transform each environment's action space into a discrete representation, creating a semi-discrete form. After that, we define our models and fine-tune all of our hyper-parameters in order to solve the given problem.

2 Modifying the environment

2.1 Making a discrete environment

A discrete action space means to take the environment's bounds for each action, and split it into a given amount of even bins. In order to do so we have tried several approaches for both the action space and observation space.

2.1.1 Discrete action space

Out first approach was to split the continuous interval of (-1,1) into equal bins. Each value that falls into the bounds of a bin will change to be the bin's represented value. We have tried to assign the median of the bins wall or the lower/upper bin wall value. Assigning the median value gave better result, but the difference was not significant. After applying the discrete mapping, each action vector will have its own corresponding action index. Each action index represent a vector of bin indices for each coordinate action (2 for luna 4 for bipedal).
Our Deep RL models will receive the state, and return the appropriate action index by the model's prediction. The action index will be translated into an action values vector. The action vector will be given to env.step as the action our agent should make. That we will handle the continuous environment as if it was discrete.
In deciding on how many bins to assign, we had to balance between the agent need to find the optimal solution at least once, while maintaining small action space so the agent will be able to find optimal solution in high prob, and to learn it. Empirically we found [6,4] to be the best distributions of bins.
We also considered using normal distribution instead of uniform, but decided to stick with uniform because in the luna environment the interval of (-0.5,0.5) has the same effect (engines off).
***In order to do so, we tried using our own discretization class, and a wrapper of the env actions.

2.1.2 Discrete state

We have tried the same idea for dividing the state space into bins, but found the models are doing better with the real values of the env sensors, instead of discrete ones.

2.2 Changing the reward system

The reward system in each environment was a set -100 for failure (crashing or falling), and a reward of 300/200 for the Bipedal/luna environments respectively. We have tried changing the reward received from the env, only for the learning process. We tried changing the reward on falling/crashing from -100 to each of the values -50, -10, and -5. In addition, we also tried changing every positive rewards to be 1.5/2/5 times the original reward. While these changes at first seemed beneficial, they made the models get stuck on an unwanted average reward (around 30) without being able to get better. We assume this is because the model keeps over estimating positive rewards without giving proper weight to crashing or negative reward, that is by definition of changing the reward, so we chose to leave the reward as is.

2.3 Disabling the environment's time limit

Each environment comes with it's own time limit, and sometimes the learning process is not efficient, or does not see enough states if the learning process is cut short. In this case, disabling the time limit may have a positive affect on the model's training. In light of this, we tried to disable the time environment's time limit. This resulted in longer training episodes, of course, but still hasn't given us a substantial growth in the model's score average.

3 Replay memory

An important concept in training models is balancing out the size of their memory of past states and results according to a selected action. Some models attempt to solve an environment with no recollection of the past, while others need the whole history to have an effective learning process.
We chose to implement a Replay Memory, which is remembering a short distance into the past.

3.1 Memory type

We tried a few types of Replay Memory. One where a new experience overwrites the oldest experience, another which was a greedy Memory where we sample experience with highest distance from the model prediction, and last, randomly replaced memory where new experiences overwrote old experiences randomly. In addition, we considered prioritized experience replay to accelerate the learning. From our trial and error process, we have found them all to have similar results.

3.2 Memory size

As mentioned before, the size of the memory is the model's look in to the past. The size of the memory, combined with the memory algorithm can greatly affect the learning process. This is why we have tried changing the memory size. The different sizes of memory we tried varied from 5K to 1M. In general bigger memory yields better results because of Catastrophic forgetting. Of course this should be balanced out so there would not be Catastrophic remembering.

3.3 Memory fill

The amount of memories at the beginning of the learning process is important. With an empty memory, the start of the process has no experience to train on, which is bad when we want our model to learn from a batch of memories on a specific state.
When training off-policy models, we tried either filling the whole memory with random experience before training, and filling just enough of batch size. For batch size of 1024 filling just the amount of batch size gave better results. When training on policy models, of course we did not filled the memory with too many random experience, since it learns from the policy result. Instead we used memory were we started to learn the model right when we collected enough memory for a batch sample.

3.4 Batch size

In the process of training a model, there is an option to let it learn from a single experience, or a batch of experiences. We decided that letting the model learn each time from a given batch will have a good effect on the learning process. In which case we let our models learn each episode from a batch randomly selected from the replay memory. The selection is uniform over all memories (as mentioned in the Replay Memory section). Batch sizes had minimal effect, for better or for worse, on the model's learning. Tested sizes were usually a power of 2, for example: 64, 128, 512 and 1024. We used power of 2 in order to best utilize the power of our gpu

4 Exploration vs. Exploitation

An important method of training a model is Exploration vs. Exploitation. This is the balance between letting the model explore the environment by testing random actions for a given state, and forcing it to make a decision of what action to take. In order to balance this behaviour we set a parameter epsilon to be the threshold deciding if we should explore or exploit. Each time we generate a random number, and if it's smaller than epsilon, we sample at random.
Each episode the epsilon value would decrease until reaching a set minimum (in our case usually 0.05). At first, the decrease rate was calculated by a decrease percentage. Later, we tested another option of a set decrease in value.

5 Additional Hyper-parameters

5.1 Optimizers

While compiling each model we used the ADAM optimizer. No other optimizers were tested.

5.2 Learning rate

We have found that our model starts learning, but at some point stops and starts to decrease. In order to cope with this, we did a Literature Review and found that a bigger batch size and smaller learning rate should stabilize the learning. After some trial and error we have found the parameters of batch size=1024 and lr=5e-5 to be optimal.
We feared that our model is getting close to optimal function approximation, but when getting a bad value (not representing the distribution), it leaves the area and adopts a bad behavior. To handle that, we also used lr decrease and batch increase over time according to the agent's results. it did a minor change in results.
In general, the lr should be big enough to learn approximation for the optimal function, yet small enough so it won't diverge when it gets close to the optimal solution. As mentioned before, we explored many lr rates, from 1e-6 to 1e-3. We also tried using different lr for the actor and the critic, and to networks in DDQN. We have found the best lr to be 5e-5.

5.3 Loss function

On our dqn based learning we tried mse, log-mse and huber. The mse function gave the best result. On out Actor-Critic we used custom loss for the actor and mse for the critic. we used clipping in the custom log to avoid log of close to 1 or 0, resulting in a calculation error.

5.4 Layers

5.4.1 Number of layers

In each model the amount of layers varied. At the beginning we used two layers for most of our models, and at the end we added some layers for testing. Adding additional layers did not give a cutting conclusion to the question "Do additional layers substantially improve the learning process?".

5.4.2 Layer activation

We have tried relu and tanh for hidden layers. For the output layer we tried softmax and linear. We received the best results while using relu and linear.

5.4.3 Layer size

We tried many layer size option. We tried 2 layers of sizes (8,8) (32,32) (64,32) (128,256) (512,1024) and similar combinations. We also tried using more hidden layers, from 2 to 8. We found fair results for 2 layers of 64 and 32.
Then we choose to use 64,32 to maximize results train time (in seconds) and efficiency.

5.5 Batch normalization

We used batch norm between hidden layers of the models, which improved the learning and made it smoother ,Especially on actor critic model.

5.6 Dropout

We tried using dropout between hidden layers. it did not improved the learning significantly.

6 Building the models

- DQN
- DDQN (2 networks)
- Dueling DQN (1 network and one copy with is updated periodically)
- ACTOR-CRITIC
- TD3-variation (using 2 critics to decrease over estimation)

At first we used off-policy Dqn model. In Dqn we explore the action space by choosing randon action in prob of eps, or by best expected action in prob 1-eps.
eps is decrasing over time from 1 to min eps. We tried decrease it by several approches: Constant value minus decrease (minus eps dec) multiply dec (epsilon is esp dec time epsilon) we also tried hard stop of epsilon, while after number of epocs it will jump to min eps. In order to learn the q value function, the model compute the loss between the expected value of choosing action a for state, and the value of immediate reward + gamma * the max value of choosing the best action of the state caused by choosing the action a.
we tried eps of 1, 0.99, 0.98,... 0.01
1, 0.999, 0.999**2,0.999**3 ... we also tried to start on eps 0.5 which gave worse results, and to stop at 0.05 eps which gave similar results.
We found it produce bad results, probably due to overestimating bad actions. Dqn model is "by definition" over estimating the value of the action he chooses. that is because it choose action by using argmax function on their estimated values. It keep choosing the best-value action and estimating next action also by it.

In order to coup with that we used two approaches:
1.DDQN- using two distinct DQN network.
while choosing action we randomly choose action by eps for random, (1-eps)/2 for network 1 or 2. In order to learn, it uses one network to estimate the action and the order to estimate next state best action value 2.Dueling DQN- using one network and a copy of it. the copy is updated each hyper parameters steps. we use the original network to choose action, and the copy to estimate value of best action next state.

Then we have tried on-policy models: We started by using actor critic model. actor chooses action and critic estimate value for action+state
after doing review literature we used semi TD3 model, which uses 2 critics, and take the minimum estimation they give in each step. Thus avoiding overestimation by the critic.

6.1 Model output

Every model has a certain output, if it is q-values, an action or an index of a chosen action. We have tried to use these different approaches in out model building. Models such as DDQN and Dueling-DQN were implemented to yield q-values, While DQN and Actor-Critic variants were built once returning an action index, and another time returning an action. The action index and action were selected as the maximal value from the model. When expecting an action index, the model gave probabilities from which the maximal was chosen.

We also tried making the output of DeepRL to be vector of action vector size (2 for luna 4 for bipedal) times the number of bins. than we split the output vector into parts of number of bins size, and use argmax to get best action according to the model. The results of this attempt were not good. At first the average score graph began to go up, but very fast it turned into low score and the agent adopted bad policy. We believe that this is because the agent learned to choose optimal bin for each coordinate, but the combination of these coordinated was not optimal.

7 Tackling the Hardcore environment

At first we've tried using our discrete env wrapper, with the actor-critic model used on Luna lander. It did poorly, probably because of the huge magnitude of the Bipedal and Bipedal Hardcore env. It takes many episodes until the agent can find an episode with close to optimal solution. If we used random actions in order to try getting the agent to walk until the end, we would have close to zero probability to do so. Because of that we've tried a different approach. We did comprehensive review of literature, and found TD3 archives good result on the Bipedal environment, from reasons that will be specified later in this document. We used TD3 model, implemented with pytorch which we tailored to our needs to fit with our classes and configurations. It gave good results and almost solved the hardcore environment for the discrete action space (we implemented wrapper for the continuous environment) in 3700 episodes. It did solved the Hardcore environment in 1750 episodes using the regular continuous environment without the wrapper (much less than 3K episodes). We remind that solving means to get an average score of 300 on 100 consecutive episodes.

8 Training the models

We have trained the models in training loops with about 3000 episodes at most. If the model preformed poorly, we stopped the process prematurely.
Most models for Luna lander converged at around 1000 episodes. For bipedal 1750 and for the Hardcore env Solved after 1750 for the continuous action space.

9 Why TD3 was so good compared to other models

TD3 (Twin Delayed DDPG) introduce 3 important improvements on other model (most closely actor-critic). 1.At first it used 2 critics, and use the minimum estimation of them, (that's where the twins name came from). By doing so, it coup with the hurting problem of over estimating the function value. 2.Clip learning size- It clip the learning difference in order to avoid too big changes in the policy on one train. Thus it prevent the model from adopting decent- yet not optimal policy fast, and encourage it to explore more. 3.Target network update period- instead of training the target model on each iteration, the model is updated every hyper parameter frequency (we used 2 as the frequency parameter). By doing so, the learning become smoother and more resilient to different experience

10 Conclusions

In reinforcement learning there is no magic solution. A model capable solving one environment is not necessarily able to solve other environment. Many trial and error must be made. We have used many models and hyperparam. It is especially important that our model will handle the over-estimation problem by using 2 critics/q networks. It is also important to keep an eye on hyperparam like lr and

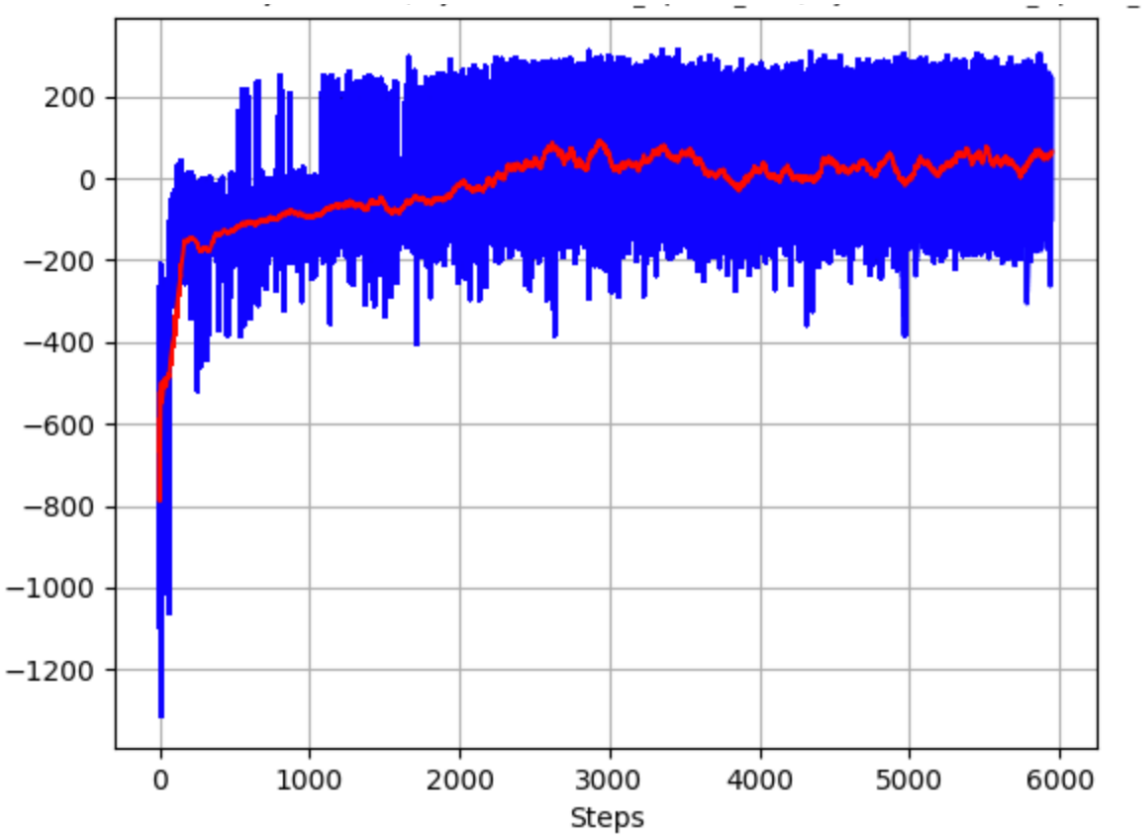

Figure 1: This is actor-critic with batch 1024 memory size 0.5M lr 5e-4. (Luna lander)

batch size, which helps to ensure smooth learning. Furthermore other Deep Learning tricks, like batch norm can improve the learning. Also we have found that using discrete space does not necessarily make the environment "simple" because of explosion of spaces, and it is better to use continuous models for continuous space

Because TD3 Handles over-estimation problem, make smooth learning, it gave best results.

References

https://jeffmacaluso.github.io/post/DeepLearningRulesOfThumb/
https://towardsdatascience.com/adapting-soft-actor-critic-for-discrete-action-spaces-a20614d4a50a
https://arxiv.org/pdf/1901.10500.pdf
prioritized memory replay: https://arxiv.org/abs/1511.05952v4
https://spinningup.openai.com/en/latest/algorithms/td3.html
https://github.com/honghaow/FORK/tree/master/BipedalWalkerHardcore
https://github.com/philtabor/Youtube-Code-Repository/tree/master/ReinforcementLearning/DeepQLearning
Our colab notebook: https://colab.research.google.com/drive/17oJuF58f5JscfLQDzbXukoQGLsaY-P25 We also have shared the colab notebook by email.

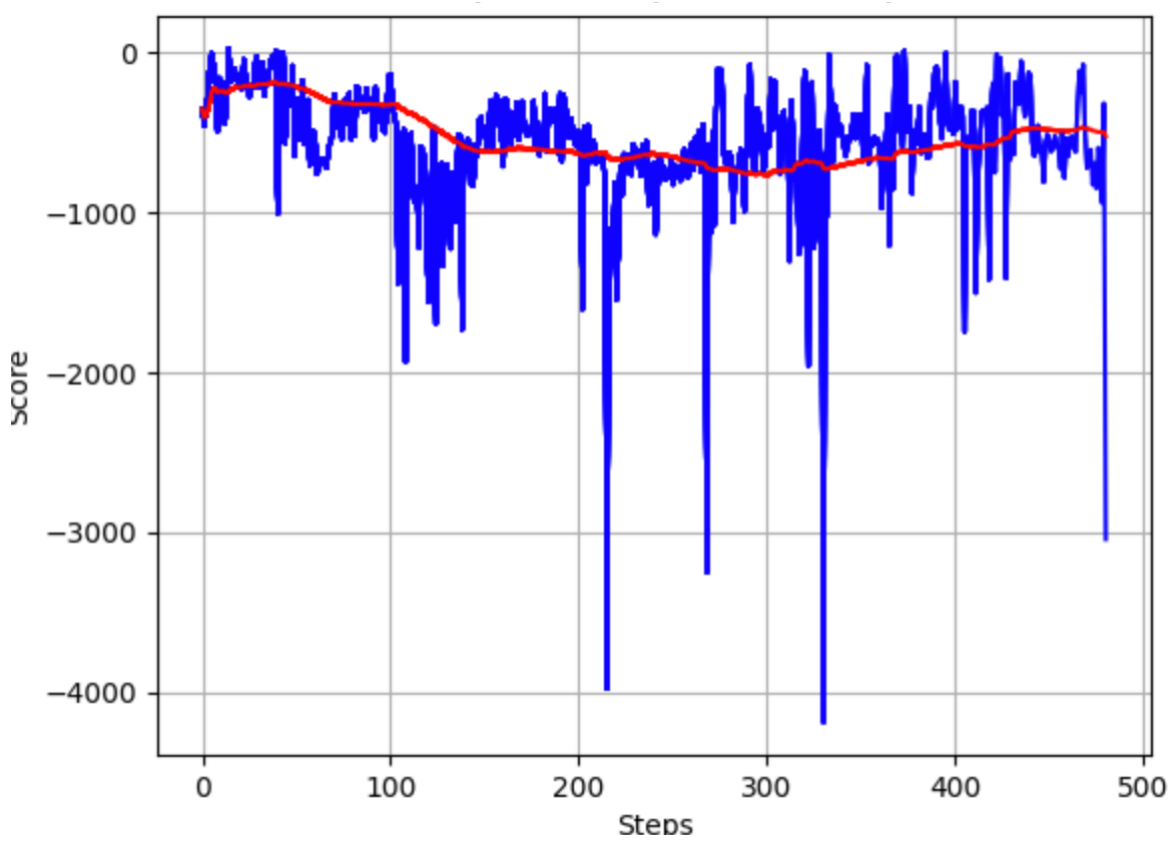

Figure 2: DQN Model. (Luna lander)

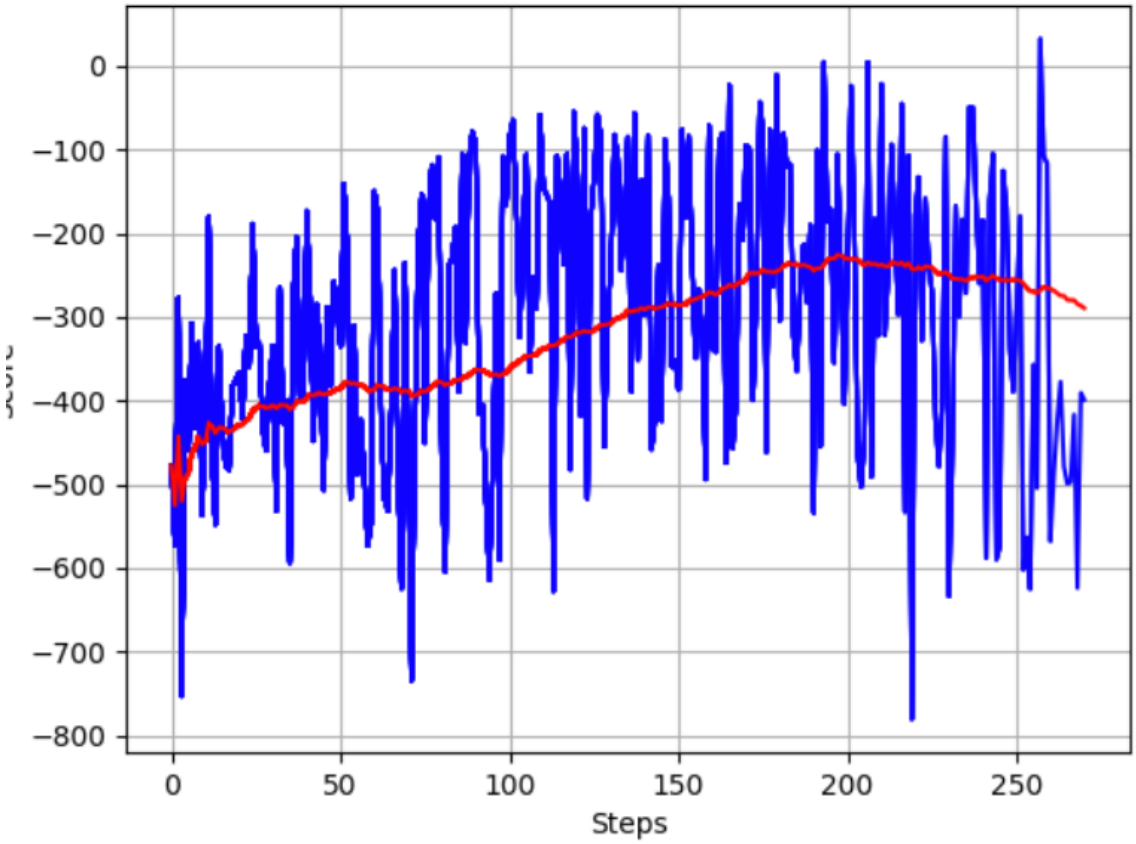

Figure 3: DDQN Model. (Luna lander)

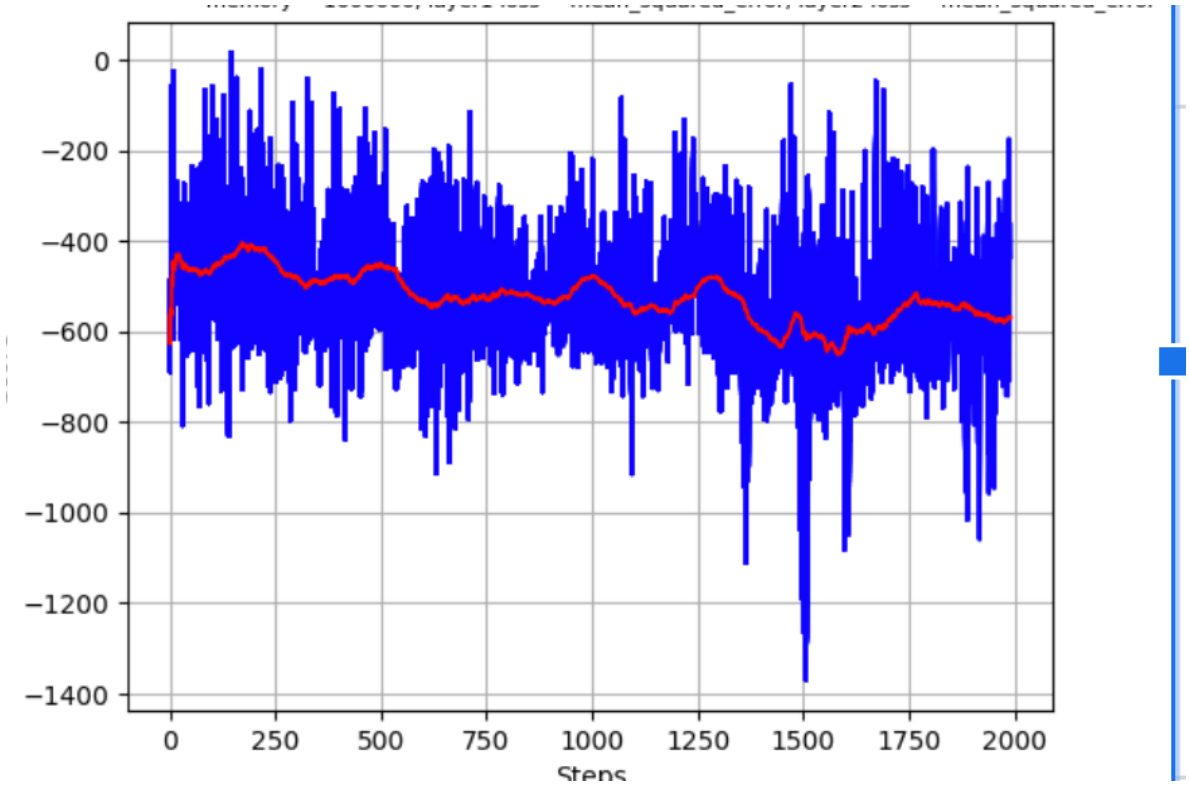

Figure 4: Dueling DDQN Model. (Luna lander)

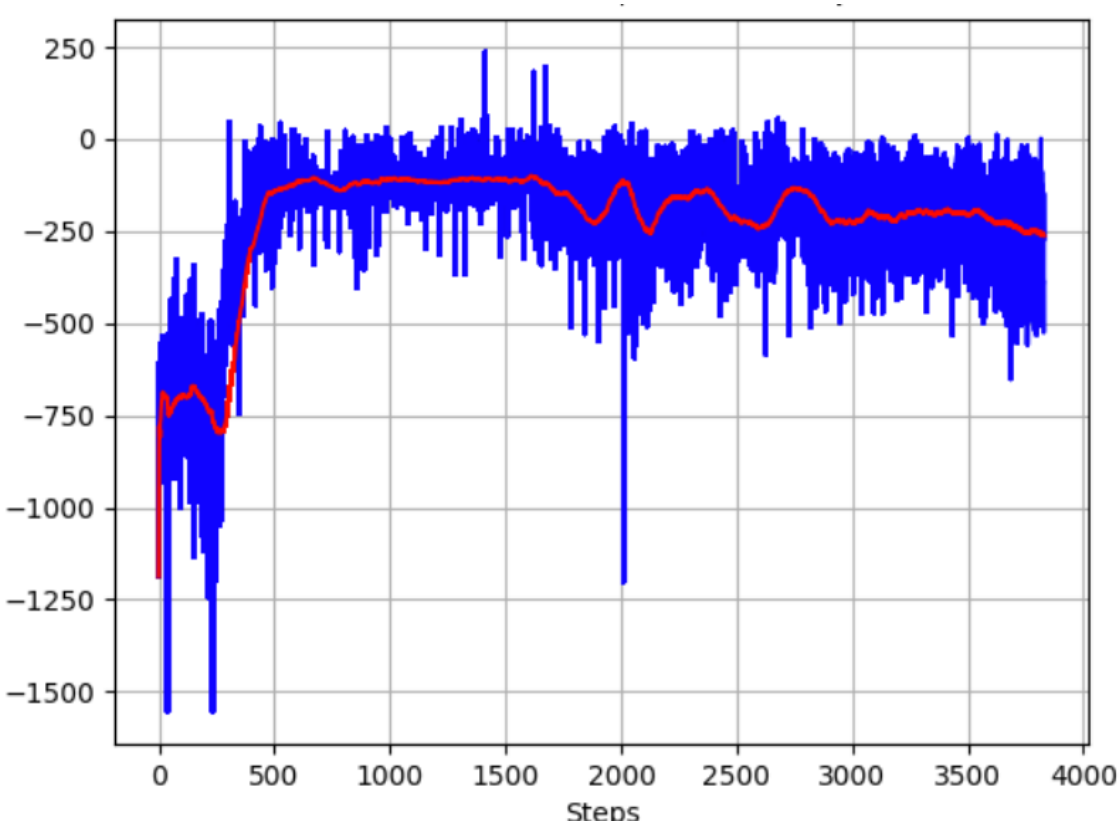

Figure 5: Action Critic with 2 critics-inspired by TD3 Model. (Luna lander)

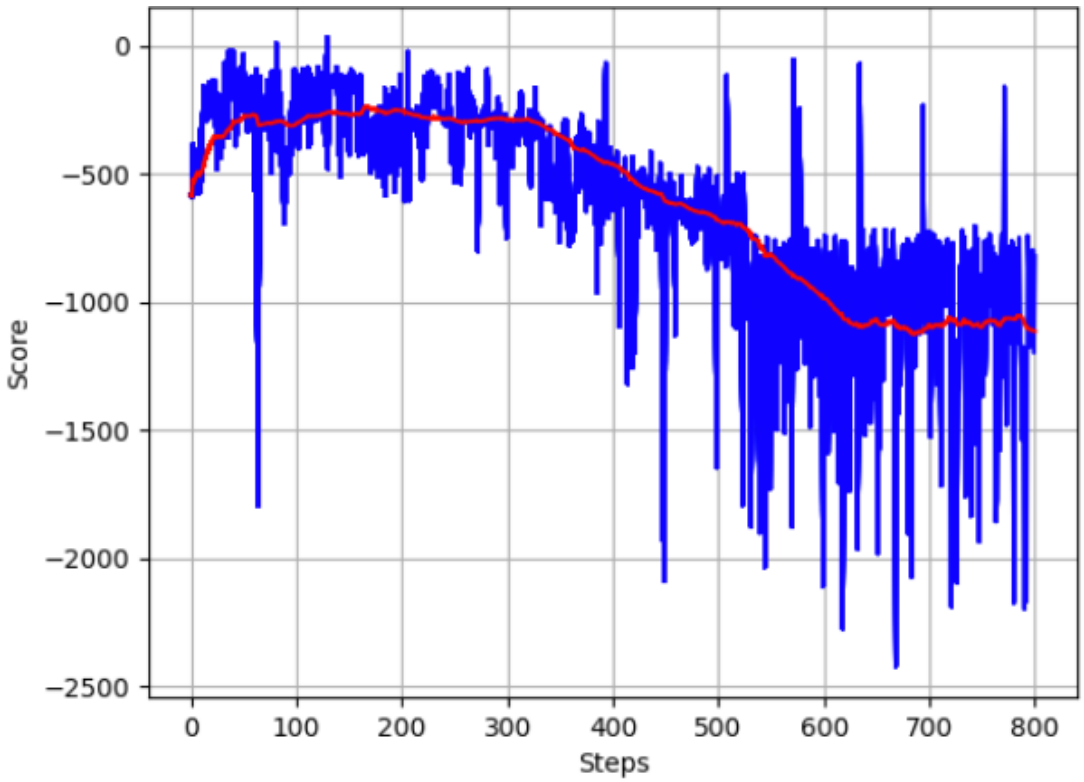

Figure 6: Catastrophic forgetting (Luna lander)

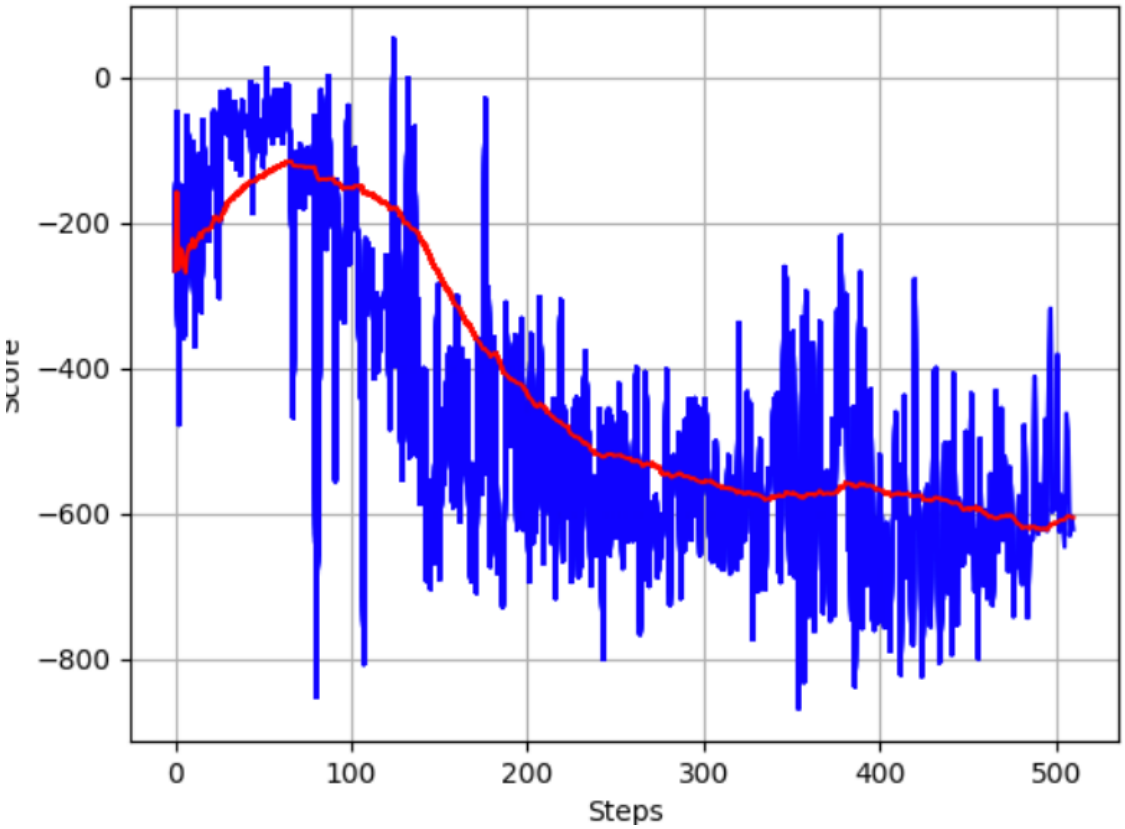

Figure 7: splitting the model output and using argmax on each part to choose coordinate separately (Luna lander)

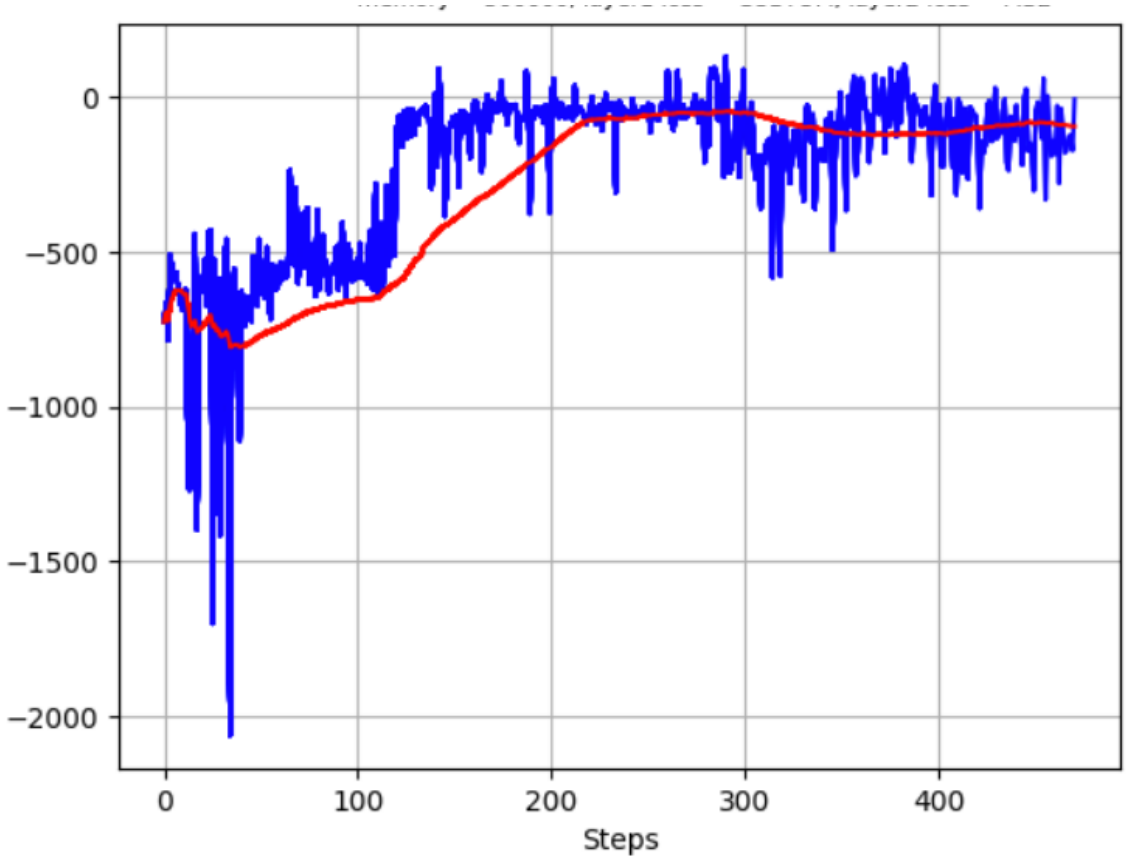

Figure 8: Big network (256X512X256) actor critic (Luna lander)

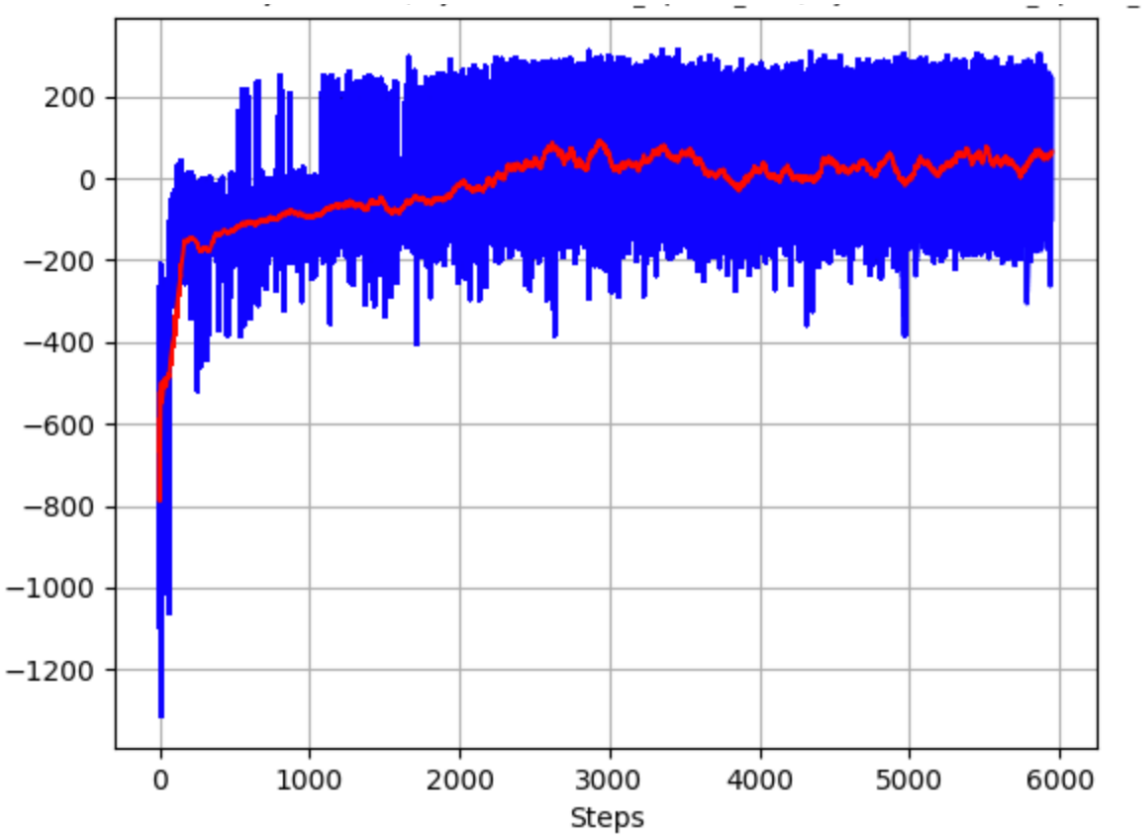

Figure 9: TD3 discrete (Luna lander)

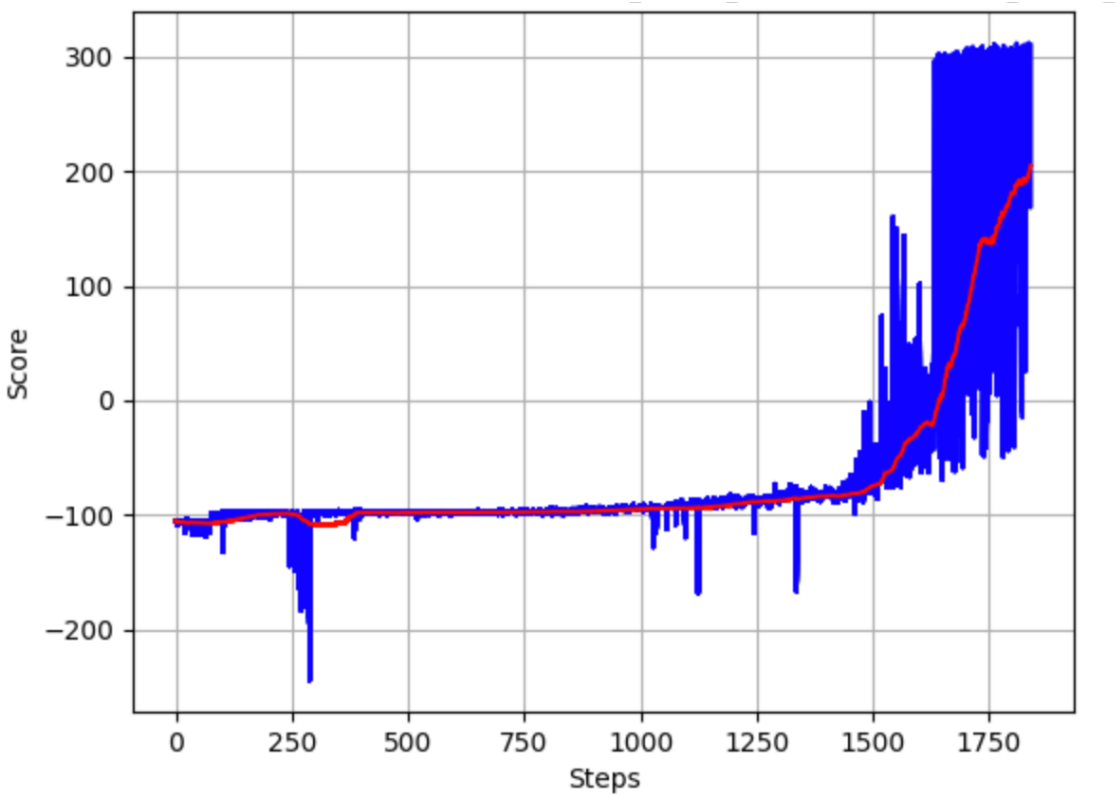

Figure 10: TD3 discrete (Bipedal)

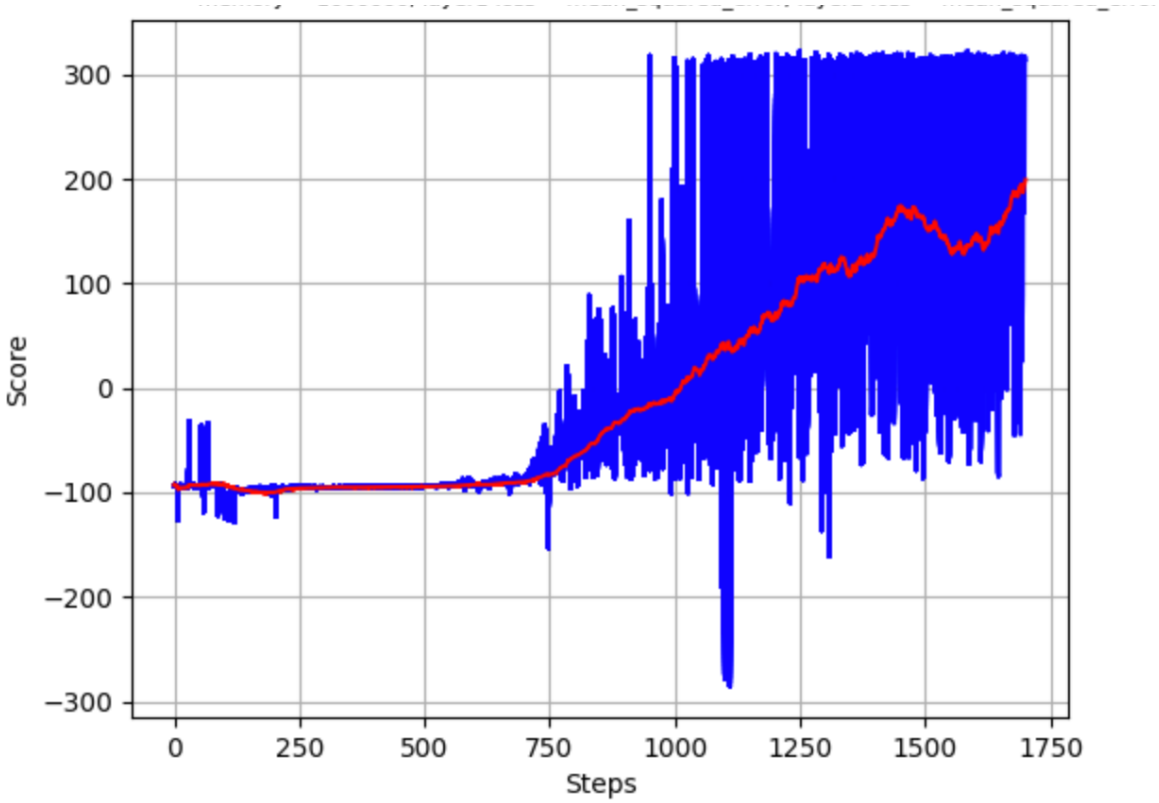

Figure 11: TD3 continuous (Bipedal)

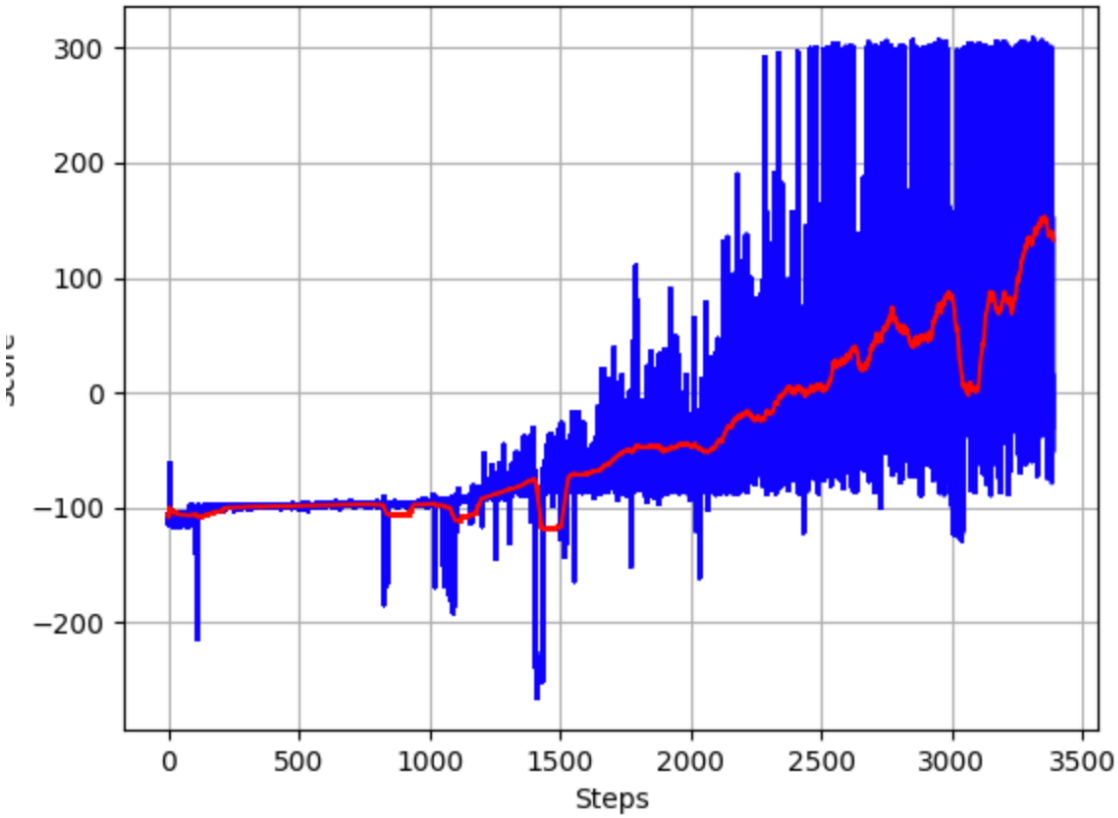

Figure 12: TD3 discrete (Bipedal Hardcore)

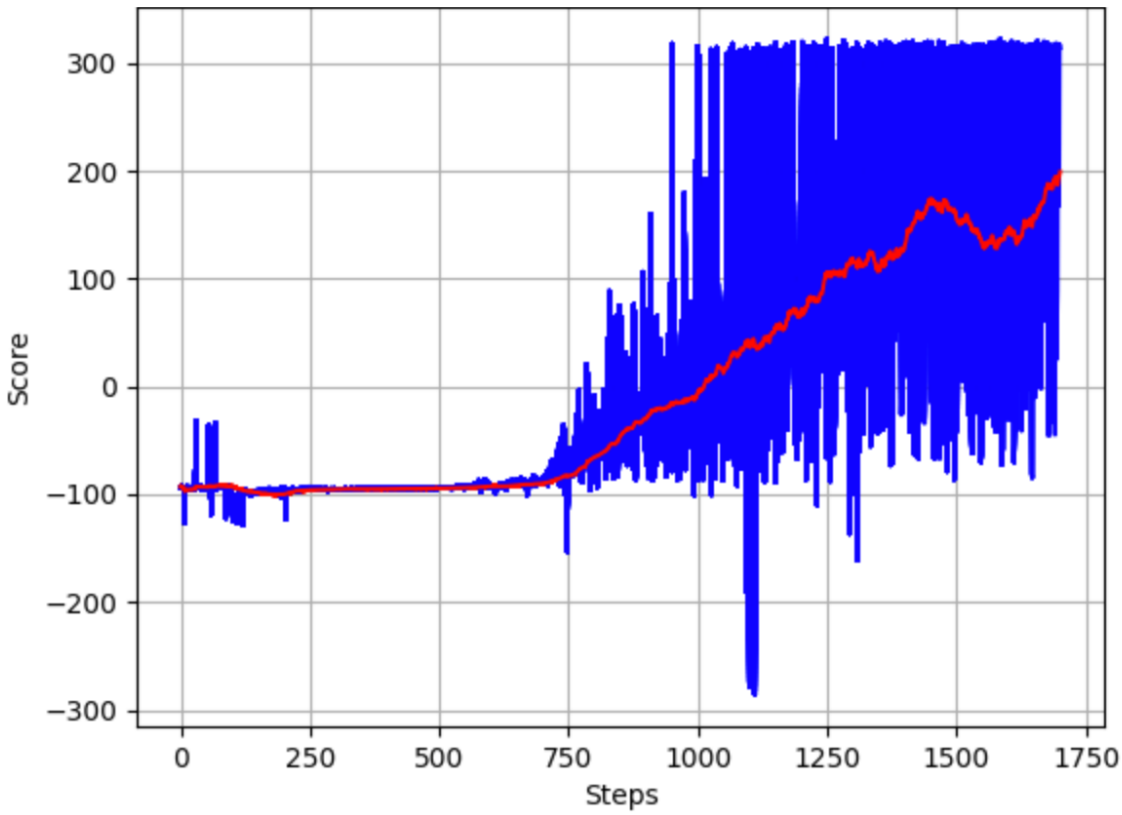

Figure 13: TD3 continuous (Bipedal Hardcore)