Skill Trees for Hierarchical Reinforcement Learning

Maxime Andre
ECE
mandre@andrew.cmu.edu

Brandon Houghton
CS
bhoughton@cmu.edu

Rishub Jain
CS
rishub@cmu.edu

Aaksha Meghawat
LTI
ameghawa@andrew.cmu.edu

Abstract

We propose a hierarchical skill learning based scheme for navigating complex multi-challenge environments. We propose to first teach the agent simpler tasks such as walking/jumping/balancing in MuJoCo environments comparing DDPG and NAF approaches. We then develop a hierarchical reinforcement learning model using the meta-controller framework that would enable the agent to operate in a more complex environment requiring a combination of the simpler skills. We hope to emulate the learning process of human beings wherein we first start by learning simple physical movements and eventually learn to combine them in more challenging situations.

1 Introduction

Complicated physical movements are usually learned by breaking them down into simpler movements and practicing them in easier terrains. Human dancers skilled in one dance form usually find it easier to pick up another dance form. Can this intuition be used in training agents/robots as well? Can this general abstraction on learning allow RL agents to tackle novel environments without extended trials? It would be important to explore these questions to train better agents that are faster at learning and adapting to new environments. More so because it is easier to optimize models and policies to learn simpler sub-tasks and combine them to tackle tougher terrains.

We want to develop a hierarchical skill learning based scheme for navigating complex multi-challenge environments. We describe in the 'Methods' section how we first teach the agent simpler tasks such as walking/jumping/balancing in MuJoCo environments designed to learn these tasks. We then develop a DQN based meta-contorller that selects the ideal sub-goal for the agent in tougher environments.

2 Related Work

Reinforcement learning is the problem of an agent that learns a behavior through trial-and-error interactions with the environment. The main challenges are the tradeoff between exploration and exploitation or learning complex behaviors [4]. In deep reinforcement learning, there is the problem of exploration which doesn't work well with sparse rewards [1].

Two main approaches exist to tackle these challenges. One is using a hand-crafted hierarchy over the actions. The second is using domain agnostic intrinsic rewards to guide the exploration process. In [2], in order to learn the skill set they used Stochastic Neural Networks [10]. The network can then be used to train a high-level policy over the skills. A similar approach is proposed in [6] where the agent has been split into a controller and a meta-controller. The meta-controller looks at the state and decides on a sub-goal, and the controller aims to complete that sub-goal. The controller performs the actions to complete the goal, and receives the reward intrinsically as opposed to directly from the environment, which is called intrinsic motivation. Once the controller reaches a terminal state, the meta-controller looks at the reward from the environment and decides on the next goal. These two components of the agent are making decisions at different times, which is called temporal abstraction. The meta-controller and controller are represented as Deep Q networks, and are trained with stochastic gradient descent. In [5], they create a framework to automatically learn the skills by performing spectral clustering and find abstract states which can be reused in other tasks without re-learning policies from scratch.

In this work we experiment with different models for the sub-tasks other than a DQN. We also experiment and design different environments that are better suited to learn the basic skills. Eventually we test the agent in a completely new and more challenging environment where it needs to employ both the skills.

3 Methods

We decided to train the agent 'HalfCheetah-v1' of the MuJoCo environment for 2 basic skills of walking and jumping. We modified the initial given environment to be suitable for training the jumping task. We then designed a more complex environment where the agent would have to use a combination of the basic skills for successful navigation (walking then jumping then walking for instance). All the environments that were designed will be described in detail in the 'Environments' section.

We experimented with the 'Deep Deterministic Policy Gradient' (DDPG) model as described in [8] to train the basic skills of walking and jumping. We also experimented with 'Normalized Advantage Functions' for continuous deep Q-learning as outlined in [3]. We present a comparison of the performance of the models for the basic skills.

We then trained a meta-controller fashioned as a 'Deep-Q Network' in the same vein as described in [9]. The meta-controller would choose the appropriate basic skill network to navigate successfully in the more complicated environment. An overview of this architecture is in Figure 1. We demonstrate the advantage of this set up by comparing it with the scenario where the agent needs to navigate in the tougher environment without a meta-controller.

We describe the different environments we designed in the 'Environments' section and the different models we used in the subsequent sections.

3.1 Environments

We started with the basic 'HalfCheetah-v1' MuJoCo environment and agent (Figure 2 a). In our experiments however we found it suitable to change the environment to make it more complex for the skills we wanted the agent to learn. We designed the following additional environments:

- Environment to train the Cheetah to jump with a box in its path (Figure 2 b).
- Harder environment for both walking and jumping with a slope in its path (Figure 2 c).
- Harder environment for both walking and jumping with boxes of different heights in the path (Figure 2 d).

3.2 Deep Deterministic Policy Gradient (DDPG)

DDPG is a policy gradient algorithm that uses a stochastic behavior policy for good exploration but estimates a deterministic target policy. It also uses the actor-critic framework. Both the actor and critic components are neural networks. These networks compute action predictions for the current state and generate a temporal-difference (TD) error signal at each time step. The input of the actor network is the current state, and the output is a single real value representing an action chosen from a continuous action space. The critic's output is the estimated Q-value of the current state and of the action given by the actor. The deterministic policy gradient theorem provides the update rule for the weights of the actor network. The critic network is updated from the gradients obtained from the

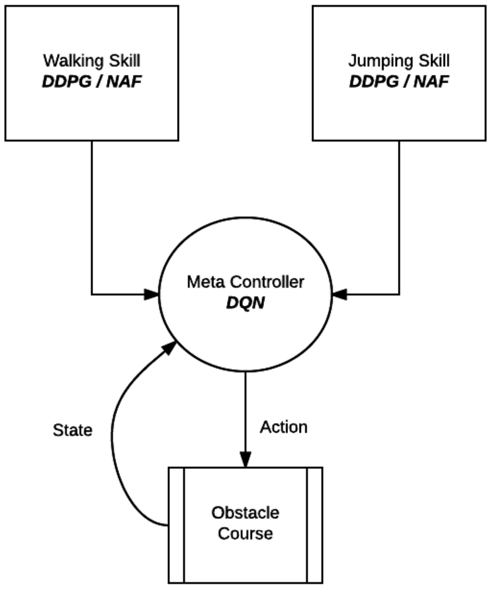


Figure 1: A 'Skill Tree' where the agent is skilled in walking, jumping and combining these 2 basic skills.

TD error signal. In addition to this, target networks are used to generate the TD target and further decouple the TD error signals. This is to avoid the problem of divergence.

Therefore the target for the critic network is calculated using the following process. Lets define:

- N is a minibatch of steps sampled from a replay buffer.
- y_i is the TD target for the i_{th} sample.
- $\theta_{\mu'}$ are the weights of the target actor network
- $\theta_{Q'}$ are the weights of the target critic network
- $Q(s_i, a_i|\theta^Q)$ is the Q-value of the critic network for the i_{th} sample
- L is the loss of the critic network

Then

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{1}$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2) \tag{2}$$

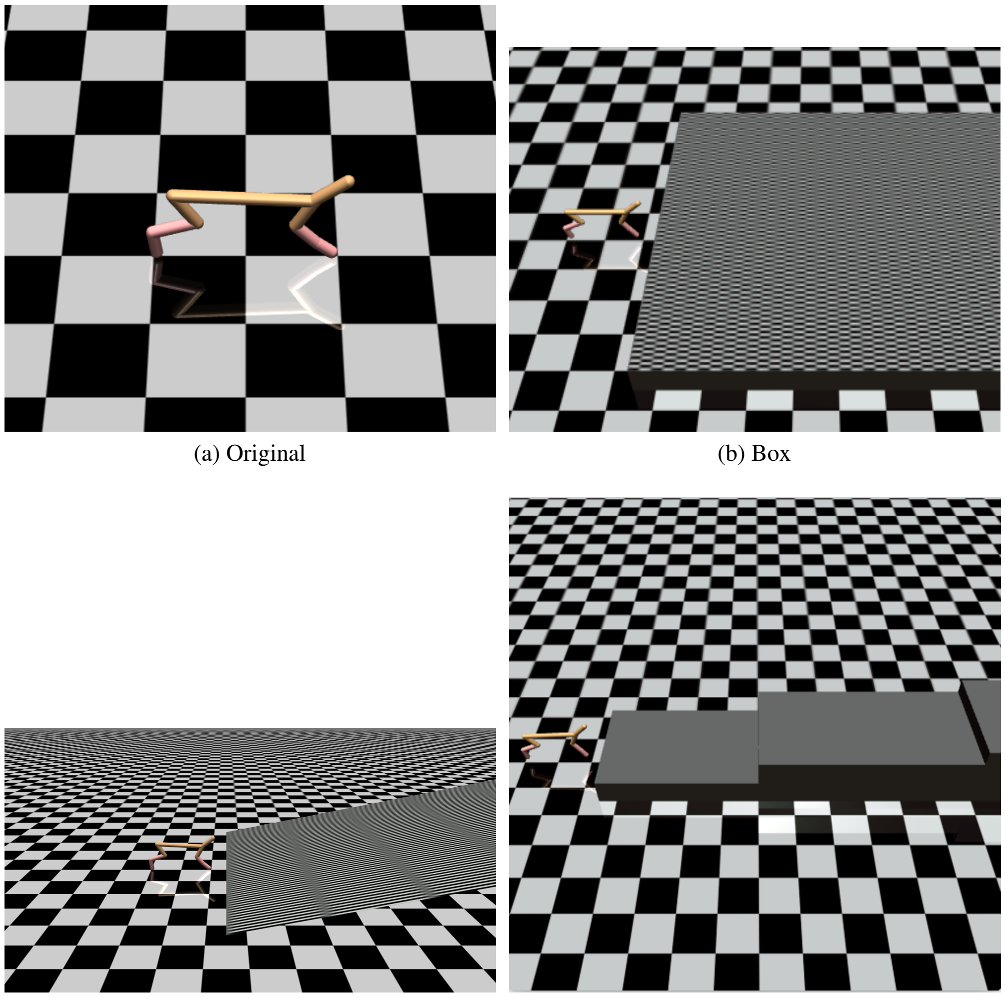
Let $\mu(a|s,\theta)$ be the stochastic policy. [7] proves that the following holds true:

$$\nabla_{\theta^\mu}\mu \approx \mathbb{E}_{\mu'}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}\right] \tag{3}$$

Using this, the update rule for the stochastic policy becomes:

$$\theta^\mu_{k+1} = \theta^\mu_k + \alpha\mathbb{E}_{\mu'^k}\left[\nabla_a Q(s,a|\theta^Q_k)|_{a=\mu(s|\theta^\mu_k)}\nabla_\theta\mu(s|\theta^\mu_k)\right]. \tag{4}$$

As the variance of the stochastic policy gradient becomes zero, it converges. This allows us to approximate the deterministic policy using the advantages of a stochastic policy and hence results in a better model in continuous action spaces.


(a) Original (b) Box

(c) Slope (d) Steps

Figure 2: Different Environments used to train HalfCheetah for the basic skills and their combination

3.3 Normalized Advantage Functions (NAF) for Q-learning

This approach is the one outlined in [8]. The main difference between this approach and the DDPG approach is that the Q-values are decomposed into value terms $V(x)$ and advantage terms $A(x, u)$. Both these terms are represented by neural networks. The goal of doing this is to enable the calculation of $argmax_u Q(x_t, u_t)$ analytically during the Q-learning update. Moreover the stochastic exploration of the action space is done using a random process which is added to the action sampled from the policy being estimated. This approach also makes use of the replay buffer experience to sample actions and target networks to decouple the policy being updated from the policy being used for sampling actions. The exact algorithm used for the NAF approach is described in Figure 3.

3.4 Deep Q Network (DQN) Meta-controller

We use the approach of a mets-controller as described in [6]. The exact model that we have used to model the meta-controller is the DQN model. This model generates the sub-goals/intrinsic goals for the controllers of the basic skills which are represented by models already described. It is the meta-controller that chooses between the skills to navigate a tougher terrain such as Figure 2 (c) and (d).

Let N be the number of time steps until the controller halts given the current goal g. π_g is the policy over goals and f is the reward signal of the environment. Given a state and a goal the meta controller

Algorithm 1 Continuous Q-Learning with NAF

```
Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network $Q'$ with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
**for** episode=1, $M$ **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
    **for** t=1, $T$ **do**
        Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
        Execute $\boldsymbol{u}_t$ and observe $r_t$ and $\boldsymbol{x}_{t+1}$
        Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in $R$
        **for** iteration=1, $I$ **do**
            Sample a random minibatch of $m$ transitions from $R$
            Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$
            Update $\theta^Q$ by minimizing the loss: $L = \frac{1}{N}\sum_i(y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
            Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        **end for**
    **end for**
**end for**
```

Figure 3: Normalized Advantage Function Algorithm taken from [8].

chooses g according to:

$$Q^*_m(s,g) = max_{\pi_g} E[\Sigma_{t'=t}^{t+N} f'_t + \gamma max'_g Q^*_m(s_t + N, g')|s_t = s, g_t = g, \pi_g] \tag{5}$$

We represent Q_m, the q-value function of the meta-controller with a Deep Q network. The DQN setup is very similar to the network described in [9].

4 Experiments

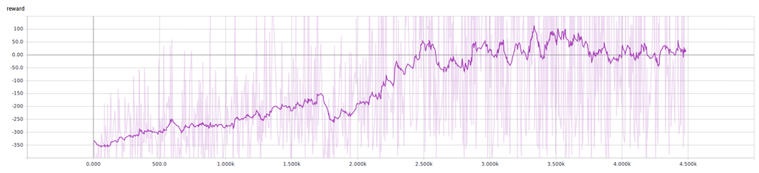
4.1 Walking

As mentioned earlier we tried the approaches of DDPG and NAF to learn the basic walking skill. The comparative performance of the DDPG and NAF approached is detailed in Figure 4. NAF performs much better than DDPG for this skill.

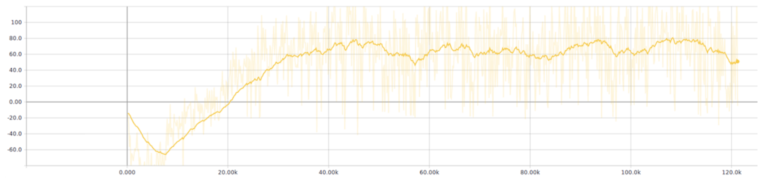
To be thorough about the performance of the walking skill, we also experimented with an environment with a slope. We tried 2 scenarios in this setting where Case 1 (Figure 2 c) was where the half cheetah directly faced the slope and had no run-up. Case 2 (Figure) was where the Cheetah had some space for walking on flatland before walking up to the slope.

As the performance graphs show (Figure 6), the agent performs much better in Case 2. This maybe because the agent acquires a stable momentum before mounting the slope as opposed to Case 1 where the agent has to directly start climbing the slope. However this may also be because it is easier to walk on flat land and that increases the total reward on average in any episode. Further experimentation would be required to ascertain the reasons for this behavior.

We believe that this maybe due to the fact that in Case 2 the agent learns to stabilize and hence is able to perform better on the slope.



(a) DDPG Performance for walking



(b) NAF Performance for walking

Figure 4: Performance of walking skill

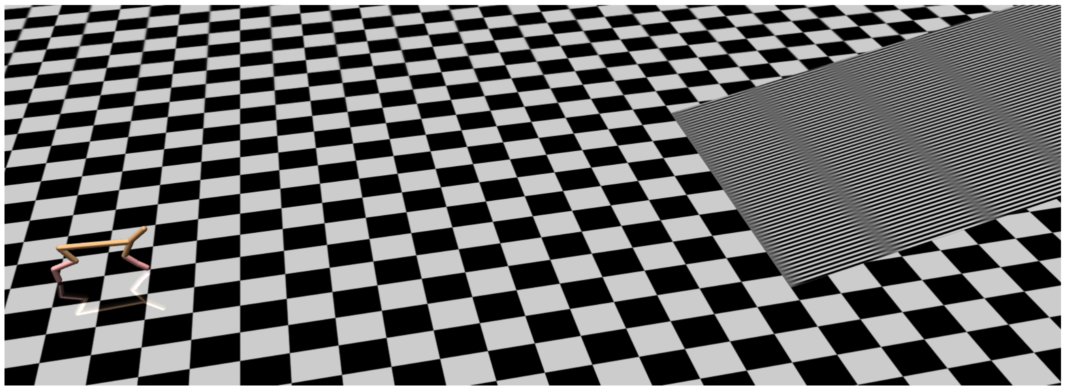
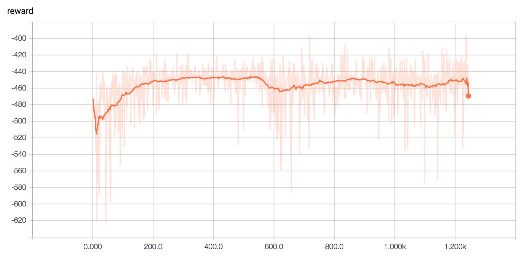
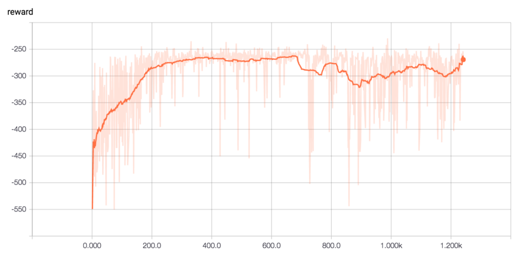


Figure 5: A variation of the Slope environment 2c where the Cheetah has space for a walk-up before climbing the slope.

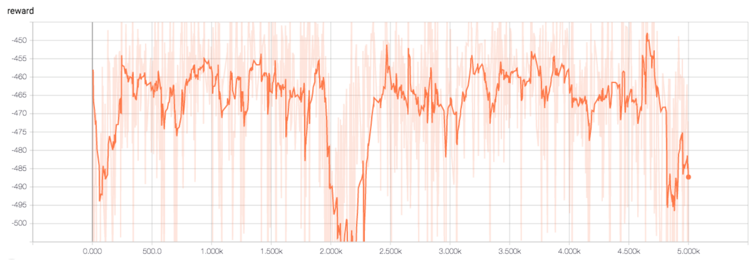


(a) Slope with no walk-up

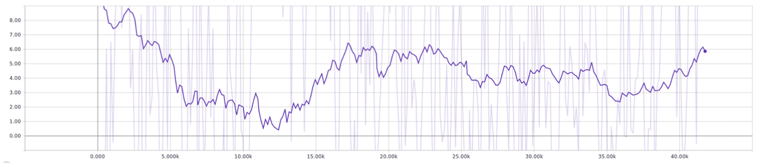


(b) Slope with walk-up

Figure 6: The performance is much better in the case where there is a walk-up to the slope.



(a) DDPG Performance for jumping



(b) NAF Performance for jumping

Figure 7: Performance of jumping skill

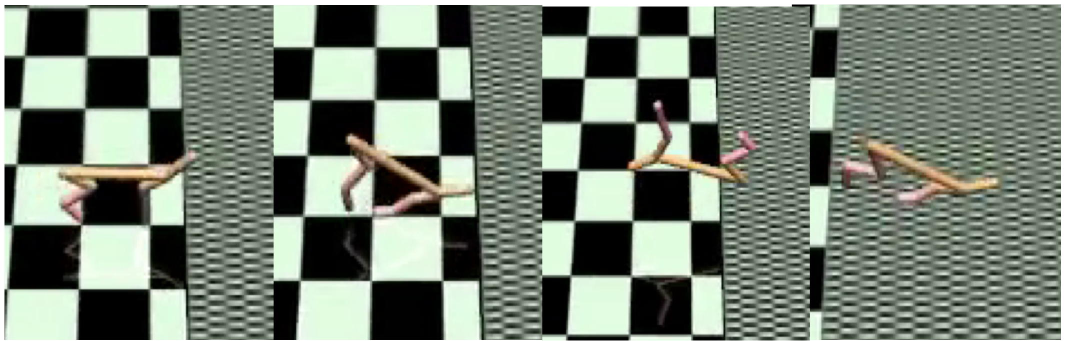


Figure 8: Agent flipping after a jump

4.2 Jumping

We faced several issues in training the models and achieving acceptable performance for this skill. The comparative performances of the DDPG and NAF models are in Figure 7.

We experimented with several reward functions to incentivise the agent to jump properly. Some of the many reward modifications included 1) Large rewards for achieving stability on the box 2) Rewarding the agent for jumping in the error function 3) Lowering and then removing the penalty on joint torque, and finally 4) Creating a set of sparse rewards to encourage the agent to get on the box. However even with 100k training episodes we were unable to get the agent to learn correctly. There were policies that had up to a 30 % success rate. However because of the multi-modality of these jumping policies, it converged to an intermediate and inferior solution in all 5 agents. When penalty was kept on the joint torques, the agent would simply give up and place it's torso on the ground waiting for the episode to end as this was its idea of an optimal policy. This was the behavior even with a reward 5 orders of magnitude greater than the penalty on its movement.

Again the NAF outperforms the DDPG model for this skill. One of the major drains on the reward in this skill was the fact that the agent would end up flipping after the jump making the activity of the jump undesirable (Figure 8).

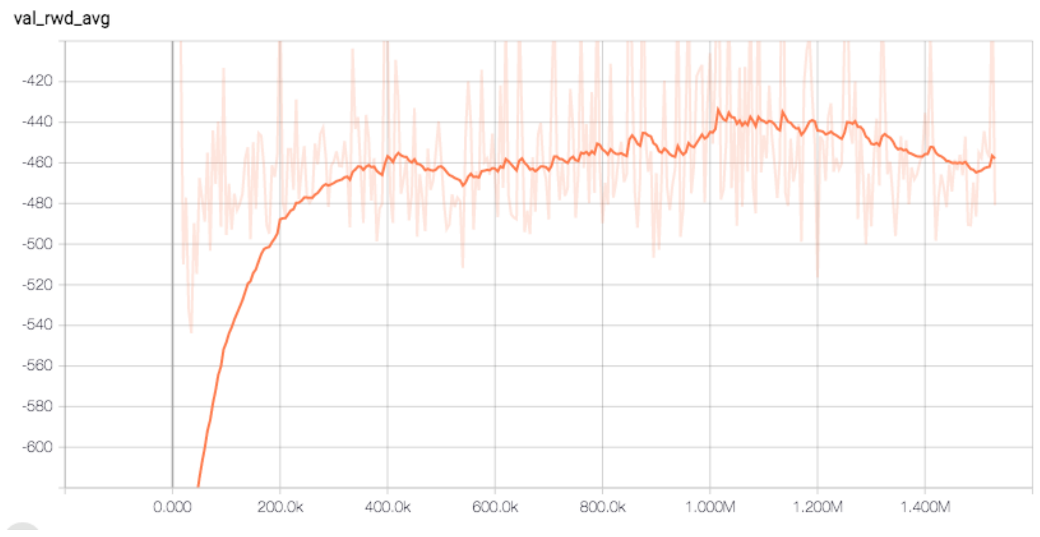


Figure 9: Performance of Meta controller in the Steps environment

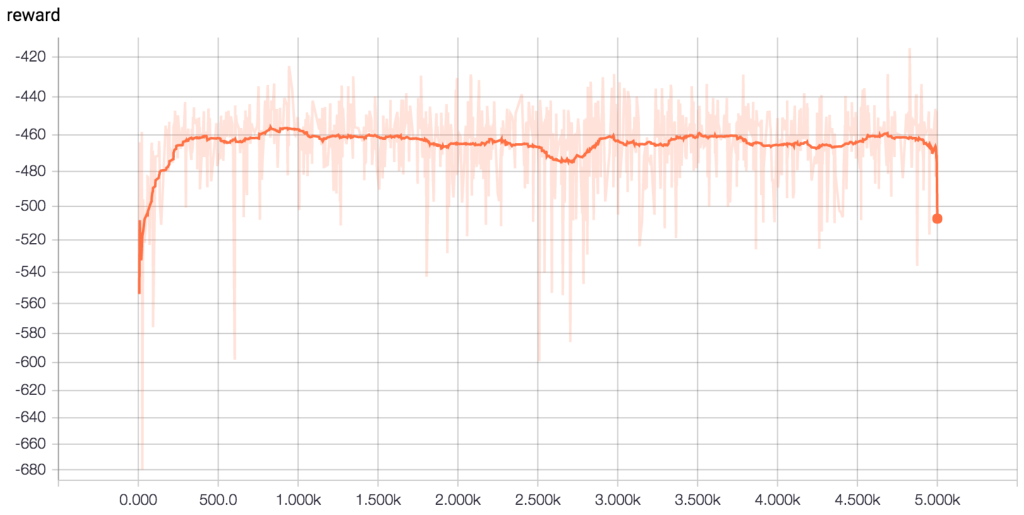


Figure 10: Baseline performance of DDPG in the Steps environment

4.3 Meta-controller

We first experimented with the DQN meta-controller with the sub-goals being fulfilled by DDPG models for the 'Steps' environment (Figure 9). The model was able to beat the baseline (Figure 10)but only by a small margin of around 20 reward points or so.

One of the main reasons why we are not able to beat the baseline by a large margin simply using DDPG models is because the jumping skill was a tough basic skill that our agent had not fully mastered.

5 Conclusions

NAF models performed much better than DDPG models for the basic skills. Therefore we definitely can improve the performance of the meta-controller with the sub-goals being fulfilled by NAF models.

One of the key problems we realized was that jumping in itself may not be a basic skill as we had hypothesized it to be. For instance, for an agent to jump without flipping (a very expensive action), the agent would have had to slow down before launching on the jump. Therefore the basic skill of jumping in itself will benefit from the use of a meta-controller or ’skill tree’.

Designing appropriate reward functions and specialized environments for basic skills may improve the performance of the agent in general.

References

[1] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[2] Carlos Florensa, Yan Duan, and Pieteer Abbeel. Stochastic Neural Networks for Hierarchical Reinforcement Learning. *submitted to ICLR 2017*, 2017.

[3] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*, 2016.

[4] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] Ramnandan Krishnamurthy, Aravind S. Lakshminarayanan, Peeyush Kumar, and Balaraman Ravindran. Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks. *CoRR*, abs/1605.05359, 2016.

[6] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

[7] Guy Lever. Deterministic policy gradient algorithms. 2014.

[8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[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] Yichuan Tang and Ruslan R Salakhutdinov. Learning stochastic feedforward neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 530–538. Curran Associates, Inc., 2013.