A Hierarchical Approach to Multi Tasking

Sidharth Ramesh
Department of ME
Indian Institute of Technology Madras
me15b132@smail.iitm.ac.in

Ganga Meghanath
Department of EE
Indian Institute of Technology Madras
gangamegha29@gmail.com

Abstract

Reinforcement learning has advanced in many areas of research. Hierarchical reinforcement is such an area which has proven to be an architecture that can be employed for solving a wide variety of tasks in a human level understanding. The paper presents two different successful architectures that have been integrated with multi-tasking algorithms inorder to make it capable of playing a set of games using the same architecture and learning agent.

1 Introduction

Reinforcement learning often focuses task specific solutions to problems. Often we require an agent to be able to perform equally well in a set of divergent tasks. Looking at the problem from a human perspective, we often have a set of actions that we perform in a multitude of situations. These set of actions can be combined together to form options. Previous works [4] attempts to use an A3C[3] to tackle the problem. In this paper, we attempt to use the feudal network framework[5] and the option-critic framework [1] to train the agent to learn multiple options to handle different situations and perform multiple tasks.

2 Evaluation Criteria

In order to assess and compare the performance of multiple games, we have adopted an evaluation criteria, which was proposed by jha et al [4]. We define this value, p, as the ratio of the maximum score achieved by the agent to the maximum baseline score. The overall performance of the agent is defined as the mean of p over all the games

$$p = score/baseline_score \tag{1}$$

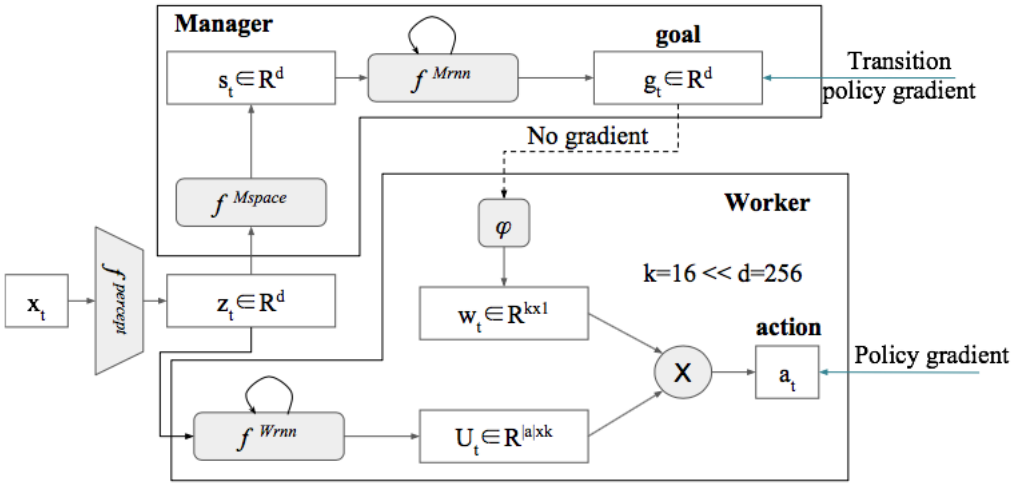
$$p_{am} = (\sum p)/k \tag{2}$$

3 Feudal Network

The framework employs a Manager module and a Worker module. The Manager operates at a lower temporal resolution and sets abstract goals which are conveyed to and enacted by the Worker, who takes the primitive actions in the given environment. The decoupled structure encourages the emergence of temporally extended sub-policies(associated with different goals set by the Manager) and facilitates long timescale credit assignment.

3.1 Architecture

The schematic illustration of the FuN architecture has been portrayed below :



3.2 Experiments

The experiments were tested in OpenAI Gym environment for the Atari games. An implementation of FuN by dmakian, available as open source on GitHub was used for running the experiments and multitasking algorithms were integrated into the same. The code was tested for a variety of games and the results were analyzed to ensure the correct implementation of the original algorithm.

A sample of the test results for individual games has been shown below.

3.2.1 Space Invaders

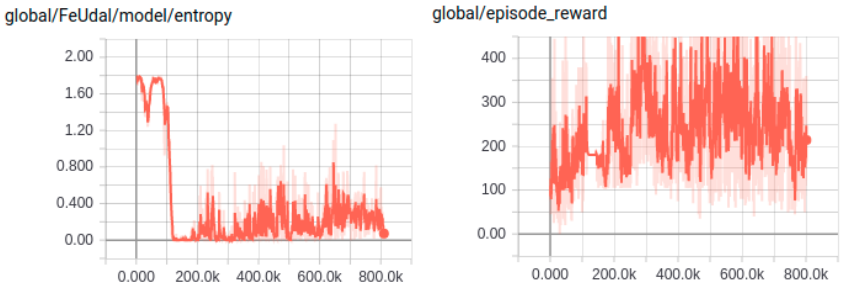


Figure 1: The plot has been generated after 800K steps inorder to show the variation trends

3.2.2 Assault

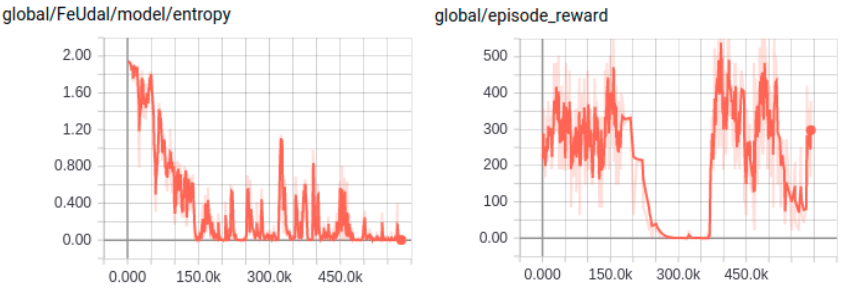


Figure 2: The plot has been generated after 800K steps inorder to show the variation trends

3.2.3 Centipede

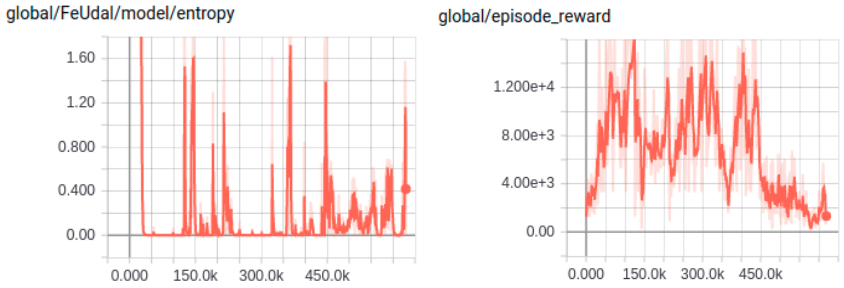


Figure 3: The plot has been generated after 800K steps inorder to show the variation trends

3.3 Active Sampling Feudal Multitasking

This method uses the adaptive active sampling technique [4] to sample games to play. The feudal network [5] contains one worker and manager. Using this setup the Multi-Tasking algorithm selects games based on their performance parameters. The baseline scores for the network was obtained from various sources. These were used to train a set of three games to measure their performance.

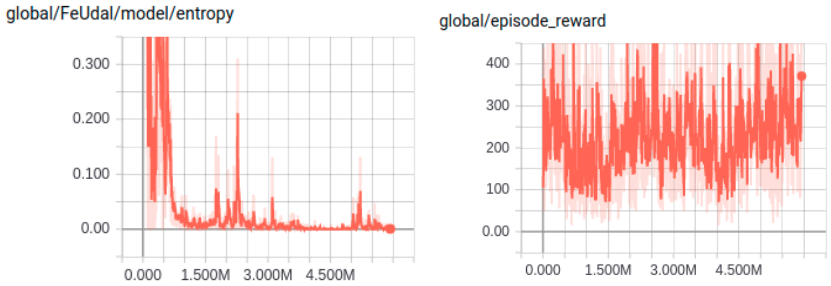


Figure 4: The plot has been generated after 800K steps inorder to show the variation trends

3.4 Inference

The feudal network architecture however did not perform to human standards when trained on. This could either be due to restricted representation power of a single manager, or that the goals were too complicated for the worker.

4 Option Critic

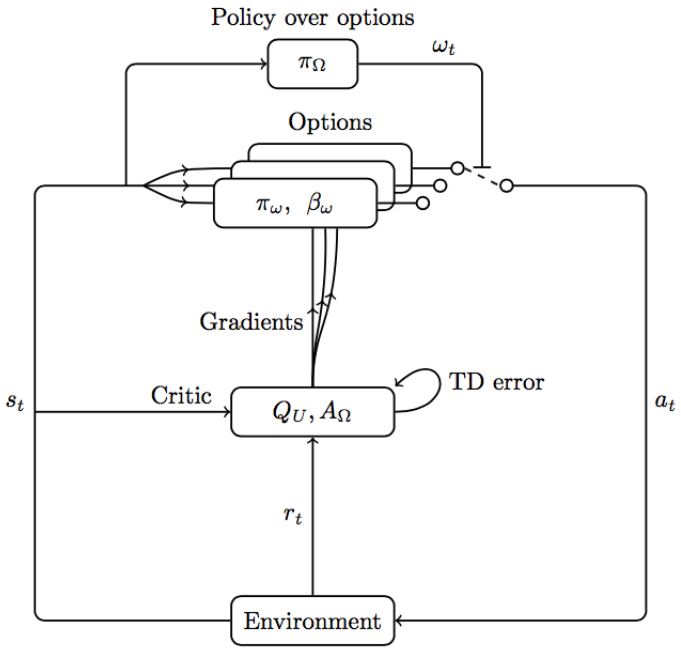
Lack of satisfactory results from the FuN architecture led to the use of the Option Critic Architecture [1] as a replacement for the Feudal Network. Option Critic comprises of two main components :

- Policy gradient methods
- Options

The existence of the underlying MDP offers the possibility of learning about many different options in parallel. The algorithm was found to learn meaningful and temporally extended behaviors.

4.1 Architecture

The schematic illustration of the Option Critic Architecture has been portrayed below :



4.2 Multitasking Algorithms

Three different multi-tasking algorithms were implemented and their performance was measured for different game combinations.

4.2.1 Active Sampling Agent

Algorithm 1 Multitasking using Active Sampling

for i in range(k) : **do**
    $p_i = \frac{1}{k}$
end for
for train_steps in range(t) : **do**
    if t **then**rain_steps $\geq l$:
        for i in range(k) : **do**
            $a_i \leftarrow s_i.average()$
            $m_i = \leftarrow \frac{ta_i - a_i}{ta_i \times \tau}$
            $p_i \leftarrow \frac{e^{m_i}}{\sum_{q=1}^{k} e_q^m}$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\boldsymbol{e}$
            $S \leftarrow S'$
        end for if S' is terminal
    end if
    $j \approx p$
    $score\_j \Leftarrow Train_one\_episode(T\_j)$
    $s.append(score\_j$
    if then$s\_j.length() > n$
        $s\_j.remove\_oldest()$
    end if
end for

4.2.2 Doubling UCB Agent

Algorithm 2 Multitasking using Doubling UCB

for i in range(k) : **do**
    $p_i = \frac{1}{k}$
end for
for train_steps in range(t) : **do**
    if t **then**rain_steps $\geq l$:
        for i in range(k) : **do**
            $a_i \leftarrow s_i.average()$
            $m_i = \leftarrow \frac{ta_i - a_i}{ta_i \times \tau}$
            $p_i \leftarrow \frac{e^{m_i}}{\sum_{q=1}^{k} e_q^m}$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\boldsymbol{e}$
            $S \leftarrow S'$
        end for if S' is terminal
    end if
    $j \approx p$
    $score\_j \Leftarrow Train_one\_episode(T\_j)$
    $s.append(score\_j$
    if then$s\_j.length() > n$
        $s\_j.remove\_oldest()$
    end if
end for

4.2.3 Doubling DQN Agent

4.3 Experiments

The implementation of the code released by the authors themselves (on GitHub) were used for conducting the experiments and the multitasking algorithm was integrated into the code and the performance was observed. The architecture was tested on the Atari environment.
The experience replay for each game (used to train the multitasking agent) were initialized separately, ie., separate replay buffers were maintained for each game. This helps avoid catastrophic forgetting [2], and stabilizes the gradient. The multitasking was carried out using a fixed baseline as well as a doubling baseline. For the doubling agent, the baselines were doubled each time the agent reaches near the threshold. This is implemented so as to remove the human baseline requirements and also to enhance the performance of the agent in a way a human would : "beat their own high score". This method was utilised by Jha et al [4], in his paper on Multitasking using A3C.

4.3.1 Performance on individual Games

As mentioned above, the codes have been released by the authors and were found to be reliable. The results obtained for the four main games mentioned in the paper [1] were verified through testing before integrating different multi-tasking algorithms into the the code.

4.3.2 Active Sampling Agent

The results obtained for different game combinations is as shown below:

++ Games	Number of steps	Baselines	Probabilities	Avg p_{am}
MS Pacman	1754123	6518	0	7.4763
Seaquest		2700	0.10703177	
Beam Rider		2200	0.41248498	
Asterix		2400	0.48048325	
Alien	278727	2700	0.12089808	0.1411
Seaquest		2700	0.41963794	
Beam Rider		2200	0.40080153	
Asterix		2400	0.05866244	
Centipede	4202610	3300	8.36400045e-10	1.5689
Space Invaders		1200	2.94769224e-02	
Assault		1900	7.16040419e-02	
Phoenix		5384	5.59409194e-02	
Star Gunner		40000	8.42978115e-01	

The high p_am in the first row is due to the fact that Pacman exceeds it's baseline by a large margin whereas others perform poorly. Hence, p_am is not a right measure of performance.

4.3.3 Doubling UCB Agent

A sample result obtained is as shown below:

Games	Number of steps	Probability	Avg p_{am}
Phoenix	3001551	0.68637276	Depends on baseline
Asteroids		0.0528381	
Assault		0.05706809	
Star Gunner		0.05571698	
Centipede		0.04436421	
Breakout		0.055396	
Space Invaders		0.04824386	

The average p_{am} was found to be less and there wasn't considerable improvement in the same after using option critic architecture.

4.3.4 Doubling DQN Agent

A sample result obtained is as shown below:

Games	Number of steps	Probability	Avg p_{am}
Phoenix	12103508	4.12525631e-04	Depends on baseline
Asteroids		7.12146679e-04	
Assault		4.62819618e-04	
Star Gunner		3.48223544e-04	
Centipede		2.07310395e-04	
Breakout		9.97856972e-01	
Space Invaders		1.91482650e-09	

The average p_{am} was found to be less and there wasn't considerable improvement in the same after using option critic architecture.

5 Future Work

The vanilla experience learning can be improved by using a prioritized experience replay. There might be a very rare event with a high reward that occurs only once. The vanilla experience replay might lose this memory when the replay buffer gets filled.

In the case of the FuN, multiple games could be played at the same time by assigning different number of threads to each game according to their probabilities of being selected.

6 Conclusion

With multiple tasks that needs to be handled, multi tasking agents are a requirement for future applications. Tasks including robots walking and performing tasks like balancing an object would require multitasking in continuous domain. In this paper we have attempted to use hierarchical methods to deploy a multitasking agent capable of playing multiple games. Although the results obtained weren't fruitful, there is still good chance that it could have been successful with more recent advances in hierarchical reinforcement learning.

References

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *CoRR*, abs/1609.05140, 2016.

[2] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[3] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. pages 1928–1937, 2016.

[4] Sahil Sharma and Balaraman Ravindran. Online multi-task learning using active sampling. *CoRR*, abs/1702.06053, 2017.

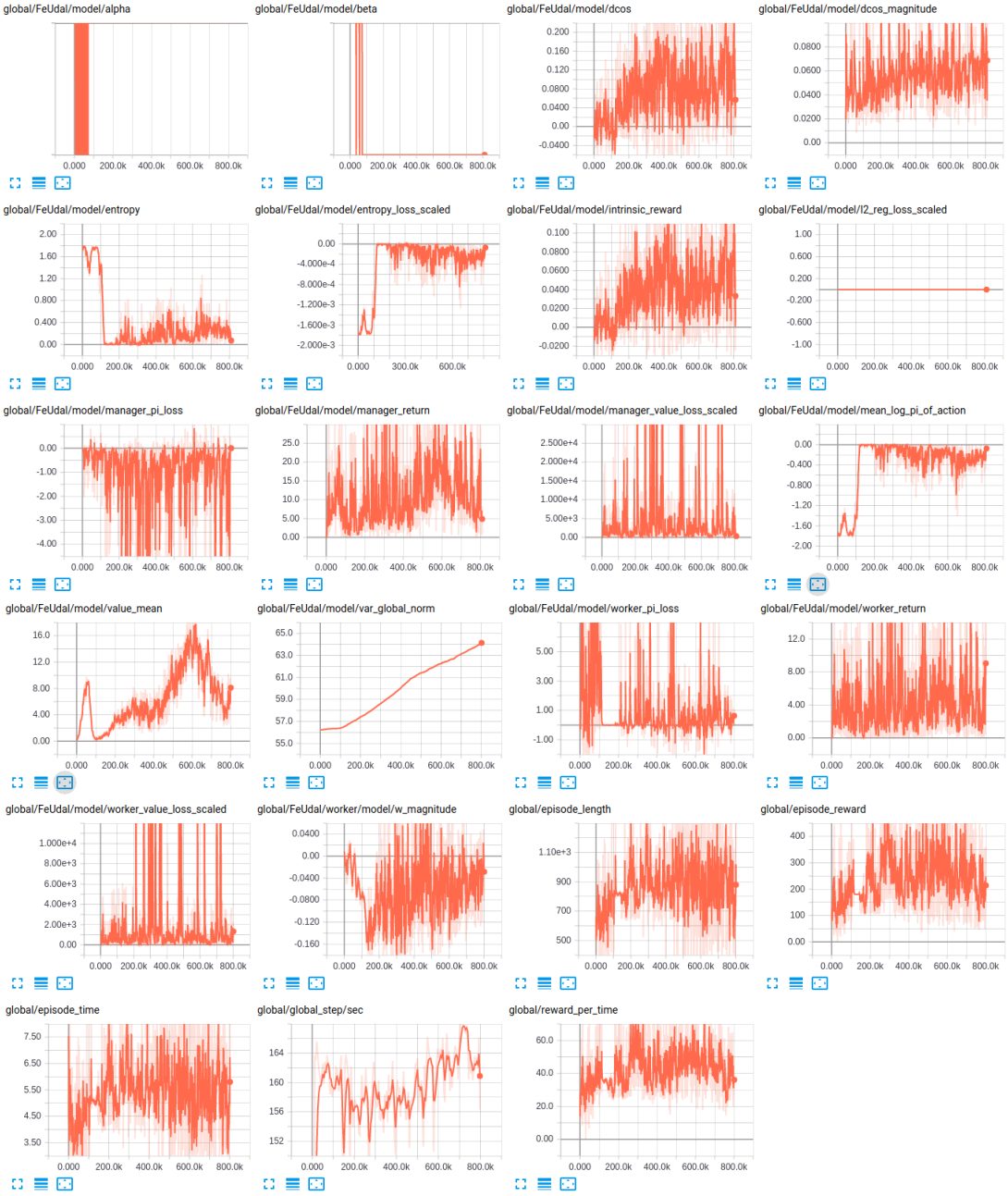
[5] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161, 2017.

APPENDIX

A FuN

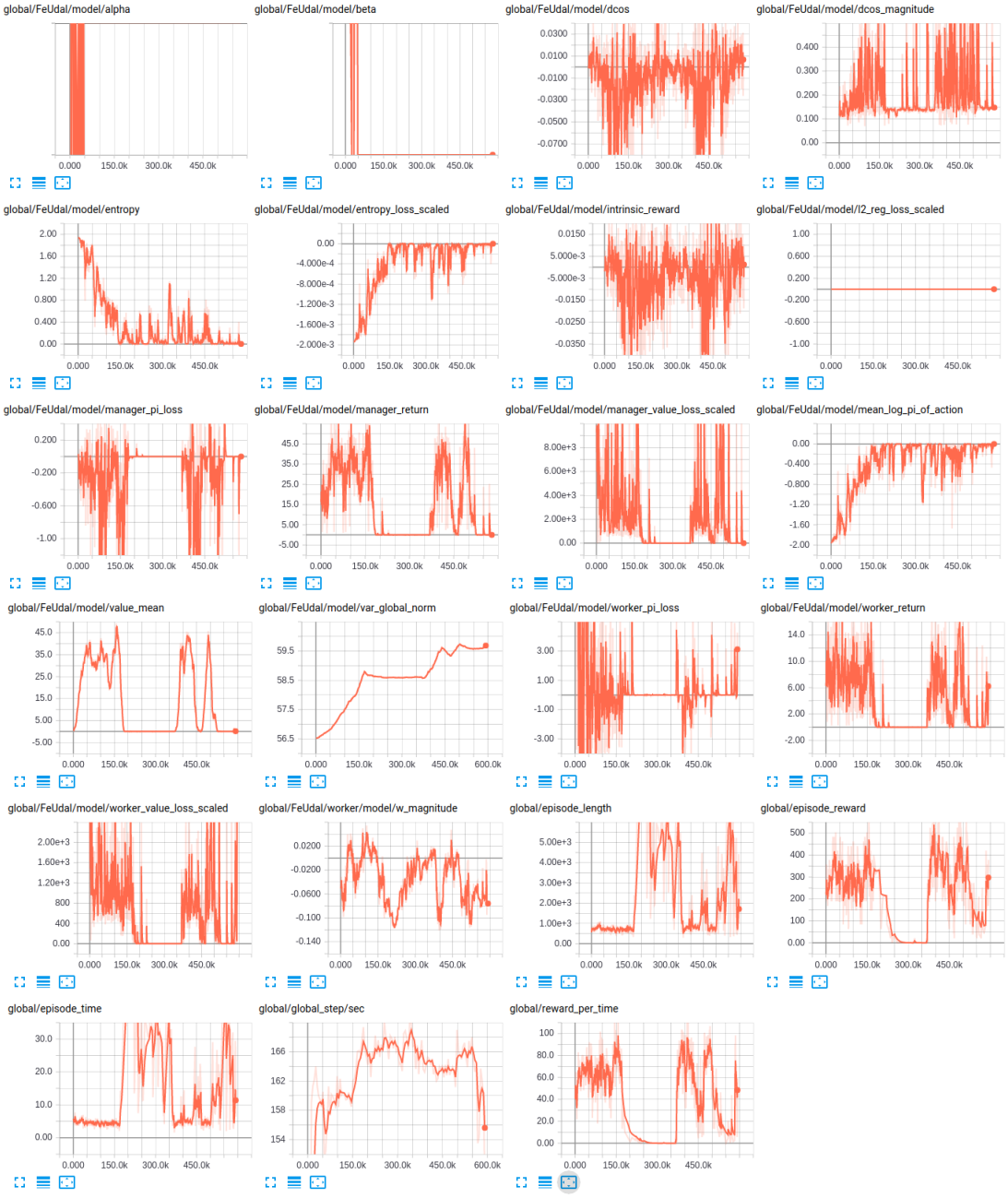
A.1 Space Invaders

The sample initial test results have been shown below :



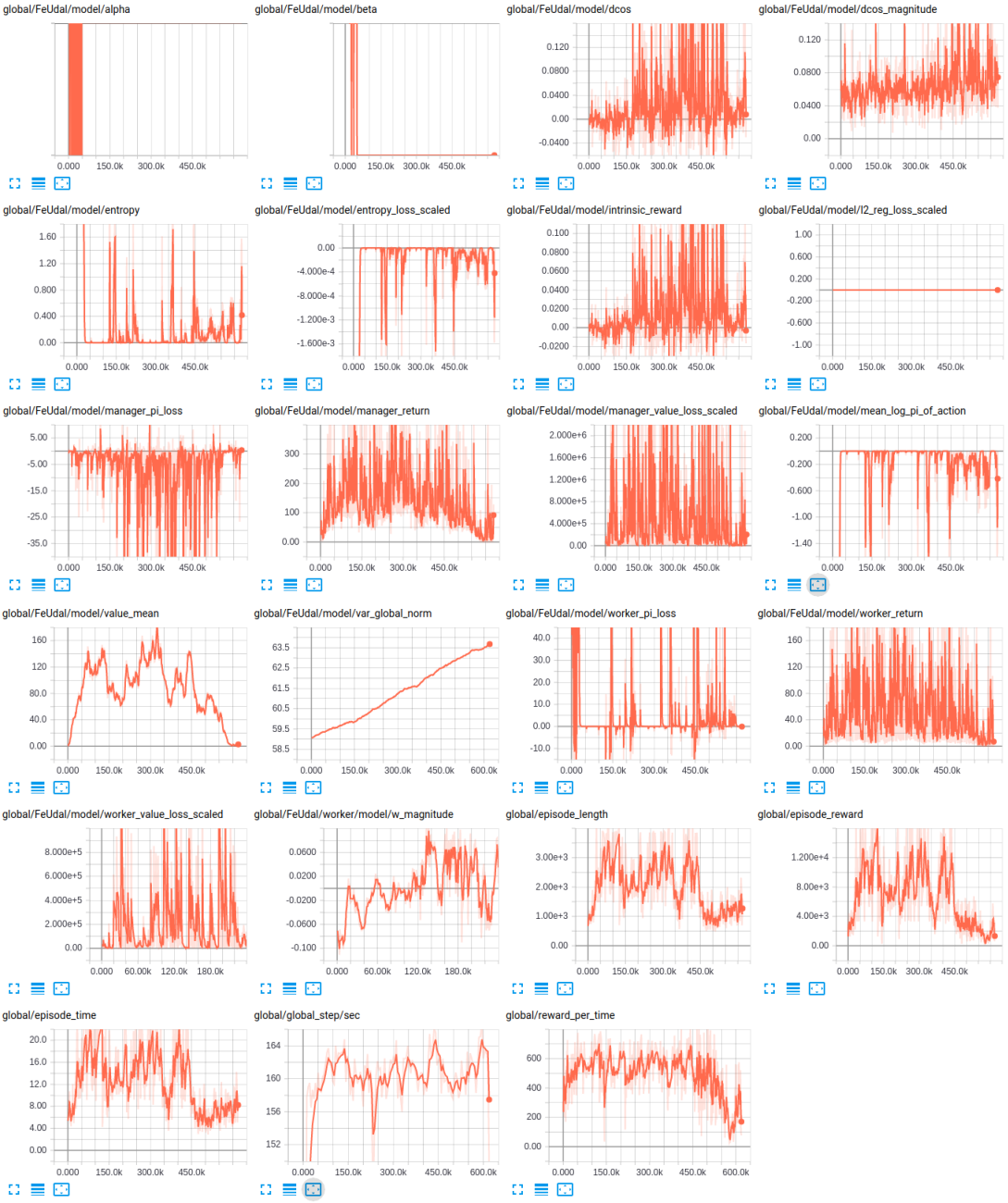
A.2 Assault

The sample initial test results have been shown below :



A.3 Centipede

The sample initial test results have been shown below :



B Active Sampling Feudal Multitasking

The complete test results have been shown below :