Special Problems: CS8903

Fall 2019

FINAL REPORT

Mihir Mavalankar
(mmavalankar3)

Supervising Faculty Member

Dr Frank Dellaert

(frank.dellaert@cc.gatech.edu)

Table of Contents

1. Problem addressed and relevance: -

The high level intuition of this project is that gaits of animals with bilateral symmetry are symmetric in nature, and we want to explore if exploiting this fact in reinforcement learning and help simulations learn more symmetric motion or that the training time for reinforcement learning models can be reduced. All the experiments are done using the PyBullet Ant simulation. (https://github.com/openai/gym/blob/master/docs/environments.md#pybullet-robotics-environments) This particular simulation was chosen due to interest in learning more about spider like simulations gaits and as it also satisfied the requirement of bilateral symmetry that is talked about here.

There is a great amount of interest currently in reinforcement learning to create models for gaits in animals and robots as it abstracts away the need to optimize trajectories of each of the moving parts itself and handle it by converting it into an MDP learning problem. While it has many advantages there are some clear drawbacks like large training time, sparse rewards and others. The relevance of this work is in addressing the large training time for model free reinforcement learning algorithms for learning gaits. If our hypothesis about bilateral symmetry is correct, we could save significant training time as our network model (weight sharing, discussed later) have half the number of parameters than other.

2. Related work: -

There are a number of papers talking about simulating animal and bipedal gaits and while a lot of them were referenced initially (all present in the bibtex file and drop box folder for Spider RL), one paper recently published was the most relevant. An excerpt from the paper is "Human and animal gaits are often symmetric in nature, which points to the use of motion symmetry as a potentially useful source of structure that can be exploited for learning. By encouraging symmetric motion, the learning may be faster, converge to more efficient solutions, and be more aesthetically pleasing." This is from the recently published paper named "On Learning Symmetric Locomotion" (https://www.cs.ubc.ca/~van/papers/2019-MIG-symmetry/) which I will refer to as the MVDP paper for brevity. We have the same high-level intuition and are trying to implement it by making the state and action vector symmetric and using weight sharing in all the policy network layers. This line of experimentation was started by us before this paper was published and after learning about this paper, we have tried to validate their findings in our experiments too.

Here we have used the Proximal policy optimization (PPO) algorithm proposed in this paper (https://arxiv.org/abs/1707.06347) from OpenAI. They have also given a implementation of PPO and other reinforcement learning algorithms in Tensorflow which on this public repository (https://github.com/openai/baselines). On advice from Dr Liu, I have used these implementations as a baseline and have added and modified models for our experiments. Using this helped us as I did not have to write the code for all of these models from scratch.

3. The approach: -

We tested the hypothesis discussed in the start with different approaches using the openai baselines framework and the tests were done on the Ant simulation mentioned before. The four approaches that we have tried out are: -

- PPO (Vanilla): - This is just the implementation of PPO that's already in openai baselines and is used as a benchmark for the other methods.
- PPO with symmetric input and output: - Here we use PPO by make the input state vector symmetric/mirrored. We also assume that this will give a symmetric output, so we rearrange the output according to the requirements for PyBullet for the simulation.
- PPO with symmetric input and output and weight sharing: - This has the same setup as the previous model but it has weight sharing added to it too.
- MVDP NET method: - This is the 'NET' method described in paper (link at the end) and implement in Tensorflow as part of the openai baselines framework.

Next, I will talk about the method of weight sharing in the policy network we have implemented here which is unique to this work.

Weight Sharing: -

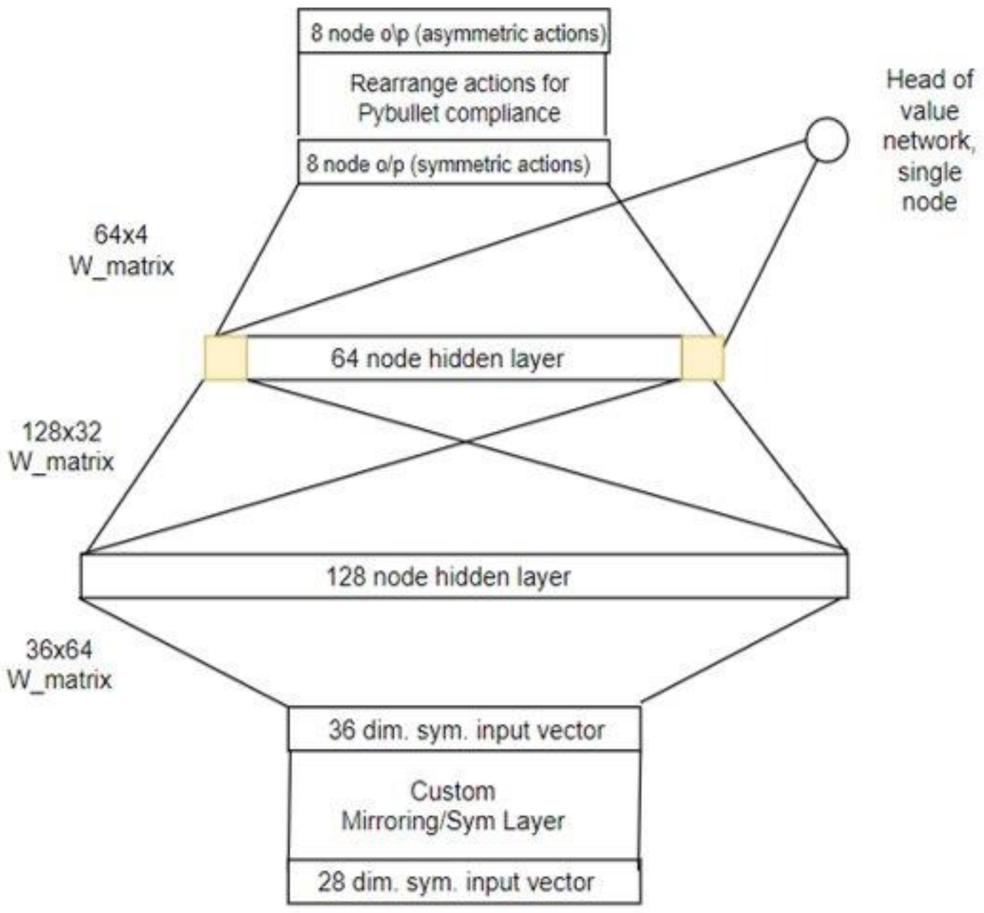
Weight sharing here is inspired by the weight sharing in convolutional neural networks where a set of weights/filters are passed over the entire image. Here we use the weights on the right side of the fully connected networks on the left side as well.

More formally for every fully connected layer we only have half the required weights in the layer and use the half weight matrix and it's flipped version for a forward pass. During back propagation only the half weight matrix is updated and hence it is computationally less expensive. The modified fully connected neural network looks like: -

$H = \text{relu}\ (\ [W \mid \text{flip}(W)\ ].X\ ) + b$

The green line in the graph shows the results for this layer. To implement this layer, we also make the input state vector symmetric/mirrored around the center of the vector. So, each side has the common state variables and one side has all the left body side state variables and the right side has the right ones. The orange line is has this sort of symmetric input and output but no weight sharing and this is used as a control for comparison.

Below is a diagram of the policy and value networks defined in the PPO algorithm with the changes that I have made for implementing weight sharing. As you can see from the diagram each fully connected layer only has half the weights as expected. Here you can also more clearly see the summarization of the input and output.


8 node o\p (asymmetric actions)
Rearrange actions for Pybullet compliance
8 node o/p (symmetric actions)
Head of value network, single node
64x4 W_matrix
64 node hidden layer
128x32 W_matrix
128 node hidden layer
36x64 W_matrix
36 dim. sym. input vector
Custom Mirroring/Sym Layer
28 dim. sym. input vector

(Diagram of the weight sharing network with symmetric input and output)

About MVDP NET method: -

The NET method in the MVDP paper was implemented by me in the open ai baselines framework. Their paper skips over some of the pats in the implementation but the paper and their code together helped me implement it. I'll just mention the main differences here which are, the use the state input and its mirror (so state vector id double the size) and run two forward passes through the policy network. The average the output and use it for gradient updates. Their policy network also has twice the parameters that we have and so that explains their better performance partly. Lastly they have tried many different methods to enforce symmetry which we could investigate further.

Modifications to OpenAI baselines: -

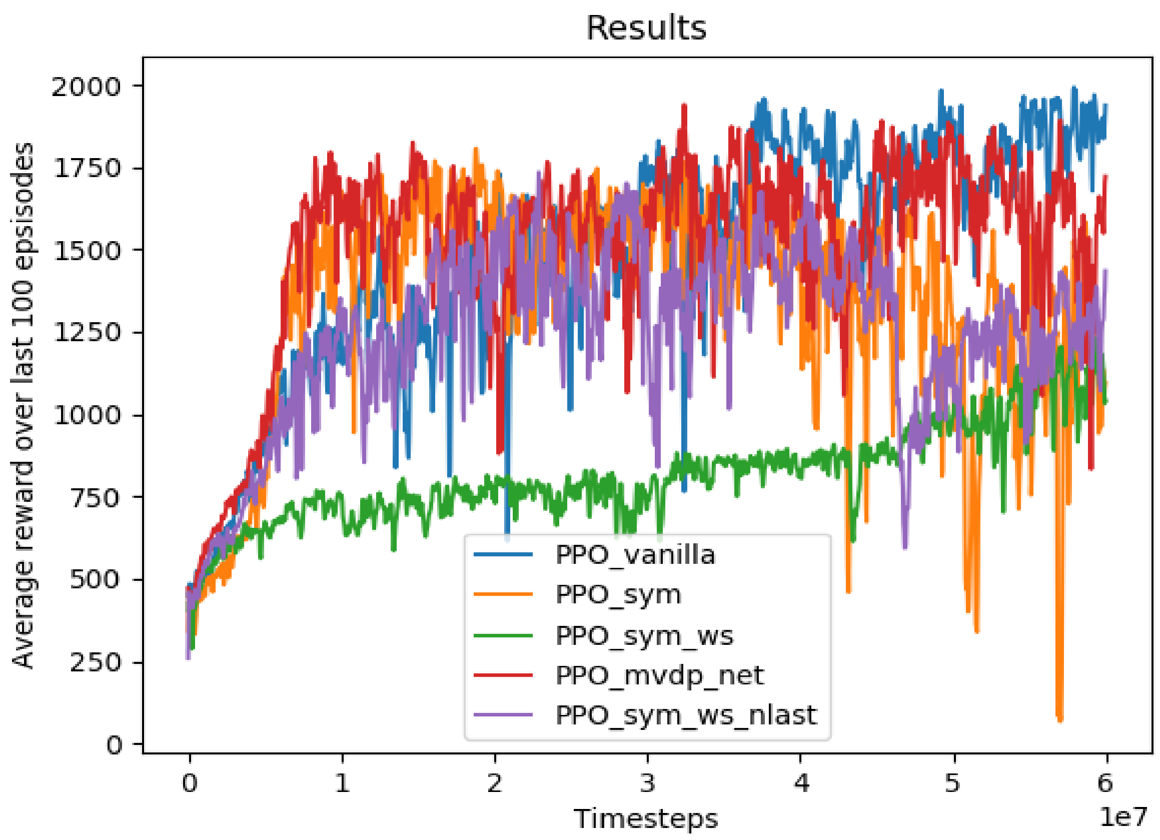
Here I give a brief overview of the code in openai baselines to better understand how I have modified the code for this project. The openai baselines code in written in Python 3.5 and uses

Tensorflow 1.14. I'll give some helpful tips here to navigate that codebase as it can be intimidating at first glance. A good knowledge of TensorFlow is necessary to fully understand the code.

The run.py file is the main launching program and it calls the a file according to the algorithm that you passed as a parameter ./ algorithm_name /<algorithm_name>.py. This file in turn calls a file called model.py. In the model.py file there are calls to files in the common folder and a2c folder which define the policy and value network, the distribution type and what layers will be used. The most import files in the common folder are policies.py, distributions.py and models.py. The util.py file in a2c folder has the definitions for each layer.

4. Results: -

The graph shows the results (average reward earned by Ant) for the **four approaches** that I trained using my modified version of openai baseline implementation. The newest one in the MVDP one. I looked at their paper and code (PyTorch) (paper skips through a lot of the implementation) and created their network as a new model in openai baselines in TensorFlow. The other two model that are ours are the Symmetric input output with/without weight sharing. Lastly (blue) I have also included results for openai baseline's Vanilla PPO model.



All of these models use PPO and are trained with the models I have made in openai baselines using TensorFlow.

Blue – PPO of openai baselines with no change
Orange – PPO with symmetric input output without weight sharing
Green – PPO with symmetric input output with weight sharing
Red – NET method in MVDP implemented by me using openai baselines in Tensorflow (See Page 4 of MVDP paper. Code link provided to for more thorough look)
Purple (Extra model) – This is the same setup as weight sharing (green) except the last layer doesn't have weight sharing

The videos of the rollouts are given in the important links section at the end.

5. Discussion of Results: -

The results of these experiments definitely need more analysis and more experiments. Our approach here is different in terms of execution from the MVDP paper and so while it is a good reference the results are not directly comparable. Although have discussed these differences and comparisons in some of the previous reports that I have submitted over the semester. Some of the important takeaways from the graphical data and the rollout videos is: -

- The Vanilla PPO model (blue) did the best, followed by the MVDP NET model (red), Symmetric input output model (orange) and last one was the symmetric input output with weight sharing (green).
- Considering the fact that half of the gradient information is not used in the weight sharing model it still did pretty well (1100 max reward) and the video does show that the ant is can walk. Another point to notice is that the model is quite stable (very little variation from episode to episode) compared to the other models as it only has half the parameters as compared to the other models.
- The video of the rollout for the weight sharing model shows that the Ant body turns itself first and then starts walking towards its goal. We need to explore if this is a one off occurrence of this rollout or if is seen in other rollouts outs and if so, explain why that happens.
- The MVDP model (red) learns very fast (reaches 1700 reward in 10M time steps) which is intuitively expected as it does two forward passes and has more state information in both those passes.
- The symmetric Input Out model (orange) loses rewards towards the end. I believe this might be due to local minima issues and would be different in another training run for this model.
- The model without weight sharing in the last layer (purple) also does very well even though there is only a small increase in the number of parameters. This needs to be investigated further.

6. Future Work: -

As mentioned in the results there is more experimentation needed and so there are multiple avenues to explore going forward. Some of which are: -

- Train more models and see the rollouts for them so that we don't make inferences only over a small number of rollouts.
- Writing unit test to understand exactly how the weights and gradients in a smaller simpler network are, and how they change.
- Test out other bipedal or quadruped simulations and see the results for those and make a comparison metric. Some of the other simulations would be the bipedal walker in PyBullet and even other symmetric robot simulations.

7. Meta-Learning: -

- I learnt a lot about reinforcement learning methods for continuous action spaces like PPO, how they and why they work, their use cases and limitations etc. This has only made me more curious to learn more about reinforcement learning in general.
- Due to this course, I now have a good understanding of the openai baselines framework and how one can use it to experiment with reinforcement learning algorithms.
- I have also learnt more about Tensorflow 1.x due this project as the openai baselines framework as all the code I wrote was for making Tensorflow models in it.
- During the development process I have learnt the importance of writing unit tests with the code and how to best utilize computing resources as these experiments are very compute intensive. I plan to make this feedback a habit going forward in any development work.

8. Important Links: -

Links to videos: -

- **Vanilla PPO (Blue):** https://www.youtube.com/watch?v=dQZUXFg6Lss
- **PPO Sym Inp/Op (Orange):** https://www.youtube.com/watch?v=jekF0PbxRjo
- **PPO Sym Inp/Op weight sharing (green):** https://www.youtube.com/watch?v=2zVFBBiuDho
- **PPO MVDP NET method (red):** https://www.youtube.com/watch?v=I_BYwuxZLHU
- **PPO Sym Inp/Op weight sharing/ except in last layer (purple):** https://www.youtube.com/watch?v=3E8fPZLulHc

My code: https://github.gatech.edu/mmavalankar3/openai_baselines
The MVDP paper: (https://www.cs.ubc.ca/~van/papers/2019-MIG-symmetry/)
MVDP Code: https://github.com/UBCMOCCA/SymmetricRL

References :(all in common bibtex file for Spiders group)