Spot Controller Writeup

https://github.com/clbeggs/Spot_Controller‘

Chris Beggs

December 12, 2020

1 Abstract

This project tackled the problem of quadruped locomotion and navigation. There are many different ways to solve this problem, control and trajectory optimization, deep reinforcment learning, etc. But in an attempt to maintain generality, Model Agnostic Meta Learning proposed by Finn et. al[1] was used. The meta learning framework was utilized for trajectory optimization of Boston Dynamic's Spot quadruped given unseen obstacles and environments.

2 Introduction

The problem of few shot learning is concerned with quickly adapting to a new set of tasks. In this project, I implemented the framework presented in *Model Agnostic Meta Learning*(MAML)[1] that allows a meta network to find the best initial weights to generalize to unseen tasks. This method was used over various trajectory optimization and deep RL techniques because of it's simplicity compared to other methods addressing the same problem.

3 Related Work

Model Agnostic Meta Learning (MAML)[1] is a meta learning model for few shot learning. It is able to generalize by learning good initial weights in the model, where training for a small number of iterations is able to adapt to a new task.
MAML++[2] is an improved version of MAML, which addresses some of the stability concerns introduced with the original MAML paper. In the improved model, they both increase model stability and reduce computational overhead. This is done by shared modified batch normalization, a modified meta update step, and shared inner loop learning rates.
Other methods were considered, such as DeepGait[3] which is a heirarchy based planner, consisting of a gait planner and controller. The planner decides footstep location, where the controller is responsible for executing the plan given by the planner. While this does achieve great results, the gait pattern is constrained, making faster gait patterns and different complex movements not possible.

4 Methodology

The MAML meta learning framework was implemented in the context of moving Boston Dynamic's Spot robot find a way to a goal point given obstacles. The stability trick adopted from [2] was the Multi-Step loss optimization method, where instead of using the original MAML

meta loss function:

$$\theta_{k+1} = \theta_k - \beta \nabla_\theta \sum_{b=1}^{B} \mathcal{L}_{T_b}(f_{\theta_N^b})$$

the following was used:

$$\theta_{k+1} = \theta_k - \beta \nabla_\theta \sum_{b=1}^{B} \sum_{i=0}^{N} v_i \mathcal{L}_{T_b}(f_{\theta_i^b})$$

Where instead of updating the meta network after the entire task rollout, it's updated after every step of each rollout.
Following the original MAML, the gradient updates for each batch of sampled trajectories of a given task $\mathcal{D}_{\mathcal{T}_i}$ were computed using vanilla policy gradient(VPG). And the outer loop optimization step, consisting of the gradient update of the meta model given the loss of the inner loop models, was computed using trust region policy optimization(TRPO).[4]

4.1 Model Architecture

Policy Net
Fully connected network, 3 hidden layers with sizes 256, ReLU activation functions.
A Normal distribution was parameterized by the output of the model:

$$a_t \sim \mathcal{N}(f_\theta(\text{Observation}), \Sigma)$$

Note: In Pytorch, gradients are able to flow through this non-differentiable operation.

Value Net
Fully connected network, 2 hidden layers with sizes 128, ReLU activation functions.
The value net was used to map observations to state value, which was then used to compute the surrogate advantage used in the trust region policy optimization implementation.[4]

4.2 Reward Function

The following were used as additions to the reward function:

- Change in distance to goal point from previous time step to current time step: $|(g - x_{t-1})| - |(g - x_t)|$
- Change in position from last time step, to encourage movement: $\alpha|x_t - x_{t-1}|$
- Body height $> \alpha$, to encourage not falling over: y
- Penalized maximum change in joint angles, to encourage smoother trajectories: $\max(\Delta\theta_{motor})$
- Penalized angle of rotation for z axis, to discourage falling and being in unrepairable state: $\beta\gamma\theta_z$, where γ is magnitude of Axis-Angle rotation param.
- Survive award, awarded at every time step

4.3 Training

As training was done on a laptop, networks had to be kept at the smallest possible size. 200 epochs, 1 task, and 5 sampled trajectories took 2 hours to train. Smaller networks were tried, but didn't result in any promising results.

5 Results

The results were unstable and very slow at training, thus I wasn't able to get good results. The best results I was able to get was Spot lunging forward, falling off balance, and making walking like motions while upside down. Final results of epochs also varied wildly given the model initialization. Lots of tweaking and tuning the model and implementation was not successful, thus a reproducible result was not found.

6 Discussion

It is a huge dissapointment that I couldn't get solid results with the time I had. Training was a big bottleneck as MAML is not known for being stable, and it took a very long time to train given my limited compute power. But, I was able to learn a lot more about policy gradients, meta learning, and pytorch with the project which is an upside. But, further time will be given to ironing out and improving the implementation.

7 Conclusion

In conclusion, the meta learning presented by Finn et. al. was shown to be unstable, and selectively adopting a subset of the improvements from MAML++ did not improve the stability. And as stated above, more work will be devoted to improving the existing implementation, using Mujoco, using robot models with smaller action spaces, and devoting more time to training larger networks.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. arXiv: 1703.03400 [cs.LG].

[2] Antreas Antoniou, Harrison Edwards, and Amos Storkey. *How to train your MAML*. 2019. arXiv: 1810.09502 [cs.LG].

[3] Vassilios Tsounis et al. "DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning". In: *CoRR* abs/1909.08399 (2019). arXiv: 1909.08399. URL: http://arxiv.org/abs/1909.08399.

[4] John Schulman et al. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.