Multitask Reinforcement Learning for Continuous Domains

Bianca Yang, Ayya Alieva, Timothy Liu, Vaishnavi Shrivastava

I. Introduction

Reinforcement learning derives its name from a related concept in behavioral psychology, where good behavior is “reinforced” with rewards. Similarly, in reinforcement learning, an agent is given rewards for behaving in a certain way. In particular, the algorithm is trained to act in a specified environment and attempts to maximize the total reward that it accumulates. This learning technique has been successfully applied to many domains, e.g. teaching robots to pick up correct objects.

Reinforcement learning usually produces models that are fine-tuned to accomplish specific tasks. An algorithm trained to recognize faces in images likely does not generalize well even to auxiliary tasks like recognizing humans in images. Another challenge with reinforcement learning is the need for expert demonstrations. In some instances, the expert demonstrations can be costly to acquire. In these cases, it can be useful to sample from related auxiliary tasks to supplement the training data. The additional data can also be used in an adversarial way - by providing examples of highly suboptimal, low reward behavior. The auxiliary data has a potential to significantly improve the model generalization by biasing the algorithm towards representations that fit multiple tasks and by leveraging the domain-specific signals contained in the related tasks.

The above highlights some possible advantages of multi-task reinforcement learning. So far, most of research in the area has been focused on discrete action spaces (i.e. Atari games). In our project, we will attempt to apply state-of-the-art multitask learning techniques to two related continuous control tasks.

II. Our goal

The purpose of multitask learning is to learn a neural network policy that behaves optimally in more than one domain. Recently, Arora et al [2] explored the possibility of multi-task learning in continuous domains by training a neural network on 6 morphologically modified variants of the Open AI gym extensions described in Henderson et al., 2017 [1]. Their approach outperformed reinforcement learning benchmarks in the Mujoco domains.

Instead of learning the same task given different configurations of the object, we will attempt to learn two different tasks on the same object. In particular, our goal is to develop a neural network which achieves fast training speed and good performance by simultaneously learning to make a humanoid stand up and walk in the extended OpenAI gym. We aim to at least perform as well as the benchmarks outlined in Henderson et al [1] .

III. Previous work

We will be using the Continuous Mujoco Modified OpenAI Gym Environments by Henderson et al. [1] to develop our model. This package includes an environment specifically designed to reward an agent for learning to stand up and walk.

Algorithm performance will be measured against the benchmarks laid out in [1].

Paper [3] describes various foundational strategies for deep multi-task learning with neural networks. Many of the architectures discussed are useful for learning the relationship between tasks. Since the tasks we are considering (standing and walking/running) likely have close dynamics, we can take advantage of architectures like cross-stitch networks, which leverage domain-specific information by outputting linear combinations of previous layers.

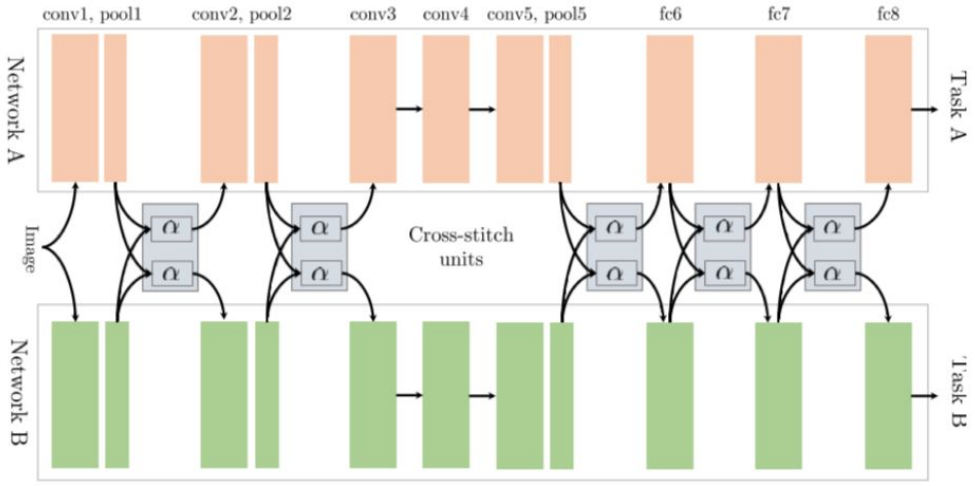

Fig 1. Cross Stitch Representation

One other useful architecture that was introduced was low supervision architectures [5], which is useful in learning how to split up the target task into sub-tasks. For example, in learning to stand from a prostrate position, figuring out how to coordinate the hands and feet separately may be separate sub-tasks. We can also cross-pollinate insights on hand/feet control in walking with insights from standing up.

We plan to base our initial algorithm off the knowledge distillation and actor to critic networks developed in [2]. The environments tested in this paper were the morphologically modified variants in the extended openAI gym. Both the actor to critic and knowledge distillation networks implement soft parameter sharing. This means that one network is trained for each environment and then the results are combined through regularization to produce policies for each task that learn on results from the other policy. The actor to critic network follows this standard framework.

The actor to critic network tries to learn the optimal policy for a Markov Decision Process parameterized by $\langle S, A, P, r, \rho 0, \gamma, T \rangle$, where S is the state space, A is the action space, $P : S \times A \times S \rightarrow R+$ is a transition function, $r : S \times A \rightarrow [Rmin, Rmax]$ is a reward function, $\gamma \in [0, 1]$ is a discount factor, and T is a time horizon. The policy maximizes the expected discounted return

. The optimal policy is

, where

The synchronous advantage actor-critic algorithm A2C is an on-policy algorithm which samples rollouts of the current policy to estimate n-step returns. A policy and value function estimate are maintained and updated

using a gradient , where A is:

.

By adding a policy entropy term, convergence to sub-optimal deterministic policies is less likely. Feed-forward neural networks are used to learn the policy and value functions. Actions are sampled from a Gaussian distribution to model a continuous environment.

On the other hand, the knowledge distillation network extends the standard framework by introducing a “teacher” policy (trained actor to critic) and a “student” policy (untrained). The student policy tries to converge to the teacher policy by following a KL-divergence objective

function: , where

.

The motivation for using distillation was that it may reduce variance. The variance of the distillation network was higher than the actor to critic model, but it may be useful to consider developing some kind of prior network as the teacher to improve training speeds.

Paper [4] aims to help a robot concurrently learn multiple skills, by building on is deep deterministic gradient descent (DDGD), developed by Lillicrap et al in 2016. The paper develops the idea of using multi-layer perceptron convolutional layers (mlpconv) in a neural network to reduce the number of parameters for single task learning, which involves data from images and sensors, and to reduce the potential for overfitting. The convolutional layers were useful in this application because the robot they were testing on had camera input as a feature. Since our environment will likely not use visual features, we will focus on using the multi-DDGD structure that is proposed in the paper, without needing to delve into the mlpconv layers of the network.

The crux of the multi-DDGD structure is that there is one critic and multiple actors. Each actor is responsible for a single task, and the critic is responsible for coordinating learning, by outputting multiple state-action values. Both the actor and critic networks are trained separately and iteratively refine each other up to some maximum number of iterations. The critic evaluates all the state-action pairs produced by all actors and is agnostic as to which actor produced which executed action. The revised

loss function is defined as ,

where Q is .

Only one actor is chosen to act at each timestep, and their actions will not be distinguished by which actor produced them. Actor and critic training will not be synchronous, and the critic will not have task-specific training, so we can choose any actor to calculate the supervising signal

.

After updating the critic, each actor will be updated using the following task-specific gradient.

.

IV. Timeline

A. April 30th - May 6th: Read all reference papers thoroughly and become familiar with the MuJoCo extended environment. Reproduce the neural network structure used in in Multi-task learning for continuous control paper for our environment. We have emailed the authors of the paper in an attempt to get the github of their code and are now awaiting a response (The link in the paper is broken).
B. May 7th - May 13th: Work on improving the performance of the first algorithm we implement to meet the benchmark.
C. May 14th - May 20th: Train a new neural network with multi-DDGD to compare performance.
D. May 21st - May 27th: Focus on improving performance of our existing algorithm by tweaking to meet benchmarks. If we are ahead of schedule and the algorithms have at least achieved benchmark performance, we will try policy-based algorithms to improve training speed and performance.
E. May 27th - June 1st: Prepare for final project presentation.

V. References

[1] Benchmark Environments for Multitask Learning in Continuous Domains
[2] Multi-task Learning for Continuous Control
[3] An Overview of Multi-Task Learning in Deep Neural Networks
[4] Multi-Task Deep Reinforcement Learning for Continuous Action Control
[5] Deep multi-task learning with low level tasks supervised at lower layers