Learning to walk, so you can run!

Billy Witrock

1 Introduction

Children learn different tasks one at a time. For example, a child would learn to write each letter before they can write words, a mathematician would learn addition before integration, or you would learn to walk before you can run. In this paper we will discuss how to do the same for a reinforcement learning agent. The question we will attempt to answer is how long should you try to learn the easier task before moving to the harder one. Or is learning to walk, then try running, but going back to walking before finally learning to run again better? Training can be a costly process, so finding the fastest ways to train agents to optimize performance helps greatly.

2 Background Related Work

The first necessary background is the paper "Playing Atari with Deep Reinforcement Learning" [2]. This reference gives me information about how make an algorithm to utilize a neural network as a Q function. Having a neural network in my model will help my agent reach maximum performance.

This paper helps set up the entire training process by saying exactly the needed steps outlined in figure 1. Following this algorithm should allow me to try an effective agent.

Another source is the openai gym documentation [1]. This will show me what the different environment states are. Luckily as discussed later, the two environments have the same inputs, which would allow for very easy transfer learning because the identity function can be used as the function to transfer knowledge from one domain to the other.

Besides from these two, I was unable to find many other related work that applied. I found alot on curriculum learning, however alot focused on source task creation, such as "Source Task Creation for Curriculum Learning"[3]. But, I was unable to find works discussing how switching back and forth between easier and harder tasks could possibly help agents learn.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 1: Deep Q-learning with Experience Replay [2]

3 Technical Approach / Methodology / Theoretical Framework

This problem is a complicated problem. First my agent will use on-policy deep Q-learning, just like in the deepmind paper that an agent learned atari [2]. This will allow my agent to learn a complex problem. The difference is I will have 3 stages of learning. First, I will run agent on the basic, easier task. Then, I will run my agent on the more complex task. And then finally I would train my agent on the simpler task again, before finally testing it on the more difficult task. Transferring between my two environments will not be difficult because the two environments have the same controls and environment states. The easier environment will be the BipedalWalker-v2 from the openai gym [1]. And then the harder task will be the BipedalWalkerHardcore-v2 from the openai gym [1]. In both environments the goal is to move forward, with the least amount of torque applied to each of its joints. The difference between the basic and hardcore mode is the hardcore mode has ladders, stumps, and pitfalls [1].

I will try to find the optimal switching time by training multiple sets of agents. The first would train on only the harder task. This could be used as a baseline to see how effective no curriculum training is. The second would be a group of agents that first train on the easier task for n steps, and then on the harder task for remaining steps. This will test to see how training on an easier task can be helpful and speed up training. And then finally, there will be a group that first trains on the easier task, switches to the harder task, and then back to the easier task. All agents will have the same total number of episodes, only the order and proportion of environments will change. Each group, when applicable, should have multiple subgroups that vary the training time for each

environment.

3.1 Task 1

Getting a baseline report. This will consist of training agents on only the hardcore environment, and looking at the average results after n episodes. And then also, training an agent on only the easier task and reporting the results on the harder task.

3.2 Task 2

Training on first the easier task and then the harder task. Here, we will vary different lengths of time for training. We are thinking of having 3 subgroups, one that splits training 1/3, 2/3. Another that splits 1/2, 1/2. And then the last group would train 2/3, 1/3. These three groups would give us a nice range of varying training.

3.3 Possible: Task 3

This subgroup has the most complicated training method. I am thinking to have 3 groups, with varying training time on different models. The proportions chosen should have to do a little with the results of task 2. We would want to try to draw some connections and try to use the more effective training time for each states.

3.4 Possible: Task 4

This task will train first on the easy environment, then the hardcore version, then back to the easy before finally doing a final round of training on the hardcore version again. This is very similar to task 3. My plan is to attempt to do both if time permitting, however it is more likely that I will chose to either do Task 3 or task 4.

4 Evaluation

My evaluation will be based on the reward system of the environment. In the environment, "reward is given for moving forward, total 300+ points up to the far end"[1]. It also receives -100 if the robot falls, and some small negative reward for applying motor torque [1]. One method of evaluation will be the final rewards the agent is able to accomplish. This will show how effective the agent will be over the n steps. Another method of evaluation could be the integral of average reward over the entire n step process. This will show how quickly the agent learns, and factors in any short term loss that could have took place when switching domains. This loss could be important because if you're doing a task in which all loss could be bad, finding the curriculum that maximizes reward the quickest is important.

My initial hypothesis is that the best method will be task 4. I feel like it has the best of curriculum learning because it has the easier task and then the harder one. However, since it goes back and forth, it allows for the agent to learn for longer on the easier task without over fitting on the easier task because it would have seen the harder one also. I think task 1 and 3 will be the worst because it towards the end, the agent would not have been with the harder environment. Therefore, it might not know as much what to do when it gets to the more complex environments.

Also, all evaluations will be done on an average of different agents with the same parameters, over a course of multiple different test runs at the end with different random states. This is needed because with an on-policy agent, the randomness makes a big difference. When testing, the agent will however, be greedy and select the best action.

5 Timeline and Individual Responsibilities

Since I am doing this project alone, I will do the whole project. I am planning on using openai's environment, and a pre-built neural network, possible sklearn or tenser flow. I believe using these packages will help allow me to focus on the more important curriculum learning question. The following is my timeline.

5.1 Step 1: (HW 4)

Run baseline tests. This means getting the environment setup and running, which I have been unable to do so far. Creating my deep Q-learning agent, and training it on either just the easy task or just the hard task.

5.2 Step 2

Complete task 2. This means starting to train on basic curriculum path. Make sure the transfer learning works, and start to make comparisons and graphs to best display the data and results.

5.3 Step 3

Complete task 4. I want to do task 4 before task 3 because I think task 4 will get better results than task 3. And, if I don't have time to complete both I want to make sure I got the best results I can.

5.4 Step 4

Complete task 3. This task is the least important to me. I believe that if I had to lose a part of the project this will be it. This task just will be able to have a nice point of comparison but is not expected to do well.

Also, throughout all the steps, I will start to make useful graphs, and draw conclusions that will allow me to write the final paper easier.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[3] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.