Homework 3 Template

Use this template to record your answers for Homework 3. Add your answers using LaTeXand then save your document as a PDF to upload to Gradescope. You are required to use this template to submit your answers. **You should not alter this template in any way** other than to insert your solutions. You must submit all 10 pages of this template to Gradescope. Do not remove the instructions page(s). Altering this template or including your solutions outside of the provided boxes can result in your assignment being graded incorrectly. You may lose points if you do not follow these instructions.

You should also export your code as a .py file and upload it to the **separate** Gradescope coding assignment. Remember to mark all teammates on **both** assignment uploads through Gradescope.

Instructions for Specific Problem Types

On this homework, you must fill in (a) blank(s) for each problem; please make sure your final answer is fully included in the given space. **Do not change the size of the box provided.** For short answer questions you should **not** include your work in your solution. Only provide an explanation or proof if specifically asked. Otherwise, your assignment may not be graded correctly, and points may be deducted from your assignment.

Fill in the blank: What is the course number?

10-703

Problem 0: Collaborators

Enter your team's names and Andrew IDs in the boxes below. If you do not do this, you may lose points on your assignment.

Name 1:	Heethesh Vhavle	Andrew ID 1:	hvhavlen
Name 2:	Karmesh Yadav	Andrew ID 2:	karmeshy
Name 3:	Aaditya Saraiya	Andrew ID 3:	asaraiya

Problem 1: REINFORCE (30 pts)

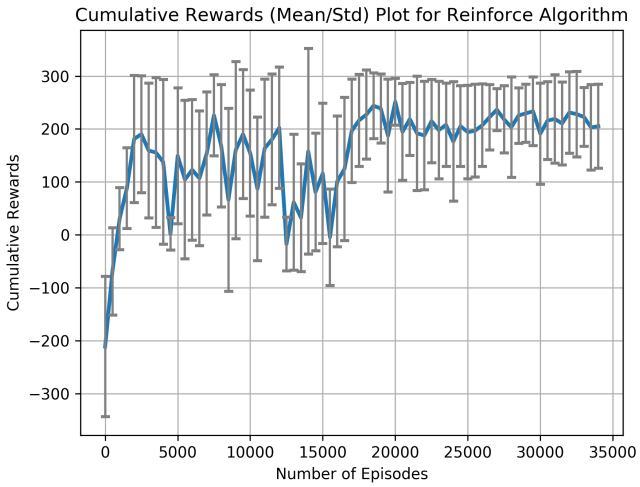
1.1 Describe your implementation (10 pts)

The policy and value network we used had the following properties:
Framework: PyTorch
Architecture: **Actor:** We used a 4-layered network with 3 layers having 64 nodes each and the output layer having 4 nodes, one for every action.
Optimiser: Adam Optimizer.
Initializer: Xavier uniform with scale factor 1.0 (The network uses ReLU activation and in He et al. 2015, it is recommended to use variance scaling initializer).
Loss: $-\frac{1}{T}\Sigma_{t=0}^{T-1} G_t log \pi_\theta(A_t|S_t)$ (negative for gradient ascent)
Sampling: We sample actions using the multinomial distributions in PyTorch which basically samples according to the log softmax output from the policy.

Hyperparameters: We used the following hyperparameters for A2C:
Training Interval - Once per episode
Gamma - 0.99
Learning Rate - 5e-4
Test Episodes - 100 episodes
Testing Interval - Once every 500 episodes
Total Episodes - 50,000 (or convergence of rewards > 200 for more than 5,000 episodes)

Algorithm / Implementation Notes: The algorithm was implemented exactly as mentioned in the write-up. The returns G_t were normalized by subtracting its mean and dividing by the standard deviation to reduce variance in learning. We also tried to update the gradients in batches and use batch norm by sampling multiple episodes of fixed trajectory length, but faced issues with handling backward pass in PyTorch and dropped the idea. Another thing that we tried was using deterministic sampling (argmax) of the action from the policy at test time, but stochastic sampling gave higher rewards for the episodes.

1.2 Learning curve and explanation of trends (20 pts)



Trends in Training:
In general, the REINFORCE algorithm took longer to converge (or solve the environment). The standard deviation of test rewards was quite high (120-150) and the rewards seem to converge at 150-170 for most hyperparamters (switching from 16 units per layer to 64 helped). A trend that we observed was that the rewards increased beyond 200, only after training the algorithm longer (5,000-10,000+) and this also led to a decrease in standard deviation of the rewards later.

Policy Performance:
The policy trained on REINFORCE performed reasonably well and achieved rewards around 200-240 most of the time (with correct hyperparams). However, by inspecting the videos of the agent, we found that the rover used to land in valleys (leading to lower rewards) sometimes or did not perfectly reach the goal position. The agent did try to fire its jet to correct its position to reach the goal region. The agent also took longer (as compared to A2C) to solve the environment.

Instability in REINFORCE:
In REINFORCE, we directly to try to optimise the policy weights based on the rewards and the log probabilities of the actions from the policy. The actions are then randomly sampled based on these probabilities. Furthermore, every action in the trajectory is given rewards based on all the states encountered after it. Thus even if this particular step wasn't helpful overall, it might get high rewards based on future actions. This could be the reason why we have high variance in our computed loss, which in turn would cause noisy gradients and policy updates leading to unstable learning.

Problem 2: Advantage Actor-Critic (40 pts)

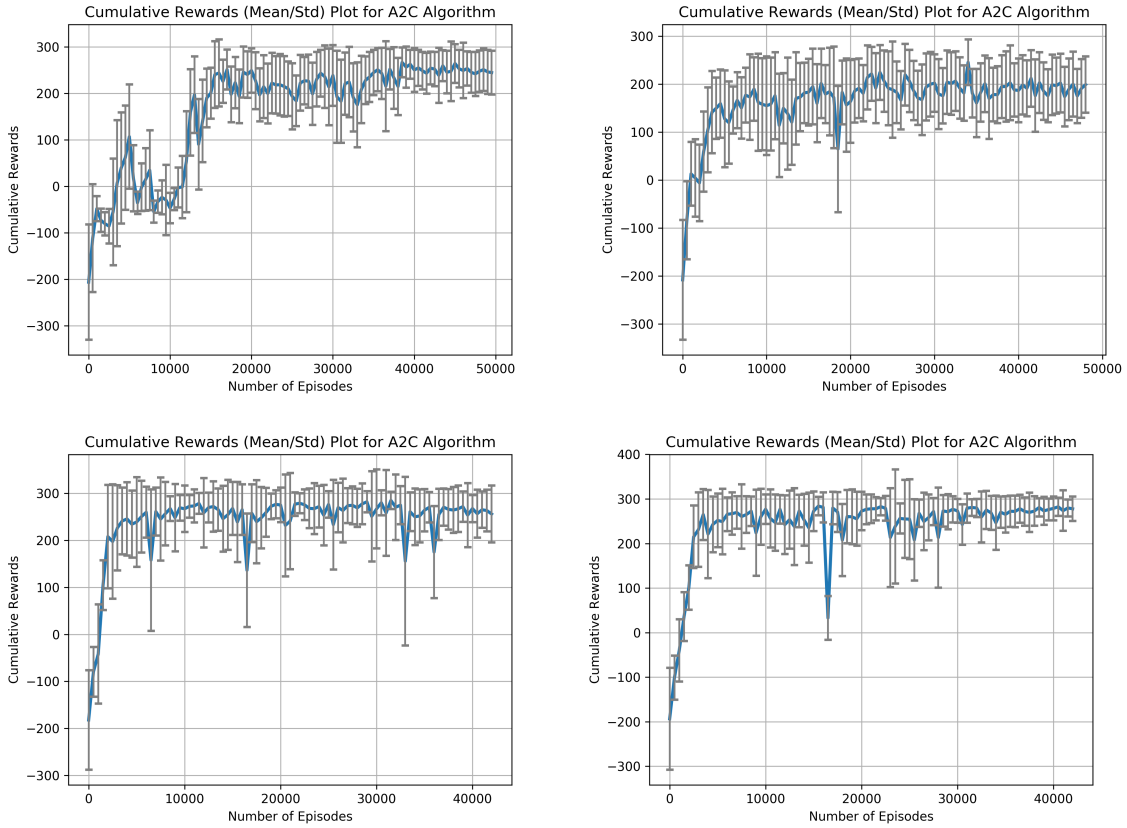
2.1 Describe your implementation (10 pts)

The policy and value network we used had the following properties:
Framework: PyTorch
Architecture: **Actor:** We used a 4-layered network with 3 layers having 64 nodes each and the output layer having 4 nodes, one for every action.
Critic: We use a 4-layered network with 3 layer of 64 nodes each and an output layer having a single node which predicts the value of the state.
Optimiser: Adam for both networks.
Initializer: Xavier uniform with scale factor 1.0 (The network uses ReLU activation and in He et al. 2015, it is recommended to use variance scaling initializer).
Actor Loss: Custom Loss: $\frac{1}{T}\Sigma_{t=0}^{T-1}(R_t - V_\omega(S_t))log\pi_\theta(A_t|S_t)$
Critic Loss: MSE: $\frac{1}{T}\Sigma_{t=0}^{T-1}(R_t - V_\omega(S_t))$
Sampling: We sample actions using the multinomial distributions in PyTorch which basically samples according to the log softmax output from the policy.

Hyperparameters: We used the following hyperparameters for A2C:
Training Interval - Once per episode
Gamma - 0.99
Actor Learning Rate - 5e-4
Critic Learning Rate - 1e-4
Reward Normalization Scale - 100.0
N-step - [1, 20, 50, 100]
Test Episodes - 100 episodes
Testing Interval - Once every 500 episodes
Total Episodes - 50,000 (or until rewards $>$ 200 for more than 5,000 episodes)
All the four networks are trained using the same random seed for comparison

Algorithm / Implementation Notes: The algorithm was implemented exactly as mentioned in the write-up. The rewards were normalized by scaling using the factor mentioned above. We also tried to update the gradients in batches and use batch norm by sampling multiple episodes of fixed trajectory length, but faced issues with handling backward pass in PyTorch and dropped the idea. Another thing that we tried was using deterministic sampling (argmax) of the action from the policy at test time, but stochastic sampling gave higher rewards for the episodes.

2.2 Learning curves and explanations of trends (20 pts)



Plots showing the mean and standard deviation of cumulative rewards for varying N-steps. (Top Left: N=1, Top Right: N=20, Bottom Left: N=50, Bottom Right: N=100)

Trends in Training:

- We tried using network with 16 hidden neurons instead of 64, both for actor and critic separately. We saw that the network with 16 hidden units performed quite well with less fluctuation in test rewards. This must be because the network learns to find the best set of features from the states.
- The rewards obtained from the environment quickly plateau but there are still some drops in the rewards obtained in between for the 64 hidden unit network.
- The instances with 100 and 50 step returns perform the best and they are quickly able to attain rewards of around 250, while the other two struggle.

Policy Performance:

A2C policy performs better than REINFORCE and gets higher rewards. Looking at the video of the agent, we see that the agent has learnt to go towards the goal region (by curving, avoiding landing in valleys), even if it was initially flying in some random direction. Rewards up to 310 were also achieved at times. The agent was also able to solve the environment faster as compared to REINFORCE.

Instability in A2C:
The major disadvantage of A2C the sample inefficiency of the trajectories collected. Therefore it takes a lot of time to train a policy using A2C. A solution is to use multiple environments in parallel which leads to speed up in training.

Also, the performance of A2C is dependent on the n-steps bootstraping of the return, which is basically another hyperparameter which requires tuning. A solution to that is to use something like Generalized Advantage Estimation (Schulman et al. 2015)

2.3 Compare and discuss REINFORCE and A2C (10 pts)

We found the following difference between REINFORCE and A2C:

- A2C requires more computing resources than REINFORCE for training due to two networks, however A2C solves the environment and achieves higher rewards with lesser training episodes compared to REINFORCE.
- A2C attains mean rewards in the range of 230-260 (N=50,100) which is much higher than REINFORCE which only attains mean rewards in the range of 170-230.
- Overall we can say A2C becomes REINFORCE as N tends to ∞. We see that out of all N's possible, values around 50-100 perform the best. This is due to the bias-variance trade-off we see in RL. REINFORCE has high variance while A2C at values like 1 has high bias.
- The variance in the rewards is around 70-80 for REINFORCE, while it is just 40-50 in A2C. This is coherent to what was taught in class regarding the variance of the different algorithms.
- Overall the fluctuations in the reward curve were lower for A2C compared to REINFORCE as we reduce the variance in gradients of policy in A2C using the baseline network, compared to directly updating the gradients in REINFORCE.
- Qualitatively looking at the performance of the LunarLander-v2 on the moon surface, we see that the A2C policy learns how to swerve if the lander starts in a difficult initial state. This is much better than REINFORCE where it would just try to push the lander towards the goal after landing.

Extra credit (up to 15 pts)

We tried implementing the A2C on the breakout environment in PyTorch. A network architecture with 3 convolution layer and two fully connected layers (with 256 hidden and 4 output units) was used. The architecture was inspired from the DeepMinds's Atari paper (Volodymyr et al. 2013). We also implemented the preprocessing step as mentioned in the paper, by cropping the top score bar and resizing the image to 84 x 84. The input images were 4 grayscale images (sampled temporally) and stacked. The algorithm was kept the same explained previously.

The implementation is provided in `breakout.py`.

Hyperparameters: We used the following hyperparameters for A2C:
Training Interval - Once per episode
Gamma - 0.99
Actor Learning Rate - 5e-4
Critic Learning Rate - 1e-4
Reward Normalization Scale - 100.0
N-step - 100
Test Episodes - 50 episodes
Testing Interval - Once every 250 episodes
Total Episodes - 50,000

However, the training was taking very long for every episode and we did not have enough time to perform hyperparameter search.

Extra (7pts)

Feedback (5pts): You can help the course staff improve the course for future semesters by providing feedback. You will receive a point if you provide actionable feedback **for each of the following categories**.

What did you find worked well for you in this assignment?

The training time for the algorithms was huge this time. According to us these things worked for us:

- Using PyTorch instead of Keras (to better understand how to implement a custom loss and take gradients).
- We have a PC with Nvidia Titan V GPU in our lab. Using that made debugging, hyperparameter search, and training easier for us.
- We started the assignment pretty early.

What was the most confusing part of this homework, and what would have made it less confusing?

The extra credit question for this assignment was quite difficult. Without any base code or implementation detail, it was difficult to plan how to get the question done. Having multiple complicated parts including a CNN, VecEnv for multiple environments and a shared optimizer, made it confusing. Computing the discounted returns over the trajectory was a little tricky to understand at first.

What was the most frustrating part of this homework, and what would have made it less frustrating?

The training time of the algorithms were long, especially the Breakout environment. Some prior guidance regarding implementation would had made it less frustrating. Some of our algorithms (even REINFORCE and A2C) took much longer than what was mentioned in the write-up and this could mislead someone to manage their time properly, but fortunately, we had started early.

What advice would you give to future students, either regarding this homework or the class in general?

Reading about classical RL before the start of the semester will help a lot to understand the topics in class. Also being up to date with the work (atleast the things they are working on) that companies like OpenAI and DeepMind are doing keeps you much more excited about every topic in the class.

Do you have any broader feedback about how lectures, recitations, and/or office hours are conducted?

Some of the lectures can maybe provide a little more intuition or help explain the algorithms in a graphical manner as well. It is difficult to follow and understand the algorithms based on only equations in the slides. The recitations can maybe cover techniques on implementing parallelization which would have been very helpful in this assignment or future assignments, where training might take very long.

Time Spent (1pt): How many hours did you spend working on this assignment? Your answer will not affect your grade.

Alone	15
With teammates	11
With other classmates	0
At office hours	0