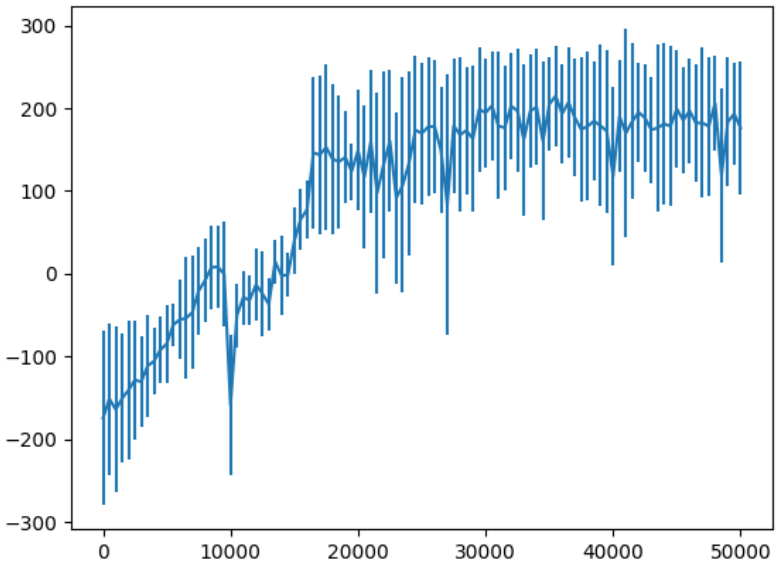
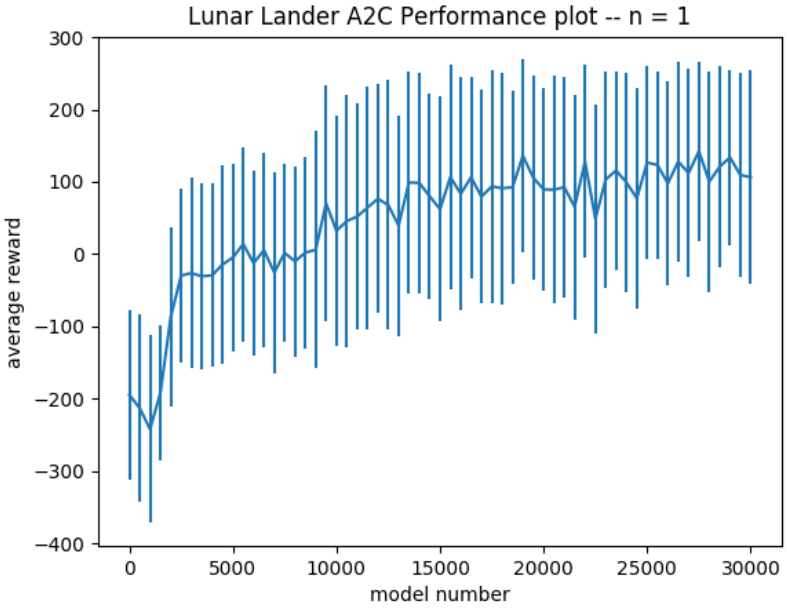
REINFORCE:

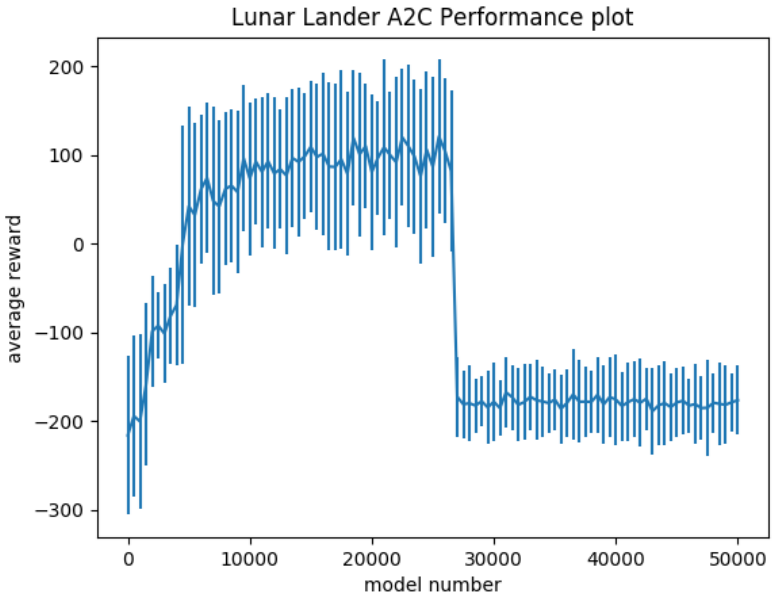
The model definition was given by the JSON file. We used gamma=1, lr = .0005. We normalized rewards by 1/100 and did not use a baseline. We normalized the rewards by the lengths of the episodes T, and due to a bug in our code, we tried training with 1/T^2 normalization, which actually performed better so we stuck with it. This effectively lowered the learning rate proportionally to the length of an episode. We tried using a baseline of the average G_t value (both mean and median) because this resembled the a2c pseudo-code, but it plateaued at around 150 total reward.



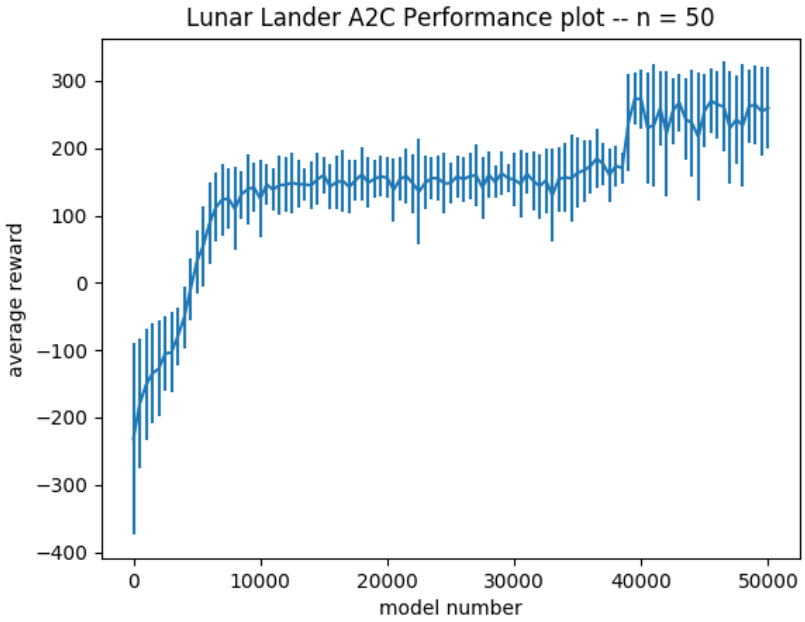
A2C:


Lunar Lander A2C Performance plot -- n = 1
average reward
model number
300
200
100
0
−100
−200
−300
−400
0
5000
10000
15000
20000
25000
30000

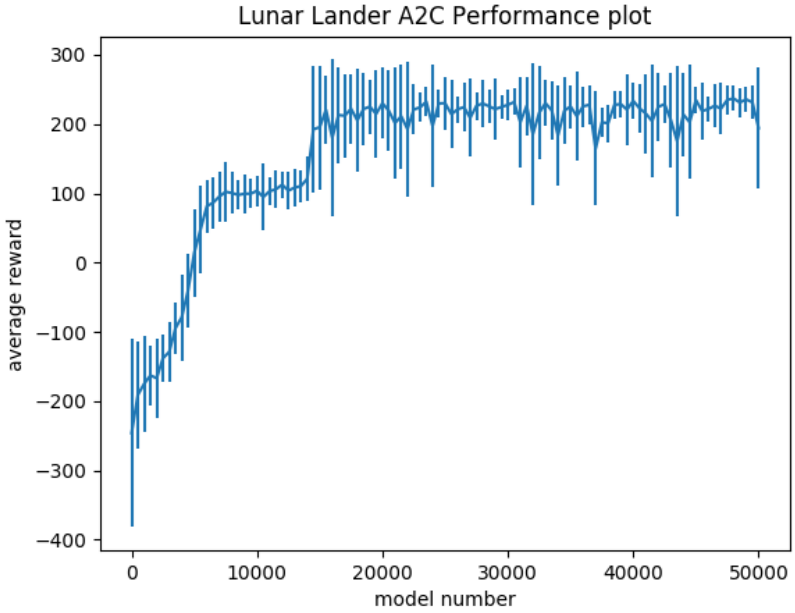
N = 1:


Lunar Lander A2C Performance plot
average reward
model number
200
100
0
−100
−200
−300
0
10000
20000
30000
40000
50000

N = 20:



N = 50:



N = 100

A2C: 1. The actor model is also taken from the model file we were given, but the critic model is chosen to be a 3-layer dense ReLU with 16 layers each, with

a final linear layer at the end. This setup has functioned for us before, so we decided not to change it unless the network was unable to learn. We used Adam as the optimizer for both models, and the default learning rate for both as well. Gamma of 1 seemed to work fine, so we kept it. We used MSE for the critic network, and the same custom loss for the actor network (our use of it was easily adjusted from our REINFORCE baseline code). We tried n = 1, 2, 50, 100, as the writeup instructs.

A2C: 3. Both n = 50 and n = 100 led to A2C solving the environment, often with reward so high that the requirement of 200 was below one standard deviation under the mean. Both n = 1 and n= 20 had exploding gradients or something else leading to NaN outputs of the policy network. We were only able to train to around 30000 episodes before this problem would lead to network failure. n = 100 trains the fastest, but n = 50 eventually reaches the highest mean reward.

The n = 100 case probably trained fastest because the value network was able to train faster because it had more immediate feedback from the real reward, and having a good value network is necessary for the policy network to train well. The difference in eventual reward is small enough to easily be due to noise of initialized weights or noise of chosen trajectories.

Compared to Reinforce, A2C had much better end-performance and variance (for n=50 and n=100). This is because A2C has a baseline for every state as opposed to a rough approximation of a constant baseline that Reinforce uses. Variance corresponds to having a policy that is not good at generalizing. Having a good baseline in general helps to learn faster because the model only interprets improvement in score as a positive change rather than viewing every episode with positive reward as a target. Because A2C gives a baseline for individual states, the model can learn more states quickly, which may reduce variance.