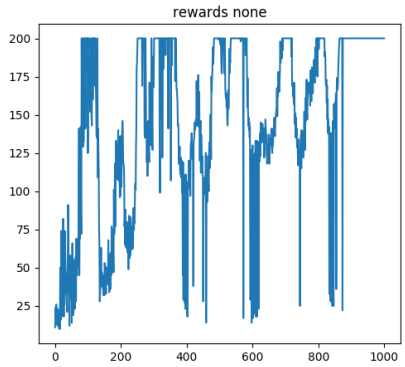
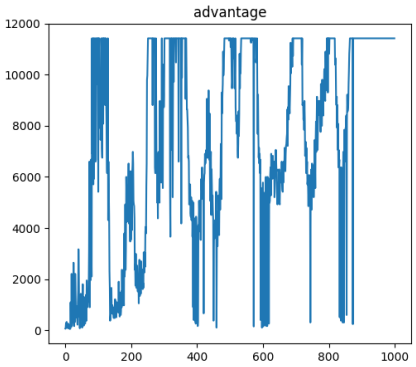
Topics in ML Assignment 2

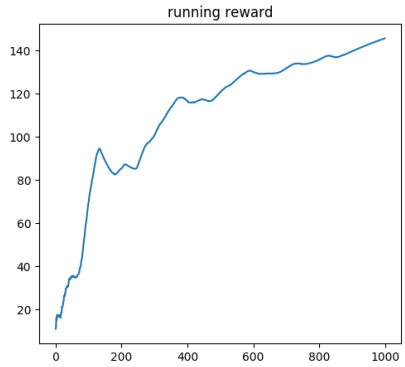
Rohit Gajawada
201401067

Prob1)
a) CartPole-v0

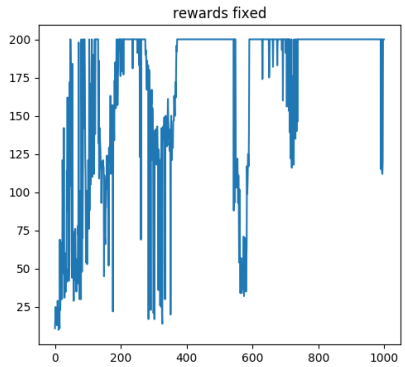
i) no baseline

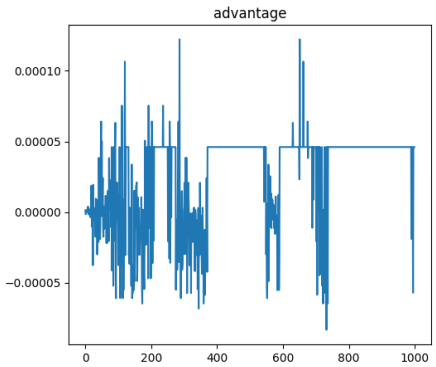




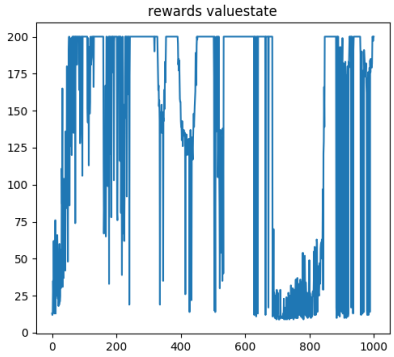


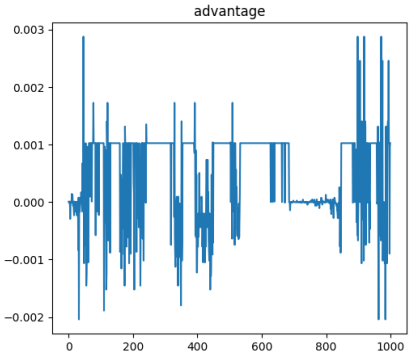
ii) fixed baseline as the mean of the estimated rewards of that trajectory

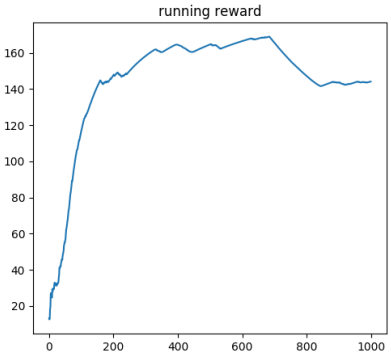




iii) value-state baseline



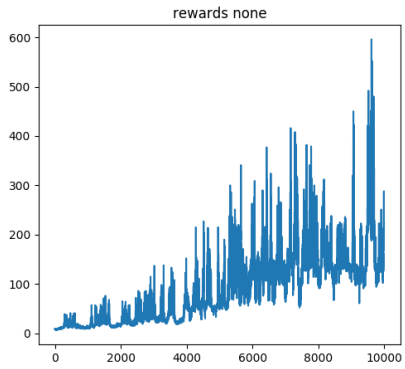


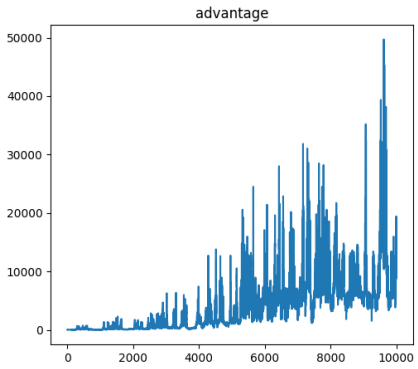


Observations:
For CartPole, all three methods end up converging. However compared to no baseline, when using a baseline the network converges much faster and is much more stable. While the fixed baseline performs much better than no baseline, using a value-state baseline results in obtaining convergence much faster and much more often at the beginning. Later on it could be less stable but that is fine as we reach convergence at beginning.
Using a baseline basically reduces the variance and allows much more smooth updates.
Details:
learning rate: 5e-3
episodes: 1000
gamma: 0.99
normal network architecture: FC(IN, 128) – RELU – FC(128, N) – Softmax
value baseline architecture: FC(IN, 128) – RELU and after this it has another head for value state: FC(128, 128) – RELU – FC(128, 1) - Tanh
other head is FC(128, N) - Softmax

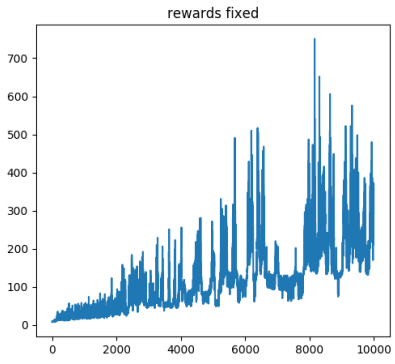
b) InvertedPendulum-v2 continuous

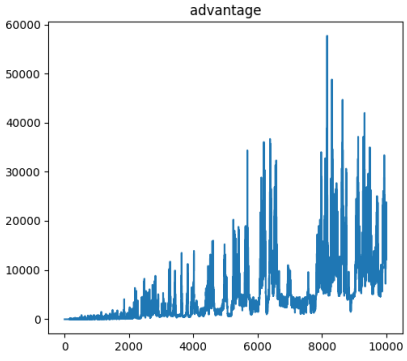
i) no baseline



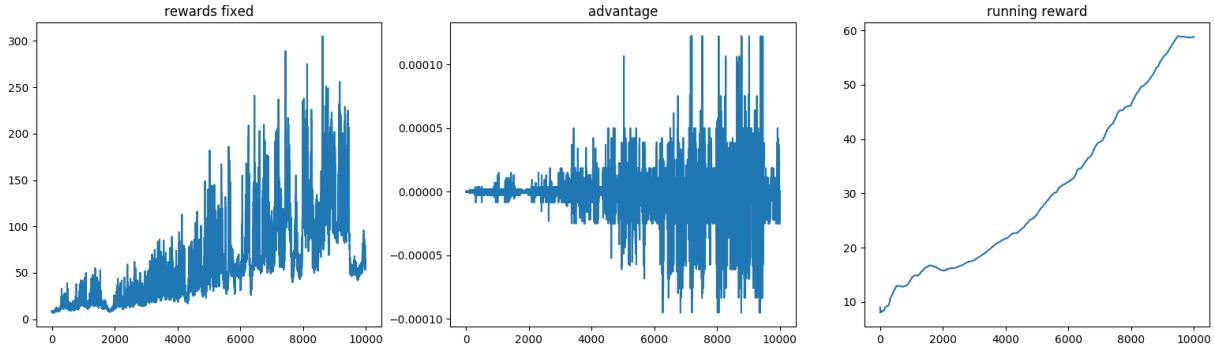


ii) fixed baseline as 10

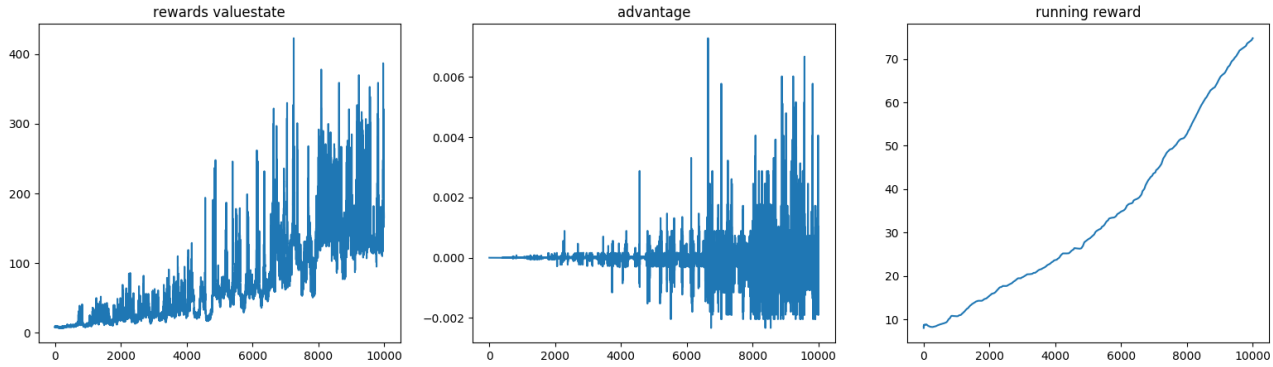




iii) fixed baseline as mean



iv) valuestate baseline



Observations:

I ran for 10000 episodes instead of 1000 as maximum values are obtained after 1000 episodes.

For Inverted-Pendulum, all three methods end up converging but the training is very sensitive to hyperparameters and small factors. Within 10000 episodes, strangely fixed baseline as 10 gave me the best results and no baseline also gives similar results. Using mean as baseline does not work too well. Using a value state baseline works better than mean baseline but still worse than no baseline within 10000 episodes. Possibility is that it will outperform the other approaches but needs much more time to train.

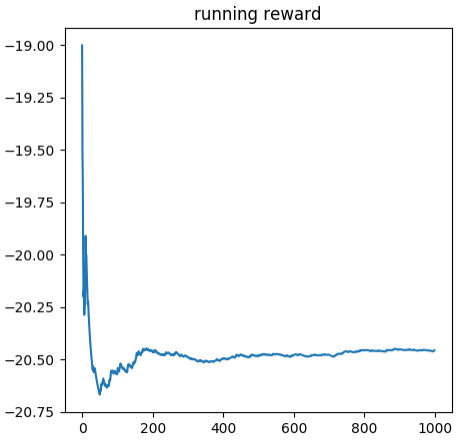
Details:

learning rate: 1e-4
episodes: 10000
gamma: 0.99

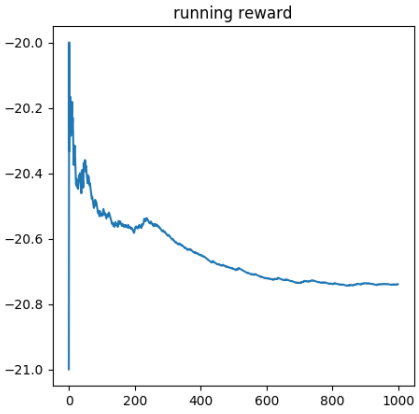
normal network architecture: FC(IN, 128) – RELU – FC(128, 1 – Tanh
value baseline architecture: FC(IN, 128) – RELU and after this it has another head for value state: FC(128, 128) – RELU – FC(128, 1) - Tanh
other head is FC(128, 1 – Tanh
After Tanh output, gaussian sampling is done with fixed stdev as 0.05

c)
Accelerating policy gradient can also be possibly done by keeping a trajectory replay buffer that stores previous trajectories. After a batch of trajectories are done, this same batch is again taken at some point in time and used for update. However these stored trajectories are offpolicy, so we must do offpolicy update and importance sampling can be used for this i.e multiply the advantage with the target policy(on-policy) divided by the behavior policy (off-policy). I was unable to experiment with this due to lack of time.

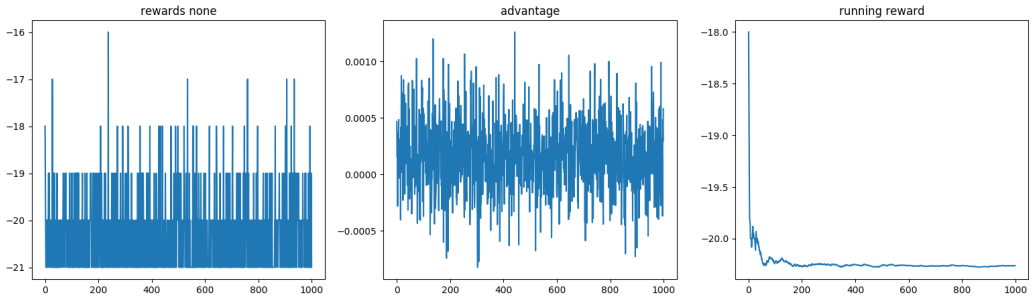
Prob2)
I)
a) Q-learning with LFA



b) Double Q-Learning with LFA

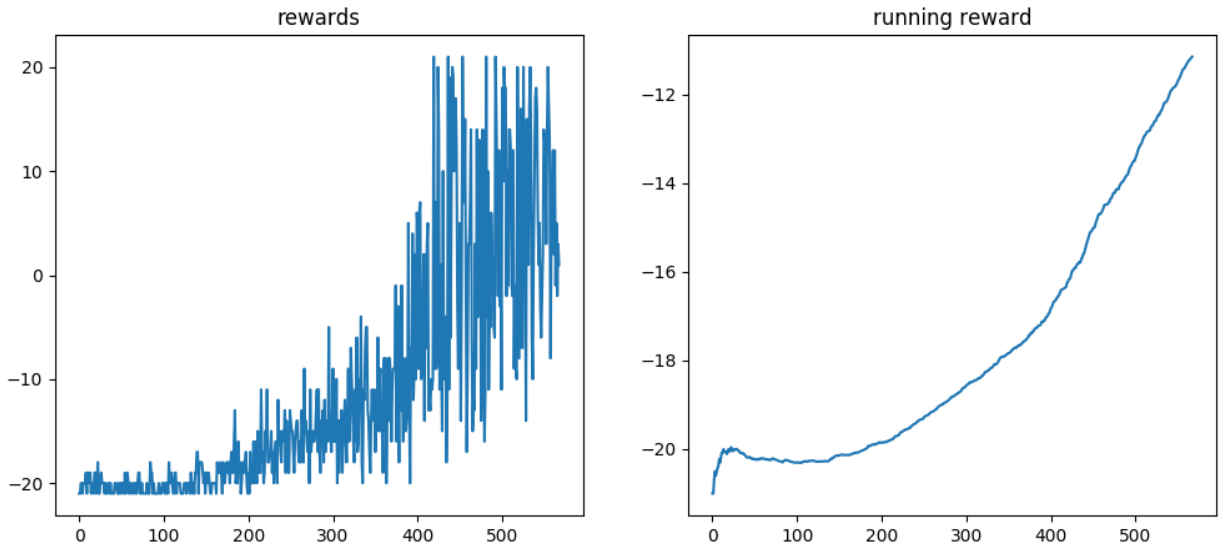


c) Policy Gradient with LFA

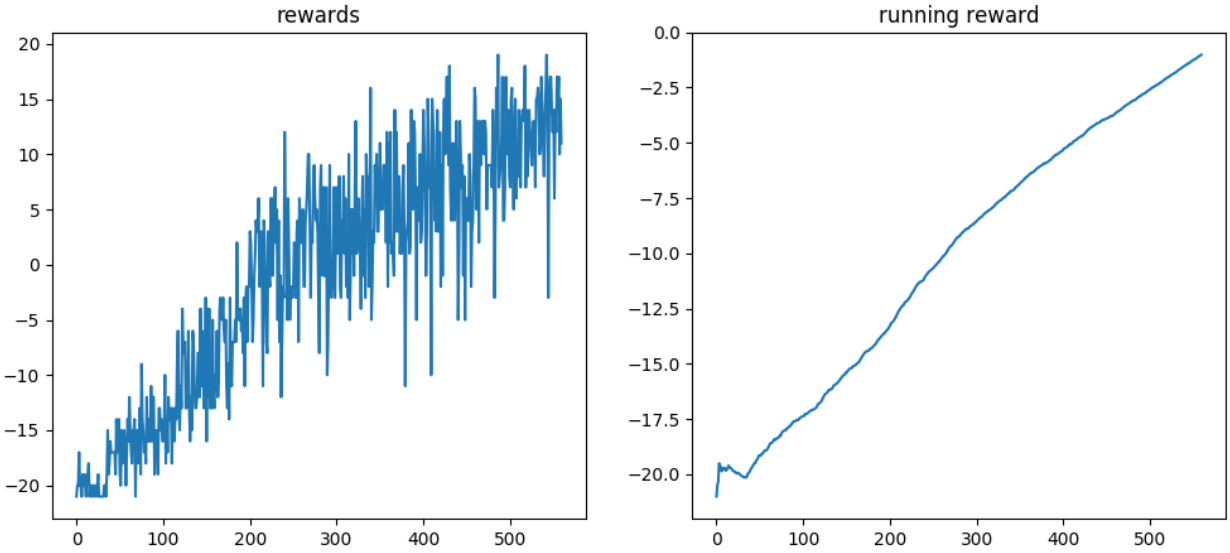


<u>Observations:</u>
As seen, using a linear function approximator gives terrible results and is no better than random. That implies that pong is too complex of a game to be approximated well with a simple linear layer.
<u>Details:</u>
I used a simple linear layer from input size to number of actions. I tried lots of hyperparameters, different types of preprocessing and even using the official atari wrappers gave no improvement. My first implementation was in PyTorch and I also implemented it in numpy to see if I got better results but it also failed.

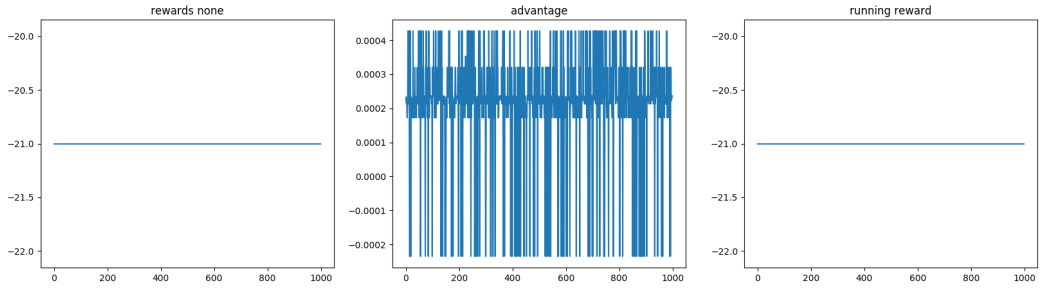
II)
a) Q-learning with neural network



b) Double Q-learning with neural network



c) Policy Gradient with neural network



<u>Observations:</u>
Both Q-Learning and Double Q-Learning give spendid results and converge after many episodes. However, double Q-learning starts to learn much fast and also converges much faster.

Using my Prob 1 setup for Pong with two different deep networks (with convolution layers or with 3 non linear layers), I was unable to get any decent results.

Details:

I ran my DQNs (Q-Learning with neural network) and Double DQNs (Double Q-Learning with neural network) around 500 episodes each until I got values around 18 to 19 due to long training time.

My running average is calculated as total sum of rewards till then divided by number of episodes from beginning.

I used the official atari gym wrappers which significantly improve the training time. I have also used my own preprocessing by taking frames simply from gym and making them into binary vectors. This does improve training speed slightly compared to directly taking the frames.

However, I finally use the openai gym wrappers for environment as this increased training time by 3 or 4 times.

For q learning, I use batchsize of 32, gamma as 0.99, and linearly decay epsilon from 1 to 0.05 over 30 episodes. I have tried experiments with higher epsilon and over longer episodes and I realized that this config is sufficient.

For my deep Q neural network, i use 3 convolution layers (inchannels, 32; 32, 64; 64, 64) followed by two linear layers (3136, 512; 512, n) with ReLU nonlinearities. I prefer to use convolutional layers because Pong is a complicated game and better results can be achieved with convolution layers as they can understand patterns and features of the game. Also when I first tried a 3 layered neural network, I found my network with convolution layers was able to start learning faster.

For double q learning, my target q network is a clone of the original q network. I update only the original q network and I copy parameters from original q net to target q net every 1000 frames. Instead of doing the 50% probability of updating either like taught in class, I found this method to be more stable.

For Pong among the three provided, double q learning gives best results.

Instructions:

Prob1)

a) Cartpole
1) no baseline: python3 policy_gradient.py --baseline='none' --episodes=1000
2) fixed baseline: python3 policy_gradient.py --baseline='fixed' --episodes=1000
3) value state baseline: python3 policy_gradient.py --baseline='valuestate' --episodes=1000

b) Inverted Pendulum
1) no baseline: python3 policy_gradient_cont.py --baseline='none' --episodes=10000
2) fixed baseline: python3 policy_gradient_cont.py --baseline='fixed' --episodes=10000
3) value state baseline: python3 policy_gradient_cont.py --baseline='valuestate' --episodes=10000

Prob2)
Pong:

LFA)
a) Q-Learning with LFA: python3 lin_dqn.py
b) Double Q-Learning with LFA: python3 lin_doubledqn.py
c) PG with LFA: python3 atari_pong_policy_gradient.py --network='lfa'

Neural Network)
a) Q-Learning with neural network: python3 dqn.py --episodes=1000 --algo='dqn'
b) Double Q-Learning with neural network: python3 dqn.py --episodes=1000 --algo='doubledqn'
c) PG with neural network: python3 atari_pong_policy_gradient.py --network='net'