Homework 2 Template (v1, released Oct. 1st)

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

AndrewId 1: ielshar

AndrewId 2: mharding

Use this template to record your answers for Homework 2. Add your answers using LaTeX and 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 13 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.

Instructions for Specific Problem Types

For questions where you must fill in a blank, 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

Question 1 Convolutional Neural Network

Question 1.a

$\mathbf{y} =$ $\boxed{[4, -3]}$

Question 1.b

1. loss value using original weights: $\boxed{37}$
2. new weights $\boxed{[0.72, 0.34, -0.84]}$
3. output using new weights $\boxed{[3.36, -1.36]}$
4. loss using new weights $\boxed{19.1392}$

Question 2 Policy Iteration and Value Iteration

Policy iteration and Value Iteration algorithms have guaranteed convergence due critically to the Gamma contraction maaping property of the tabular Bellman update operation. In other words, in finite MDPs, a Bellman update operation on any value function will monotonically improve the value function, which in turn can be used to create an improved policy. However, if a function approximator for V(s) or Q(s, a) such as a neural network is used instead of this finite tabular representation, then we lose the Gamma contraction mapping property of the approximator updates. This is because an exact Bellman update operation may not be performed on the function approximator as there are no guarantees od convergence or the neural network might even diverge, and therefore the updates cannot be guaranteed to strictly improve the Value or Q-value functions.

Question 3 Continuation Function

Question 3.a

1. Derive the recurrence relation

$$C(s_1, a_1) = \gamma \sum_{s_2} T(s_1, a_1, s_2) \cdot V(s_2) \tag{1}$$

$$= \gamma \sum_{s_2} T(s_1, a_1, s_2) \cdot \max_{a_2}(R(s_2, a_2) + C(s_2, a_2)) \tag{2}$$

(2) follows (1) as we can show that:

$$V(s_2) = \max_{a_2} \left(R(s_2, a_2) + \gamma \sum_{s_3} T(s_2, a_2, s_3) V(s_3) \right) \tag{3}$$

$$= \max_{a_2} \left(R(s_2, a_2) + C(s_2, a_2) \right) \tag{4}$$

2. Fill in the table

i)

$$V(s) = \max_{a}(R(s, a) + C(s, a))$$

ii)

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot V(s')$$

iii)

$$Q(s,a) = R(s,a) + C(s,a)$$

iv)

$$C(s,a) = \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Question 3.b

1. True, knowing $Q(s,a)$ is better than knowing $V(s)$ if we don't know T, R because $Q(s,a)$ captures all necessary information from T and R, since $\pi^*(s) = \text{argmax}_a(Q(s,a))$ but if we use $V(s)$ to compute the optimal action we need to know $T(s,a,s')$ since $\pi^*(s) = \text{argmax}_a(R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s'))$.

2. False, knowing $C(s,a)$ is better because it captures T and we only need to know $R(s,a)$ whereas $V(s)$ still needs T to compute state values and that information about T is lacking. $\pi^*(s)$ can be computed using only $\pi^*(s) = \text{argmax}_a(R(s,a) + C(s,a))$.

Question 4 Importance Sampling

1. Expectation: $\boxed{0.8}$

   Variance: $\boxed{0.774286}$

2. Direct Sampling

For 10 samples, expected value and variance: 0.5175211497062785 0.22445729082239635
For 1000 samples, expected value and variance: 0.7801754107151546 0.7679549069879695
For 10k samples, expected value and variance: 0.7900644740717525 0.764102137090607

By the strong law of large numbers the as the number of samples increases, the sample mean and variance converges a.s to $\mathbf{E}_p[f(x)]$ and $\mathrm{Var}(f(x))$ (seed used: 10703)

3. Importance Sampling

Q(x) = norm(3,1):
For 10 samples, expected value and variance: 4.295353386364986 166.05054642381438
For 1000 samples, expected value and variance: 1.2630579295295916 55.00040929485666
For 10k samples, expected value and variance: 0.8143918731813137 34.32446852702005
Q(x) = norm(0,1):
For 10 samples, expected value and variance: 0.35818306780353104 0.49598389429460665
For 1000 samples, expected value and variance: 0.7440814888529712 3.703013244355498
For 10k samples, expected value and variance: 0.7658318235847844 3.9772672572375267
Q(x) = $15/16 \cdot x^2 \cdot (1+x)^2$:
For 10 samples, expected value and variance: 0.7999999999999999 8.628166150854816e-33
For 1000 samples, expected value and variance: 0.8000000000000002 3.4944072910962007e-32
For 10k samples, expected value and variance: 0.8 1.1985755378701749e-32
(seed used: 10703)

The proposal distribution $q(x) = 15/16x^2 \cdot (1+x)^2$ gives the lowest variance. This is because its support of values is the same as that of $p(x)$, i.e. $-1 \leq x \leq 1$ and the ratio of $p(x)/q_3(x)$ when multiplied with $f(x)$ equals a constant: $p(x)/q_3(x) \cdot f(x) = 4/5$. Therefore, the variance of $p/q_3 \cdot f$ must aprox. 0 using importance sampling from proposal distribution q_3. The $q_1 = N(3,1)$ proposal distribution converges the slowest because it samples most often outside the range $-1 \leq x \leq 1$ of $p(x)$ since its mean is 3. Many samples were needed to inform the sample mean and sample variance about samples of $f(x)$ within the range of p.

4. Weighted Importance Sampling

Q(x) = norm(3,1):
For 10 samples, expected value and variance: 1.65367501514988
For 1000 samples, expected value and variance: 0.6347482121569935
For 10k samples, expected value and variance: 0.8159765021680583
Q(x) = norm(0,1):
For 10 samples, expected value and variance: 0.5678804940372946
For 1000 samples, expected value and variance: 0.750091065121246
For 10k samples, expected value and variance: 0.7790952791280585
Q(x) = $15/16 * x^2 * (1+x)^2$:
For 10 samples, expected value and variance: 1.7487719726369466
For 1000 samples, expected value and variance: 0.9252790478202617
For 10k samples, expected value and variance: 0.8969532357821838
(seed used: 10703)

In the weighted importance sampling, we see that the sample mean approaches a different value than importance sampling did. Weighted importance sampling is a biased estimator.

Question 5 MC and TD Methods

1. S0: $\boxed{5}$ S1: $\boxed{25}$ S2: $\boxed{20}$ S3: $\boxed{17}$

   S4: $\boxed{5}$ S5: $\boxed{-5}$ S6: $\boxed{-5}$

2. S0: $\boxed{20}$ S1: $\boxed{23}$ S2: $\boxed{21}$ S3: $\boxed{28}$

   S4: $\boxed{31}$ S5: $\boxed{23}$ S6: $\boxed{13}$

3. S0: $\boxed{25.296875}$ S1: $\boxed{28.59375}$ S2: $\boxed{29.1875}$ S3: $\boxed{32.375}$

   S4: $\boxed{29.75}$ S5: $\boxed{20.5}$ S6: $\boxed{13}$

Question 6 Programming Question

6.1, 6.2, 6.3 (a)

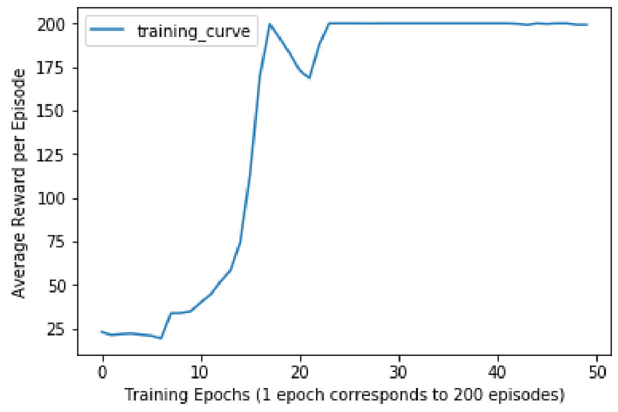
*Dueling architecture's 2 streams were merged using equation (9) from [4].

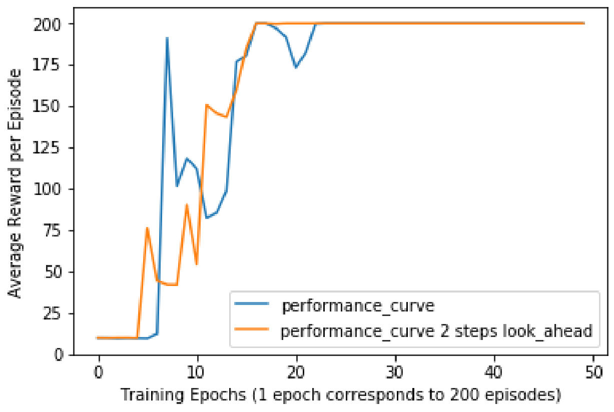
Table 1: Implementation settings

	Settings	DQN	DDQN	Dueling
CartPole-v0	NN MLP Architecture	2 hidden layers each of size 32 with relu activation	2 hidden layers each of size 60 with relu activation	2 streams with 1 relu hidden layer of size 128 each connected to 1 shared hidden layer of size 64 with relu, merged as noted.
	optimizer	Adam	Adam	Adam
	Burn-in transitions	10000	10000	10000
	Initial epsilon	0.5	1	0.5
	Final epsilon	0.1	0.1	0.1
	Exploration decay steps	2×10^5	2×10^5	5×10^4
	minibatch size	32	32	32
	no. of steps until target network is updated	100	100	100
	steps before minibatch	4	1	4
	learning rate	0.0001	0.0001	0.0001
MountainCar-v0	NN MLP Architecture	3 hidden layers each of size 60 with relu activation	3 hidden layers each of size 60 with relu activation	2 streams with 1 relu hidden layer of size 64, sharing 3 hidden relu layers of size 64 each, streams merged as above
	optimizer	Adam	Adam	Adam
	Burn-in transitions	10000	10000	10000
	Initial epsilon	0.5	1	0.7
	Final epsilon	0.1	0.1	0.1
	Exploration decay steps	2×10^5	2×10^5	2×10^4
	minibatch size	32	32	32
	no. of steps until target network is updated	100	100	20
	steps before minibatch	4	4	16
	learning rate	0.0001	0.00007	0.00007

6.1 (b) and (c)

CartPole-v0 DQN training and performace plots:

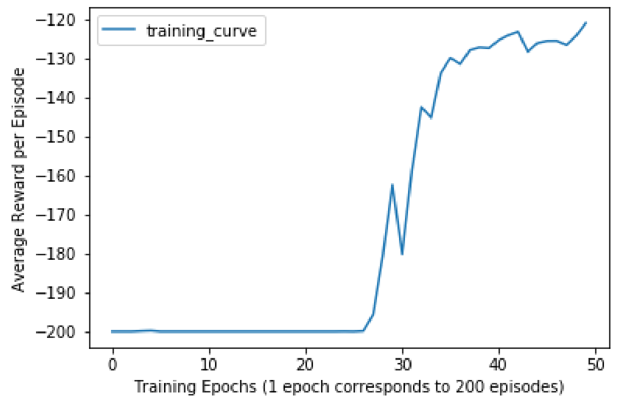


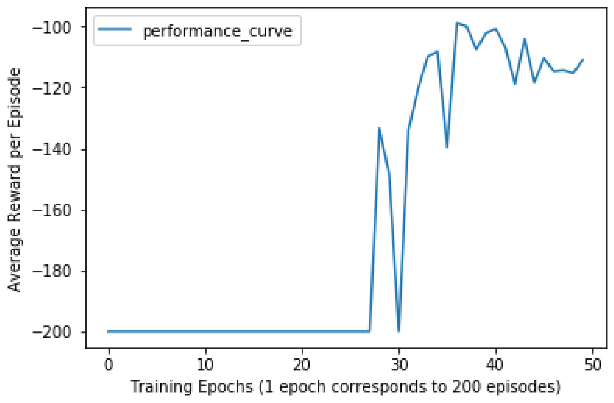


The training curve shows that our agent have learned to perform optimally in CartPole after about 25 epochs where it is stabilized there after.

The performance curves show that the two step look ahead policy performes better than the greedy policy w.r.t Q, where Q is learned from DQN, this result is expected, except at some point around 10 epochs in the training. This could be due to the fact the agent haven't learned Q values close to the true Q values just yet, we see however that this behavior is no longer the case after around 20 epochs where Q has converged to the true values.

Mountaincar-v0 DQN training and performace plots:





The training curve shows that our agent took about 27 training epochs in order to learn the Q value that induce a policy that reaches the flag in Mountain Car. The reason behind this could be due to the fact that Mountain car is a harder environment than cartPole since a random policy which we start with is highly unlikely to reach the flag. This problem could be mitigated by having the NN model predict initially Q values that are some how optimitic. The training curve seems to have converged to average values between -130 and -120 at the end training (min $\epsilon = 0.1$ is reached), which is pretty good for this environment.

The performance curves using ϵ-greedy policy (ϵ=0.05) is consistent with the training curve. The average reward per episode seems to be around -115 near the end of training.

6.1 (d): please refer to the submitted videos.

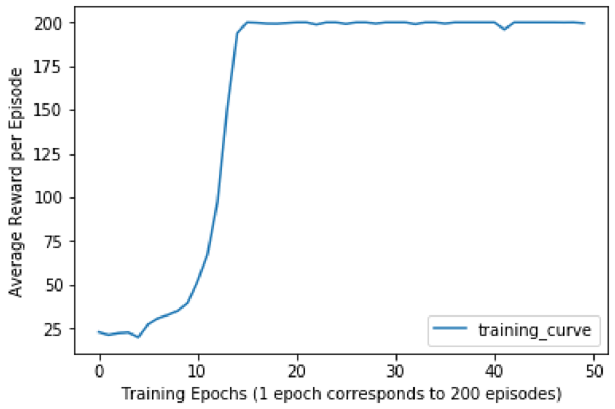
6.1 (e): The below table was generated using a **greedy** policy. The results shows that our agent was able to pass all episodes with max reward for CartPole and performs very well with MountainCar (can reach the flag by approx. 110 steps on avg.).

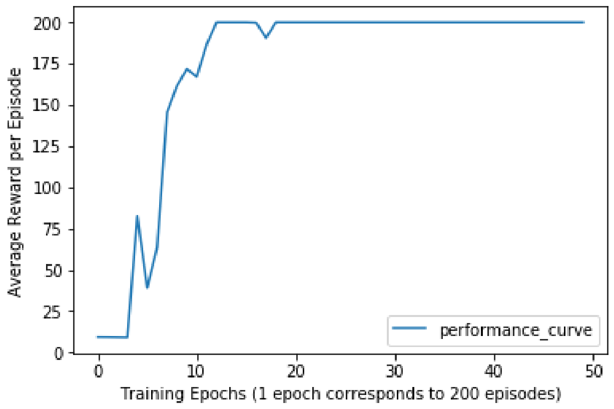
Table 7: DQN: Avg. total reward per 100 episode +/- std.

Setting	**mean ± std.**
CartPole DQN	200.0 ± 0.0
CartPole DQN 2-step LA	200.0 ± 0.0
MountainCar DQN	-109.85 ± 11.6115

6.2 (b) and (c)

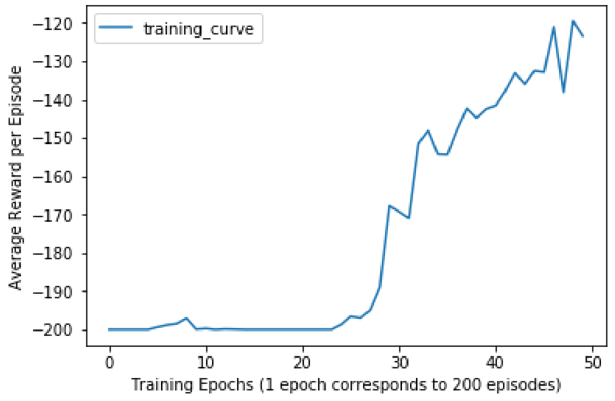
CartPole-v0 DDQN training and performace plots:

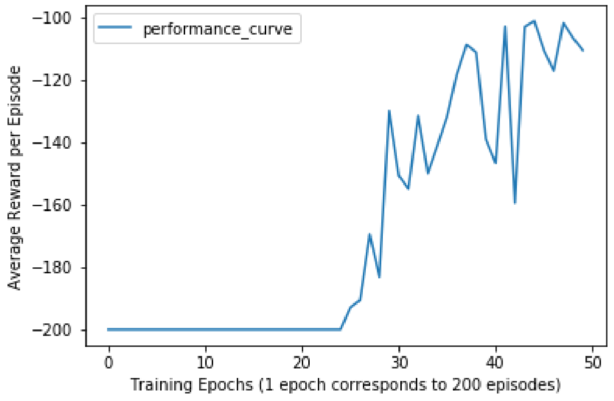




The training curve for DDQN shows that our agent have learned to perform optimally in CartPole after about 15 epochs where it is stabilized there after.
The performance curve shows that the agent have learnt to act optimally where the average reward achieved per episode after 20 epochs is the maximum achievable.
Compared to DQN, DDQN performs consistently better since it solves the Q-values overestimation problem of DQN. Dueling converges quicker than DDQN for this environment, however, DDQN has more robust performance after convergence.

Mountaincar-v0 DDQN training and performace plots:





The training curve for DDQN shows that our agent took about 24 training epochs in order to learn the Q values that induce a policy that reaches the flag in Mountain Car. It also shows that our agent is learning consistently overtime where the avg. reward per episode hits -120 towards the end of training. It is worth mentioning that in our case finding the right hyperparameters for the DDQN to achieve this performance in MountainCar was not an easy task.
The performance curves using ϵ-greedy policy (ϵ=0.05) is consistent with the training curve. The average reward per episode seems to be around -115 on average near the end of training.
DDQN seems to have less effect w.r.t DQN results on this environment the reason may be that fact that it was harder for us to find the best parameters for DDQN. Perhaps better results could be achieved with more parameter tweaking.

6.1 (d): please refer to the submitted videos.

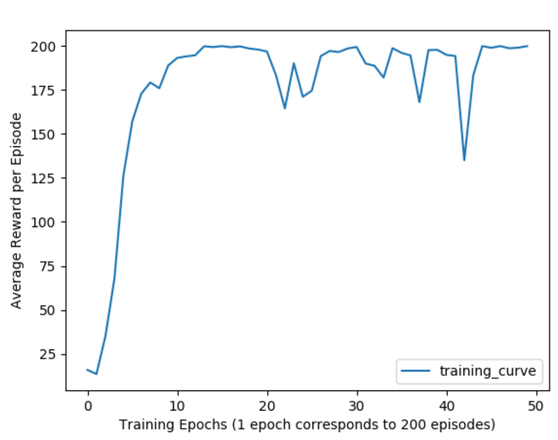
6.1 (e): The below table was generated using a **greedy** policy. The results shows that our agent was able to pass all episodes with max reward for CartPole and performs very well with MountainCar (can reach the flag by approx. 108 steps on avg.). Note that the mean and std. for MountainCar are better than that of DQN.

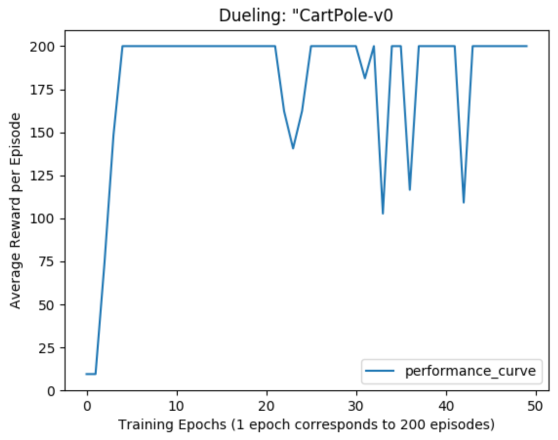
Table 16: DDQN: Avg. total reward per 100 episode +/- std.

Setting	**mean ± std.**
CartPole DDQN	200.0 ± 0.0
MountainCar DDQN	-108.59 ± 9.8347

6.3 (b) and (c)

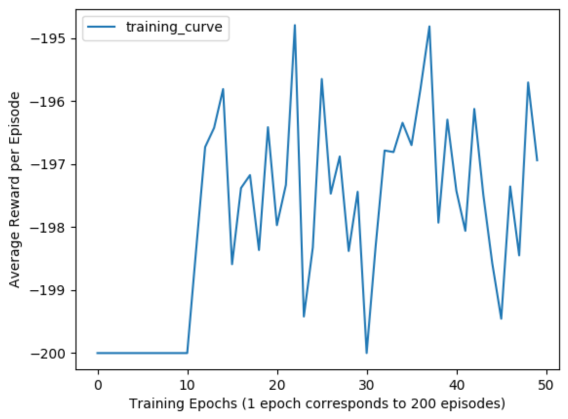
CartPole-v0 Dueling training and performace plots:

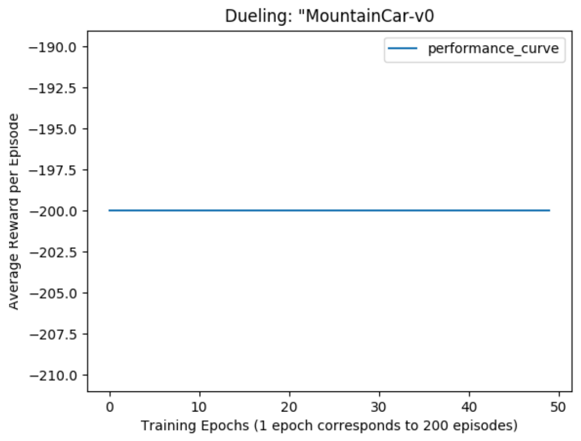




The training curve for the Dueling network shows sooner convergence on the CP task (around training epoch 10) than the other two Q Network models. We believe the drops in performance after epoch 20 and through epoch 40 could be avoided or subdued with a quicker decay of our epsilon value, as Dueling model converged much sooner than the epsilon decayed to its minimal value. The performance curve for the Dueling network is not as stable as that of the DDQN network after 25 training epochs, which may be due again to the differing epsilon decay rates and the fact that the DDQN model ran a minibatch update 4x's as often as the Dueling network. and could therefore have learned more robustly from its epsilon-greedy policy.

Mountaincar-v0 Dueling training and performace plots:





The MC environment was quite difficult for the Dueling network. Hyperparameters tuned included the width of the 2 network streams and the shared layers, epsilon decay rate, learning rate, and optimizer. Also, we experimented with a hyperparameter included in the Atari DQN paper (arXiv), which was the number of frames skipped/over which to repeat previous actions. Ultimately, the best performing instance of the model could stumble upon epsidoe returns above -200 (in the range of [-80, -190] after 10 training epochs, however, this were sporadic enough that the average epoch returns remained close to -200. According to the performance plot, the Dueling network never learned to reach the goal with the best hyperparameters we used. However, with more tuning, we believe that the Dueling network shows promise of converging quicker than DDQN and DQN since on multiple occasions the Dueling model reached the flag sooner than trainig epoch 10. We expected Dueling network 2-stream architecture to again outperform DQN and DDQN as it did in terms of quicker convergence for the CarPole environment. 6.3 (d): please refer to the submitted videos.

6.3 (e): The below table was generated using a **greedy** policy.

Table 24: Dueling: Avg. total reward per 100 episode +/- std.

Setting	**mean ± std.**
CartPole Dueling	200.0 ± 0.0
MountainCar Dueling	-200.0 ± 0.0

Table 25: Summary: Avg. total reward per 100 episode +/- std.

Setting	**mean $\pm$ std.**
CartPole DQN	200.0 $\pm$ 0.0
CartPole DQN (2 step look ahead)	200.0 $\pm$ 0.0
CartPole DDQN	200.0 $\pm$ 0.0
CartPole Dueling	200.0 $\pm$ 0.0
MountainCar DQN	-109.85 $\pm$ 11.6115
MountainCar DDQN	-108.59 $\pm$ 9.8347
MountainCar Dueling	-200 $\pm$ 0.0

The comparison between DQN, DDQN and dueling have been addressed in the previous sections.