Bozidar Antic
Shubham Krishna

Self Driving Cars
Assignment 2

18.12.2019

1

a) We worked with 2 different baseline architectures which had almost similar performance:

- Conv5(s1)(10) → leaky_relu(0.2) → Conv3(s1)(20) → leaky_relu(0.2) → Conv3(s1)(30) → leaky_relu(0.2) → Conv3(s1)(60) → leaky_relu(0.2) → FC(100) → leaky_relu(0.2) → FC(output) {in future text referenced as ***net1***}
- Conv8(s4)(16) → leaky_relu(0.2) → Conv4(s2)(32) → leaky_relu(0.2) → Conv3(s1)(64) → leaky_relu(0.2) → FC(512) → leaky_relu(0.2) → FC(output) {in future text referenced as ***net2***}

  Notation: Conv3(s2)(10): Convolutional layer of filter size 3, stride 2 and 10 output channels.
  FC(100): Fully Connected layer with 100 neurons

Baseline performance of **net1** is 276.617871 and of **net2** is 233.2595. Both networks have sensor values appended to first Fully connected layer.

(i) It would be a problem, because Markov property (assumption) would not hold. Without the use of sensor values, single frame (current state) does not hold enough information on previous states. Even for human, this task can be hard for example one could not tell if car is moving or is stationary, just by looking at a single frame. Without using sensor data this can be resolved by using multiple (stack of) frames as a current state representation.

(ii) We do not need to stack frames as sensor data gives enough information of previous states for Markov property to hold (e.g. acceleration - car's movement, steering sensor (in combination with acceleration) - car's previous location etc.)

b) (i) Using only a single network for predicting the target as well as the output leads to an unstable and oscillatory training process. As when we update the single network, our output moves closer to the target value but simultaneously our target moves further away from the output (prediction).

(ii) We use replay memory in order to break the correlation between past consecutive frames. Learning from a batch of past consecutive frames breaks the independence assumption and leads to inefficient training and result in poor performance.

c) (i) In the beginning of training process deep q-network hasn't learned enough about the value of each action so exploiting doesn't make much sense but exploring all the possible actions in order to learn which actions are more valuable than others (in terms of reward) does. However, at later stages of training exploitation is more valuable but still shouldn't be relied upon completely as sacrificing immediate rewards (by exploring) may lead to states that can yield better future rewards. ϵ greedy algorithm uses a threshold ϵ which acts as a probability with which model will explore and $1 - \epsilon$ as probability to exploit. Value of ϵ can be modified (usually reduced) over training time, because with time, network learns more about value of each action making exploitation a better option than exploration.

d) **(net2)** By observing the reward graph Figure(1) we can say that rewards oscillate over time but increasing trend can be observed. In the beginning rewards are mostly negative due to high value of ϵ and lack of knowledge of agent (q-network hasn't learned much). After 50th episode, rewards become consistently positive. As time passes exploitation becomes more dominant and with that rewards also increases - but the reason is not purely because of lower value of ϵ (which is expected to result in smaller immediate future reward) but also because network is learning. Loss function in standard supervised learning problem is much more stable (should be consistently decreasing) and even though it can display slight 'jumps' over time, oscillations are not nearly as prominent as in the case of reinforcement learning.

e) **(net2)** Observed performance resulted in average score of 233. The agent learned how to drive straight and take not so sharp turns but failed to take sharp ones. It also learned to get back on the tracks but when the speed was high it started to drift when trying to come back on tracks. In some cases, when managed to return to track, car was facing opposite direction. Keeping lane and driving straight, also accelerating in the beginning was learned successfully. Comparing to imitation learning agent, this agent learned to brake before black region every time. However, imitation learning agent displayed much more stable driving as it learns from the start of the training to imitate expert's (perfect) driving, while reinforcement learning needs (a lot of) time in order to discover good driving techniques. But the advantage is that it is not dependent on expert's performance and in theory has a potential to perform much better as it optimizes objective which describes good performance. For our baseline model we did not explore many of hyperparameters of model and training process itself and poor performance is most likely consequence of that.

2

a) Changing discount factor to 0.8 reduced **net1** performance to 205.374110, driving style had a notable change of agent braking and not moving after certain point. Increasing it to extreme value of 1.1 completely ruined the score to -1.898112. When the value of $\gamma \geq 1$, it means agent is too far future sighted. The agent takes every action by giving more importance to expected further future rewards rather than immediate future rewards and hence bringing a lot of factors in consideration before taking any action. These factors can be irrelevant and will bring too much variance in the actions and making it hard to learn. The discount factor determines the importance between immediate future rewards and the further future rewards. Training rewards are displayed in Figure(2).

b) The action_repeat parameter k is used an indicator that the agent will see and select the best action on every kth frame (starting from the first one) and repeat the same action for the next k-1 frames irrespective of what would be the 'best' action in each of those. This also makes sense assuming that for the good driving action will not change very frequently in small number of frames. Decreasing this parameter to 1 resulted in slight drop in performance to 256.051751, it also displayed oscillatory turning while driving on the straight road. Increasing action repeat to value 6 leads to improvement of average score to 313.068252. Agent learned to brake before turns but it sometimes stopped completely. Training plots and rewards can be seen in Figure(3)

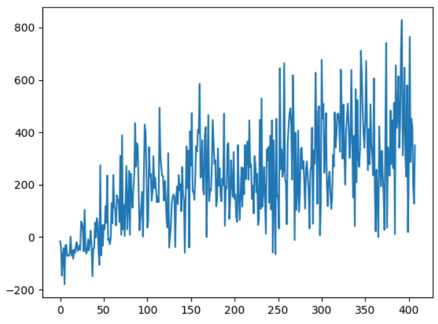
c) **net1** Addition of a null action improved agent's driving by reducing unnecessary turning while driving on the straight road. This further led to an improvement in a score of 377.390752! Even though it can be concluded that optimal agent would never do null action - optimally it would always try to drive as fast as possible and apply just the right amount of brake to be able to make successful turn, agent trained with these training hyperparameters (first of all number of episodes) benefited from null action. We changed intensity of turn action from 1.0 to 1.5 and noticed that it seemed to help but not significantly, as agent achieved 287.856758. It started displaying strange behaviour of stopping suddenly and not moving and sometimes struggled taking sharp turns. We also expanded initial set of actions with this higher intensity turns (so set of actions now contained 6 actions) and this improved evaluation score to 298.915573 but agent started driving on the left border on the road - which might not be the consequence of these new actions but rather unfortunate training and need for exponential increase in training data with increasing output dimension. This is also the reason why it is not always the best idea to increase action space. Also continuous actions are problematic with DQN approach as taking a maximum value of continuous set of actions is not an easy task. There exist multiple solutions to this problem as restricting function to be convex, using optimization to find minimum, actor critic algorithms etc. Training loss and reward graphs can be seen in Figure(4)

d) **net1** double q 341.908321 [still speeding a bit but better braking before turns, overall more stable] In DQN we have $Q^*(s,a) = E(r + \gamma max_{a'} Q^*(s',a'))$ From this equation, it is clear that when we take the maximum of the Q values in the next state we are overestimating as the Q values are noisy. When we take the maximum over the Q values of next state, DQN uses the same values to select and estimate an action which makes it more likely to overestimate. The Double Q learning solves the problem of overestimation by using two different function approximators: one for selecting the best action and the other for calculating the value for this action and since these function approximators are trained on different samples, it is unlikely that it will lead to overestimation. When we used double Q learning it improved overall average score to 341.908321. Driving became more stable, almost no unnecessary turns while driving straight, better braking before taking turns but the agent still struggled some time with control while driving high speeds. Training loss and rewards can be seen in Figure(5)

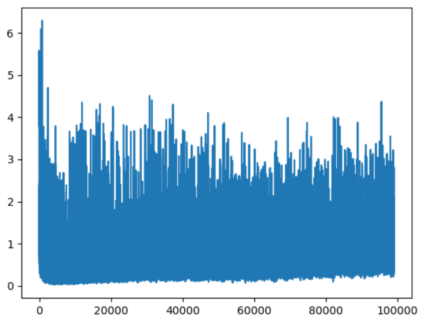
e) Before going into best resulting agents, it is interesting to mention that by modifying **net1** architecture, so that it receives full image (not cropped) as input and by not including explicit extracted sensor data into fully connected layer, agent acquired score of **455.062542**! We then trained 2 different architectures with modified training parameters using some of the approaches from section 2. Previously mentioned experiment led us to believe that extracted sensor data when appended to the fully connected layer

brought 'noise' for the agent in this training conditions and possibly not having enough nonlinearities (as net1 had only 1 fully connected layer) worsened the effect so we added wider and deeper fully connected block to **net1** (neurons: FC1(400), FC2(300), FC3(200)). This network will be referenced as **deepernet1**. Second network utilized previous discovery of using full images with **net1** architecture, slightly modified by adding additional fully connected layer of 100 neurons. This network will be referenced as **nosensnet1**.
Both networks were trained using double Q learning approach, with action_repeat set to 5 and γ to 0.99. **deepernet1** was trained for 350k steps while **nosensnet1** for 800k steps.
deepernet1 achieved score of 502.172749 and displayed very stable driving, with no mistakes like going out of the track. It also successfully used brakes before every turn with no stopping while doing that.
nosensnet1 achieved score of 517.824080 but in comparison to **deepernet1** it was a bit more unstable. There were drives where it stopped while braking before turns, it went out of the track. However it was able to achieve significantly better scores in single drives, reaching 700.
It seems that these approaches solved most of the problems displayed in part 1, but still scores of 900+ were not achieved. One idea would be not to sample experience buffer uniformly but based on 'importance' of each sample, as not every sample is equally useful in terms of learning.
Training images can be seen in Figure(6)

3 Appendix

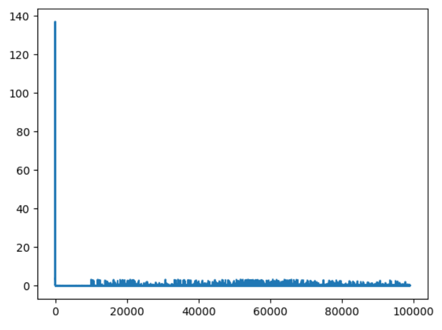


(a) Training Rewards

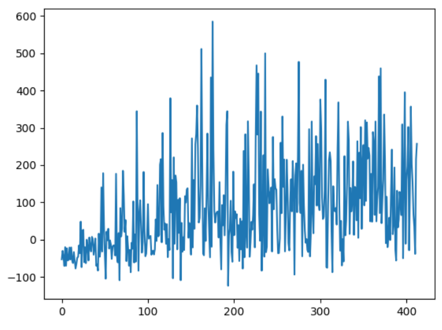


(b) Training Loss

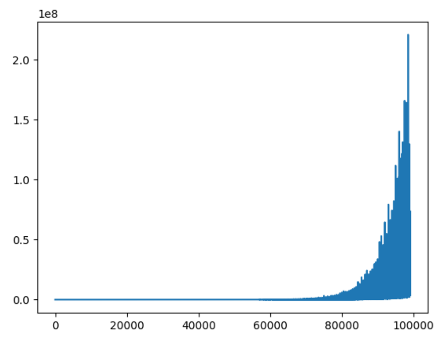
Figure 1: Part 1d)



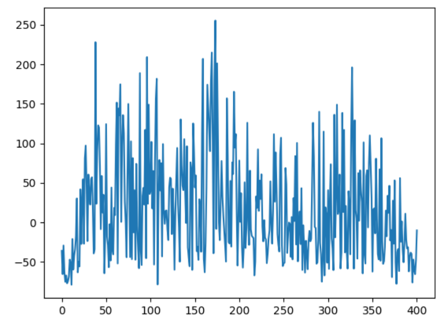
(a) net1 discount 0.8 loss



(b) net1 discount 0.8 rewards

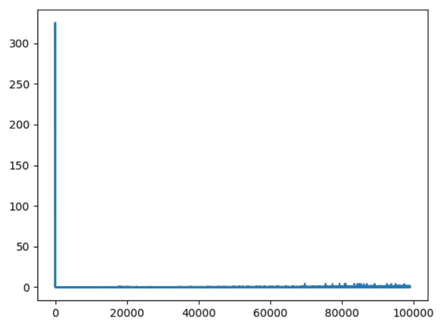


(c) net1 discount 1.1 loss

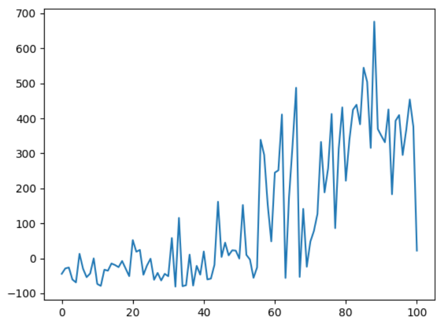


(d) net1 discount 1.1 rewards

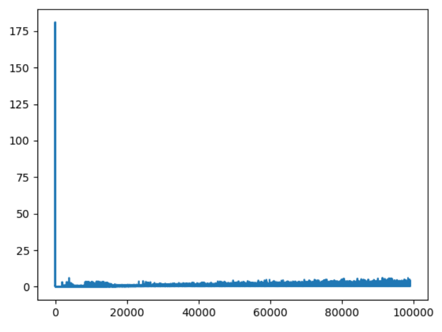
Figure 2: Part 2a)



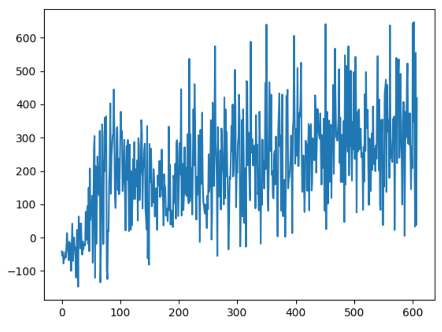
(a) net1 action repeat 1 loss



(b) net1 action repeat 1 rewards

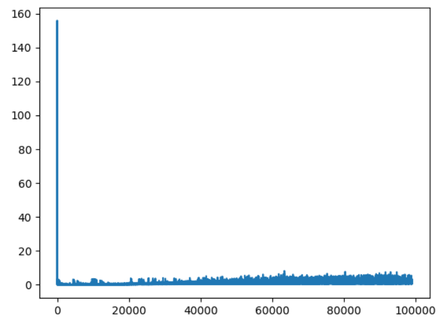


(c) net1 action repeat 6 loss

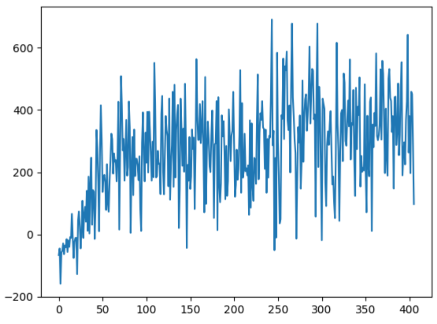


(d) net1 action repeat 6 rewards

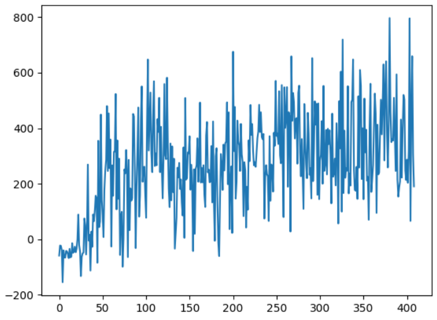
Figure 3: Part 2b)



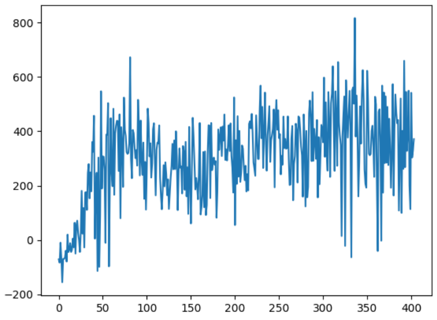
(a) net1 no action loss



(b) net1 no action rewards

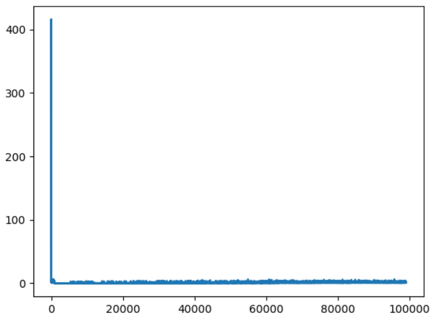


(c) net1 turn 1.5 rewards

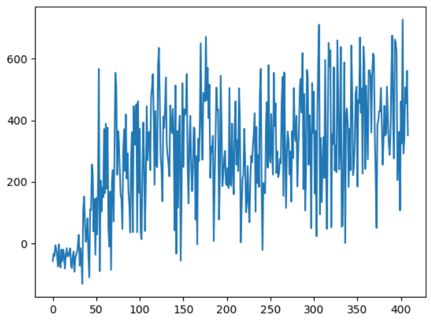


(d) net1 added action turn 1.5 rewards

Figure 4: Part 2c)

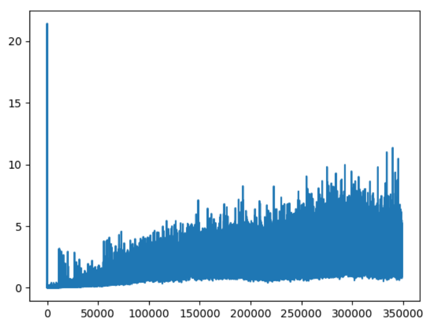


(a) net1 doubleQ loss

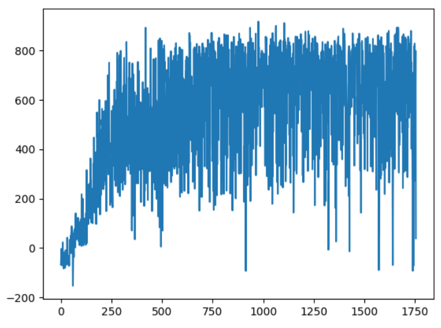


(b) net1 doubleQ rewards

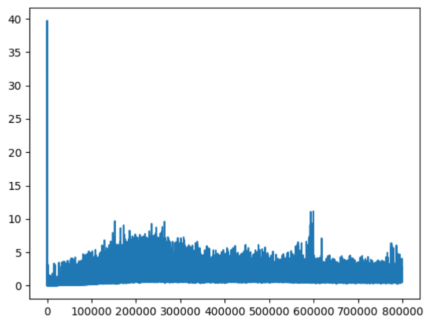
Figure 5: Part 2d)



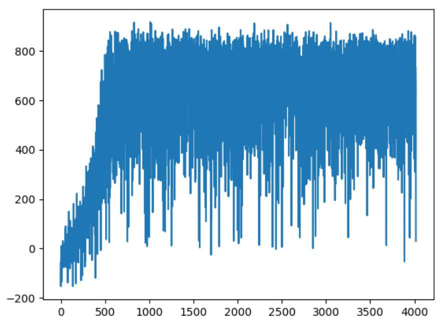
(a) deepernet1 loss



(b) deepernet1 rewards



(c) nosensnet1 loss



(d) nosensnet1 rewards

Figure 6: Part 2e)