SDC, Exercise 2 - Deep Q-Learning , Driton Goxhufi

Freitag, 22. November 2019 08:42

Questions:

Q 2.1

a) Deep Q-Network: Implement the Network.
   My DQN architecture follows the structure shown in the Nature Paper: "Human-level control through deep reinforcement learning". I added some generalization enhancements like DropOut Layers and leaky relu's, which helped a lot.

   Architecture:

```
DQN(
  (conv_net): Sequential(
    (0): Conv2d(1, 16, kernel_size=(8, 8), stride=(2, 2))
    (1): Dropout2d(p=0.2, inplace=False)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
    (4): Dropout2d(p=0.5, inplace=False)
    (5): LeakyReLU(negative_slope=0.2)
  )
  (head): Sequential(
    (0): Linear(in_features=14112, out_features=256, bias=True)
  )
  (q_values): Linear(in_features=256, out_features=4, bias=True)
)
```

   i) Q:
   Would it be a problem if we only trained our network on a crop of the game pixels, which would not include the bottom status bar and would not use the extracted sensor values as an additional input to the network?

   i. Answer:
   Apart from information loss, hence a decrease in performance, it should not cause big problems.
   Yes it could be a problem, because our task properties doesn't fulfill the Markov assumption. Hence future actions depend on past actions. For example if our agent is on the track and right in front of a curve, it is suitable to know how to adjust the current speed of the agent, to get a good estimate of steering the wheels -> estimate good future action
   So if we crop of the game pixels and don't use extracted sensor values like described, the task properties would flip

to fulfilling the markov assumption and future actions just depend on the current state.

ii) **Q:**
Why do we not need to use a stack of frames as input to our network as propose3d to play Atari games in the original Deep Q-Learning papers?[1][2] [https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf][https://arxiv.org/pdf/1312.5602.pdf]

   i. **Answer:**
   Because in Q-Learning we just need the current state(observations) and the possible actions to compute the next step.

b) Deep Q-Learning:

i) **Q:**
Why do we utilize fixed targets with a separate policy and target network?

   i. **Answer:**
   - Beause we produce our target values during training. By freezing the target network for x-steps before updating it with a copy of the policy network weights, we guarantie a stable behavior.
   - if we dont utilize fixed targets, the convergence of training can be much harder because weights gets updated after every step and then a very noisy target values are used to update each step.

ii) Why do we sample training data from a replay memory instead of using a batch of past consecutive frames?

   i. **Answer:**
   Because consecutive frames are very similar and the agent might be enforced to stick to a specific action, because the greediest action is always the same one. **--> Prone to oscillation!**
   Instead with picking random samples from a replay memory, we force the agent to see and learn from different frames without the consecutive attribute.

c) Action selection:

i) **Q:**
Why do we need to balance exploration and exploitation in a reinforcement learning agent and how does the ϵ-greedy algorithm accomplish this?

i. **Answer:**
We need to balance it, because if the agent just does exploration, we might never converge to the desired goal and if we only follow exploitation, the agent tend to get stuck in a "loop" of known paths.
The ϵ-greedy algorithm tackles this by a build in random mechanism. The algorithm choose with a probability of $1 - \epsilon$ the greedy action(eploitation) else, it does a random action to force exploration.

d) Training:

i) How quickly is the agent able to consistently achieve positive rewards?

ii) What is the relationship between the ϵ-greedy exploration schedule and the development of the cumulative reward which the agent achieves over time?
Answer:
The ϵ-greedy exploration schedule adjusts the value of ϵ in a manner of decaying over time. It starts with a p -value of 1.0 and decays until it reaches 0.02.
The idea is to lower ***random*** actions after a fixed number of timesteps, because the network should become better(reach higher cumulative reward values) in solving the task as the training progresses.

iii) How does the loss curve compare to the loss curve that you would expect to see on a standard supervised learning problem?

e) Evaluation:

i) Where does the agent do well and where does it struggle?

ii) How does its performance compare to the imitation learning agent you have trained for Exercise 1?

iii) Discuss possible reasons for the observed improvement/decline in performance compared to your imitation learning agent from Exercise 1.

Q2.2 - Further Investigations and Extensions

(I couldn't put any time into this section, mainly because my course partner dropped the SDC-course very surprisingly... i hope i can catch up in the last exercise)

a) Discount Factor

i) We typically use this factor, because we want that things that our network learned recently, should have more value.