Reinforcement Learning

Student name and id: Nicklas Hansen (s153077)

Hand-in for week: 1

Exercise 1.1: *Self-play*

The model would likely converge faster when playing itself, as it could learn from both perspectives at the same time. Additionally, it would likely learn more sophisticated strategies, as it does not have a bias towards certain strategies like a human does and it may outperform human players eventually.

Exercise 1.2: *Symmetries*

Updates to value of a given board state also affect all mirrored boards because of symmetry. A model learns faster that way as state space decreases, but if a human opponent is biased towards certain symmetries and does not take advantage of symmetry, the model would not have learned that information and thus performs comparatively worse.
Therefore, depending on the opponent, an agent should not consider symmetries as identical, but might want to use that knowledge early on in the training process to learn something meaningful faster.

Exercise 1.3: *Greedy Play*

In most real-life examples, greed is necessary in order to incrementally improve the given agent, without exhausting all possible states in the state space (which is virtually impossible for many RL problems). Too much greed, however, hinders the agent from exploring strategies that are more valuable in the long run (i.e. lower reward in near future, but high expected value). Exploitation vs. exploration is thus an important trade-off in RL and basing your learning on greed alone would generally perform worse than a non-greedy strategy in the long run.

Exercise 1.4: *Learning from Exploration*

Consider the following two scenarios: agent A does not learn from exploratory moves and only considers the optimal move at each time step - agent B, however, learns from exploratory moves at a fixed, non-zero rate, and the expected value at future states thus take into account the potential from "suboptimal" paths that later might turn out favorable. Not learning from exploration may put the agent's learning strategy in a local minima, resulting in a poor converge. Therefore, including exploratory learning is most likely the best strategy and should yield better results, unless we are in the odd case where agent A in fact does converge at the optimal solution despite its lack of exploratory learning. Though, as agent B performs exploratory moves at a fixed rate, it will continue to perform poor actions despite actually converging at a good strategy, which will negatively impact its reward.

Exercise 1.5: *Other Improvements*

In the previous exercise, the exploration rate was fixed. Using an incrementally decreasing exploration rate may offer a better learning scheme over a fixed exploration rate assuming a stationary problem, i.e. the opponent does not change their play-style over time. Once the agent has learned sufficiently, a fixed rate may result in unnecessary losses due to some degree of forced exploration despite already knowing the optimal solution.

Exercise 2.1

If $\varepsilon = 0.5$, the probability for a greedy action g at time-step t is $Pr(A_t = g_t) = 0.5 + (1-\varepsilon)/2 = 0.75$.

Exercise 2.2: *Bandit example*

After $t = 3$, the sample-average for action 1 is -1 and the sample-average for action 2 is -0.5. Therefore, a greedy action would be either action 3 or action 4 (both with a sample-average of 0). As the agent selects action 2 anyway means that A_4 must have been a randomly sampled move. Additionally, A_5 is also a randomly sampled action, as it selects $A_5 = 3$ (sample-average 0) despite discovering that the reward of action 2 is greater than 0.
Any of the 5 actions can have been randomly sampled as a greedy action might as well be selected randomly.

Exercise 2.3

Although the $\varepsilon = 0.01$ action-value method converges slowly, it will eventually converge at an optimal action percentage of $1 - \varepsilon = 0.99$, producing a greater cumulative reward than the other two actions in the long run. As seen in the "average reward" graphs of Figure 2.2, $\varepsilon = 0.01$ almost is on par with $\varepsilon = 0.1$ after 1000 steps, so one can only imagine its average reward once it has converged.

Exercise 2.4

If the step-size parameters α_n are not constant, the estimate Q_n is a weighted average of previously received rewards with a different weighting from that given by equation (2.6) in the book, but let's base our derivation on that.
Just like the case of constant step-size, we rewrite the equation to an expression with only one recursion and then expand that recursion until we get the initial case of Q_1:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n(R_n - Q_n) \\
&= \alpha_n R_n + (1-\alpha_n)Q_n \\
&= \alpha_n R_n + (1-\alpha_n)[\alpha_{n-1}R_{n-1} + (1-\alpha_{n-1})^2 Q_{n-1}] \\
&= \alpha_n R_n + (1-\alpha_n)(1-\alpha_{n-1})R_{n-1} + (1-\alpha_n)(1-\alpha_{n-1})Q_{n-1} \\
&= \left[\sum_{i=1}^{n} R_i \left(\prod_{j=i}^{n-1} 1-\alpha_{j+1}\right)\right] + \left(\prod_{i=1}^{n} 1-\alpha_i\right) Q_1
\end{aligned}$$

Exercise 2.5 *(programming)*

Code available at `https://github.com/s153077/rl_sutton_barto/blob/master/Week1.ipynb`.
Figure 1 shows the output of the exercise. The first plot shows the average reward for each of the agents, whereas the second plot shows the optimal action percentage over time.
It can be concluded that on the non-stationary bandit problem presented, the constant step-size (orange) agent learns much faster and appears to converge towards its optimal action percentage of $1 - \varepsilon = 0.9$. In part because of the non-stationary nature of the problem, the other agent quickly converges on a selected action, which does produce a decent reward, but is sub-optimal and comparatively worse than the optimal action.
It can be concluded that constant step-size performs far better in a non-stationary environment.

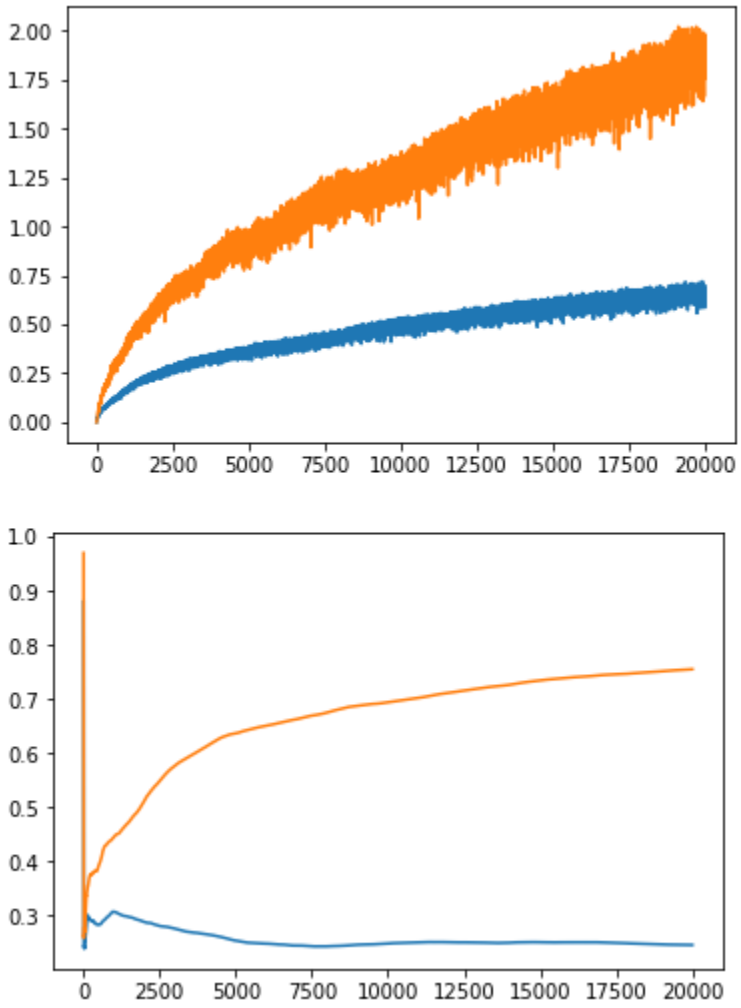

Figure 1: Results from exercise, average over 100 runs. (blue) sample-average step-size. (orange) constant step-size.

Exercise 2.6: *Mysterious Spikes*

Early in the learning process, an optimistic method forces the agent to try different actions as it is disappointed in all of them. Due to the optimistic bias, it believes that all actions are good and thus try each of them a number of times until the Q value has decreased sufficiently for all of the poor performing actions.
Therefore, it discovers an optimal action faster than the realistic ε-greedy method that explores on a fixed rate.

Exercise 2.7: *Unbiased Constant-Step-Size Trick*

The new step-size $\beta_n = \alpha/\bar{o}_n$ uses a recursively defined trace starting at 0 and defined as follows:

$$\bar{o}_n = \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{for } n \geq 0, \quad \text{with } \bar{o}_0 = 0$$

Inserting the new step-size into the same equation as derived in Exercise 2.4, we get the following expression:

$$\begin{aligned}
Q_{n+1} &= Q_n + \beta_n(R_n - Q_n) \\
&= \beta_n R_n + (1 - \beta_n)Q_n \\
&= [\bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1})]R_n + [1 - \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1})]Q_n \\
&= \left[\sum_{i=1}^{n} R_i \left(\prod_{j=i}^{n-1} 1 - (\bar{o}_j + \alpha(1 - \bar{o}_j))\right)\right] + \left(\prod_{i=1}^{n} 1 - (\bar{o}_{i-1} + \alpha(1 - \bar{o}_{i-1}))\right) Q_1
\end{aligned}$$

Exercise 2.8: *UCB Spikes*

Initially, UCB selects randomly among untried actions. As the problem is a 10-armed testbed, UCB is occupied with untried actions for the first 10 steps. On the 11th step, a huge spike in performance appears. This is due to the UCB method now utilizing the formular

$$A_t = \arg\max a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

which for step 11 means that the variance term is equal for all actions. UCB thus maximizes based on expected reward $Q_t(a)$, causing a spike at that specific time step. At the following steps, however, the variance has decreased by a factor of 2 and other, possibly less rewarding, actions are selected for some time.
If $c = 1$ rather than 2, the variance term is also decreased by a factor of 2 for all actions, which means that - depending on the estimation of reward - some relatively well performing actions are likely favored over poorly performing actions despite the variance term.

Exercise 2.9

The logistic function is given by

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

where L denotes the maximum value, k controls rate of growth and x_0 is the offset. In binary classification, a special case of the logistic function where $L = k = 1$ and $x_0 = 0$ is commonly used (denoted σ):

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

The defined $\sigma(x)$ has a couple of convenient properties. The function is defined for all values of x, but consistently produces values in the $[0; 1]$ range, which is easily interpreted as the probability of belonging to one class or the other (or in the context of reinforcement learning, *actions*).
Another function with such a property is the *softmax* function, which is a generalization of $\sigma(x)$. For binary problems, the two functions are equivalent, but the logistic function is faster to compute and may thus be the better choice for binary classification problems if performance is key. Softmax can, however, be applied to problems with any number of classes.

Softmax $\phi(x)$ is defined as follows:

$$\phi(x)_j = \Pr(y = j \mid x) = \frac{e^{x_j}}{\sum_{n=0}^{N} e^{x_n}}$$

where k is the number of classes (which would be 2 in a binary classification problem) and y is the target class.
For $k = 2$, the following holds:

$$\begin{aligned}\sigma(x)_j = \frac{e^{x_j}}{1 + e^{x_j}} &= \frac{e^{x_j}}{\sum_{n=0}^{2} e^{x_n}} = \phi(x)_j \\ &= \frac{e^{x_j}}{e^{x_{\neg j}} + e^{x_j}} \\ &= \frac{e^{x_j}}{1 + e^{x_j}}\end{aligned}$$

where j and $\neg j$ are classes in a binary classification problem.

Exercise 2.10

The two cases presented are as follows:

$$\text{Case A: } \Pr(q_1 = 0.1,\ q_2 = 0.2) = 0.5$$
$$\text{Case B: } \Pr(q_1 = 0.9,\ q_2 = 0.8) = 0.5$$

The problem is an associate search task, as we on each time step can identify whether we face case A or case B.
A good strategy for solving a problem like this within the framework of what has already been discussed would be to apply a system of learning methods, such that each distinct case has its own learning method and value estimates. As each distinct case is stationary, an agent would be expected to perform optimally in each of the two cases (in the long run), which would produce an average reward of $\text{mean}(\{0.9, 0.2\}) = 0.55$.