# Exercise 1

Breno Aberle - 876438
ELEC-E8125 - Reinforcement Learning

September 21, 2020

## 1 Task 1

In task 1, we were supposed to train the model with 200 and then test the trained model with 500 time steps per episode. In figure 1 the respective reward history for the training is plotted.

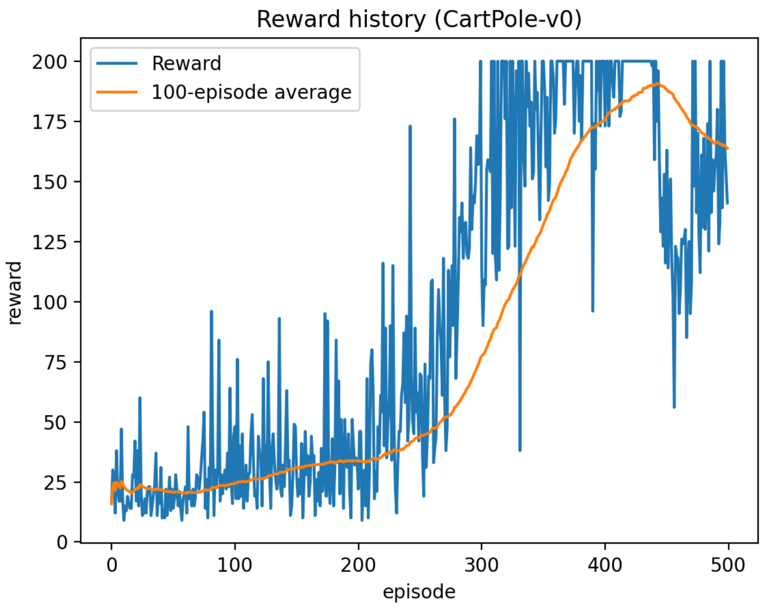


Figure 1: Reward history for training the model with 200 time steps.

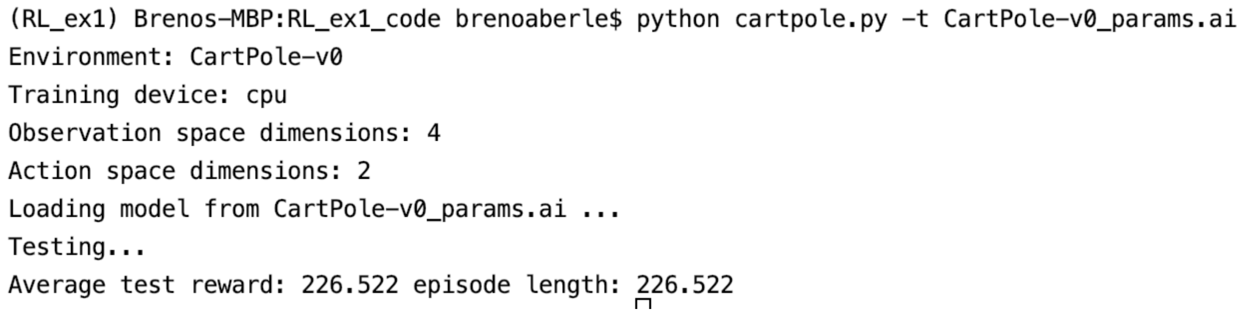
## 2 Question 1.1

*Can the same model, trained to balance for 200 timesteps, also balance the pole for 500 timesteps? Briefly justify your answer.*

First of all, testing the trained model with 500 time steps via the following command

```
python cartpole.py -t CartPole-v0_params.ai --render_test
```

we get the output shown in figure 2.



```
(RL_ex1) Brenos-MBP:RL_ex1_code brenoaberle$ python cartpole.py -t CartPole-v0_params.ai
Environment: CartPole-v0
Training device: cpu
Observation space dimensions: 4
Action space dimensions: 2
Loading model from CartPole-v0_params.ai ...
Testing...
Average test reward: 226.522 episode length: 226.522
```

Figure 2: Output of testing model with 500 time steps.

So we can see that the average test reward and episode length is 226.52. So in that case the model trained with 200 time steps can not balance the pole for 500 time steps.

## 3 Task 2

The table 1 shows the average test reward for each experiment.

| Experiment number | Average test reward |
|---|---|
| 1 | 226.53 |
| 2 | 482.02 |
| 3 | 190.47 |
| 4 | 500.00 |
| 5 | 146.99 |
| 6 | 499.58 |
| 7 | 307.56 |

Table 1: Repeating the experiment and documenting the respective average test reward.

Furthermore, we can also evaluate the performance by rendering the test. For that, I need to describe what I see. Depending on the experiment, the cart is usually not able to balance the pole for 500 timesteps during testing. However, after several trainings the cart is able to balance the pole for 500 timesteps.

## 4 Question 1.2

*Are the behavior and performance of the trained model the same every time? Analyze the causes briefly.*

In the table 1 we can see that the behavior and performance of the trained model is not the same every time. The models are trained for 200 timesteps and that does not mean that they always achieve an average test reward of 500 for 500 timesteps. However, it could be that a model trained on 200 timesteps achieves an average test reward of 500 like in experiment 4. Due to stochastics, the parameters are trained in a way that they can even pass a test of 500 timesteps. However, the average test reward varied in my case between 190.47 up to 500.00.

## 5 Question 2

*Why is this the case? What are the implications of this stochasticity, when it comes to comparing reinforcement learning algorithms to each other? Please explain.*

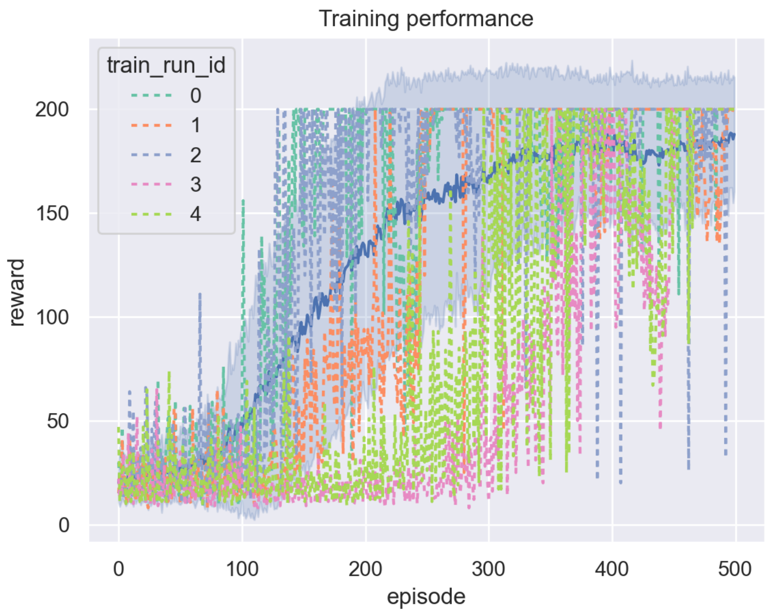


Figure 3: Mean and standard deviation throughout 100 independent training procedures.

The academic research paper [1] deals with stochastic Reinforcement Learning. It is stated that most learning episodes the rewards as well as the punishments are often non-deterministic, and underlying stochastic elements exist. These stochastic elements can't be known in advance. They are numerous and tend to be connected to with the reward and punishment patterns. If stochastic elements were not presented the outcome would be every time the same. In addition, learning evnironments are often noisy. The impact of stochastic on several independent training procedures can be seen in figure 3.

## 6 Task 2

### 6.1 Version 1

*Balance the pole close to the center of the screen (close to $x$ = 0).*

So, the first thought of this exercise is to give less reward the more away the cart is from the center. For that, I needed to identify the min and max values of the observation variables. The position ranges from -2.4 to 2.4 and the velocity from -inf to inf. Therefore, we need to define the highest reward in x=0 and for every distance unit further away from the center the reward gets smaller. Consequently, out of intuition I was starting with 5 - abs(state[0]). This didn't work out that, because the cart most of the time left the boundaries. Even after several training the result was still bad and therefore it is not a stochastic incident. Then I tried to make negative rewards possible with 1.5 - abs(state[0]). It worked better, but not every trained model was good. Then I tried 2.4 - abs(state[0]), so every reward is ¿= 0 with lower rewards than before. The better results are achieved with 2 - abs(state[0]). I was also thinking of using a function like 1/x so the reward grows exponentially grows the closer you get to the center. While working also on version 2 I could transfer the knowledge from version 2 to version 1 to improve it. So, it is basically the same function as for version 2 but we don't need the target position parameter.

```python
def new_reward(state, version=3):
    if (version == 1): # reward function task 2.1
        if (state[0] == 0):
            reward = 4
        else:
            reward = min(4, 1 / abs(state[0]))

    elif (version == 2): # reward function task 2.2
        target_position = args.position
        if (state[0] == target_position):
            reward = 5
        else:
            reward = reward = min(5, 1 / abs(state[0] -
               target_position))

    elif (version == 3): # reward function task 2.3
        velocity = state[1]
        if (-1.8 <= state[0] <= 1.8):
            reward = 2 + abs(velocity)*2
        elif (state[0] < -1.8):
            if (state[1] > 0):
                reward = 1
            else:
                reward = -1 # -1 # maybe -(velocity**2)
        elif (state[0] > 1.8):
            if (state[1] < 0):
                reward = 1
            else:
                reward = -1
        else: # just to be sure all cases are met
            reward = 0.001

    else: # so that code still runs with default setting
        reward = 1

    return reward
```

The code above shows my implemented reward function. I have implemented all three reward function into one function and can change between the three of them by assigning 1,2, or 3 to the *version* variable. I implemented it that way out of simplicity and clarity.

## 6.2 Version 2

*Balance the pole in an arbitrary point of the screen (x = x0), with x0 passed as a command line argument to the script.*

My first thought was to use the linear approach of from Version 1 and shift it to the target position x0. However, that doesn't work at all. That is because we don't have the benefit of starting at the center x = 0 like in v1. It seems to be more difficult than expected. The difficult part is to reach x0. As a consequence, I tried an 1/x approach to give exponentially higher rewards the closer we get to x0. In that case we achieve useful results. As a next step, we try to put it to the next level with 1/(x**2) but that was worsening the results. However, x=x0 is not defined and we treat that position as a special case by assigning it with 4. It worked best by choosing a number close to 4. See the final version in the code above.

### 6.3 Version 3

*Keep the cart moving from the leftmost to rightmost side of the track as fast as possible, while still balancing the pole.*

Implementing the third reward function was by far the most challenging task and consumed a lot of time. Firstly, I have split the position into the three areas [-2.4 ¡= S1 ¡ -1.8 ¡= S2 ¡= 1.8 ¡ S3 ¡= 2.4]. On the one hand, in S1 the agent receives positive reward for positive velocity and negative reward for negative velocity. On the other hand, in S3 the agent receives positive reward for negative velocity and negativ reward for positive velocity. The reason is to indicate that in S1 and S3 the agent should turn around. Furthermore, in S2 the agent receives by default higher reward by being in the section S2 plus additional reward for velocity. That is to make sure that the cart drives faster.

I tried a lot of different scenarios which I will explain in the following. One simple and acceptible solution was 1 + abs(velocity). However, you can only see some useful result with at least 1000 timesteps. To square the velocity didn't tell and that makes sense. The velocity ranges approximately from -1.5 to 1.5. So most of the values range form [-1, 1]. If you square or put numbers below 1 to the power they get smaller and consequently the reward gets smaller. If I put the reward in S2 for higher velocity too high it also has a negative impact. Because the cart tends to go out of bound. Putting the rewards higher in S1 and S3 helped slightly but didn't had a great impact. For clarity I kept them simple with -1 and 1.

## 7 Task 3

The figure 4 shows the training performance of 10 trained models and the average of them. The training are independent. We trained with 1000 episodes and the y-axis shows the reward.

## 8 Question 3

By testing and rendering the trained model file we can observe the behaviour. The cart moves from right to left and back while it increases the speed in the middle. In figure 5 we

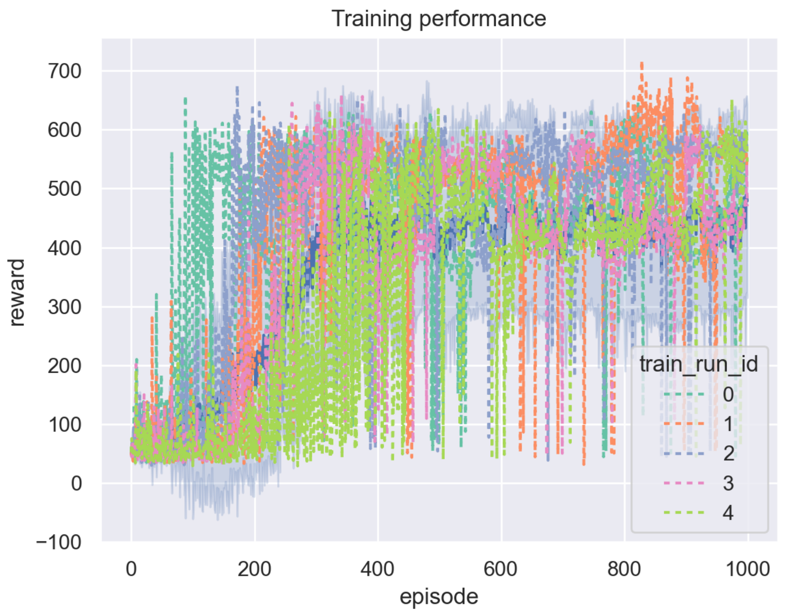


Figure 4: Training performance for 10 trained models of the third reward function.

can see the velocity history. It looks like a sinus function. The velocity ranges between 1.5 and -1.5. This shows that the cart is bouncing smoothly back and forth.

## References

[1] N. L. Kuang, C. H. C. Leung, and V. W. K. Sung, "Stochastic reinforcement learning," *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2018.

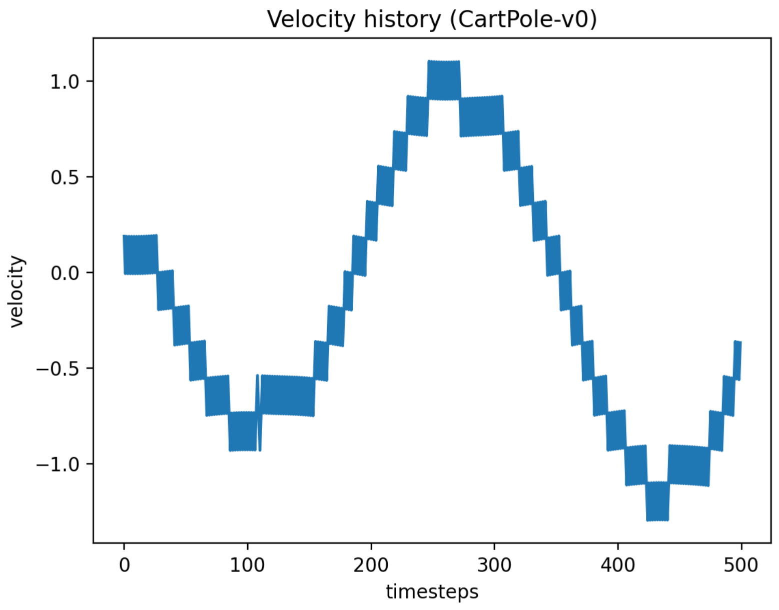


Figure 5: Mean and standard deviation throughout 100 independent training procedures.