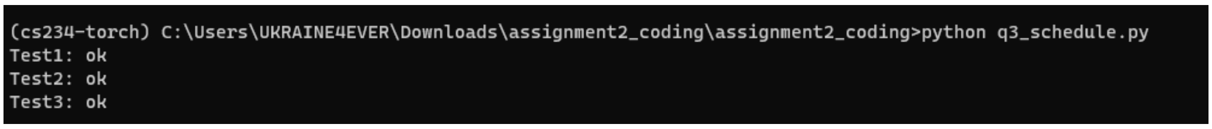
Daria Herasymchuk. Assignment 2.

Firstly, I created a new Conda environment cs234-torch. Then installed the package requirements by running “pip install -r requirements.txt” on a terminal, and installed MinAtar.

Tabular Q-Learning.

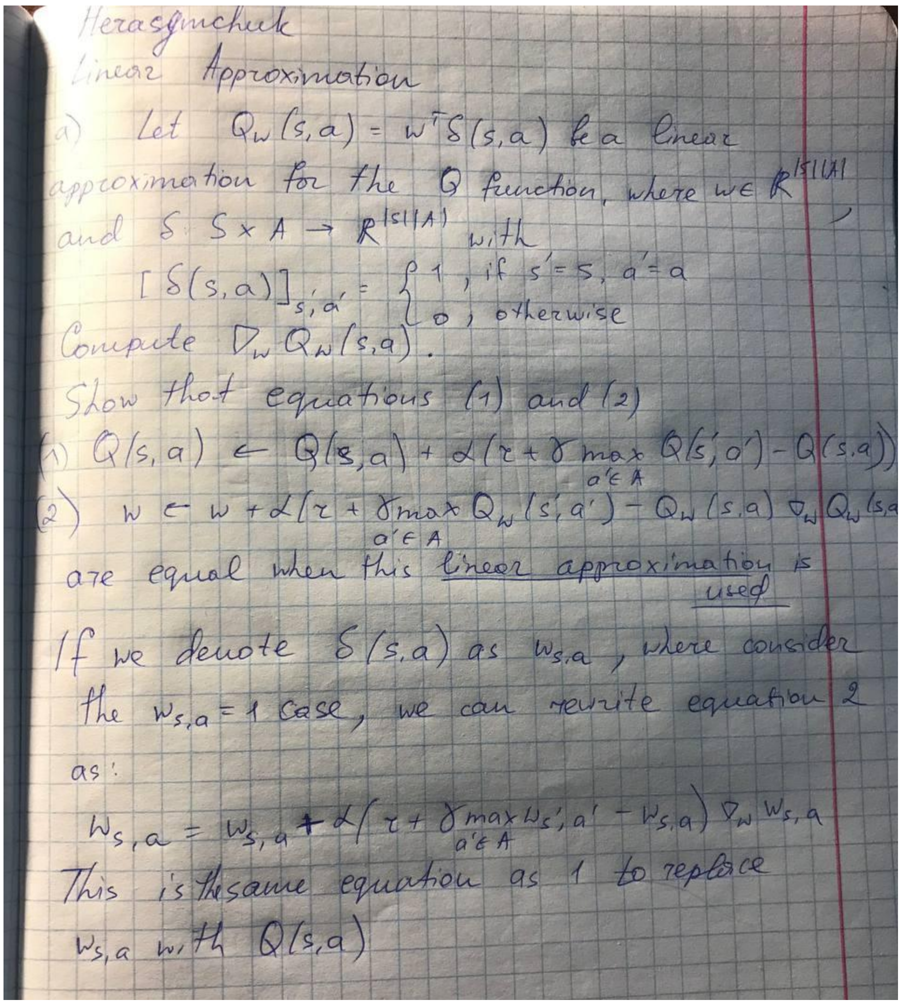
(a) (coding, 3 pts) Implement the get action and update functions in q3 schedule.py. Test your implementation by running python q3 schedule.py.



```
(cs234-torch) C:\Users\UKRAINE4EVER\Downloads\assignment2_coding\assignment2_coding>python q3_schedule.py
Test1: ok
Test2: ok
Test3: ok
```

Tests were successful.

Linear Approximation.



Herasymchuk

Linear Approximation

a) Let $Q_w(s,a) = w^T \delta(s,a)$ be a linear approximation for the Q function, where $w \in \mathbb{R}^{|S||A|}$ and $\delta: S \times A \to \mathbb{R}^{|S||A|}$ with

$$[\delta(s,a)]_{s',a'} = \begin{cases} 1, & \text{if } s'=s, a'=a \\ 0, & \text{otherwise} \end{cases}$$

Compute $\nabla_w Q_w(s,a)$.

Show that equations (1) and (2)

$$(1) \quad Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a))$$

$$(2) \quad w \leftarrow w + \alpha(r + \gamma \max_{a' \in A} Q_w(s',a') - Q_w(s,a)\, \nabla_w Q_w(s,a$$

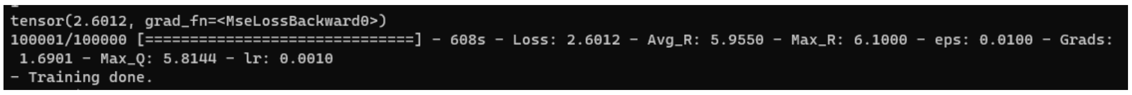
are equal when this <u>linear approximation is used</u>

If we denote $\delta(s,a)$ as $w_{s,a}$, where consider the $w_{s,a} = 1$ case, we can rewrite equation 2 as:

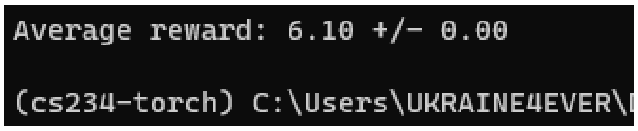
$$w_{s,a} = w_{s,a} + \alpha(r + \gamma \max_{a' \in A} w_{s',a'} - w_{s,a}) \nabla_w w_{s,a}$$

This is the same equation as 1 to replace $w_{s,a}$ with $Q(s,a)$

Then linear approximation was implemented in PyTorch launching `python q4_linear_torch.py`:



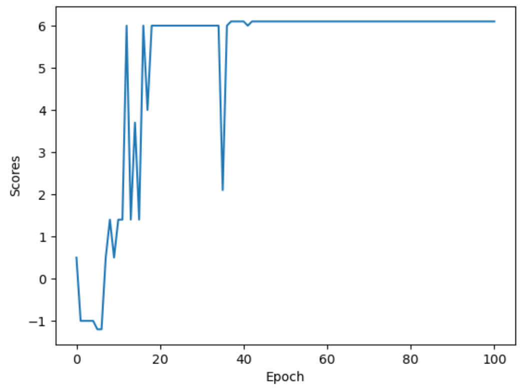
```
tensor(2.6012, grad_fn=<MseLossBackward0>)
100001/100000 [==============================] - 608s - Loss: 2.6012 - Avg_R: 5.9550 - Max_R: 6.1000 - eps: 0.0100 - Grads:
 1.6901 - Max_Q: 5.8144 - lr: 0.0010
- Training done.
```



```
Average reward: 6.10 +/- 0.00

(cs234-torch) C:\Users\UKRAINE4EVER\
```

The plot scores.png from the directory results/q4 linear

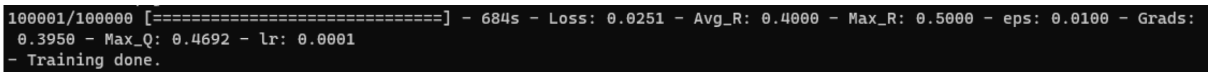


Implementing DeepMind's DQN.

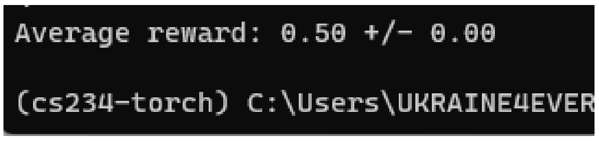
A smaller version of the deep Q-network was implemented, and the implementation was locally tested on CPU on the test environment by running python q5 nature torch.py

The following architecture was used:

- One convolution layer with 16 output channels, a kernel size of 3, stride 1, and no padding.
- A ReLU activation.
- A dense layer with 128 hidden units.
- Another ReLU activation.
- The final output layer.

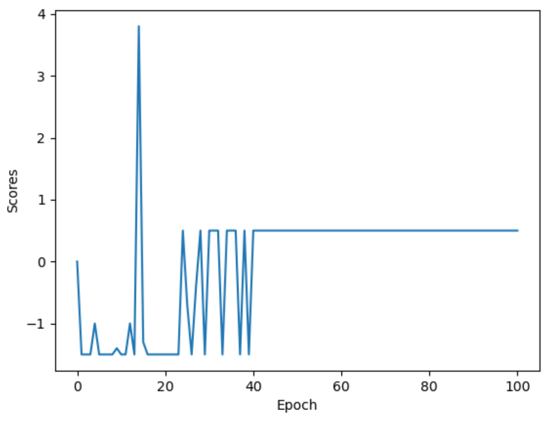


```
100001/100000 [==============================] - 684s - Loss: 0.0251 - Avg_R: 0.4000 - Max_R: 0.5000 - eps: 0.0100 - Grads:
 0.3950 - Max_Q: 0.4692 - lr: 0.0001
- Training done.
```

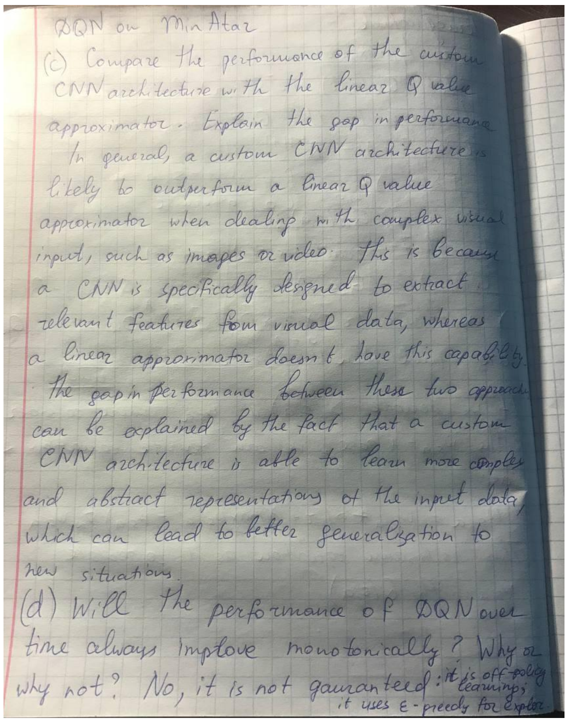


```
Average reward: 0.50 +/- 0.00

(cs234-torch) C:\Users\UKRAINE4EVER
```



DQN on MinAtar.



DQN on MinAtar

(c) Compare the performance of the custom CNN architecture with the linear Q value approximator. Explain the gap in performance

In general, a custom CNN architecture is likely to outperform a linear Q value approximator when dealing with complex visual input, such as images or video. This is because a CNN is specifically designed to extract relevant features from visual data, whereas a linear approximator doesn't have this capability.

The gap in performance between these two approaches can be explained by the fact that a custom CNN architecture is able to learn more complex and abstract representations of the input data, which can lead to better generalization to new situations.

(d) Will the performance of DQN over time always improve monotonically? Why or why not? No, it is not gauranteed: it is off-policy learning; it uses ϵ-greedy for explor.