Foo Lin Geng 1002504

***Coding Homework (Week 8)***

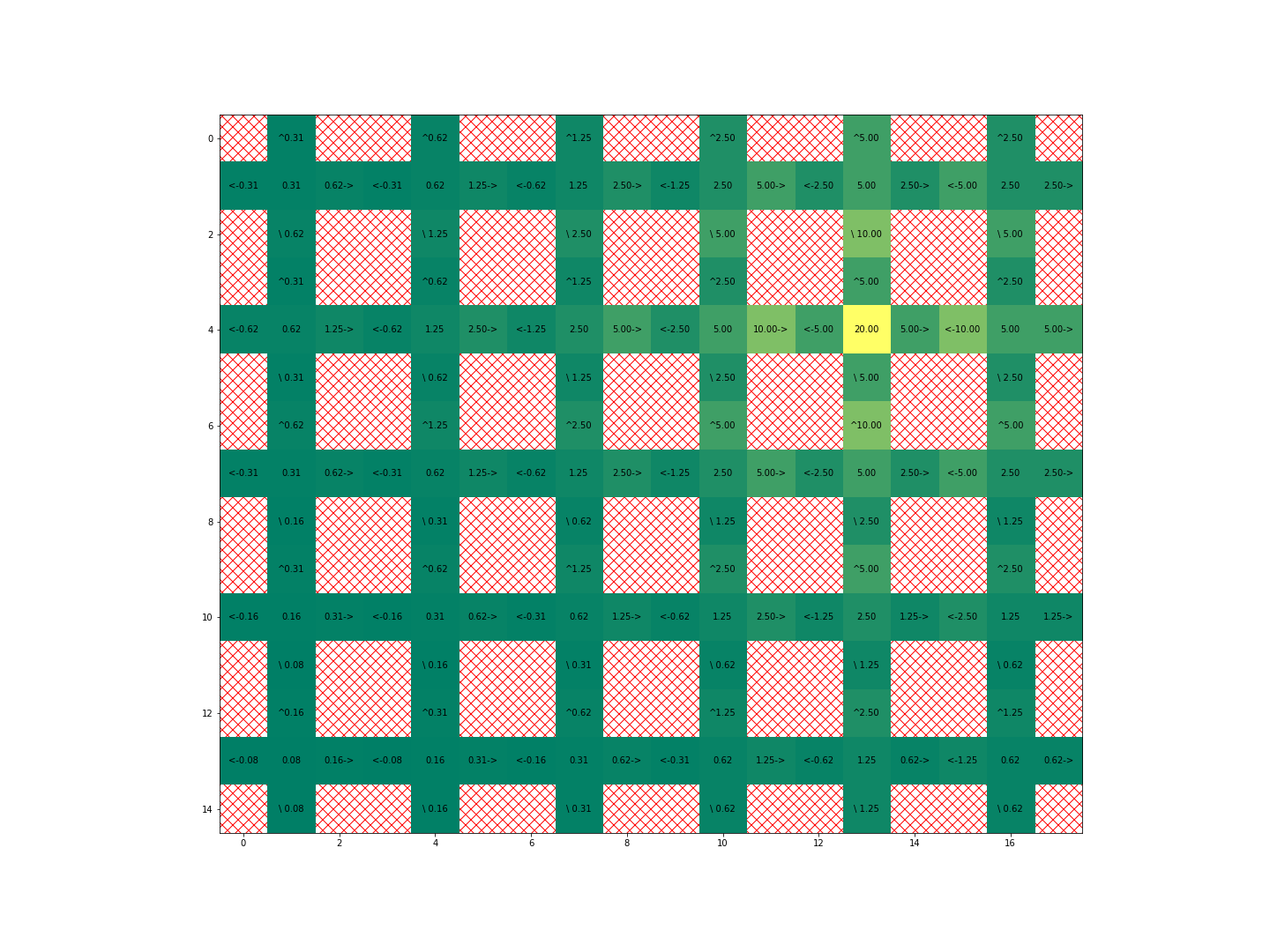
**Task 1**

The finished code is in **simpleq2\_studentversion.py**.

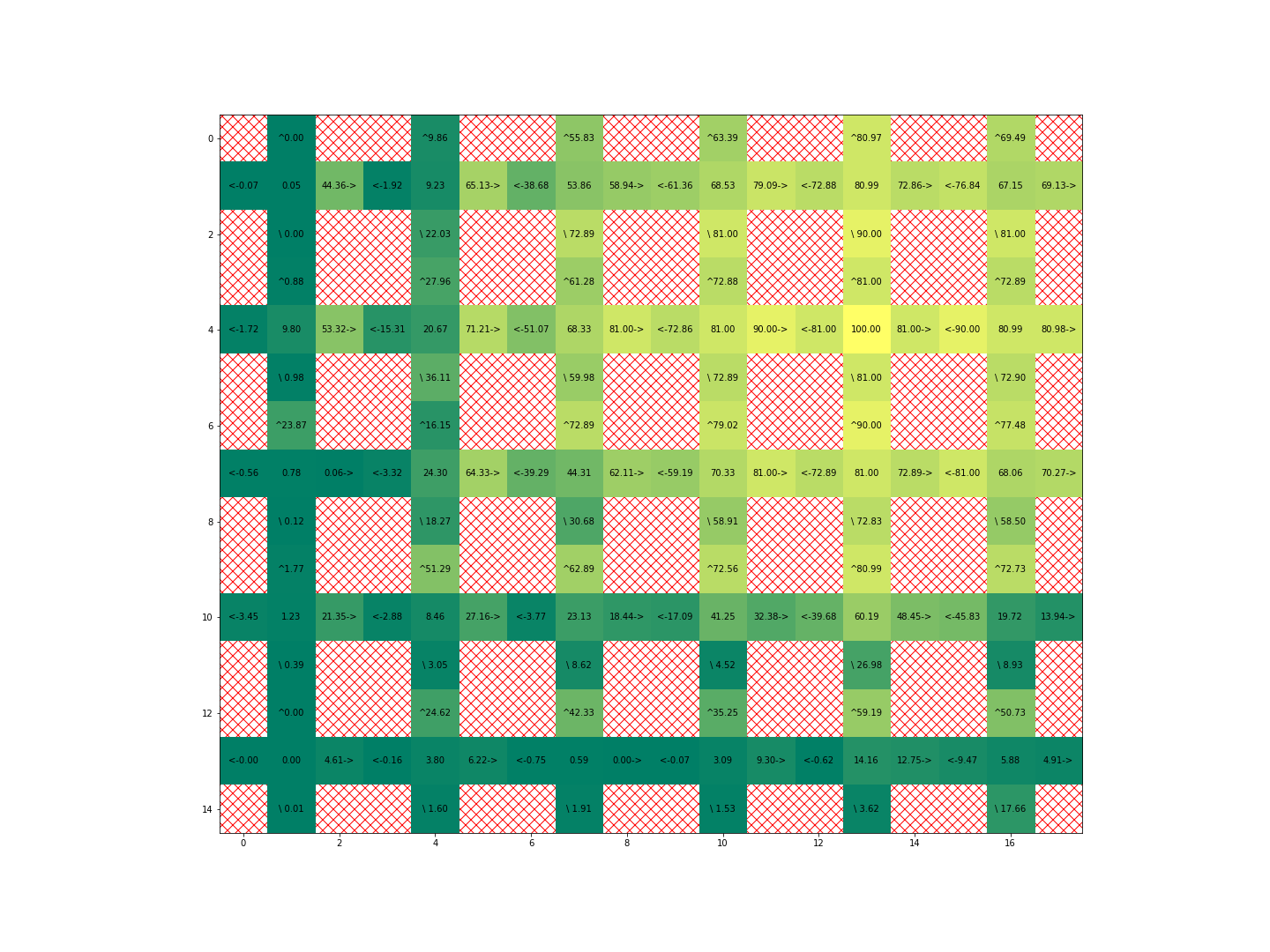
Both functions were completed:

1. **runmdp()** which gets Q values using Q-iteration/value iteration.
2. **trainsth()** which gets Q values using Q learning TD(0) algorithm

**runmdp()** uses a gamma value of 0.5. The resulting **Q(s,a)** plot is:



**trainsth()** uses a gamma value of 0.9. The resulting **Q(s,a)** plot is:



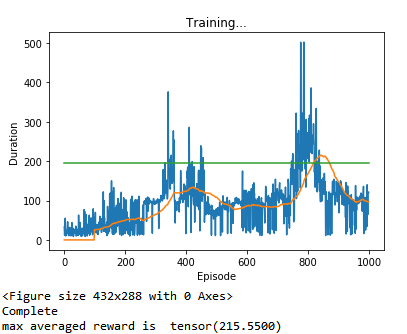
**Task 2**

The finished code is in **week8codinghmk\_part2.py.**

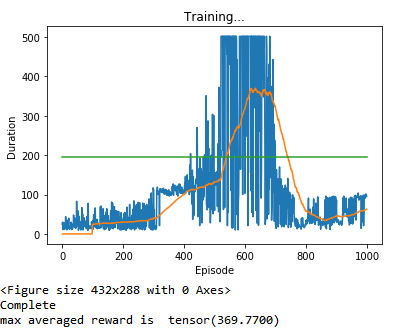
Following the instructions, I capped episode length at 500, ran using a linear epsilon decay, and single DQN, with the settings:

BATCH\_SIZE = 128  
GAMMA = 0.99  
EPS\_START = 0.9  
EPS\_END = 0.02  
EPS\_END\_STEPS = 12000  
TARGET\_UPDATE = 20  
REPLAY\_MEM\_SIZE = 50000  
LEARNING\_RATE = 0.005  
WEIGHT\_DECAY = 0.000001

The rewards over 1000 iterations are shown below:

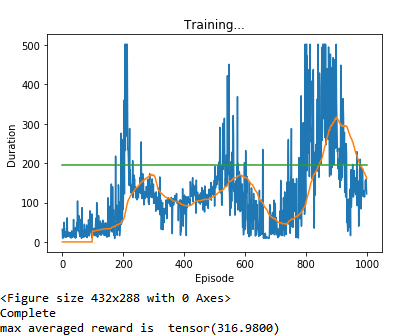


The max 100-step averaged reward is 215.55.

**Case 1a) LR change to 0.01 (two times larger)**

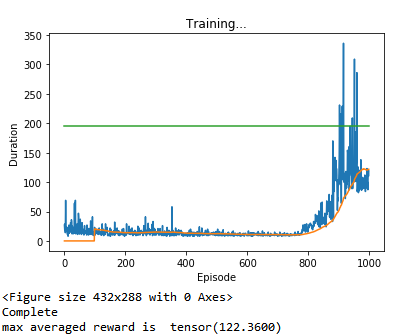
The max 100-step averaged reward is 369.77. There is a marked improvement.

**Case 1b) LR change to 0.05 (10 times larger)**



The max 100-step averaged reward is 316.98. The improvement is good but less than Case 1a.

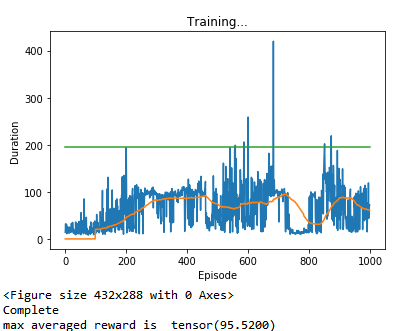
**Case 1c) LR change to 0.0001 (50 times smaller)**



The max 100-step averaged reward is 122.36. The learning rate is too low!

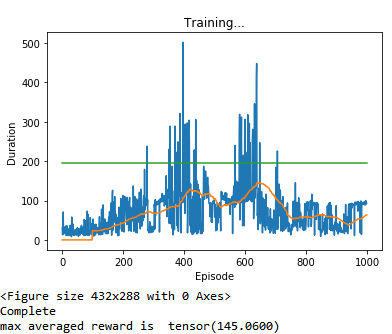
**Overall**, I would suggest that the learning rate should be set around 0.01. The **algorithm seems to be rather sensitive to values of learning rate.**

**Case 2a) REPLAY\_MEM\_SIZE change to 20000 (smaller than 50000)**



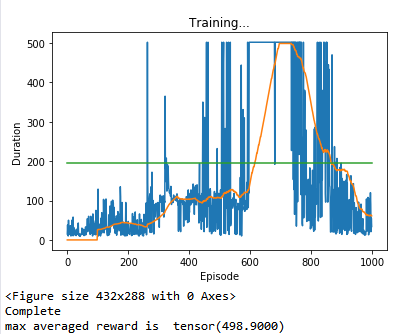
The max 100-step averaged reward is 95.52. It appears that reducing the size of the replay memory had a huge decrease on performance.

**Case 2b) REPLAY\_MEM\_SIZE change to 80000 (larger than 50000)**



The max 100-step averaged reward is 145.06. It appears that increasing the size of the replay memory had a huge decrease on performance.

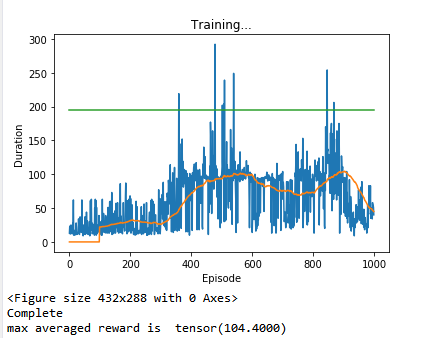
**Case 2c) REPLAY\_MEM\_SIZE change to 60000 (slightly larger than 50000)**



The max 100-step averaged reward is 498.90. This is a really good performance, although it could also be due to luck.

**Overall**, the **algorithm seems very, very sensitive to the size of the replay memory**. A small tweak makes the difference between a very good performance, and a mediocre one.

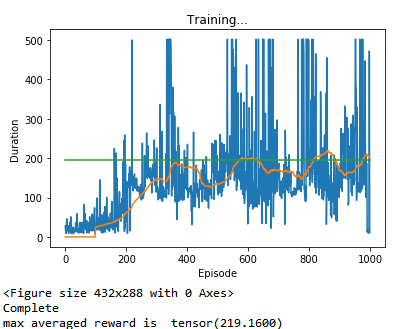
**Case 3) Neural network layer output from 24/24/2 to 30/30/2 (making it wider)**



The max 100-step averaged reward is 104.4. Making the neural network wider has decreased performance.

This shows that the algorithm is rather sensitive to the neural network parameters as well.

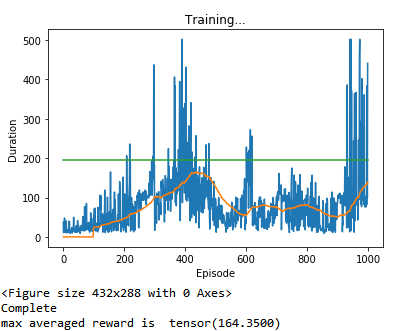
**Case 4) Loss from Huber Loss to MSE Loss**



The max 100-step averaged reward is 219.16. Changing the loss only led to a minimal improvement from 215.55 to 219.16.

This means that the algorithm is not extremely sensitive to the type of loss function.

**Case 5) From Q-learning to Double Q-learning**



The max 100-step averaged reward is 164.35. It appears that under the current parameters, Q-learning outperforms double Q-learning.

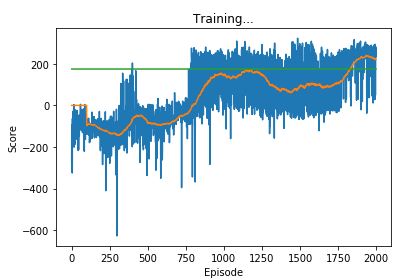
**Task 3**

The finished code is in **week8codinghmk\_part3.py.**

I ran using a linear epsilon decay, and single DQN, with the settings:

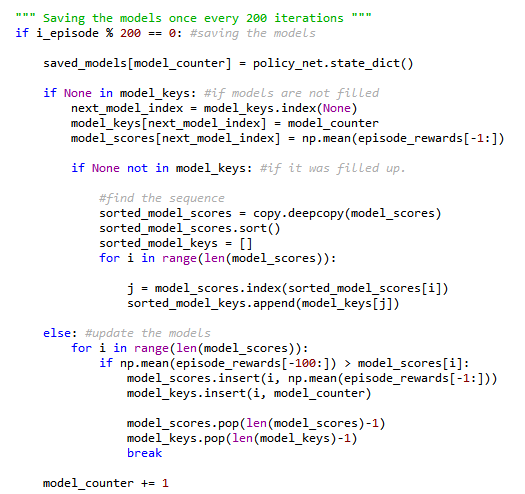
BATCH\_SIZE = 128  
GAMMA = 0.999  
EPS\_START = 0.9  
EPS\_END = 0.02  
EPS\_END\_STEPS = 20000  
TARGET\_UPDATE = 15  
REPLAY\_MEM\_SIZE = 30000  
LEARNING\_RATE = 0.001

The rewards over 2000 iterations are shown below:



We observe that the agent has successfully learnt how to play Lunar Lander in about 1800 episodes.

**The code for saving models:**



**The code for prediction using the saved models.**

