APPLIED ROBOTICS
CW-REPORT

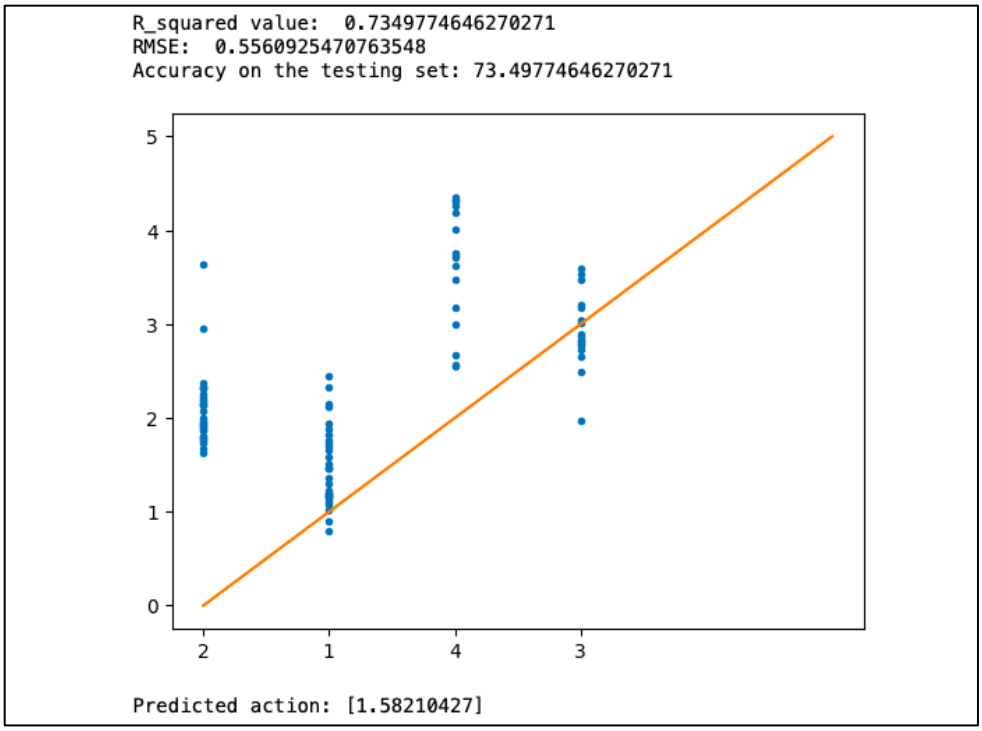
6ELEN018W.1

Anatol Satler | W1810699
[E-Mail-Adresse]

1. Neural Network Control of the Robot

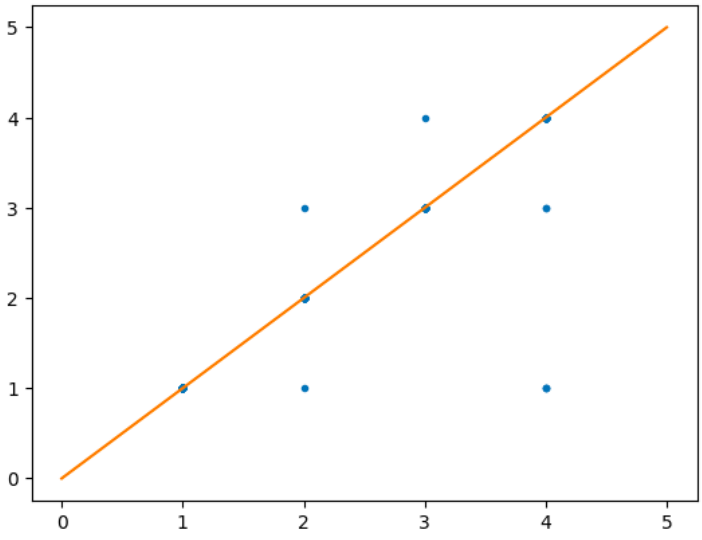
I split the data into an 80/20 ratio, 80% present the training data while the other 20% represents the testing data. After that I Initialized the **MLPRegression** with the default function but with 2 layers (50, 30) with a max iteration of 10,000.
Trained the data and returned the accuracy of the training.
The 2 layers (50, 30) returned the best results in my case with an accuracy of 75%, while just one layer of 50 nodes would be approx. 57% of accuracy.

Then I repeated the steps from the first cell, let the model predict data from the X_test values, and plotted them. In the graphic, they are being visualized against each other, where each point represents the predicted value against the actual value. Additionally, a line is plotted using **plt.plot(x, y)** where **x** and **y** form a line with a slope of 1, representing the ideal scenario where predicted values perfectly match the actual values.
I calculated **RMSE** and **R_squared** to make the model comparable and determine its error rate.
R_squared value is 73%. Generally, a higher r-squared indicates more variability is explained by the model. However, it is not always the case that a high r-squared is good for the regression model.
RMSE value is 0.55. In regression tasks, lower RMSE values indicate better model performance, as a smaller RMSE signifies that the model's predictions are closer to the actual values. Which can be interpreted in my case as acceptable but not good
The accuracy of the model changed by 2%.
The final prediction using **[4, 2, 2, 4]** is printed as Predicted action: **[predicted_value]**, representing the model's prediction for the given input.



Additionally, to make a comparison I exchanged **MLPRegression** with the **MLPClassifier** model. The solutions had huge differences. Starting with the accuracy of the model with 91.39%. RMSE equals 0.58, but it is normally not used for classifier models. R_squared with 91%.

```
R_squared value:  0.9139784946236559
RMSE:  0.5865884600854132
```




```
Accuracy on the testing set: 91.39784946236558
```

Next cell returns, a rounded value of the amount of actions. I inserted a value into the mlp and performed a predict method which returned the value 1.

In the last Cell, I simulated the robot going through the maze with help of the **MLPRegression** prediction method. Which returns a rounded number that then moves the robot in a direction towards the goal.

2. Reinforcement Learning Control of the Robot

1st and 2nd cells just calculate and re-evaluate the rewards for the robot moving in the space. From -100 to 100 as a reward, if the box is next to an obstacle the reward towards this side, depending on the direction, will be -100. The other way around if a box is next to the goal it is going to be 100 in the direction.

```
s1: ->  {'N': 0, 'E': -100.0, 'S': -100.0, 'W': 0}
s2: ->  {'N': 0, 'E': -100.0, 'S': 0, 'W': -100.0}
s3: ->  {'N': 0, 'E': -10.0, 'S': -10.0, 'W': -100.0}
s4: ->  {'N': 0, 'E': -100.0, 'S': 100.0, 'W': -100.0}
s5: ->  {'N': 0, 'E': 0, 'S': -10.0, 'W': -10.0}
s6: ->  {'N': -100.0, 'E': 0, 'S': -100.0, 'W': 0}
s7: ->  {'N': 0, 'E': 0, 'S': 0, 'W': 0}
s8: ->  {'N': -100.0, 'E': 100.0, 'S': -109.0, 'W': 0}
s9: ->  {'N': 0, 'E': 0, 'S': 0, 'W': 0}
s10: ->  {'N': -100.0, 'E': 0, 'S': -100.0, 'W': 100.0}
s11: ->  {'N': -100.0, 'E': -100.0, 'S': -100.0, 'W': 0}
s12: ->  {'N': 0, 'E': 0, 'S': -190.0, 'W': -100.0}
s13: ->  {'N': -10.0, 'E': -10.0, 'S': -100.0, 'W': -100.0}
s14: ->  {'N': 100.0, 'E': -100.0, 'S': 0, 'W': 0}
s15: ->  {'N': -10.0, 'E': 0, 'S': -100.0, 'W': -10.0}
s16: ->  {'N': -100.0, 'E': -190.0, 'S': -100.0, 'W': 0}
s17: ->  {'N': -100.0, 'E': -100.0, 'S': -100.0, 'W': -100.0}
s18: ->  {'N': 0, 'E': 0, 'S': -100.0, 'W': -190.0}
s19: ->  {'N': 0, 'E': 0, 'S': 0, 'W': 0}
s20: ->  {'N': -100.0, 'E': 0, 'S': -100.0, 'W': 0}
s21: ->  {'N': -100.0, 'E': -100.0, 'S': 0, 'W': 0}
s22: ->  {'N': -190.0, 'E': -100.0, 'S': 0, 'W': -100.0}
s23: ->  {'N': -100.0, 'E': -100.0, 'S': 0, 'W': -100.0}
s24: ->  {'N': 0, 'E': -100.0, 'S': 0, 'W': -100.0}
s25: ->  {'N': -100.0, 'E': 0, 'S': 0, 'W': -100.0}
```

```
Value of Q[ 0 0 ] =  {'N': 0, 'E': 0.0, 'S': 0.0, 'W': 0}
Value of Q[ 0 1 ] =  {'N': 0, 'E': 0.0, 'S': -19.0, 'W': 0.0}
Value of Q[ 0 2 ] =  {'N': 0, 'E': 90.0, 'S': 0.0, 'W': 0.0}
Value of Q[ 0 3 ] =  {'N': 0, 'E': 0.0, 'S': 100.0, 'W': 0.0}
Value of Q[ 0 4 ] =  {'N': 0, 'E': 0, 'S': 0.0, 'W': 90.0}
Value of Q[ 1 0 ] =  {'N': 0.0, 'E': -100.0, 'S': 34.86784401000001, 'W': 0}
Value of Q[ 1 1 ] =  {'N': 0.0, 'E': 90.0, 'S': 38.742048900000015, 'W': 31.381059609000012}
Value of Q[ 1 2 ] =  {'N': 0.0, 'E': 100.0, 'S': -19.0, 'W': -19.0}
Value of Q[ 1 3 ] =  {'N': 0, 'E': 0, 'S': 0, 'W': 0}
Value of Q[ 1 4 ] =  {'N': 0.0, 'E': 0, 'S': 0, 'W': 0}
Value of Q[ 2 0 ] =  {'N': 31.381059609000012, 'E': 38.742048900000015, 'S': 38.742048900000015, 'W': 0}
Value of Q[ 2 1 ] =  {'N': -19.0, 'E': -19.0, 'S': 43.04672100000001, 'W': 34.86784401000001}
Value of Q[ 2 2 ] =  {'N': 90.0, 'E': 90.0, 'S': 47.829690000000014, 'W': 38.742048900000015}
Value of Q[ 2 3 ] =  {'N': 100.0, 'E': 81.0, 'S': -19.0, 'W': -19.0}
Value of Q[ 2 4 ] =  {'N': 0.0, 'E': 0, 'S': 72.9, 'W': 90.0}
Value of Q[ 3 0 ] =  {'N': 34.86784401000001, 'E': 43.04672100000001, 'S': 43.04672100000001, 'W': 0}
Value of Q[ 3 1 ] =  {'N': 38.742048900000015, 'E': 47.829690000000014, 'S': 47.829690000000014, 'W': 38.7420489000
00015}
Value of Q[ 3 2 ] =  {'N': -19.0, 'E': -19.0, 'S': 53.144100000000016, 'W': 43.04672100000001}
Value of Q[ 3 3 ] =  {'N': 90.0, 'E': 72.9, 'S': 59.049000000000014, 'W': 47.829690000000014}
Value of Q[ 3 4 ] =  {'N': 81.0, 'E': 0, 'S': 65.61000000000001, 'W': -19.0}
Value of Q[ 4 0 ] =  {'N': 38.742048900000015, 'E': 47.829690000000014, 'S': 0, 'W': 0}
Value of Q[ 4 1 ] =  {'N': 43.04672100000001, 'E': 53.144100000000016, 'S': 0, 'W': 43.04672100000001}
Value of Q[ 4 2 ] =  {'N': 47.829690000000014, 'E': 59.049000000000014, 'S': 0, 'W': 47.829690000000014}
Value of Q[ 4 3 ] =  {'N': -19.0, 'E': 65.61000000000001, 'S': 0, 'W': 53.144100000000016}
Value of Q[ 4 4 ] =  {'N': 72.9, 'E': 0, 'S': 0, 'W': 59.049000000000014}
```

The 3rd cell simulates going through the room/maze with 10 iterations. One iteration is when the robot from its random position finds a way to the goal and this process repeats 10 times.

To evaluate how well the robot performed I calculated the average steps per episode. It varies depending on the random numbers. Through the random numbers, a livelock or an error can occur.

The first picture will show the model has an average of 33.8 while on another try, it is 17.8.

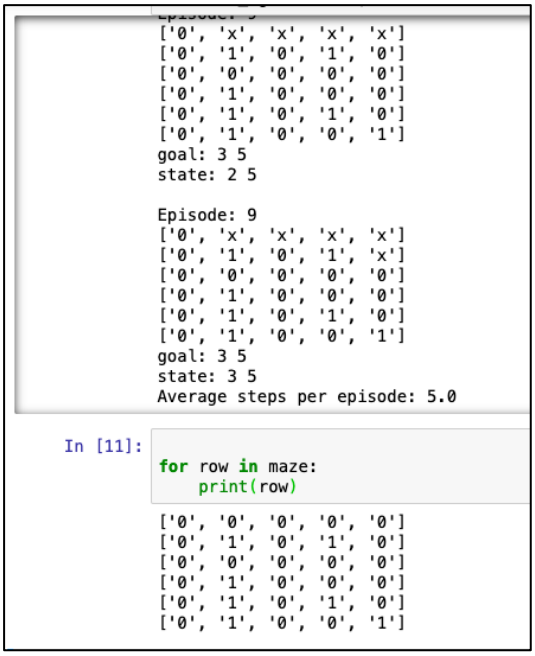
This makes it hard to compare the first simulation in part 1 with this algorithm. The average is hard to compare with the RMSE and R-squared. So, I can't tell which is performing better.

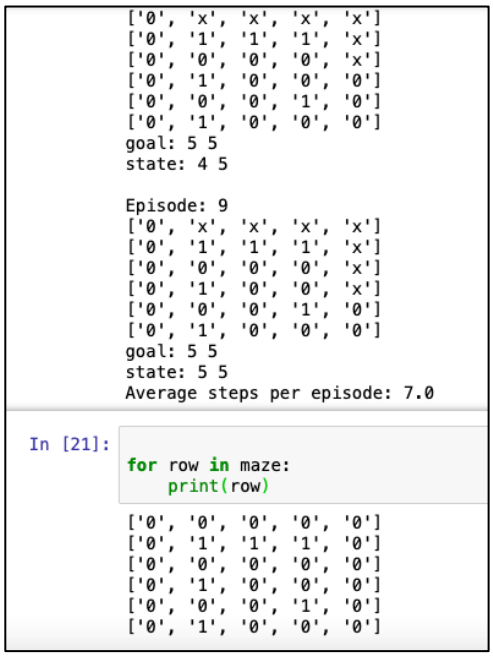
```
(4, 2) ( E ) -> (4, 3)
(4, 3) ( E ) -> (4, 4)
(4, 4) ( N ) -> (3, 4)
(3, 4) ( N ) -> (2, 4)
(2, 4) ( W ) -> (2, 3)
(2, 3) ( N ) -> (1, 3)
[1, 0] ( S ) -> (2, 0)
(2, 0) ( E ) -> (2, 1)
(2, 1) ( S ) -> (3, 1)
(3, 1) ( E ) -> (3, 2)
(3, 2) ( S ) -> (4, 2)
(4, 2) ( E ) -> (4, 3)
(4, 3) ( E ) -> (4, 4)
(4, 4) ( N ) -> (3, 4)
(3, 4) ( N ) -> (2, 4)
(2, 4) ( W ) -> (2, 3)
(2, 3) ( N ) -> (1, 3)
[1, 2] ( E ) -> (1, 3)
[4, 0] ( E ) -> (4, 1)
(4, 1) ( E ) -> (4, 2)
(4, 2) ( E ) -> (4, 3)
(4, 3) ( E ) -> (4, 4)
(4, 4) ( N ) -> (3, 4)
(3, 4) ( N ) -> (2, 4)
(2, 4) ( W ) -> (2, 3)
(2, 3) ( N ) -> (1, 3)
Average steps per episode: 33.8
```

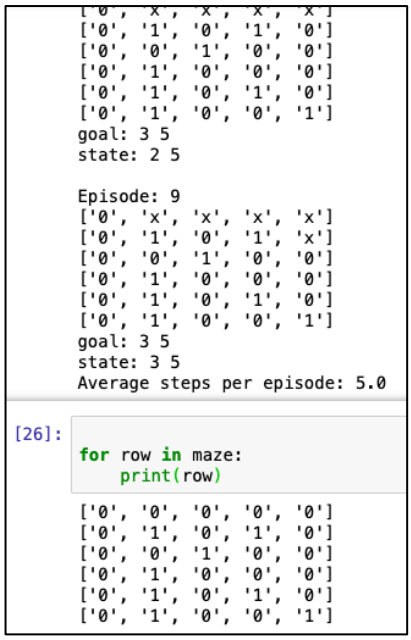
```
[1, 2] ( E ) -> (1, 3)
[2, 2] ( N ) -> (1, 2)
(1, 2) ( E ) -> (1, 3)
[4, 2] ( N ) -> (3, 2)
(3, 2) ( W ) -> (3, 1)
(3, 1) ( N ) -> (2, 1)
(2, 1) ( W ) -> (2, 0)
(2, 0) ( N ) -> (1, 0)
(1, 0) ( N ) -> (0, 0)
(0, 0) ( E ) -> (0, 1)
(0, 1) ( E ) -> (0, 2)
(0, 2) ( E ) -> (0, 3)
(0, 3) ( S ) -> (1, 3)
[1, 2] ( E ) -> (1, 3)
[0, 4] ( W ) -> (0, 3)
(0, 3) ( S ) -> (1, 3)
[2, 3] ( N ) -> (1, 3)
[3, 1] ( N ) -> (2, 1)
(2, 1) ( W ) -> (2, 0)
(2, 0) ( N ) -> (1, 0)
(1, 0) ( N ) -> (0, 0)
(0, 0) ( E ) -> (0, 1)
(0, 1) ( E ) -> (0, 2)
(0, 2) ( E ) -> (0, 3)
(0, 3) ( S ) -> (1, 3)
[0, 1] ( E ) -> (0, 2)
(0, 2) ( E ) -> (0, 3)
(0, 3) ( S ) -> (1, 3)
[0, 3] ( S ) -> (1, 3)
[3, 4] ( N ) -> (2, 4)
(2, 4) ( N ) -> (1, 4)
(1, 4) ( W ) -> (1, 3)
Average steps per episode: 17.8
```

3. Reinforcement Learning Control of the Robot for Different Mazes

In the next 3 cells, I repeated the functions and read a matrix from a txt file. Then I applied the algorithm to simulate the robot moving through the room. I return the average steps per episode which is for the “Matrix.txt” and “Matrix3.txt” file exactly 5. It changed for the 3rd test file “Matrix3.txt.” to an average of 7 steps per episode.







4. Plagarism

I confirm that I understand what plagiarism is and have read and understood the section on Assessment Offences in the Essential Information for Students. The work that I have

submitted is entirely my own. Any work from other authors is duly referenced and acknowledged.
Anatol Satler