GUC
German University in Cairo
Faculty of Engineering and Material Science
Department of Mechatronics Engineering

Reinforcement Learning and Optimal Control (MCTR1024)

Quiz 2: Linear Regression

Due Date: Tuesday, 30-April-2024

This take-home quiz is groups of 2 students.

Student #1	
Name:	Salma Essam Shafik
GUC ID:	49-9180
Tutorial Number:	T-03

Student #2	
Name:	Anthony Rezkalla
GUC ID:	49-5963
Tutorial Number:	T-02

Please note that cheating will not be tolerated and that it is your responsibility to ensure the genuineness of your work.

Problem: Linear Regression:

Machine learning, more specifically the field of predictive modeling, is primarily concerned with minimizing the error of a model or making the most accurate predictions possible, at the expense of explain-ability.

As such, linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables, but has been borrowed by machine learning. Thus, linear regression is both a statistical algorithm and a machine learning algorithm.

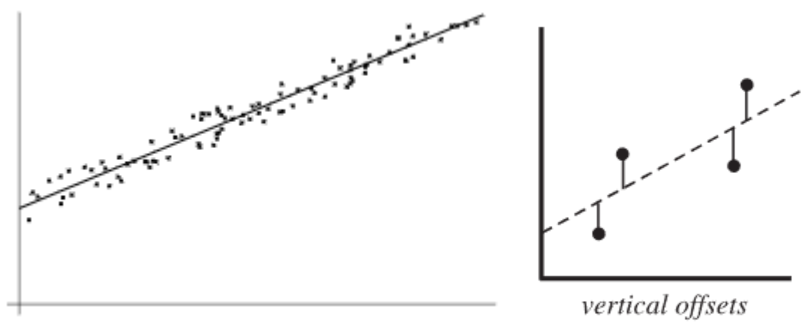


Figure 1

Linear regression attempts to model the relationship between two variables (X, Y) by fitting a linear equation to observed data (as shown in Figure 1). One variable is considered to be an explanatory variable X, and the other is considered to be a dependent variable Y. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Linear model:

The following model is suitable for a multi-variate n-dimensional feature vector $\underline{\boldsymbol{x}}$.

$$\hat{y}(\underline{\boldsymbol{x}}) = \underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + b \; ; \; \underline{\boldsymbol{x}} \in \mathbb{R}^n$$

For the special case of a univariate feature vector x, the linear model is as follows.

$$\hat{y}(x) = w\, x + b$$

Estimating the linear model parameters using gradient descent:

The linear model parameters are the weights w or $\underline{\boldsymbol{w}}$ (representing the slope of the line) and the bias b (representing the y-intercept). These parameters are obtained by minimizing the mean squared error of the vertical offsets (shown in Figure 1) between the predictions $\hat{y}$ and the actual outputs $y_i \in Y$ in the training data.

$$\underline{\boldsymbol{w}}^*, b^* = \underset{\underline{\boldsymbol{w}},b}{\operatorname{argmin}}\, J(\underline{\boldsymbol{w}}, b)$$

$$J(\underline{\boldsymbol{w}}, b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}(x_i)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \left(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}} + b\right)\right)^2$$

Where N is the size of the training data points (X, Y).

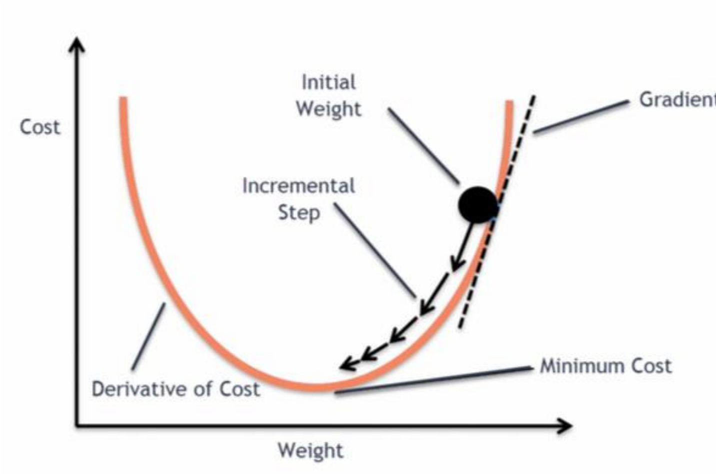


Figure 2

Gradient descent finds the optimal values of the weights and bias $\underline{\boldsymbol{w}}^*, b^*$ by moving tiny steps in the descending (i.e. decreasing) direction of the cost function $J(\underline{\boldsymbol{w}}, b)$ (i.e. mean squared error between predictions and actual outputs). This is illustrated in Figure 2 and is achieved using the following relations.

$$\underline{\boldsymbol{w}} \leftarrow \underline{\boldsymbol{w}} - \alpha\frac{\partial J}{\partial \underline{\boldsymbol{w}}}$$

$$b \leftarrow b - \alpha\frac{\partial J}{\partial b}$$

Where α is the learning rate (determining how tiny the steps towards the global minimum point are) and the gradients of the cost function with respect to the weights and bias are:

$$\frac{\partial J}{\partial \underline{\boldsymbol{w}}} = \frac{1}{N}\sum_{i=1}^{N}\left(-2\underline{\boldsymbol{x}}_i\left(y_i - \left(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b\right)\right)\right)$$

$$\frac{\partial J}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}\left(-2\left(y_i - \left(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b\right)\right)\right)$$

Algorithm:

1. Load/Generate the training data (X, Y) of size N.
2. Initialize the weights and bias $\underline{\boldsymbol{w}}, b$ to small random numbers close to zero.
3. Initialize the learning rate α.
4. Repeat for E iterations (also known as epochs):
    a. For each training data point $(\underline{\boldsymbol{x}}_i, y_i)$ in (X, Y):
        i. Calculate the prediction: $\hat{y}_i = \underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b$
        ii. Calculate the respective component of the gradients $\frac{\partial J}{\partial \underline{\boldsymbol{w}}}$ and $\frac{\partial J}{\partial b}$.
    b. Compute the gradients $\frac{\partial J}{\partial \underline{\boldsymbol{w}}}$ and $\frac{\partial J}{\partial b}$.
    c. Update the weights and bias: $\underline{\boldsymbol{w}} \leftarrow \underline{\boldsymbol{w}} - \alpha\frac{\partial J}{\partial \underline{\boldsymbol{w}}}$ and $b \leftarrow b - \alpha\frac{\partial J}{\partial b}$.

Analytical Solution using Least Squares Method:

This linear regression problem has a closed-form analytical solution using least squares method. A detailed explanation can be found in the link: https://mathworld.wolfram.com/LeastSquaresFitting.html and http://mathforcollege.com/nm/mws/gen/06reg/mws_gen_reg_spe_multivariate.pdf .

However, the cost function used is the sum of squared errors between predictions $\hat{y}$ and actual outputs $y_i \in Y$ in the training data.

$$\underline{\boldsymbol{w}}^*, b^* = \underset{\underline{\boldsymbol{w}}, b}{\operatorname{argmin}}\, J(\underline{\boldsymbol{w}}, b)$$

$$J(\underline{\boldsymbol{w}}, b) = \sum_{i=1}^{N}\left(y_i - \hat{y}(x_i)\right)^2 = \sum_{i=1}^{N}\left(y_i - \left(\underline{\boldsymbol{w}}^T\underline{\boldsymbol{x}}_i + b\right)\right)^2$$

For 1D (i.e. univariate) feature vector x, the closed form solution is given as follows.

$$\begin{bmatrix} b^* \\ w^* \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix}$$

For a multi-variate n-dimensional feature vector $\underline{\boldsymbol{x}}$, the closed form solution is as follows.

$$\begin{bmatrix} b^* \\ \underline{\boldsymbol{w}}^* \end{bmatrix} = (X^T X)^{-1} X^T Y$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \; and \; X = \begin{bmatrix} 1 & \underline{\boldsymbol{x}}_1^T \\ 1 & \underline{\boldsymbol{x}}_2^T \\ \vdots & \vdots \\ 1 & \underline{\boldsymbol{x}}_N^T \end{bmatrix}$$

Where N is the number of the training data points.

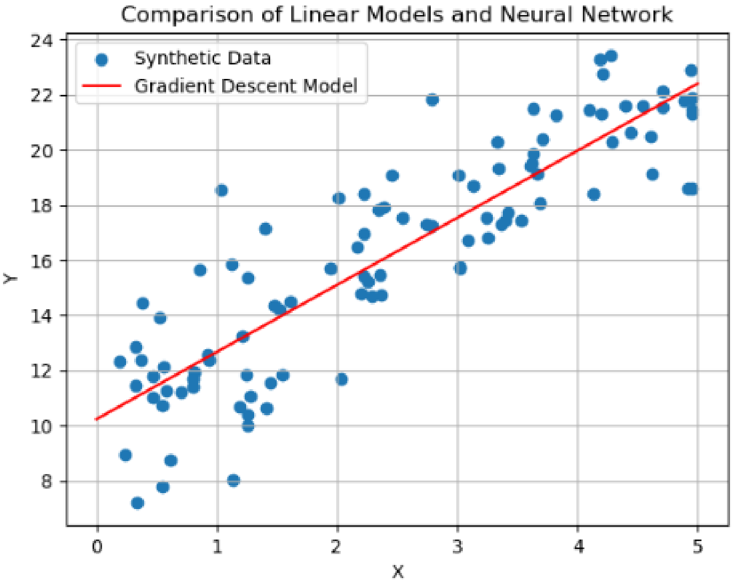
Required:

1. Write 2 python functions; one for training the linear regression model using gradient descent and the other for making predictions using the linear gradient descent model. You may use only **numpy** package.
2. Write another python function to find the analytical solution to the linear regression problem using least squares method.
3. Generate synthetic data and test your linear regression model on it. Compare the 2 obtained linear models; using gradient descent and least squares method.
   Plot the scattered data and the lines of best fit obtained using both methods. You may use **matplotlib** package for plotting purposes.
4. Redo requirement 1 using **scikit-learn** package.
   You may use these links for guidance:
   https://scikit-learn.org/stable/modules/sgd.html
   https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
5. Redo requirement 3. Compare the model obtained using scikit-learn and the model obtained using the least squares method.
6. Redo requirement 1 using a simple neural network of your design. Use **Tensorflow** for this purpose.
   You may use this link:
   https://www.tensorflow.org/tutorials/quickstart/beginner
7. Redo requirement 3. Compare the model obtained using neural networks and the model obtained using the least squares method.

Submission Guidelines:

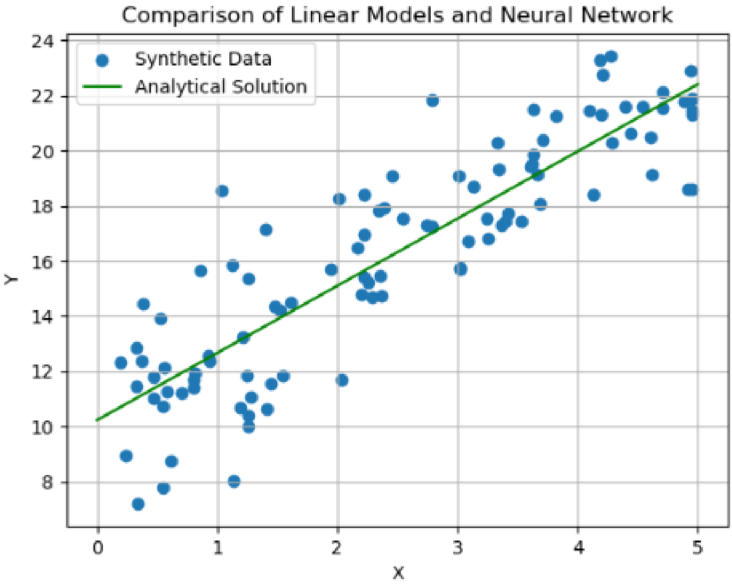
- Submit a zip file with the following:
  - This pdf form including all the required plots.
  - A “pynb” Python notebook containing all the code.
- Submission will be through the GUC online submission system.
- The submission deadline is on Tuesday, 30-April-2024, at 23:59.
- Please note that cheating will not be tolerated and that it is your responsibility to ensure the genuineness of your work.

```
Final weight: 2.4368808970510196
Final bias: 10.226167798131037
Final weight for the closed form solution: [2.4368809]
Final bias for the closed form solution: [10.2261678]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```



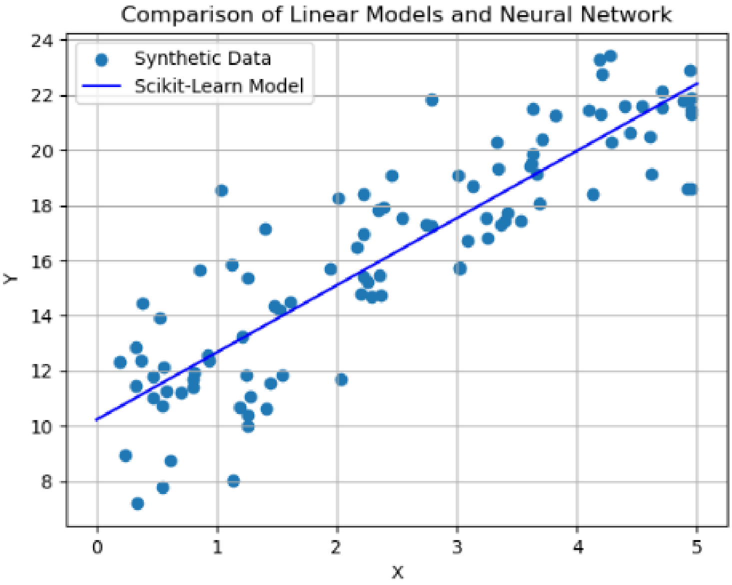

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
The actual value would have been:  [16.25]
However the trained model gave us:  [16.31837004]
```

```
Final weight: 2.4368808970510196
Final bias: 10.226167798131037
Final weight for the closed form solution: [2.4368809]
Final bias for the closed form solution: [10.2261678]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```



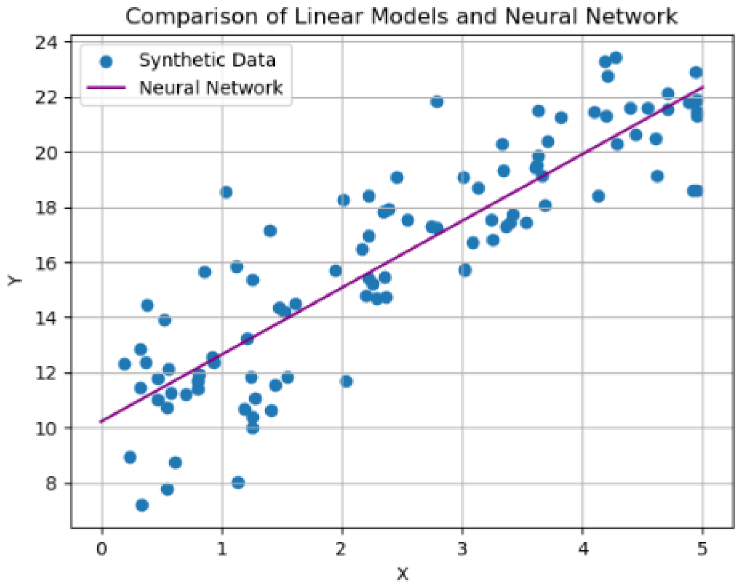

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
The actual value would have been:  [16.25]
However the trained model gave us:  [16.31837004]
The closed form solution is:  [16.31837004]
```

```
Final weight: 2.4368808970510196
Final bias: 10.226167798131037
Final weight for the closed form solution: [2.4368809]
Final bias for the closed form solution: [10.2261678]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```



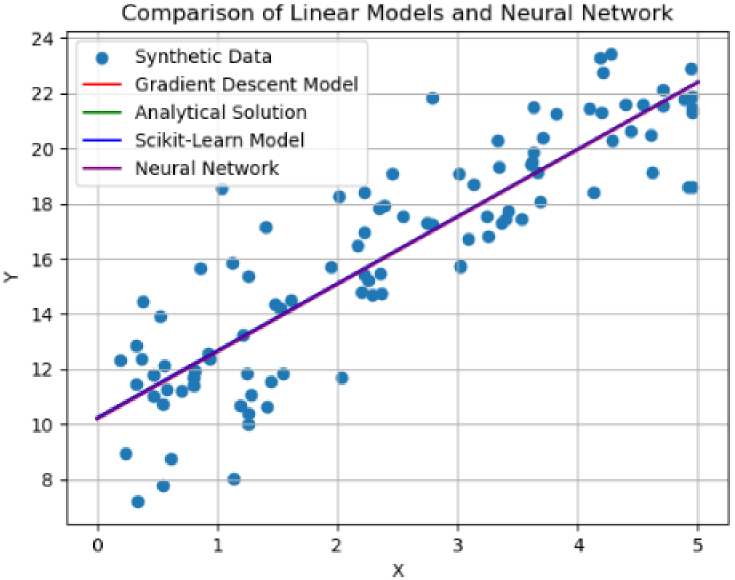

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
The actual value would have been:  [16.25]
However the trained model gave us:  [16.31837004]
The closed form solution is:  [16.31837004]
The scikit-learn model output is:  [16.31837004]
```

```
Final weight: 2.4368808970510196
Final bias: 10.226167798131037
Final weight for the closed form solution: [2.4368809]
Final bias for the closed form solution: [10.2261678]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
The actual value would have been:  [16.25]
However the trained model gave us:  [16.31837004]
The closed form solution is:  [16.31837004]
The scikit-learn model output is:  [16.31837004]
The neural network output is:  [[16.275784]]
```

```
Final weight: 2.4368808970510196
Final bias: 10.226167798131037
Final weight for the closed form solution: [2.4368809]
Final bias for the closed form solution: [10.2261678]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
The actual value would have been:  [16.25]
However the trained model gave us:  [16.31837004]
The closed form solution is:  [16.31837004]
The scikit-learn model output is:  [16.31837004]
The neural network output is:  [[16.276577]]
```