CS 412 HW-2 SUMMARY REPORT

Link to the Colab Notebook:
https://colab.research.google.com/drive/1m8M1hyBN2nEvljMG6m8jEH82eFc9hSI_?usp=sharing

This report is about the application of different Linear Regression techniques with the use of Sklearn and NumPy libraries. Two main sections are present in the report: Linear Regression of linear relationship and polynomial regression of non-linear relationship. Under the linear relationship section, 3 techniques were applied: Sklearn's linear regression model, ordinary least squares (manually), and gradient descent optimization. Under the non-linear relationship section, 2 techniques were applied: Sklearn's linear regression model, and manual polynomial regression. Further details are given below.

Part 1: Linear Relationship

Data used for the part 1 has relationship between x and y variables. 50 samples are generated by the given Python function in notebook with the underlying linear function *2.5x + 0.5 = y* and range [0, 1] for variable x. This data then splitted into two sets, 50% for training and 50% for validation. Aim is to find a representing function close to the underlying function by linear regression.

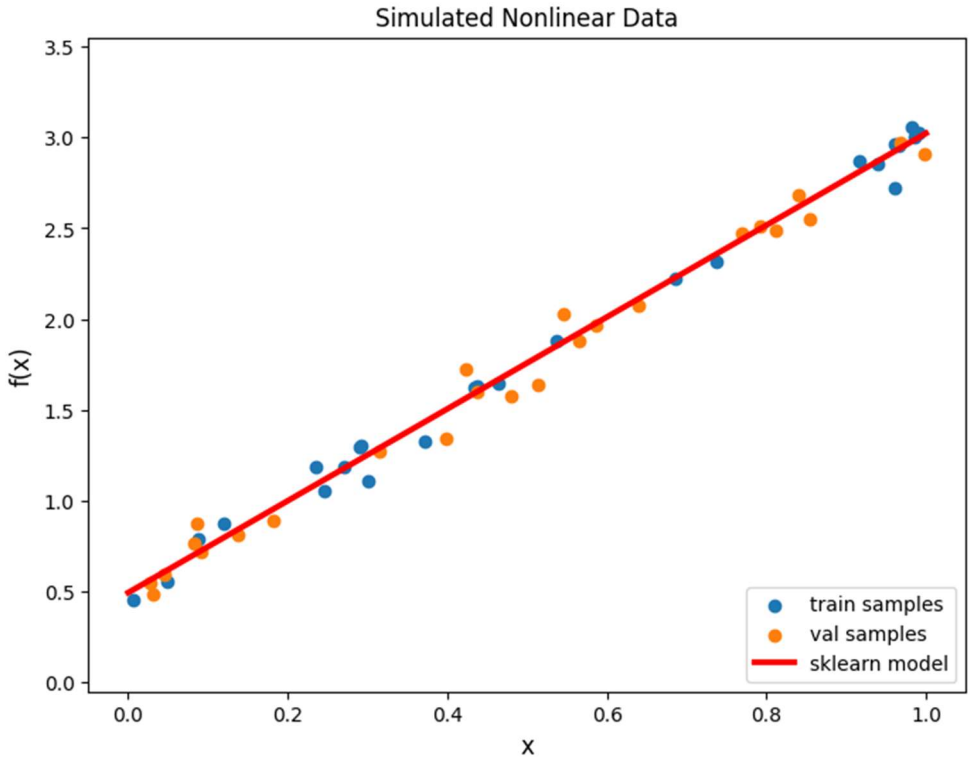
Part 1.a: Sklearn Linear Regression Model

First, the LinearRegression() model of Sklearn library is initialized and trained with training set. Then, trained model made predictions with validation set. Finally, mean squared error is calculated with these predictions and y values of validation set. Results are given in the table below:

Mean Squared Error of Sklearn Model	0.00795462682779033
Slope of Linear Regression	2.52838262
Intercept of Linear Regression	0.49447669

It can be observed that slope and intercept values are pretty close to ones of the original underlying function.

Here is the visual representation of generated data and the linear regression function on a graph, fitting on the data nicely:



Part 1.b: Ordinary Least Squares, Manual Implementation

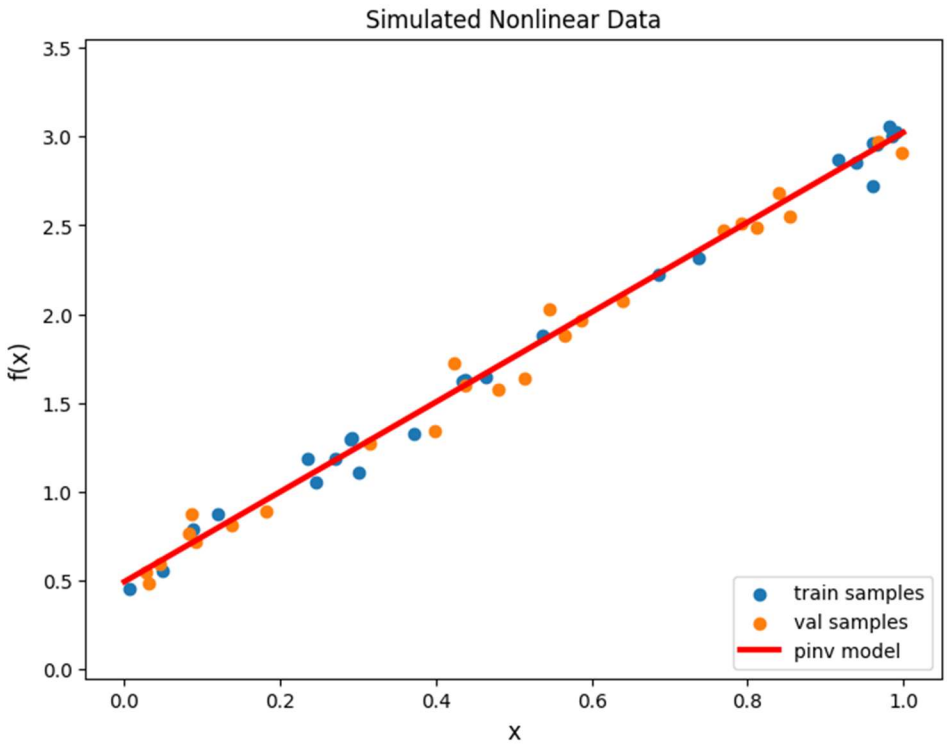
Before taking the pseudo-inverse of training matrix x, both the training-x and validation-x matrix have been extended with a column of ones from the front, which will be constructing the intercept term. Then, the pseudo-inverse of the training-x matrix was taken and multiplied with the training-y matrix. This multiplication resulted in the weight vector, which includes the coefficient and intercept terms.

By multiplying the validation-x matrix with the weight vector, we obtain the predictions of the model for the validation set. Finally, the MSE of these predictions with validation-y data was calculated. Results are given in the table below:

MSE of manual model	0.007954626827790374
Slope of Linear Regression (w[1])	2.52838262
Intercept of Linear Regression (w[0])	0.49447669

Here, again we see that slope and intercept values are close to the ones of the real underlying function. On top of that, MSE turned out to be very close to the one we found with Sklearn model.

Here is the visual representation of generated data and the linear function with the weights on a graph, fitting to the data:



Part 1.c: Gradient Descent Optimization

This technique again needs extended matrices, and previously computed ones are used in this section. Gradient descent also needs some random values for weights to start with the iteration. In this homework, weights started with zeros. The step size/learning rate was given as 0.1.

After this, iteration started. There were 1000 iterations. At each iteration, steps below were applied:

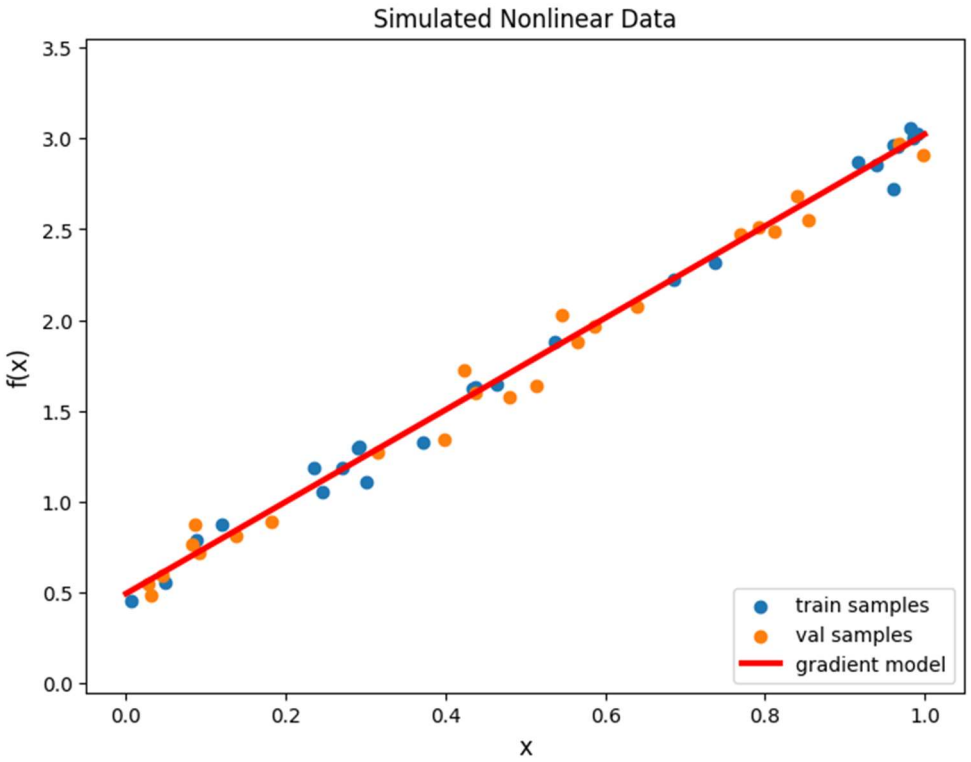
1- Calculated prediction values with current weights and training-x matrix/data
2- Error was calculated by finding the difference between real and predicted values
3- Gradient of cost function was computed with respect to weights, which included the error and the transpose of training-x matrix
4- Weights are updated by subtracting learning rate*gradient
5- To monitor the development of the model, MSE is calculated with validation-y matrix and validation-x matrix*weights

MSEs at every 100 steps are printed. Results are given in the table below:

MSE error at step 1	2.5459
MSE error at step 100	0.0586
MSE error at step 200	0.0161
MSE error at step 300	0.0092
MSE error at step 400	0.0081
MSE error at step 500	0.0079
MSE error at step 600	0.0079
MSE error at step 700	0.0079
MSE error at step 800	0.0080
MSE error at step 900	0.0080
MSE error at step 1000	**0.0080**
Slope of Linear Regression (w[1])	**2.52814518**
Intercept of Linear Regression (w[0])	**0.49461488**

With the results obtained, we can say that gradient descent has given very close results to the two other models. We can see that slope and intercept values are close to the real underlying function values (2.5 and 0.5 respectively). Also, we see that MSE at the last iteration is again very close to the MSE values obtained previously, which are around 0.0079.

Here is the visual representation of generated data and the linear function with the weights on a graph:



Again in the last graph, we can see that model fits the data well.

Part 2: Non-Linear Relationship

In the second part of the homework, for non-linear relationship, given data files are used rather than generating some with code. This data again has 50 samples and was splitted into two sets, namely training and validation. Both sets have 50% of the data. Aim is to find a representing function close to the underlying function by polynomial regression.

Part 2.a: Sklearn Linear Regression and Polynomial Features

Data in in the format of (x, y) tuples. However, since relationship we are talking about is not linear, we may need the powers of x, i.e. polynomial features. Sklearn library provides a class with same name to calculate the powers of x up until a given degree. For example, if degree is 3, then polynomial features transforms [x] to [1 x x^2 x^3]. In this way, training-x and validation-x are transformed. After this transformation, steps for training the LinearRegression() model and predictions are same with part 1.a.

It was asked to check 4 different degrees: 1, 3, 5, and 7. To see all at the same time, process described above repeated in a for loop. Results are in the table below:

MSE of sklearn model with order 1	0.06362852709262727
MSE of sklearn model with order 3	0.012059223742868292
MSE of sklearn model with order 5	0.007475463079281102
MSE of sklearn model with order 7	0.011549227838827407

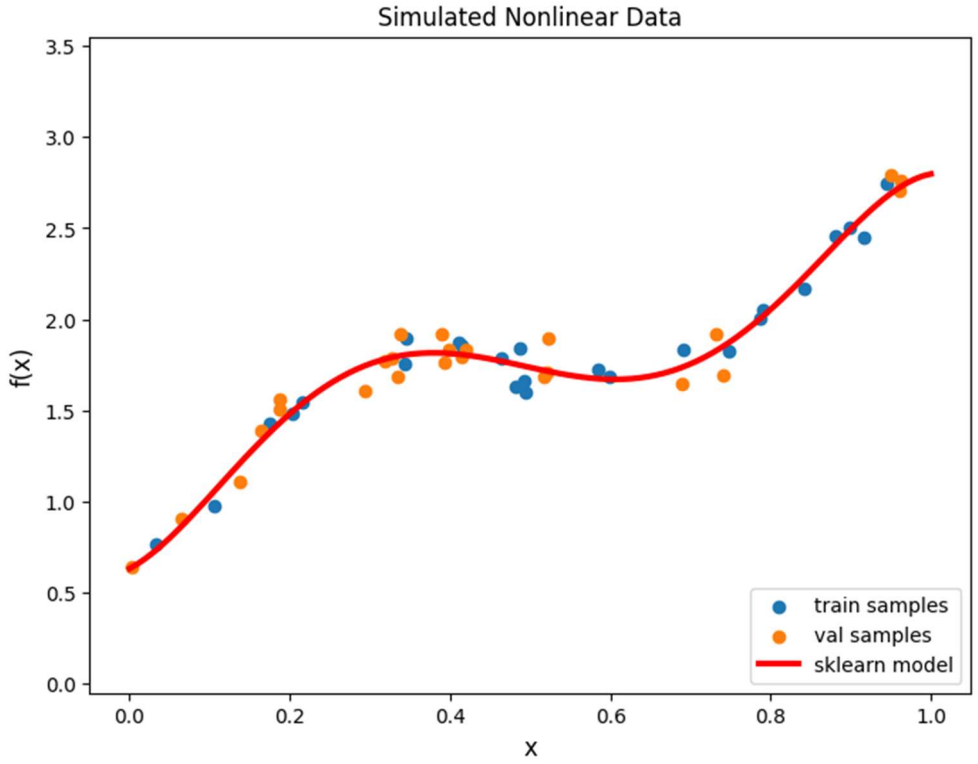
From the table above, we understand that order 5 would be the best fit in this case. We can understand it from the smallest MSE value that calculated with validation set. When order is 1,

comparing to others, we can clearly see that this model is underfitting. When order is 3 and 7, which are before and after order 5, we see that their MSE values are close and higher than of order 5. However, these cases are different: order 3 is underfitting compared to order 5 and order 7 is overfitting. In general, choosing too low order would lead to underfitting for the case of non-linear regression, and too large order is too flexible, thus can easily overfit.

After deciding that order 5 is the best in this case, new model is created with order 5. These are the coefficients and intercept:

1 (intercept)	0.63284179
x	2.30261338
x^2	25.94068538
x^3	-105.99696315
x^4	135.01003494
x^5	-55.09169475

And here is the visualization of data and the polynomial regression model with order 5:



Part 2.b: Manual Polynomial Regression

In this part, order is fixed as 3. To construct the data matrix in the form stated in the notebook, polynomial features is again utilized as it converts data just into specified format. After the transformation of data (namely training-x and validation-x), rest of the steps are similar to part 1.b: pseudo-inverse of data matrix (training-x) is calculated and multiplication of this pseudo-inverse with training-y matrix produces the weights.

After we got the weights, we can predict. Multiplying the validation-x matrix with the weight vector produces the predictions for validation set. Finally, MSE is calculated with validation-y matrix and predictions. Results are given below:

MSE of manual model	**0.012059223742868296**
Coefficient of 1 (intercept) (w [0])	0.39883796
Coefficient of x (w [1])	8.68921502
Coefficient of x^2 (w [2])	-18.07027007
Coefficient of x^3 (w [3])	12.18401084

One can note that MSE calculated in this part is very close to the MSE calculated for degree 3 in the part 2.a.

Here is the visual representation of the data and the polynomial function with the weights on a graph, we can see that it is more loosely fitting compared to order 5: