Deep Learning Spring-2023

Assignment 1 (Part-01)

Linear Regression Using Stochastic Gradient Descent

(With or without chatGPT)

Submission:

Submit all of your codes and results in a single zip file with name FirstName_RollNumber_01.zip

- Submit single zip file containing
  - codes (b) report.pdf (c) Saved Models (d) Readme.txt
- There should be **Report.pdf** detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just analyze the results. Your report should include your comments on the results of all the steps, with images, for example what happened when you changed the learning rate etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- The assignment is only acceptable in .py files. No Jupyter notebooks i.e. ipynb files will be accepted.
- In the root directory, there should be a python file, a report and a folder containing saved models.
- Root directory should be named as **FirstName_RollNumber_01**
- Follow all the naming conventions.
- Each convention/rule violation, there will be a 3% penalty.
- Email instructor and TA's if there are any questions. You cannot look at others code or use others code, however you can discuss with each other. Plagiarism will lead to a straight zero with additional consequences as well.
- 2% (of obtained marks) deduction per day for late submission.
- The **weightage** of this assignment is **2%** of the total marks.

Due Date: 11:59 PM on Wednesday, 8th February 2023

Note: For this assignment (and for others in general) you are not allowed to search online for any kind of implementation. Do not share code or look at the other's code. You should not be in possession of any implementation related to the assignment, other than your own. In case of any confusion please reach out to the TA's and instructor (email them or visit them).

Objectives: In this assignment you will write the code for implementing the linear regression using Numpy arrays. The goals of this assignment are as follows.

- Hand-on experience on Numpy arrays

- Understand how to design and implement an efficient layered architecture for linear regression
- For each layer:
    - Initialize the number of features and learnabe parameters
    - Implement feedforward
    - Keep track of the gradients
    - Understand the mechanism of optimization
- Compare the performance of the model on different hyper-parameters i.e. batch size, learning rate, array initialization
- Able to use ChatGPT

NOTE: You can only use Numpy for code implementations. It's recommended that you use VS Code for debugging.

Report: You have to write a report explaining, what is your implementation logic? How you came up with the optimized weights (how you find learning rate?). Which numpy array initialization performs best on the given dataset? Compare the results of your own code and the ChatGPT. Finally, your comment?

Task 1: Implement linear regression using Stochastic Gradient Descent via Numpy arrays

1. Linear Regression using Stochastic Gradient Descent:

Linear regression is an algorithm that provides a linear relationship between independent variables and a dependent variable to predict the outcome of future events.

1.1. Feed Forward: Mathematically, we can predict the outcome of the future events through the following equation:

$$\hat{y} = X^T w \quad (1)$$

You can calculate a prediction of the outcome $\hat{y}$, using the above formula. X is the input data variable, where θ is the learnable parameter. The dimensions of the above variables should be:

- $X \in R^{d \times N}$: is a matrix of input data. d is the features size and N is the number of samples.
- $w \in R^{d \times 1}$: weights/coefficient of linear function.
- $\hat{y} \in R^{N \times 1}$: predicted value, one value for each sample.

1.2. Compute Loss: A loss will be computed between ground truth targets (y) and predicted targets ($\hat{y}$). T

$$L(y,\hat{y}) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \quad (2)$$

1.3. Gradient: The gradient of the equation (2) can be computed by the following equation:

$$grad = \frac{\partial L}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_i \quad (3)$$

1.4. Update the Parameter:

The learnable parameter *w* can be updated by the following equation:

$$w = w - \text{learning\_rate} * \text{grad} \quad (4)$$

2. Dataset

For this task we will be using the California Housing Dataset. This dataset comes as part of sklearn library. There are 20640 samples, each sample consists of 8 features and target value, i.e. median house price. Our objective is to design a linear for predicting median house price. The target column values lie between the range 0.15 - 5. To load the dataset, you can use the following lines:

```
from sklearn.datasets import fetch_california_housing
dataset = fetch_california_housing()
X = dataset.data
Y = dataset.target[:,np.newaxis]
```

The function `fetch_california_housing()` returns a dictionary-like object, named **dataset** in above code, with the following attributes.

feature_names	MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude
data	ndarray, shape (20640, 8) Each row corresponds to the 8 feature values in order.
target	ndarray, shape (20640,) Each value corresponds to the average house value in units of 100,000.
frame	pandas DataFrame We are not working with pandas in this assignment. So, don't worry about it.
feature_names	list of length 8 Array of ordered feature names used in the dataset.
DESCR	Description of the California housing dataset.

Table-1: output of `fetch_california_housing()`

The above code first imports the `fetch_california_housing` function from `sklearn.datasets library.` Then using the fetch_california_housing() reads the dataset. In the end

i. *X*: numpy array of shape `(20640, 8)` where 20640 is the number of samples and 8 is the number of features.

ii. `Y`: numpy array of shape `(20640, 1)`, target value for each sample.

3. Normalization:

Normalization allows us to convert all the features to the same scale. For each feature we will compute its mean and standard deviation. Then we will subtract the mean from each observation and divide it by standard deviation to get the normalized values.

$$x_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

You calculate a normalized value (Z), using the above formula. The symbols are:

- x_{ij}: is the jth feature of the ith sample.
- μ_j, σ_j: mean and standard deviation of the jth feature

NOTE:

- You must only compute mean and variance of the training data.
- Same mean and standard deviation should be used when inferring over the validation or training data.
- Mean and standard deviation should be saved as part of the model.

4. Stochastic Gradient Descent(SGD):

SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously.

It is also common to sample a small number of data points instead of just one point at each step and that is called "mini-batch" gradient descent. Mini-batch tries to strike a balance between the goodness of gradient descent and speed of SGD.

5. Implementation Details:

5.1. Data Splitting: After reading the dataset (both samples and their ground-truth), we should divide into three subsets, training, validation and testing.

```
train_X, train_Y, val_X, val_Y, test_X, test_Y = data_split(X, Y)
```

Where $X \in R^{d \times N}$, $Y \in R^{N \times 1}$.

train_X, traind_Y : assign first 0.7*N samples to the training set (14,448 in our case).

val_X, val_Y : assign next 0.2*N samples to the validation set (it should be of size 4128).

test_X, test_Y : assign last 0.1*N samples to the testing set (2064 in our case).

Note: these are numpy arrays.

5.1. Data preprocessing:

Data loader has already been explained in section 1. Compute mean and standard deviation, and then pass the input X to the normalize() function.

```
Me = set_mean(X)
```

```
St = set_standardDeviation(X)

[X] = normalize(X, Me, St)
```

The argument of normalize() function is X where $X \in R^{d \times N}$. normalize() function will return the numpy array of shape (d x N). Kindly refer to section 3. related to the in-depth details for the implementation of this function. Don't forget to save Me and St in the model.

5.3. Initialize Network:

`net = linear_regression_network(`N`)`

For example if you pass following parameters to this function:

`net = init_network(8)`

- Use `np.random.randn(), np.zeros(), or np.ones()` to initialize the weights matrices or you can use any other weight initialization method you learned in the class.

It should return you the network architecture with parameters initialized:

`self.theta =` d`x1`

5.4. Training

`[model, loss_epoch_tr, loss_epoch_val] = train(net, train_X, train_Y, val_X, val_Y, batch_size, n_epochs, lr)`

- This function returns a trained model
- `net` network architecture
- `train_X` and `train_Y` will be used for training
- `val_X` and `val_Y` will be used for validation
- `lr is the` learning rate
- `batch_size` tells how many examples to pick for each iteration.
- `n_epochs` how many training epochs to run

Note: *Please make sure your code is modular. You can divide your training process into following functions*

Implement the mini-batch gradient descent algorithm.

- Apply one for loop to iterate over the number of epochs, then used a nested for loop to iterate over the number of the samples that you set in the batch. Then, implement the following functions inside the nested for loop.

5.4.1. y_hat = feed_forward(X_b)

This function will forward through your input examples. Kindly refer to section 5.2. for the dimensions of X_b**.** Kindly refer to Section 1.1 for the details of the implementation of feed forward function. The feed_forward() function will return the predicted output. Note: size of X_b during training will be determined by the batch size.

5.4.2. loss = l2_loss(groundTruth, y_hat)

This function will tell how far are your predictions from the ground-truth. Apply L2 loss as a cost function (KIndly refer to the section 1.2 for the details). Kindly refer to section 5.2. for the

dimensions of train_Y**.** Refer section 5.4.1. for the details of y_hat. Note during training groundTruth = train_Y and during testing it will be test_Y.

5.4.3. grad = compute_gradient(train_X, train_Y, y_hat)

This function will find the gradient. Kindly refer to section 1.3 related to the in-depth details for the implementation of this function. Refer section 5.2 for finding the details of train_X, train_Y. Refer section 5.4.1. for finding the details of y_hat.

5.4.4. self.theta = optimization(lr, grad, theta)

This function will multiply the learning rate with gradient and subtract it from the *theta* to update the *theta*. Kindly refer to section 1.4 related to the in-depth details for the implementation of this function. Refer section 5.4 for finding the details of lr. Refer section 5.4.3 for finding the details of grad.

self.theta = self.theta - lr * gradient

5.5. train_model = pickle.dump(model, open('model.pkl', 'wb'))

After the training, save the trained model via pickle library or some other library. To save the ML model using Pickle all we need to do is pass the model object into the dump() function of Pickle. This will serialize the object and convert it into a "byte stream" that we can save as a file called model.pkl.

5.6. test_model = test_function(train_model, test_X, test_Y)

After the training, do inference on the test dataset. **First load the trained model using pickle.load('model.pkl', 'rb').** Refer section 5.2 for finding the details of test_X, and test_Y. Pass the test_X to feed_forward() function to get the predictions. Then, pass the predictions and test_Y to the loss() function to get the loss of the test dataset.

Merging all functions together:

You will now see how the overall model is structured by putting together all the building blocks (functions that you implemented in the previous parts) together, in the right order. Now implement a main function in which you have to call all the above functions in the correct order to train and test your network.

Note: You are not restricted to implement the assignment in a way that is explained above, you can break down or merge the several functions, but you are required to implement in a modular way.

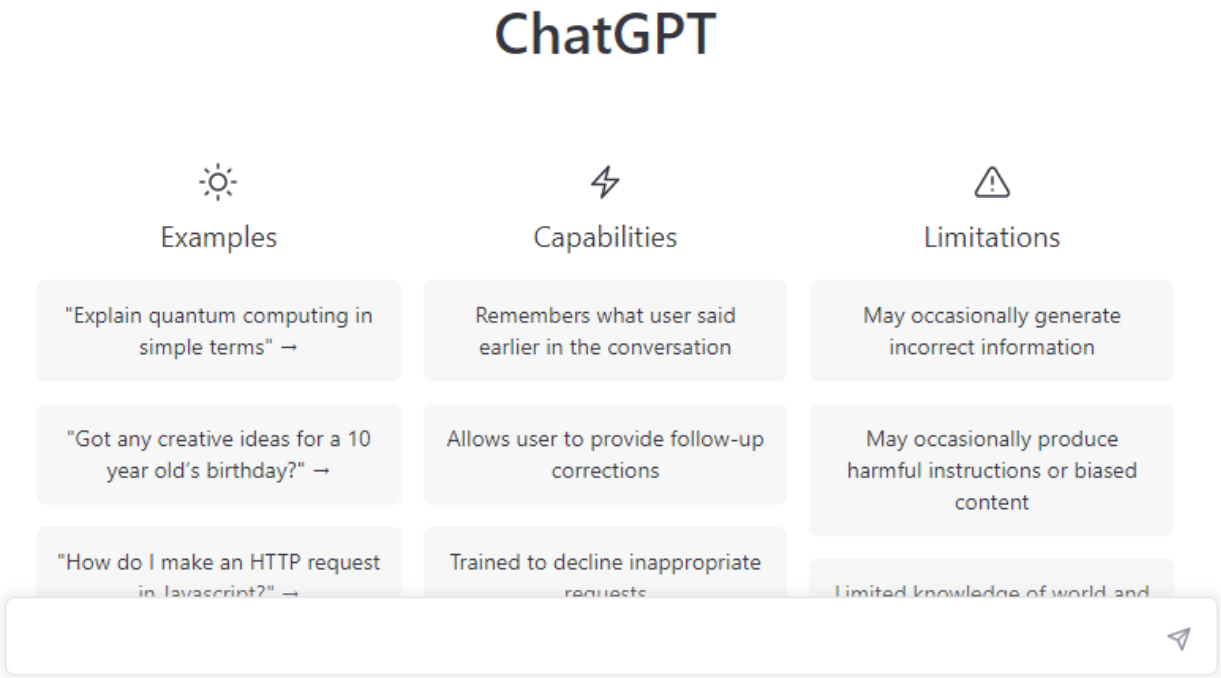
Task 2: Implement linear regression using Stochastic Gradient Descent via ChatGPT

ChatGPT is a sibling model to InstructGPT, which is trained to follow an instruction in a prompt and provide a detailed response.

Instructions:

- Open chatGPT and make an account
- Use the following link to move to the prompt window:

https://chat.openai.com/chat



- Write a prompt command to write a code for line regression using stochastic gradient descent

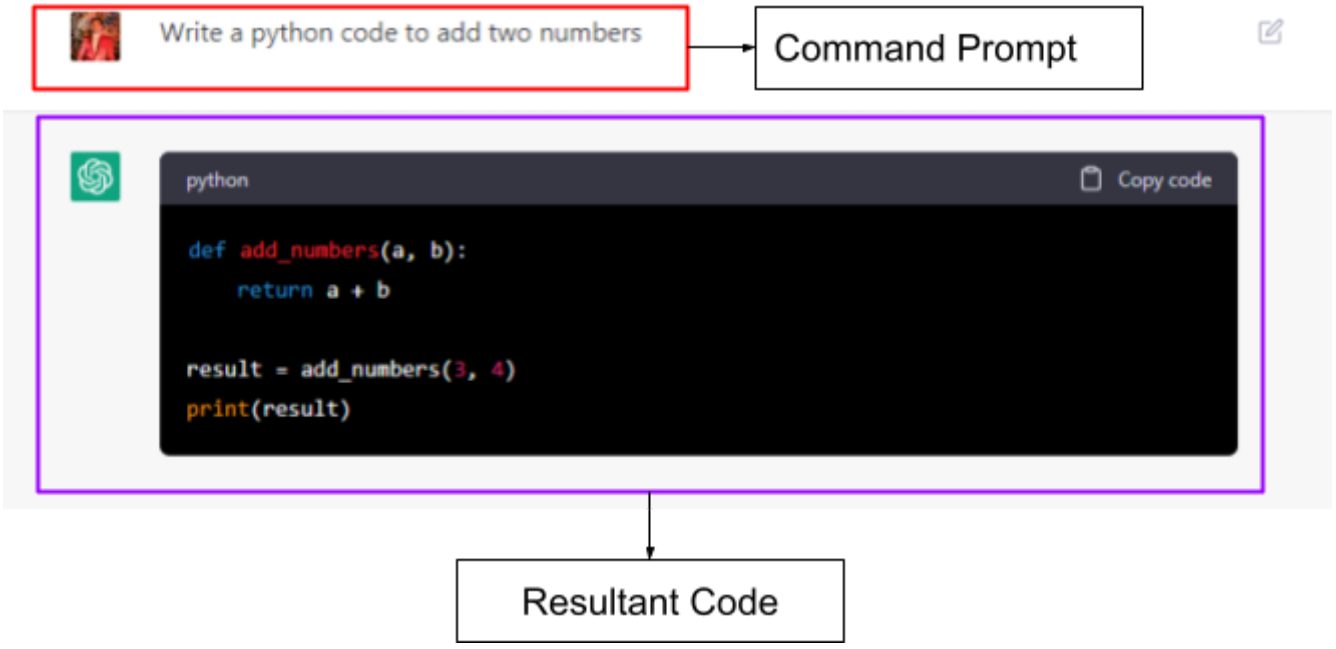


Fig2. Example demonstration to add two numbers using chatGPT

Report

For this assignment, and all other assignments and projects, you must write a report. In the report you will describe the critical decisions you made, important things you learned, or any decisions you made to write your algorithm a particular way. For each experiment, you are required to provide analysis of various hyperparameters.

1. Train model 100 epochs.
2. Plot training, validation and testing loss for both training and testing data with normalization and without normalization. Sort the ground truth values and prediction values of both training and testing data and plot it. Report the difference in their prediction and loss curves. Analyze and write down your findings.
3. Plot the loss curve by changing hyper-parameters.
    - learning_rate : 0.01, 0.00001, 0.9
    - number of epoch : 100, 150
    - initialization: all three options

    Make a table indication mean loss over validation data for each.

    Pick best hyper-parameters by looking at the validation set, train the model using those and report results on the Testing Dataset.
4. Add the prompt command of chatGPT in the report. Also, add the screenshots of the generated code from the chatGPT. Note : you can use the code but you cannot access the referenced material or links or any other material pointed out by the chatGPT.
5. Analyze the differences between your own code and the code generated from the chatGPT by plotting the loss curves and prediction curves or any other thing you find interesting
6. A code without report or report without code will not be graded.
7. Your report should be generated from your own code that you have submitted.

Marks Division:

1. The marks division is as below:
    a. Working code [55 points]

        i. Task 1
            1. Feedforward [10 points]
            2. Optimization [15 points]
            3. Normalization [05 points
            4. Loss [05 points]
            5. Training Loop [05 points]
            6. Code commenting [05 points]
        ii. Task 2
            1. Feedforward [2.5 points]
            2. Optimization [5 points]
            3. Loss [2.5 points]

    b. Report [35 points]
        i. Task 1
            1. Loss curve [10 points]
            2. Prediction Curve [05 points]
            3. Analysis (in different experiments) [05 points]
        ii. Task 2
            1. Loss curve [05 points]
            2. Prediction curve [2.5 points]

3. Analysis (in different experiments) [2.5 points]

iii. Analysis

1. Between your code and chatGPT [05 points]

c. Evaluation and Viva [10 points]