CS240-Week-2: Linear Regression and Gradient Descent

Instructions

- This lab intends to introduce you to linear regression and gradient descent along with `PyTorch`, a versatile python package extensively used in machine learning. This lab also reinforces the understanding of basis functions used for non-linear regression.
- This lab will be **graded**.
- Please read the problem statement and the submission guidelines carefully.
- All code fragments need to be written within the *# Student Code Start* and *# Student Code End* blocks in the given Python files. Do not change any other part of the code.
- **Do not** use any *print* statements in the final submission since the submission will be evaluated automatically.
- For any doubts or questions, please contact either the TA assigned to your lab group or one of the 2 TAs involved in making the lab.
- The submission will be evaluated automatically, so stick to the naming conventions and follow the directory structure strictly.
- The deadline for this lab is **22 January, 5:30 PM**.
- The submissions will be checked for plagiarism, and any form of cheating will be appropriately penalized.

The directory structure should be as follows (nothing more, nothing less). **Since your submission will go through an automated grading tool if these structures are not properly maintained, it may not be graded properly (even if the program is correct)**.

```
cs240_rollno_lab2 /
     |- q1 /
         |- Q1.py
         |- l2_loss.txt
         |- test_err.txt
         |- w_closed.txt
         |- w_trained.txt
     |- q2 /
         |- q2.py
```

After creating your directory, package it into a tarball: **cs240_rollno_lab2.tar.gz**

Command to generate tarball:
`tar -czvf cs240_rollno_lab2.tar.gz cs240_rollno_lab2/`

1 Question 1: Linear Regression [10 + 10 + 10 + 30 = 60 marks]

The objective of this problem is to implement basic linear regression, using both the closed form solution and gradient descent. We will be using `PyTorch` for this question. Go through the documentation of `PyTorch` here. Use the following command on the linux terminal to install the `PyTorch` package:

```
pip install torch
```

- First, implement the closed form solution of linear regression as discussed in the class. You can use the linear algebra functionalities present in `torch.linalg` [10 Marks]
- Secondly, complete all the loss functions and the gradient function. Note that you may need basics of matrix calculus for this. [10 + 10 Marks]
- Finally, implement the gradient descent algorithm as discussed in the class. [30 Marks]

`Q1.py` has sections of code where you need to implement the above things.

1.1 Linear Regression

As discussed in the lectures, the setup that we have is as follows:

- **Dataset:** $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{n}$, where $\mathbf{x}^{(i)}$ are the input data points and $y^{(i)}$ are the corresponding observed output. Each input data point is a d-dimensional vector, i.e., $\mathbf{x}^{(i)} \in \mathbb{R}^d$, and the outputs are scalars, i.e., $y^{(i)} \in \mathbb{R}$.
- **Weights:** $w \in \mathbb{R}^d$ where d is the dimension of the dataset.
- **Hypothesis Function:** $h_w(\mathbf{x}) = w^\top \mathbf{x}$ is the hypothesis function to be optimised to predict the outputs for any point $\mathbf{x}$ in the d-dimensional space.

For the task of optimisation we define the loss function as follows:

$$L_2 \textbf{ Loss:} \quad \mathcal{L}(w) = \frac{1}{n}\sum_{i=1}^{N}(y_i - w^\top \mathbf{x}^{(i)})^2.$$

Also recall that the closed-form solution for linear regression is given by:

$$w^* := (X^T X)^{-1} X^T y$$

Refer to the pseudocode below for implementing Gradient Descent for the last sub-part of this question

Algorithm 1 Gradient Descent(X, Y, w, η)

```
ϵ ← 1e − 15
old_loss ← 0
while abs(old_loss − f(X, Y, w)) > ϵ do
    old_loss ← f(X, Y, w)
    dw ← ∇f(X, Y, w))
    w = w − η ∗ dw
end while
```

1.2 Code

To run the file use the following command line:

```
python3 Q1.py <dataset01.csv> --seed <seed_value> --eta <learning_rate>
```

The gradient descent submission will be evaluated for different datasets with different seeds (these seeds are used to initialise d-dim weight tensors passed to the Gradient descent algorithm) and different learning rates η. The code has been divided into several functions for your convenience. The following are the functions present in the code file:

- `train_test_split()`: This function generates an 80:20 train:test split given a dataset. The significance of the train:test split will be discussed later in the class. You do not need to edit this function.
- `w_closed_form()`: Write the code for the closed-form solution of the linear regression problem in this function. This function writes the value of weights obtained to a file `w_closed.txt`.
- `l2_loss()`: Write the code to calculate L2 Loss in this function.
- `l2_loss_derivative()`: Write the code to compute gradient of the L2-Loss function. (You may need basic matrix calculus for this question).
- `train_model()`: Implement gradient descent using the pseudo-code given in the code as a reference. Note that you will have to **optimise the `l2_loss` on train dataset** i.e. optimise `l2_loss(X_train, Y_train, w)` and update w in the direction of decreasing training loss. Append the test error (`l2_loss(X_test, Y_test, w)`) to the array `test_err`. This will be used in the **autograder to verify the gradient descent** and to plot a graph.

As a sanity check, you can compare the weights obtained after gradient descent and those given by closed-form solution. Ideally, these should be very similar to each other.

1.3 Plot (ungraded learning exercise)

The plot generated at the end shows how the error on the test set varies. The important thing to note here is that while the training error decreases continually, the error on the test set starts to increase after a point. This is called **overfitting** and will be taught in future lectures.

2 Question 2: Basis Functions [15 + 25 = 40 Marks]

We will use basis functions to try and improve the model's fit from Q1. As we saw for our given test cases, the loss for the second test case was huge. But if you try to plot the data points, you can see some polynomial resemblances. We can transform it using a Polynomial Basis Function, as shown,

$$\phi(x) = [1, x, x^2, \ldots, x^n]$$

Here, n is the degree of the polynomial. Complete the following:

- Firstly, We have added a file `Q2.py` in the Q2 folder. All the functions in the file are similar to Q1 except one. You are expected to fill the functions in the same manner as in Q1. We will be verifying the outputs in the same manner.
- Secondly, complete the `transform_features(X, degree)`, which takes the feature matrix X and parameter degree and returns the basis function feature matrix $\Phi(X)$. Please see the classnotes to find the structure of the matrix Φ. [15 Marks]
- Thirdly, Find the minimum optimal degree for the polynomial. The more the degree of the polynomial, the more features we will end up with. This might seem like a more fruitful option, but this leads to overfitting (which will be covered in future lectures). For this, you have to start with degree=1, and go till higher degrees (till degree = 10). Create a file named `output.txt` which contains 11 lines: 10 lines of which contains the results for the different degrees "⟨`degree`⟩ ⟨`training l2 loss`⟩ ⟨`test l2 loss`⟩", and at the end your file should be printing "`optimal degree` ⟨`optimal_degree`⟩". Hint: optimal degree will have both lesser training loss as well as test loss, and try to choose the lowest possible degree. [$2 \times 10 + 5$ Marks]