# ISE - 364/464: Introduction to Machine Learning
# Homework Assignment 3

This assignment requires the accompanying Python file "HW_3_Code.py" to complete problems 4 and 5 (found in the .zip file).

The goal of this assignment is to provide a series of problems that strengthen knowledge in the mathematics, geometry, and intuition behind linear regression, a deeper understanding of the properties of the MSE loss function when using a linear model, and lastly, an introduction to working with the sigmoid function (which is applied in logistic regression models).

**Grading:** This assignment is due on Coursesite by E.O.D. 10/18/2024. All problems are worth the same number of points. If a problem has multiple parts, each of those parts will be worth equal amounts and will sum to the total number of points of the original problem (Example: If each problem is worth a single point, and problem 1 has 4 parts, each part will be worth 1/4th of a point). ISE - 364 students are only required to answer problems 1 through 4; however, you are allowed to answer the 5th graduate-level question (if done so correctly, you will receive extra credit in the amount that the 5th problem will be worth for the ISE - 464 students). ISE - 464 students are required to answer all 5 problems.

**Submitting:** Only electronic submissions on Coursesite are accepted. Students should submit a .zip file for this assignment, with two files inside: one file for the homework write-up and another for the code the students will write.

## 1 Problems

1. (**Properties of the MSE Loss Function for Linear Regression**) Recall that the Mean Squared Error (MSE) loss function for a linear regression model is defined as $J(\theta) := \frac{1}{2m} \|X\theta - y\|_2^2$, where $X \in \mathbb{R}^{m\times n}$ is the design matrix, $y \in \mathbb{R}^m$ is the target vector, and $\theta \in \mathbb{R}^n$ is the vector of parameters.

   **a)** Prove that the Hessian of this loss function is positive definite. That is, show that $\nabla^2 J(\theta) \succ 0$ for all values of $\theta$. To do this, assume that the design matrix $X$ has full-column rank (its columns are linearly independent, i.e., for all nonzero $z \in \mathbb{R}^n$, it follows that $Xz \neq 0$).
   *(Hint: We derived the gradient of this loss function in class; I would use this as a starting place.)*

   **b)** What does this result about the Hessian tell you regarding the nature of the solution one will obtain when minimizing the MSE loss function for a linear regression model?

**2.** (**Properties of the Sigmoid Function**) Consider the sigmoid function defined as $\sigma(z) := \frac{1}{1+e^{-z}}$, where $z \in \mathbb{R}$. Prove that $\sigma(-z) = 1 - \sigma(z)$.

**3.** (**Linear Regression Loss Function By Hand**) Suppose that you are given two datapoints defined by the feature vectors $x^{(1)} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$ and $x^{(2)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, along with the corresponding target values of $y^{(1)} = 4$ and $y^{(2)} = -2$.

**a)** Write the MSE loss function $J(\theta)$ with parameters $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$ using the linear model $h_\theta(x) = \theta^\top x$.

**b)** What is the value of this loss function when the current choice of parameters is $\theta = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$?

**c)** Write the value of the gradient $\nabla J(\theta)$ evaluated at the point $\theta = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$.

**4.** (**Training a Linear Regression Model via Gradient Descent**) For this problem, you will use Python to train a least-squares linear regression model via gradient descent. You will do this by writing a series of functions that will be called in your implementation of the gradient descent algorithm to minimize the MSE loss function for a linear model. Use the "HW_3_Code.py" file accompanying this document in the .zip file. This is the Python file that you will use to write your code. The functions that you are required to write have been pre-defined for you as well as all of the code needed for initialization (generating the dataset and visualization functions). Follow the comments in the .py file; they should guide you where to write your code.
Once you are finished, you should plot your own Linear Regression training plot (as shown in the right plot of Figure 1) as well as a 3D plot of your learned hyperplane (as shown in the bottom plot of Figure 1). Notice that you should "hyperparmeter tune" your learning rate and maximum number of iterations to achieve a "decent" convergence to the optimal solution (I have purposely not displayed plots that have fully converged; yours should).
Some things to look at:

- The optimal MSE loss function value for the parameters you obtained from the normal equations. This is the value you want your gradient descent algorithm to converge to.
- What is the gradient evaluated at your final learned parameter value? Is it close to 0?
- Try two different learning rate schemes: a fixed step size (it stays a constant value for every iteration, i.e., $\alpha_k = c \in (0, 1)$ for all iterations $k$) and a decaying step size (you can try using $\alpha_k = \frac{1}{k}$ or $\alpha_k = \frac{1}{\log(k)}$). You should notice that certain learning rate schemes are more effective than others to converge to a solution.
- How difficult is it to learn the optimal parameters with gradient descent compared to the normal equations?

Write a brief summary of your findings and submit your .py file along with your written (or typed) answers in a .zip file.

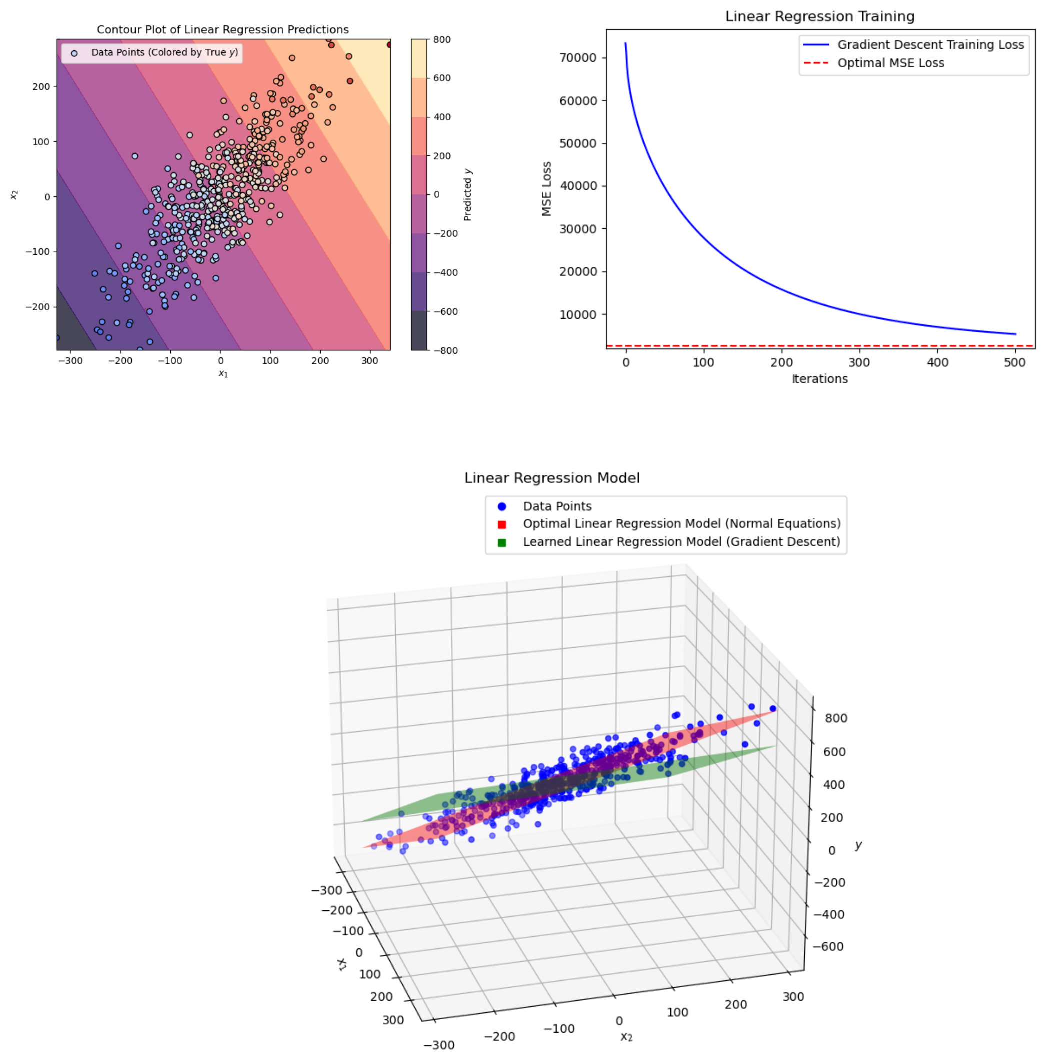

Figure 1: Three methods of illustrating training a linear regression model as well as visualizing the learned hyperplane on the dataset that it was trained on. (Top Left) A contour plot of the least-squares model colored by the model's predicted value of $y$ with the original datapoints overlaid on it (colored by the true value of $y$). (Top Right) A plot displaying the training progress in the loss function over the number of gradient descent iterations. (Bottom) A 3D scatter plot of the 2 feature dimensions and the target dimension along with the optimal least-squares hyperplane (obtained by the normal equations) and the trained (via gradient descent) hyperplane to minimize the MSE loss (this hyperplane is not the same as the other because I intentionally did not train to full convergence).

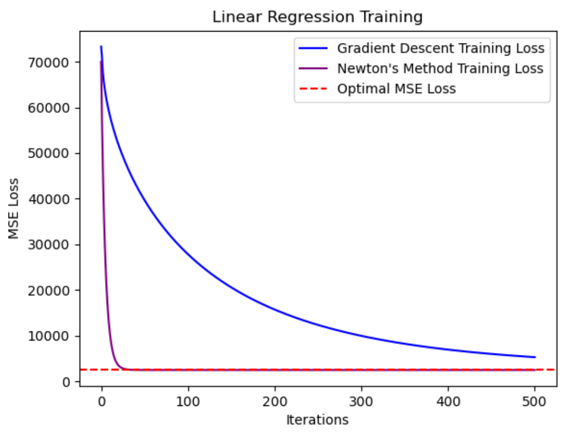

Figure 2: Optimization performance of both gradient descent and Newton's Method when training a least-squares linear regression model.

5. **(ISE-464 Graduate Students)** Using the same Python code file that you wrote for problem 4, add another algorithm to implement Newton's Method to minimize the MSE Loss. This will require you to define another function that computes the Hessian Matrix as well as a function to compute the Newton Direction. You should obtain a performance plot similar to the one displayed in Figure 2. Display your own performance plot that you obtain and write a brief summary of your observations comparing the two algorithms. Simply include your added code in the same file you submit for problem 4.
*(Hint 1: You should have derived the expression for the Hessian of the MSE Loss in problem 1.)*
*(Hint 2: I would recommend, instead of writing an entirely new algorithm, just edit you current gradient optimization algorithm with an option to toggle between using either the gradient descent direction or the Newton Direction; you may want to rename the algorithm to be a more fitting "opt_algo" instead of "gradient_desc" but that is up to you ;).)*