HW 2: ECE 601 Machine Learning for Engineers

Important Notes:

(a) When a HW question asks for writing a code, you would need to include the entire code as well as the output of the program as well as any other analysis requested in the question.

(b) Don't panic about the length of the HW assignment. HW assignments are treated as opportunities for improving learning and understanding, so I might include some extra text to help you better understand the concepts or learn about a point that was not covered during the class. The actual work needed from you is indeed manageable.

1. In this problem, we investigate the issue that gradient descent can converge to different values based on the initial values. Here the goal is to find to minimize the following function in the range $0 \leq x \leq 6$.

$$y = f(x) = 36x - 8(x-1)^2 + (x-3)^4$$

(a) Write a python code that plots the following function in the range $0 \leq x \leq 6$.

$$y = 36x - 8(x-1)^2 + (x-3)^4$$

(b) By looking at this plot, discuss why GD will converge to two different values based on the initial conditions.

(c) Find the two optimizing points (x_1^*, y_1^*) and (x_2^*, y_2^*) by implementing Gradient Descent. Which one would you accept?

(d) Look at the graph again, specifically, note that there is a local maximum between 3 and 4. Let's call it x_{max}. Mathematically (not using the code), describe the set of initial values x for which GD converges to (x_1^*, y_1^*), and the set of initial values x for which GD converges to (x_2^*, y_2^*). (Assume we know the value of x_{max}. Note: x_{max} can be easily found using Gradient Ascent, i.e., instead of subtracting a multiple of the Gradient you add it in each iteration. Equivalently, you can run Gradient Descent for $-f(x)$ to find x_{max}. You do not need to find x_{max} for this homework.)

2. **The Impact of Learning Rate:** The learning rate α can have a significant impact on the performance of gradient descent. Specifically,

   If the learning rate is too small, gradient descent can be very slow to converge. This means it will take a large number of iterations to reach the minimum of the cost function, which can be computationally inefficient and time-consuming.

   Conversely, if the learning rate is too large, gradient descent may overshoot the minimum, potentially leading to divergence or causing the algorithm to oscillate back and forth across the minimum without settling. This can prevent the algorithm from finding the optimal solution.

   In this problem, we would like to investigate this phenomenon. Specifically, consider the simple case where our cost function has been evaluated and given by the following function:

$$J(w) = (w-2)^2$$

   Write a python program to implement gradient descent with three values of learning rates $\alpha = .01, .1, 1.01$. Plot the cost value $J(w)$ as a function of the iteration number for the first 10 iterations. For all cases, assign the initial value $w_0 = 0$. Based on the resulting graphs, explain the impact of α.

   Note: The above curves (cost values vs the number of iterations) are examples of *learning curves*. In machine learning (ML), a learning curve is a graphical representation of the learning process of a model over time or experience. Specifically, it plots the model's performance on the training set and the validation set as a function of the number of training samples or iterations.

3. **Trigonometric (Harmonic) Regression:** In this problem, the goal is to implement a similar model fitting and selection similar to the polynomial regression done in class, with a different format. Let's consider a general case, were the model is given by

$$f(x; w) = \sum_{j=0}^{n} w_j f_j(x)$$

   In polynomial regression, $f_j(x) = x^j$ (note here w_0 is the same as the bias term b in polynomial regression). In this setting,

$$w = [w_0, w_1, w_2, \cdots, w_n]$$

   is the wight vector. The value n determines the model complexity. After fitting and finding the weights w_j's, we can write

$$\hat{y}^{(i)} = f(x^{(i)}; w) = \sum_{j=0}^{n} w_j f_j(x^{(i)})$$

So the cost function can be written as before

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}\left[\sum_{j=0}^{n} w_j f_j(x^{(i)}) - y^{(i)}\right]^2$$

So, we can write

$$\frac{dJ}{dw_j} = \frac{1}{m}\sum_{i=1}^{m} f_j(x^{(i)})(\hat{y}^{(i)} - y^{(i)}), \quad j = 0, 1, 2, \cdots, n$$

(a) Consider a model in the above setting, where $f_j(x) = \cos(2j\pi x)$. Here the assumption is that the data is scaled so that $0 \leq x \leq 1$.

(b) Look at the provided Jupyter notebook (HW 1 2 Cosine Example). Go through it carefully running all the cells.

(c) After you feel comfortable understanding it, start your notebook and write your code to accomplish the same steps. Feel free to modify or implement in a more efficient way (not required). Download the same data and make sure you get the same results. It is OK if your code is very similar to the provided code, the point is that you go through this process yourself to make sure you know all the details. In your submission, include the entire notebook with the results. This exercise goes through all the model selection, fitting, and regularization steps covered in class, so it can be very helpful for reinforcing all the concepts.

Note: The concept similar to polynomial regression, where sine or cosine functions are used instead of powers of x, is known as **Trigonometric Regression** or **Harmonic Regression**. In trigonometric regression, the model employs sine and cosine functions to fit periodic or oscillatory data, making it particularly useful for modeling phenomena that exhibit a cyclical pattern, such as seasonal effects, tidal movements, or time series with regular frequency.

A typical trigonometric regression model might be expressed as:

$$y = a_0 + a_1\cos(wx) + b_1\sin(wx) + a_2\cos(2wx) + b_2\sin(2wx) + \ldots + \epsilon$$

where:

- y is the dependent variable.
- x is the independent variable.
- $a_0, a_1, b_1, a_2, b_2, \ldots$ are the coefficients of the model.
- w is the frequency of the sine and cosine terms.
- ϵ represents the error term.

Like polynomial regression, trigonometric regression can be implemented using linear regression techniques because the model is linear with respect to the coefficients, even though it is nonlinear in terms of the independent variable x. This type of regression is particularly effective when the underlying data pattern is repetitive and can be captured by sinusoidal functions.