036049: Applied ML4SE, Winter 2023
Homework 2

Due: February 13, 2024

This homework is related to the material in Chapter 2 of UDL on supervised learning and linear regression.

Theory

1. Please do Problem 2.1 in UDL text (10 points)
2. Please do Problem 2.2 in UDL text (10 points)

Computation

1. Consider a simple linear regression relating the dependent variable y to an independent variable x:

$$y = \beta_0 + \beta_1 x + \epsilon$$

   where β_0 is the $y-$intercept, β_1 is the slope of the line, and ϵ is the error. The goal is to estimate values of β_0, β_1 so the best fit line results. This is done by minimizing the sum of the squares of the errors:

$$E = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

   This can be done setting to zero the partial derivatives of E with respect to β_0 and β_1 which can then be solved to get:

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

   with $\bar{x}, \bar{y}$ the means of x and y, resp. Consider the following data points:

x	1	2	3	4	5	6
y	1	3	2	5	4	6

   Write a Python program using `numPy` and `matplotlib` libraries to implement the above linear regression method for the above training data. Plot the data points and the best fit line. (25 points)

2. Another approach to estimate β_0, β_1 is via the *gradient descent optimization algorithm* to minimize a function. This approach iteratively updates the β_0 and β_1 values and moves in the direction of the steepest descent as measured by the negative of the gradients $\frac{\partial J}{\partial \beta_0}$ and $\frac{\partial J}{\partial \beta_1}$, where J is the cost function.

   Cost function: The cost function which we want to minimize is the summation of the squared errors. We start by choosing initial values for β_0, β_1 (denoted $\boldsymbol{\beta} = [\beta_0, \beta_1]$) and update them in the opposite direction of the gradient until we converge. The learning rate α controls the step size at each iteration. We first define the cost function $J(\boldsymbol{\beta})$ which is the sum of squared errors but averaged over the number of observations m and multiplied by factor 1/2 for mathematical convenience during the derivation (that is, the derivative of a squared function). Let $\mathbf{x}^{(i)}$ be the feature vector for the i^{th} training example (row 1 in the table of the previous question) and $y^{(i)}$ the corresponding target value (row 2 in the table of the previous question). The cost function is defined:

$$J(\boldsymbol{\beta}) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_{\boldsymbol{\beta}} \left(\mathbf{x}^{(i)} \right) - y^{(i)} \right]^2 \tag{1}$$

where $h_{\boldsymbol{\beta}} \left(\mathbf{x}^{(i)} \right) = \boldsymbol{\beta}^T \mathbf{x}$ is hypothesis function (model) which gives our predicted value of y for given x. The generalized hypothesis function is defined as follows:

$$h_{\boldsymbol{\beta}} \left(\mathbf{x}^{(i)} \right) = \beta^T \mathbf{x}^{(i)} = \beta_0 x_0^{(i)} + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots \tag{2}$$

In the case of linear regression $x_0^{(i)} = 1$ (for all i) and $x_1^{(i)} = x^{(i)}$ are as provided in the table of the previous problem. The hypothesis function for in the linear regression reduces to:

$$h_{\boldsymbol{\beta}} \left(\mathbf{x}^{(i)} \right) = \beta_0 x_0^{(i)} + \beta_1 x_1^{(i)} \tag{3}$$

Gradient of cost function: The idea of gradient descent algorithm is to update each parameter β_j in $\boldsymbol{\beta}$ such that cost function $J(\boldsymbol{\beta})$ is minimized. The update rule for β_j is obtained by taking derivative of cost function with respect to β_j and setting it to 0.

$$\frac{\partial J}{\partial \beta_j} = \frac{1}{m} \sum_{i=1}^{m} \left[h_{\boldsymbol{\beta}} \left(\mathbf{x}^{(i)} \right) - y^{(i)} \right] x_j^{(i)} = 0 \tag{4}$$

$$\text{Update rule for } \beta_j \rightarrow \quad \beta_j := \beta_j - \alpha \frac{\partial J}{\partial \beta_j} \tag{5}$$

where α is learning rate and $x_j^{(i)}$ is j^{th} feature of i^{th} training example. Write a Python program using `NumPy` and `matplotlib` libraries for the same data set as in the previous example. Write Python function to define the cost function and gradient descent functions. Call your gradient descent function using $\alpha = 0.01$ and use 1000 iterations. Plot the cost history to check convergence and also plot the data points and the best fit line using the estimated β_0, β_1. Consider varying the learning rate. (30 points)
(30 points)

3. *Regularization* is a technique to deal with overfitting and poor generalization by adding a term to the cost function that favors small coefficients. Two examples are Lasso and Ridge regularization which add $\lambda \sum_{j=1}^{p} |\beta_j|$ and $\lambda \sum_{j=1}^{p} |\beta_j^2|$, resp. to the cost function $J(\boldsymbol{\beta})$. Study the effect of each of these by implementing them in your linear regression code above for different values of λ. (10 points)

4. Consider a shallow neural network with:

   - Two input features (neurons)

- One hidden layer with two hidden units, and
- One output layer (for binary classification)

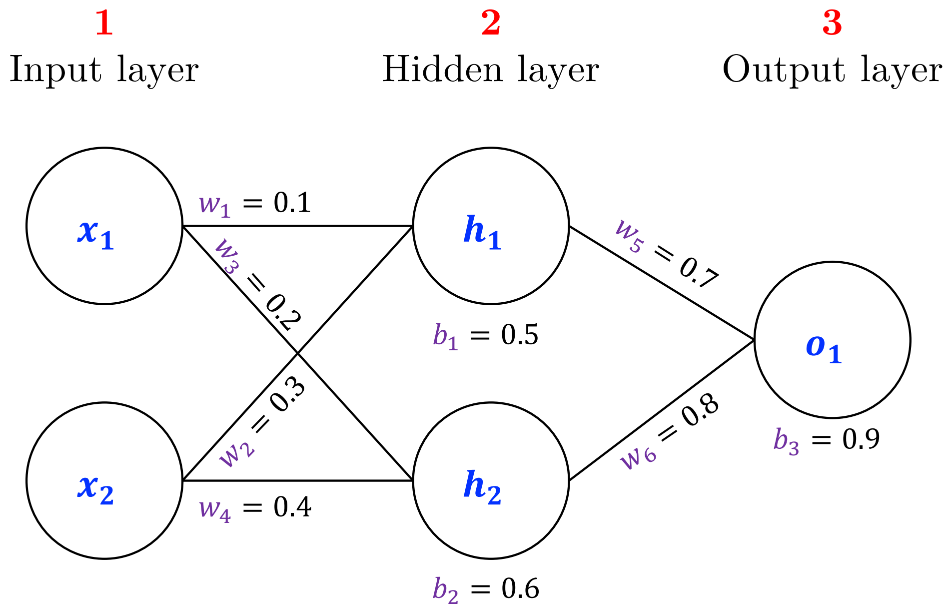

Figure 1: Neural network map

Consider $X = \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} 0.3 & 0.7 \end{bmatrix}$ as the input layer values. Initialize the neural network with the following weights and bias:

- Layer 1 $\rightarrow$ 2 weights: $W_1 = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$
- Hidden layer bias: $B_1 = \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.6 \end{bmatrix}$
- Layer 2 $\rightarrow$ 3 weights: $W_2 = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix}$
- Output layer bias: $B_2 = [b_3] = [0.9]$

Use the sigmoid activation function for both hidden and the output layers. Considering the output value to be true (i.e. $o_1 = 1$), perform hand calculation for one iteration of a forward pass. Compute the squared error function i.e. error $= \frac{1}{2}(\text{output-target})^2$. (15 points)