ECE 174: Introduction to Linear and Nonlinear Optimization with Applications
Fall 2020

Mini Project # 2
Due: Wednesday December 16, 11:59pm

Instructions: In this mini proejct, you will train a simple neural network using non-linear least squares algorithms, and test them to approximate certain non-linear functions. The goal of the programming assignment is to implement non-linear least squares algorithms, so you must write your own code to implement the desired iterative algorithms.

- You can write the source code in any language of your choice (Python, MATLAB). The code should be properly commented and you will be required to submit the source code. You should write your own code and not copy from any resource on the internet. If you consult any resource online before writing your own code, you must cite the source, add a comment to your code with a link to the source you got it from. Any uncited code you did not write yourself, or any code which is substantially similar to another student's, will be referred to the Office of Academic Integrity.
- In addition to the source code, you will submit a detailed report that describes the simulations and explains the results using figures and mathematical interpretations. Your report must address all the questions asked in this assignment. We do not have any detailed requirements on the formatting, but we expect the report to be presented in a complete and professional manner, with appropriate labels on items and comments which demonstrate understanding of the results.
- Your grade on the programming assignment will be based on the correctness of the code as well as the quality of the report. Since our evaluation of your code is based on its results as presented in your report, correct code is very important. We have every intention of awarding partial credit, but for instance if a small bug causes your program's output to be unusable throughout the report, we cannot award much credit even if the bug was very small.

---

1. **Reading:** Read Sec. 18.1-18.3 of the textbook.
2. **Simple Neural Networks:** A neural network is widely used parametric model to approximate non-linear functions. These networks are parameterized by a set of weights. Neural networks are often referred to as "universal approximators", which means that they can approximate a large class of non-linear functions by suitably tuning the weights. In this assignment, you will program a simple neural network to approximate different types of

non-linear functions using “non-linear" least squares algorithms learned in lectures. A neural network is a model of the form

$$\hat{y} = f_{\mathbf{w}}(\mathbf{x}) \tag{1}$$

where $\hat{y} \in \mathbb{R}$ is the scalar output, $\mathbf{x} \in \mathbb{R}^n$ is the vector of inputs and $\mathbf{w} \in \mathbb{R}^p$ is the vector of weights that parameterize the network. In this exercise, we will consider a simple neural network of the following form

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) =& w_1\phi(w_2x_1 + w_3x_2 + w_4x_3 + w_5) + w_6\phi(w_7x_1 + w_8x_2 + w_9x_3 + w_{10}) \\ &+ w_{11}\phi(w_{12}x_1 + w_{13}x_2 + w_{14}x_3 + w_{15}) + w_{16} \end{aligned} \tag{2}$$

Clearly, we have $n = 3, p = 16$. The function $\phi : \mathbb{R} \to \mathbb{R}$ is the sigmoid function given by

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3. **Training and Testing the Neural Network:** The goal of this assignment is to determine the weights $\{w_i\}_{i=1}^{16}$ for approximating different non-linear functions. We will compute these weights using non-linear least squares algorithms learned in class. Suppose we want to use the neural network defined by (**??**) to approximate a non-linear function $g(\mathbf{x}), \mathbf{x} \in \mathbb{R}^3$. Let

$$y^{(n)} = g(\mathbf{x}^{(n)}), n = 1, 2, \cdots N$$

be samples of the function $g(\mathbf{x})$ at N points $\mathbf{x}^{(n)}, n = 1, 2, \cdots N$. Consider the sum of squared errors

$$\sum_{n=1}^{N} \underbrace{\left(f_{\mathbf{w}}(\mathbf{x}^{(n)}) - y^{(n)}\right)^2}_{r_n^2(\mathbf{w})} \tag{3}$$

   (a) Using the sigmoid function defined above, compute an expression for the gradient $\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x})$. You can leave the expression in terms of ϕ and the derivative of ϕ. *

   (b) Compute the Derivative matrix $\mathbf{Dr}(\mathbf{w})$ using the gradient from previous part (read page 383 of the textbook). Notice that $\mathbf{r}(\mathbf{w}) = [r_1(\mathbf{w}), \cdots, r_N(\mathbf{w})]$ where $r_n(\mathbf{w})$ is defined in (**??**).

   (c) We will train the neural network to approximate the non-linear function $g(\mathbf{x}) = x_1x_2 + x_3$. To this end, generate $N = 500$ random points $\mathbf{x}^{(n)} = [x_1^{(n)}, x_2^{(n)}, x_3^{(n)}]^T, 1 \leqslant n \leqslant N$ in $\mathbb{R}^3$ and assign $y^{(n)} = x_1^{(n)}x_2^{(n)} + x_3^{(n)}$. Use these $(\mathbf{x}^{(n)}, y^{(n)})_{n=1}^N$ pairs to train the neural network by minimizing the following function (aka training loss) with respect to $\mathbf{w}$:

$$l(\mathbf{w}) = \sum_{n=1}^{N} r_n^2(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2$$

   Notice for $\lambda = 0$, it reduces to a simple non-linear least squares and for non-zero λ, it represents a regularized non-linear least squares problem.

*During implementation however, you will need to write a function that explicitly computes the derivative of ϕ for different iterations.

i. Write your own code implementing the Levenberg-Marquardt algorithm to approximately solve the above non-linear least squares problem. Your code should take $(\mathbf{x}^{(n)}, y^{(n)})_{n=1}^{N}$ as inputs and produce the learned $\mathbf{w}$ as the output. Choose a very small value of λ (say $\lambda = 10^{-5}$) and experiment with a few choices of λ. Choose a suitable stopping criterion (Read Page 393 of the textbook) and specify it in your report. Plot the value of training loss $l(\mathbf{w})$ versus iterations to show how your training loss decreases with iterations. Try different initializations for the Levenberg-Marquardt algorithm and see the effect on final model training loss error.

ii. Test the model on another set of N_T randomly generated points $\tilde{\mathbf{x}}^{(n)}, n = 1, 2, \cdots N_T$ (which are different from those used to train the network). Show how you compute the error for test points (try different values of N_T between 50 and 300) and report the values. Summarize your training and testing error metrics for different initializations, different choices of λ and N_T. Comment on your results.

iii. Repeat part (i) and (ii) for a different non-linear function $g(\mathbf{x})$ of your choice. Define the function $g(\mathbf{x})$ that you choose, and describe how you will re-use/modify your code from previous part.

iv. **Extra (no additional credit, for fun):** You may also want to visualize the function that you learned using the neural network and compare it against the actual function $g(\mathbf{x})$. Since this is a function of 3 variables, you will need to use contour plots. Visualize the contour plots of the actual function and its approximation. Comment on the results.