Generalized Least-Squares in Neural Networks

Lloyd Sangwoo Ko

March 11, 2025

1 Introduction

This report explores the use of generalized least-squares problems in neural networks. We derive the necessary expressions, implement them in MATLAB, and analyze the numerical results.

2 Introduction

Neural networks are widely used in classification tasks, where the goal is to learn a function that maps input data to discrete class labels. In this exercise, we consider an extremely simplified neural network with only one neuron, which can be viewed as a generalized linear classifier with a nonlinear activation function.

2.1 Background on Neural Networks and Least-Squares Optimization

The neuron takes an input vector $a_i \in \mathbb{R}^n$ and produces an output through a sigmoid activation function, parameterized by a weight vector $x \in \mathbb{R}^n$:

$$\omega(a_i^T x),$$

where the sigmoid function is defined as:

$$\omega(z) = \frac{1}{1 + e^{-z}}.$$

The sigmoid function is commonly used in logistic regression and binary classification because it maps real-valued inputs to a range between 0 and 1, which can be interpreted as probabilities.

For supervised learning, we are given a set of training data:

- Feature vectors: $a_1, a_2, \ldots, a_m$ (each $a_i \in \mathbb{R}^n$)
- Binary labels: $b_1, b_2, \ldots, b_m$ (each $b_i \in \{0, 1\}$)

Our goal is to train the neural network by adjusting the weights x such that the predicted outputs match the given labels as closely as possible.

2.2 Defining the Least-Squares Loss Function

To quantify the error between the network's predictions and the true labels, we define the error function for each data point:

$$g_i(x) = \omega(a_i^T x) - b_i.$$

This represents the difference between the predicted probability and the actual class label.

The total loss function is then given by the sum of squared errors:

$$f(x) = \frac{1}{2} \sum_{i=1}^{m} \left(g_i(x)\right)^2.$$

This is a generalized least-squares problem, which is a fundamental approach in machine learning and optimization. The factor of $\frac{1}{2}$ is included to simplify differentiation.

2.3 Optimization: Computing Gradients and Hessians

To minimize $f(x)$, we use gradient-based optimization techniques, such as gradient descent or Newtons method. These methods require computing the gradient $\nabla f(x)$ and Hessian matrix $\nabla^2 f(x)$.

Using the chain rule, we derive:

$$\nabla g_i(x) = a_i \omega'(a_i^T x),$$

where

$$\omega'(z) = \omega(z)(1 - \omega(z)).$$

For the Hessian:

$$\nabla^2 g_i(x) = \omega''(a_i^T x) a_i a_i^T,$$

where

$$\omega''(z) = \omega'(z)(1 - 2\omega(z)).$$

The second derivative provides curvature information, which is useful for Newtons optimization methods.

2.4 Implementation and Testing

The goal of this exercise is to:

1. Derive the gradient $\nabla g_i(x)$ and Hessian $\nabla^2 g_i(x)$.
2. Implement these functions in MATLAB.
3. Compute and report the function values, gradient, and Hessian for a specific example.

The provided MATLAB functions allow us to evaluate the optimization landscape and understand how the loss function behaves as we adjust the neural networks parameters.

3 Mathematical Derivations

3.1 Gradient Computation

Applying the chain rule:

$$\nabla g_i(x) = \omega'(a_i^T x) \nabla (a_i^T x).$$

Since $\nabla (a_i^T x) = a_i$, we obtain:

$$\nabla g_i(x) = a_i \omega'(a_i^T x).$$

3.2 Hessian Computation

Differentiating $\nabla g_i(x)$:

$$\nabla^2 g_i(x) = \omega''(a_i^T x) a_i a_i^T.$$

Thus:

$$\nabla g_i(x) = a_i \omega'(a_i^T x), \quad \nabla^2 g_i(x) = \omega''(a_i^T x) a_i a_i^T.$$

4 MATLAB Implementations

4.1 Implementation of $g_i(x)$

```
function [gi, gradgi, Hessiangi] = mygi(ai, x, bi)
    z = ai' * x;
    [sx, sdx, sddx] = mySigmoid(z);
    gi = sx - bi;
    gradgi = ai * sdx;
    Hessiangi = sddx * (ai * ai');
end
```

Listing 1: MATLAB function for $g_i(x)$

4.2 Implementation of $f(x)$

```
function [f, gradf, Hessianf] = myf(A, x, b)
    [n, m] = size(A);
    f = 0;
    gradf = zeros(n, 1);
    Hessianf = zeros(n, n);
    for i = 1:m
        ai = A(:, i);
        bi = b(i);
        [gi, gradgi, Hessiangi] = mygi(ai, x, bi);
        f = f + 0.5 * gi^2;
        gradf = gradf + gi * gradgi;
        Hessianf = Hessianf + gradgi * gradgi' + gi *
            Hessiangi;
    end
end
```

Listing 2: MATLAB function for $f(x)$

5 Results and Analysis

5.1 Given Data

The script was executed with the following data:

$$A = \begin{bmatrix} 1 & 0.5 & -1 \\ 2 & -1 & 0.5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad x = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}.$$

5.2 Computed Values

$$f(x) = 0.7280,$$

$$\nabla f(x) = \begin{bmatrix} -0.1007 \\ 0.1743 \end{bmatrix},$$

$$\nabla^2 f(x) = \begin{bmatrix} -0.0647 & -0.0091 \\ -0.0091 & -0.0441 \end{bmatrix}.$$

6 Conclusion

This report derives and implements the gradient and Hessian for a single-neuron neural network using generalized least-squares. The numerical results demonstrate how these derivatives can be computed efficiently for use in machine learning optimization.