# Report: Gauss-Newton Training Algorithms and Hessian Approximation

**Student: Ahaan Tagare, Student ID: 33865799**
**Coursework 2**
**IS71040B: NEURAL NETWORKS (2024-25)**

## 1. Introduction

This report provides an analytical evaluation of the Gauss-Newton training algorithm for multilayer perceptron (MLP) neural networks, emphasizing the Hessian approximation and its role in optimizing network weights. The analysis is based on Coursework 2, which implemented Newton's method in two parts: a simple network (Part 1) and a sunspot prediction task (Part 2), comparing its performance against classical backpropagation. The key formulas, sourced from Haykin (1999)—Equation 4.108 (page 246) for the Hessian approximation and Equation 4.121 (page 257) for the weight update—are examined in their exact forms as provided, alongside results from the MATLAB execution. The report looks at how fast the method learns, how accurate it is, and the balance between speed and computer effort, using the given results.

Neural network training relies on optimization techniques to minimize error. Backpropagation, a first-order method, adjusts weights using gradients, often slowly. Gauss-Newton, a second-order method, incorporates curvature via the Hessian, aiming for faster convergence. This report breaks down these algorithms, their math basics, and how they perform in the coursework.

## 2. Gauss-Newton Algorithm: Detailed Structure and Formulas

The Gauss-Newton training algorithm is a method that improves how neural networks learn by using more advanced calculations. Instead of just following the steepest path to reduce error (like basic methods), it also considers how the error changes, helping the network learn faster and more accurately., distinguishing it from simpler gradient-based methods. It approximates the error surface as a quadratic function around the current weights, enabling precise weight updates that account for both the direction and curvature of the error landscape. This approach is particularly suited to problems with a sum-of-squares error function, such as regression tasks, where the error is defined as $E = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$, with $y_i$ as targets and $\hat{y}_i$ as predictions.

The algorithm's structure is iterative and methodical:

1. **Initialization**: Start with an initial weight vector $\mathbf{w}(0)$, a regularization parameter $\lambda$, and a predefined number of iterations (epochs).
2. **Iterative Optimization**:

- Compute the network output $\hat{y}$ for input $\mathbf{X}$ using forward propagation through layers with weights $\mathbf{w}$.
- Calculate the error $e = y - \hat{y}$ for each sample, where $y$ is the desired output.
- Derive the Jacobian $\mathbf{J} = \frac{\partial \hat{y}}{\partial \mathbf{w}}$, which captures how sensitive the output is to weight changes across all parameters.
- Compute the gradient and approximate the Hessian matrix using accumulated Jacobian terms.
- Update weights with a Newton step that balances direction and curvature.

3. **Termination**: Continue until convergence (e.g., error below a threshold) or after the set epochs.

The Gauss-Newton method refines the classical Newton approach by simplifying the Hessian computation, avoiding the need for second derivatives $\frac{\partial^2 E}{\partial \mathbf{w}^2}$, which are computationally intensive and sensitive to noise. Instead, it relies on first-order derivatives (the Jacobian), making it practical for MLP training while retaining second-order benefits. This simplification assumes the network is near a local minimum, where $\hat{y}_i \approx y_i$, reducing the error's residual impact on curvature.

The key formulas, as provided and corrected, are:

**2.1 Gradient**

$$\mathbf{g}(n) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial E_i}{\partial \mathbf{w}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{J}_i^T (y_i - \hat{y}_i)$$

This formula computes the gradient, averaging the error's sensitivity to weights over $N$ samples. For each sample $i$, $E_i = \frac{1}{2}(y_i - \hat{y}_i)^2$ is the squared error, and $\frac{\partial E_i}{\partial \mathbf{w}} = \mathbf{J}_i^T (y_i - \hat{y}_i)$ weights the Jacobian by the error, indicating the direction of steepest error increase. Averaging ensures a balanced update direction across the dataset.

**2.2 Hessian Approximation**

$$\mathbf{H}(n) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{J}_i \mathbf{J}_i^T + \lambda \mathbf{I}$$

Equation 4.108 from Haykin (1999, p. 246) defines the Hessian as $\mathbf{H}(n) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{J}_i \mathbf{J}_i^T$, which approximates curvature using the outer product of the Jacobian. In this implementation, we modify it by adding a regularization term $\lambda \mathbf{I}$ (e.g., $\lambda = 10^{-3}$) to ensure $\mathbf{H}$ is invertible and to address singularities when $\mathbf{J}_i$ has dependent rows. The term $\mathbf{J}_i \mathbf{J}_i^T$ captures how weight changes affect outputs across dimensions, averaged over $N$ samples.

**2.3 Weight Update**

$$\Delta \mathbf{w}(n) = -\mathbf{H}^{-1}(n) \mathbf{g}(n)$$

This formula, given as Equation 4.121 in Haykin (1999, p. 257), updates the weights by scaling the negative gradient $\mathbf{g}(n)$ with the inverse Hessian $\mathbf{H}^{-1}(n)$, assuming it exists. This step enables efficient convergence by solving the quadratic system $\mathbf{H}\Delta\mathbf{w} = -\mathbf{g}$, using curvature information for optimization. The Hessian H represents the second-order derivatives of the loss function, capturing how the gradient itself changes at different points in the optimization process.

## 3. In-Depth Analysis of Gauss-Newton and Hessian Approximation

The Gauss-Newton algorithm's strength lies in its iterative refinement of weights using curvature, unlike backpropagation's reliance on small $\eta$-scaled gradient steps (e.g., $\mathbf{w} = \mathbf{w} - \eta\nabla E$). By modeling the error surface quadratically, it computes a step $\Delta\mathbf{w}$ that theoretically jumps to the minimum in one iteration for a perfect quadratic, though practical error surfaces require multiple steps. This efficiency stems from the Hessian approximation, which simplifies the full Newton method's Hessian:

$$\mathbf{H}_{\text{full}}(n) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{J}_i\mathbf{J}_i^T + (y_i - \hat{y}_i)\frac{\partial^2 \hat{y}_i}{\partial \mathbf{w}^2}\right]$$

Gauss-Newton drops the second term $(y_i - \hat{y}_i)\frac{\partial^2 \hat{y}_i}{\partial \mathbf{w}^2}$, assuming small residuals near convergence, reducing complexity from $O(n^2m)$ (for $m$ outputs) to $O(n^2)$ for the Jacobian-based term.

The Hessian approximation (Equation 4.108) is critical. The term $\frac{1}{N}\sum_{i=1}^{N}\mathbf{J}_i\mathbf{J}_i^T$ aggregates curvature information across samples, where $\mathbf{J}_i$ is a vector of partial derivatives $\frac{\partial \hat{y}_i}{\partial w_j}$ for each weight $w_j$. This matrix's eigenvalues reflect the error surface's steepness, guiding the weight update's magnitude and direction. The regularization $\lambda\mathbf{I}$ shifts these eigenvalues upward, ensuring positive definiteness and numerical stability, especially in early training when residuals may be large.

Equation 4.121's weight update leverages this approximation, inverting $\mathbf{H}$ to scale $-\mathbf{g}$. This inversion, while computationally intensive ($O(n^3)$ for $n$ weights), allows Gauss-Newton to adapt weights more effectively than backpropagation, particularly in regions with flat or highly curved error surfaces. The method's reliance on $\mathbf{J}$ assumes a locally linear output response, valid for sigmoid or linear activations, aligning with the MLP structures in Parts 1 and 2.

## 4. Analytical Framework

### 4.1 Part 1: Simple Network

Part 1 trains a small MLP with 2 inputs, 3 hidden neurons, and 1 output, using initial weights $[-0.25, 0.33, 0.14, -0.17, 0.16, 0.43, 0.21, -0.25, 0, 0, 0]$, input $\mathbf{X} = [1; 0]$, and target $y = 1$. The Hessian approximation (Equation 4.108) is computed as $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}$ (

$\lambda = 10^{-5}$), and weights update via Equation 4.121. We have taken the weights 9, 10, 11 to create a fully connected neural network

### 4.2 Part 2: Sunspot Prediction

Part 2 applies both Gauss-Newton and backpropagation to a 10-5-1 MLP for sunspot prediction, using normalized data split 80-20. Gauss-Newton uses $\lambda = 10^{-3}$, while backpropagation uses a learning rate of 0.1. Performance is measured via training loss and normalized mean squared error (NMSE).

## 5. Results Analysis

### 5.1 Part 1 Results

After 100 epochs, Gauss-Newton achieves:

- **Final Error**: 0.000000, indicating perfect convergence to the target.
- **Final Weights**: $[-0.2231, 0.5875, 0.1236, -0.17, 0.16, 0.5427, 0.3591, -0.1122, 0, 0.0128, 0]$.
- **Weight Changes**: Notable adjustments (e.g., $w_2 : 0.33 \rightarrow 0.5875$, $\Delta = 0.2575$) reflect adaptation to minimize error.
- **Jacobian**: $[0.1340, 1.0000, -0.0279, 0, 0, 0.4445, 0.6428, 0.5309, 0, 0.0825, 0]$, showing sensitivity of output to weights.

The error plot (semilogy) shows rapid convergence, using the Hessian's curvature information.

### 5.2 Part 2 Results

#### Training Loss

- **Backpropagation**: Starts at 0.3354, decreases steadily to 0.0888 over 100 epochs, reflecting gradual improvement.
- **Gauss-Newton**: Begins at 3.2639, drops sharply to 0.2959 (Epoch 2), then stabilizes at 0.0505, showing faster initial convergence.

#### NMSE on Test Set

- **Backpropagation**: 0.5259, indicating moderate prediction accuracy.
- **Gauss-Newton**: 0.2971, significantly better, showing superior generalization.

#### Plots

- **Loss Comparison**: Gauss-Newton (red) falls faster than backpropagation (blue), plateauing lower.
- **Prediction Comparison**: Gauss-Newton (red) tracks actual data (black) more closely than backpropagation (blue).

## 6. Comparative Analysis

### 6.1 Convergence Speed

Gauss-Newton excels in Part 1, reaching zero error, and in Part 2, reducing loss from 3.2639 to 0.0505 faster than backpropagation's 0.3354 to 0.0888. The Hessian approximation (Equation 4.108) enables larger, curvature-informed steps, unlike backpropagation's small gradient steps.

### 6.2 Accuracy

Part 1's perfect fit and Part 2's lower NMSE (0.2971 vs. 0.5259) demonstrate Gauss-Newton's precision, using second-order information to better fit the error surface.

### 6.3 Computational Cost

Gauss-Newton's Hessian computation ($O(n^3)$ for inversion, where $n = 61$ in Part 2) is costlier than backpropagation's $O(n)$, evident in slower per-epoch execution despite fewer effective iterations.

## 7. Conclusion

The Gauss-Newton algorithm is a powerful optimization technique that outperforms traditional backpropagation in both convergence speed and accuracy. This is due to its use of second-order information, specifically the curvature of the error surface, which allows it to make more precise and effective weight adjustments during training. In simpler terms, instead of relying solely on the gradient (or slope) to find optimal weights, the Gauss-Newton method also accounts for the steepness of the slope, leading to quicker and more accurate updates.

The effectiveness of the Gauss-Newton algorithm was clearly demonstrated in Coursework 2. In Part 1, the algorithm achieved zero error, perfectly matching the target values. In Part 2, it produced a significantly lower NMSE (Normalized Mean Squared Error) than other methods, indicating that the model's predictions were much closer to the actual values. These results emphasize how Gauss-Newton's combination of gradient and curvature information leads to improved model performance.

However, while the Gauss-Newton method excels in speed and accuracy, it comes with a higher computational cost. The need to compute the Hessian matrix (which represents the curvature) and its inverse increases the time and resources required, particularly for large datasets or complex models. This makes the method less scalable for very large tasks.

In the implementation, the weight update formula, $-H^-1 * g$, was correctly applied, along with regularization of the Hessian, ensuring that the algorithm's performance aligned with theoretical expectations. This approach produced reliable and robust results, though the computational burden remains a challenge.

To address these limitations, future improvements could focus on incorporating an adaptive lambda (a regularization parameter). By dynamically adjusting lambda, it would be possible to strike a balance between accuracy and computational efficiency, making the Gauss-Newton algorithm more suitable for larger datasets and more complex models without sacrificing performance. This enhancement would help scale the method to a broader range of problems while keeping the computational cost manageable.

## 8. Practical Benefits and Challenges

The Gauss-Newton method helps neural networks learn faster and more accurately than backpropagation, as seen in Parts 1 and 2. It uses the error surface's shape to take smarter steps, cutting down training time and improving predictions, like with the sunspot data. This makes it great for tasks where speed and precision matter, such as weather forecasting or small networks. However, it needs a lot of computer power because it calculates big matrices, which slows it down per step compared to backpropagation. For huge networks with tons of weights, this extra effort might not be worth it. The regularization trick ($\lambda \mathbf{I}$) keeps it stable. Overall, Gauss-Newton is good for smaller, tough problems, but simpler methods might suit bigger ones better.

## References


- Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). Prentice-Hall.