Neural Network from Scratch (Mathematical Representation and Pseudocode

Om Patel, Jenis Patel, Vrund Shah
Group: The Sigmas
Professor: Santosh Parajuli

March 23, 2025

Introduction

This document presents a detailed breakdown of a neural network project implemented from scratch. The goal is to understand the underlying mathematics of a multi-layer neural network, including forward propagation, loss function, backpropagation, and optimization using gradient descent. This pseudocode and mathematical formulation are designed for clarity, especially for those who prefer a more theoretical approach rather than code-based explanations.

1 Initialization

In a neural network, we start by initializing the weights and biases for each layer. The number of neurons in each layer is defined in the `layer_dims` list, and the parameters are initialized as follows:

- For each layer l, initialize the weights $W^{[l]}$ and biases $b^{[l]}$ as follows:
  - $W^{[l]}$ is a matrix of size $(n^{[l-1]} \times n^{[l]})$
  - $b^{[l]}$ is a vector of size $(1 \times n^{[l]})$
- Where $n^{[l-1]}$ is the number of neurons in the previous layer and $n^{[l]}$ is the number of neurons in the current layer.

2 Forward Propagation

For each layer, we compute the pre-activation $Z^{[l]}$, which is then passed through an activation function to produce the output $A^{[l]}$.

Mathematical Formulation:

For Layer 1 to L (from input to output):

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

Where:

- $A^{[l-1]}$ is the activation of the previous layer (for the first layer, $A^{[0]} = X$)
- $Z^{[l]}$ is the pre-activation output for layer l
- $W^{[l]}$ is the weight matrix for layer l
- $b^{[l]}$ is the bias vector for layer l

Then apply an activation function $\sigma^{[l]}$ to compute the output $A^{[l]}$:

$$A^{[l]} = \sigma^{[l]}(Z^{[l]})$$

Where $\sigma^{[l]}$ could be sigmoid, ReLU, or any other activation function.

The output from the last layer, $A^{[L]}$, is the final prediction of the neural network.

3 Loss Calculation

For a classification task, we use Cross-Entropy Loss. The cross-entropy loss for a single example is calculated as:

$$L = -\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i)$$

Where:

- y_i is the true probability (from the true labels, one-hot encoded)
- $\hat{y}_i$ is the predicted probability (the output of the neural network)

For the entire dataset with m examples, the total loss is:

$$L = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{[i]} \cdot \log(\hat{y}_j^{[i]})$$

Where:

- $y_j^{[i]}$ is the true label of the j-th class for the i-th sample
- $\hat{y}_j^{[i]}$ is the predicted probability for the j-th class for the i-th sample

4 Backpropagation

In backpropagation, we compute the gradient of the loss function with respect to each parameter (weights and biases). The error term $\delta^{[l]}$ for layer l is computed as follows:

For the output layer L, the error term is:

$$\delta^{[L]} = A^{[L]} - Y$$

Where:

- $A^{[L]}$ is the output of the network (predicted probabilities)
- Y is the true label

For hidden layers $l = L-1, L-2, \ldots, 1$, the error term is propagated backward using:

$$\delta^{[l]} = (W^{[l+1]})^T \cdot \delta^{[l+1]} \cdot \sigma'^{[l]}(Z^{[l]})$$

Where:

- $\sigma'^{[l]}(Z^{[l]})$ is the derivative of the activation function for layer l
- $W^{[l+1]}$ is the weight matrix of the next layer

5 Gradient Calculation

The gradients for weights and biases are computed as follows:

For weights $W^{[l]}$:

$$\frac{\partial L}{\partial W^{[l]}} = \frac{1}{m} \cdot \delta^{[l]} \cdot (A^{[l-1]})^T$$

For biases $b^{[l]}$:

$$\frac{\partial L}{\partial b^{[l]}} = \frac{1}{m} \cdot \sum_{i=1}^{m} \delta_i^{[l]}$$

Where $\delta^{[l]}$ is the error term for layer l, and $A^{[l-1]}$ is the activation of the previous layer (for the first layer, $A^{[0]} = X$).

6 Gradient Descent Update

Finally, we update the weights and biases using the gradients computed during backpropagation. The weight update rule is:

$$W^{[l]} = W^{[l]} - \alpha \cdot \frac{\partial L}{\partial W^{[l]}}$$

The bias update rule is:

$$b^{[l]} = b^{[l]} - \alpha \cdot \frac{\partial L}{\partial b^{[l]}}$$

Where α is the learning rate.

7 Training Algorithm (Pseudocode)

The training process is carried out by performing the following steps iteratively for a fixed number of epochs.

- **Input:** X (features), Y (labels), `layer_dims` (list of neurons in each layer), learning rate, epochs
- **Initialize parameters** W and b for each layer

For epoch = 1 to epochs:

- Perform forward propagation:

$$Z[l] = W[l] \cdot A[l-1] + b[l]$$

$$A[l] = \text{activation}(Z[l]) \quad \text{(activation could be sigmoid, ReLU, etc.)}$$

- Compute the loss L using Cross-Entropy Loss or MSE
- Perform backpropagation:

$$\delta[L] = A[L] - Y$$

$$\delta[l] = (W[l+1])^T \cdot \delta[l+1] \cdot \text{derivative(activation)}(Z[l])$$

- Compute gradients for weights and biases:

$$dW[l] = \frac{1}{m} \cdot \delta[l] \cdot (A[l-1])^T$$

$$db[l] = \frac{1}{m} \cdot \sum(\delta[l])$$

- Update parameters using gradient descent:

$$W[l] = W[l] - \text{learning rate} \cdot dW[l]$$

$$b[l] = b[l] - \text{learning rate} \cdot db[l]$$

Output: Optimized weights W and biases b