MACHINE LEARNING

ASSIGNMENT REPORT

Multi-Layer Perceptron Training & Digits Classification

Student Name: Tran Tung Duong

Student ID: 24022309

Class: K69A - AI1

Hanoi, November 2025

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

VIETNAM NATIONAL UNIVERSITY

Contents

Chapter 1

Problem 1: Multi-Layer Perceptron Training

1.1 Problem Statement

Consider a Multi-Layer Perceptron with:

- Input layer: $x \in \mathbb{R}^d$
- Hidden layers with weights $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ and biases $b_\ell \in \mathbb{R}^{d_\ell}$
- Activation function: $\sigma(z)$ (e.g., ReLU)
- Output layer with softmax activation for multi-class classification
- Loss function: Cross-entropy loss

(1) How to compute the output of layer ℓ through forward propagation?

(2) How to compute the gradient of loss with respect to the output of layer ℓ using the gradient from layer $\ell + 1$?

(3) How to compute the gradient of loss with respect to weights $W_{\ell,a,b}$ and biases $b_{\ell,a}$ of layer ℓ?

(4) How to update parameters W_ℓ and b_ℓ using SGD with batch size 1 and learning rate η?

1.2 Solution

1.2.1 (1) How to compute the output of layer ℓ through forward propagation?

Forward propagation computes the output layer by layer through the network.

For each hidden layer $\ell = 1, 2, \ldots, L-1$:

$$z_\ell = W_\ell a_{\ell-1} + b_\ell \tag{1.1}$$

$$a_\ell = \sigma(z_\ell) \tag{1.2}$$

Apply softmax activation to get output probabilities:

$$a_L^{(a)} = \text{softmax}(z_L^{(a)}) = \frac{\exp(z_L^{(a)})}{\sum_{a'=1}^{C} \exp(z_L^{(a')})} \tag{1.3}$$

The loss function is the cross-entropy loss computed for a single sample:

$$\ell(\mathbf{z}, y) = -\sum_{a=1}^{C} \mathbb{1}[y = a] \log a_L^{(a)} \tag{1.4}$$

Let $\text{MS} = \sum_{a'=1}^{C} \exp(z_L^{(a')})$ (softmax denominator). Then $a_L^{(a)} = \frac{\exp(z_L^{(a)})}{MS}$.

Expanding the loss:

$$\ell(\mathbf{z}, y) = -\sum_{a=1}^{C} \mathbb{1}[y = a] \log \left(\frac{\exp(z_L^{(a)})}{MS} \right) \tag{1.5}$$

$$= -\sum_{a=1}^{C} \mathbb{1}[y = a] \left(z_L^{(a)} - \log(MS) \right) \tag{1.6}$$

$$= -\mathbb{1}[y = a] z_L^{(a)} + \log(MS) \tag{1.7}$$

Now compute the gradient of loss with respect to pre-activation $z_L^{(a)}$:

$$\frac{\partial \ell}{\partial z_L^{(a)}} = -\sum_{a'=1}^{C} \mathbb{1}[y = a'] \frac{\partial}{\partial z_L^{(a)}} z_L^{(a')} + \frac{\partial}{\partial z_L^{(a)}} \log(MS) \tag{1.8}$$

$$= -\mathbb{1}[y = a] + \frac{1}{MS} \frac{\partial}{\partial z_L^{(a)}} \sum_{a'=1}^{C} \exp(z_L^{(a')}) \tag{1.9}$$

$$= -\mathbb{1}[y = a] + \frac{1}{MS} \exp(z_L^{(a)}) \tag{1.10}$$

$$= -\mathbb{1}[y = a] + a_L^{(a)} \tag{1.11}$$

In vector form, let y be the one-hot encoding of true label y (with 1 at position y and 0 elsewhere), and let p be the output vector with components $a_L^{(a)}$:

$$\boxed{\delta_L = \nabla_{z_L} \ell = p - y} \tag{1.12}$$

This shows that the error gradient at the output layer is simply the difference between output probabilities and the true label.

1.2.2 (2) Backpropagation for Hidden Layer

For hidden layers $\ell = L-1, L-2, \ldots, 1$, compute the gradient of loss with respect to the output a_ℓ (post-activation of layer ℓ).

Using the chain rule, the loss depends on a_ℓ through the pre-activation of the next layer:

$$\frac{\partial L}{\partial a_\ell^{(a)}} = \sum_{b=1}^{d_{\ell+1}} \frac{\partial L}{\partial z_{\ell+1}^{(b)}} \frac{\partial z_{\ell+1}^{(b)}}{\partial a_\ell^{(a)}} \tag{1.13}$$

Since $z_{\ell+1}^{(b)} = W_{\ell+1}^{(b,:)} a_\ell + b_{\ell+1}^{(b)}$, we have $\frac{\partial z_{\ell+1}^{(b)}}{\partial a_\ell^{(a)}} = W_{\ell+1}^{(b,a)}$.

Therefore:

$$\frac{\partial L}{\partial a_\ell^{(a)}} = \sum_{b=1}^{d_{\ell+1}} \delta_{\ell+1}^{(b)} W_{\ell+1}^{(b,a)} \tag{1.14}$$

In matrix form:

$$\frac{\partial L}{\partial a_\ell} = W_{\ell+1}^T \delta_{\ell+1} \tag{1.15}$$

where $\delta_{\ell+1}^{(b)} = \frac{\partial L}{\partial z_{\ell+1}^{(b)}}$ is the error term at layer $\ell + 1$.

Now, compute the gradient of loss with respect to the pre-activation z_ℓ:

$$\frac{\partial L}{\partial z_\ell^{(a)}} = \frac{\partial L}{\partial a_\ell^{(a)}} \frac{\partial a_\ell^{(a)}}{\partial z_\ell^{(a)}} \tag{1.16}$$

Since $a_\ell^{(a)} = \sigma(z_\ell^{(a)})$, we have $\frac{\partial a_\ell^{(a)}}{\partial z_\ell^{(a)}} = \sigma'(z_\ell^{(a)})$.

Therefore:

$$\frac{\partial L}{\partial z_\ell^{(a)}} = \sum_{b=1}^{d_{\ell+1}} \delta_{\ell+1}^{(b)} W_{\ell+1}^{(b,a)} \sigma'(z_\ell^{(a)}) \tag{1.17}$$

In vector form, using element-wise multiplication $\odot$:

$$\boxed{\delta_\ell = \nabla_{z_\ell} L = (W_{\ell+1}^T \delta_{\ell+1}) \odot \sigma'(z_\ell)} \tag{1.18}$$

This formula propagates the error gradient from layer $\ell+1$ back to layer ℓ, modulated by the derivative of the activation function.

1.2.3 (2) How to compute the gradient of loss with respect to the output of layer ℓ using the gradient from layer $\ell + 1$?

For layer ℓ, compute the gradient of loss with respect to weights $W_{\ell,a,b}$ and biases $b_{\ell,a}$.

Gradient with respect to weights:

Using the chain rule:

$$\frac{\partial L}{\partial W_{\ell,a,b}} = \frac{\partial L}{\partial z_\ell^{(a)}} \frac{\partial z_\ell^{(a)}}{\partial W_{\ell,a,b}} \tag{1.19}$$

Since $z_\ell^{(a)} = \sum_{b=1}^{d_{\ell-1}} W_{\ell,a,b} a_{\ell-1}^{(b)} + b_{\ell,a}$, we have:

$$\frac{\partial z_\ell^{(a)}}{\partial W_{\ell,a,b}} = a_{\ell-1}^{(b)} \tag{1.20}$$

Therefore:

$$\frac{\partial L}{\partial W_{\ell,a,b}} = \delta_\ell^{(a)} a_{\ell-1}^{(b)} \tag{1.21}$$

In matrix form, the gradient for all weights in layer ℓ is:

$$\boxed{\nabla_{W_\ell} L = \delta_\ell (a_{\ell-1})^T \in \mathbb{R}^{d_\ell \times d_{\ell-1}}} \tag{1.22}$$

where $\delta_\ell \in \mathbb{R}^{d_\ell}$ is the error vector at layer ℓ.

Gradient with respect to biases:

Using the chain rule:

$$\frac{\partial L}{\partial b_{\ell,a}} = \frac{\partial L}{\partial z_\ell^{(a)}} \frac{\partial z_\ell^{(a)}}{\partial b_{\ell,a}} \tag{1.23}$$

Since $z_\ell^{(a)} = \sum_{b=1}^{d_{\ell-1}} W_{\ell,a,b} a_{\ell-1}^{(b)} + b_{\ell,a}$, we have:

$$\frac{\partial z_\ell^{(a)}}{\partial b_{\ell,a}} = 1 \tag{1.24}$$

Therefore:

$$\frac{\partial L}{\partial b_{\ell,a}} = \delta_\ell^{(a)} \tag{1.25}$$

In vector form, the gradient for all biases in layer ℓ is:

$$\boxed{\nabla_{b_\ell} L = \delta_\ell \in \mathbb{R}^{d_\ell}} \tag{1.26}$$

These gradients show that the weight updates are proportional to both the error term δ_ℓ and the input activation $a_{\ell-1}$, while bias updates depend only on the error term.

1.2.4 (3) How to compute the gradient of loss with respect to weights $W_{\ell,a,b}$ and biases $b_{\ell,a}$ of layer ℓ?

Stochastic Gradient Descent updates parameters using gradients computed from a single training sample (batch size = 1). The update rule minimizes the loss by moving

parameters in the direction of the negative gradient.

Weight update:

For each layer $\ell = 1, 2, \ldots, L$, the weights are updated as:

$$W_{\ell,a,b}^{\text{new}} = W_{\ell,a,b}^{\text{old}} - \eta \frac{\partial L}{\partial W_{\ell,a,b}} \tag{1.27}$$

where $\eta > 0$ is the learning rate that controls the step size.

Substituting the gradient from step (3):

$$W_{\ell,a,b}^{\text{new}} = W_{\ell,a,b}^{\text{old}} - \eta \delta_{\ell}^{(a)} a_{\ell-1}^{(b)} \tag{1.28}$$

In matrix form:

$$\boxed{W_{\ell}^{\text{new}} = W_{\ell}^{\text{old}} - \eta \nabla_{W_\ell} L = W_{\ell}^{\text{old}} - \eta \delta_{\ell} (a_{\ell-1})^T} \tag{1.29}$$

Bias update:

Similarly, for each layer $\ell = 1, 2, \ldots, L$, the biases are updated as:

$$b_{\ell,a}^{\text{new}} = b_{\ell,a}^{\text{old}} - \eta \frac{\partial L}{\partial b_{\ell,a}} \tag{1.30}$$

Substituting the gradient from step (3):

$$b_{\ell,a}^{\text{new}} = b_{\ell,a}^{\text{old}} - \eta \delta_{\ell}^{(a)} \tag{1.31}$$

In vector form:

$$\boxed{b_{\ell}^{\text{new}} = b_{\ell}^{\text{old}} - \eta \nabla_{b_\ell} L = b_{\ell}^{\text{old}} - \eta \delta_{\ell}} \tag{1.32}$$

Training algorithm:

The complete SGD update process is:

1. **Initialize** parameters W_ℓ and b_ℓ for all layers $\ell = 1, \ldots, L$
2. **For each epoch**:
   (a) **For each training sample** (x, y):

i. Forward propagation: Compute z_ℓ and a_ℓ for all layers

ii. Compute loss: $L(y, a_L)$

iii. Backpropagation: Compute δ_ℓ for all layers from L to 1

iv. Update: $W_\ell \leftarrow W_\ell - \eta\delta_\ell(a_{\ell-1})^T$ and $b_\ell \leftarrow b_\ell - \eta\delta_\ell$

3. **Repeat** until convergence (typically when loss stops decreasing or after fixed number of epochs)

The learning rate η is a critical hyperparameter: too large values cause divergence, while too small values lead to slow convergence. The SGD update rule with batch size 1 introduces noise that can help escape local minima and improve generalization.

1.3 Conclusion

The training of a Multi-Layer Perceptron is a fundamental technique in deep learning that enables neural networks to learn complex patterns from data. Through systematic application of forward propagation, backpropagation, and parameter updates using stochastic gradient descent, the network iteratively improves its predictions by minimizing the loss function.

The mathematical derivations presented above establish the complete framework for training neural networks: computing network outputs through layers, propagating errors backward to calculate parameter gradients, and updating weights and biases in the direction of the negative gradient. These principles form the foundation of modern deep learning and are essential for understanding how neural networks learn from training data.

The SGD algorithm with its variants (momentum, adaptive learning rates) has enabled the training of very deep networks and remains central to contemporary machine learning practice. Understanding these mathematical foundations is crucial for developing and optimizing neural networks for various applications.

Chapter 2

Problem 2: Digits Classification with Machine Learning

2.1 Problem Statement

Implement a complete machine learning pipeline to classify handwritten digits from the Digits dataset using multiple algorithms: Logistic Regression, Decision Tree, K-Nearest Neighbors, and Artificial Neural Network. Perform hyperparameter tuning and compare the performance of all models.

2.2 Project Overview

This project develops and evaluates four machine learning models for digit classification on the scikit-learn Digits dataset containing 1,797 handwritten digit images with 64 features (8×8 pixels). The dataset is split into training (60%), validation (20%), and test (20%) sets using stratified sampling to ensure balanced class distribution.

The implementation includes:

- Data preprocessing with standardization and encoding
- Hyperparameter optimization using GridSearchCV with 5-fold cross-validation
- Model evaluation on test set with accuracy and per-class recall metrics
- Comparative analysis of model performance and generalization

The four models achieve the following test accuracies:

- Artificial Neural Network (ANN): 97.78%
- K-Nearest Neighbors (KNN): 97.22%
- Logistic Regression (LR): 96.94%
- Decision Tree (DT): 81.11%

2.3 Implementation Details

GitHub Repository:

`https://github.com/TranTungDuong02082006/Digits-Classification-with-ML-Models`

The repository contains the full Jupyter Notebook with all experiments, hyperparameter configurations, model training, and results visualization.