Low-Level Implementation of a DNN and MNIST Classifier

Sihoo Lee

Abstract

This study explores the fundamental principles of Artificial Intelligence by implementing a Deep Neural Network (DNN) from scratch using TypeScript, without relying on high-level deep learning libraries such as PyTorch or TensorFlow. The research covers the mathematical foundations of differentiation, Backpropagation, Gradient Descent, and activation functions (Sigmoid, Softmax), translating these concepts directly into code. The implemented model was tested on the MNIST handwritten digit dataset, achieving an accuracy of approximately 97%. This process aims to provide a comprehensive understanding of the internal mechanisms of AI models.

Contents

1 Introduction

The initial motivation for this research stems from a prior project titled "Contactless UI Implementation using Hand Pose AI," submitted to the Korea Code Fair in 2022. That project involved manipulating on-screen programs by recognizing poses through cosine similarity comparisons of hand position data provided by Google MediaPipe.

The utility of such AI models, which continuously provide real-time coordinates of hands and fingers from webcam image streaming data, was evident in enabling the creation of various applications. Beyond the application of pre-existing models, there was a necessity to investigate how the internal components of AI are constructed and operated.

While foundational knowledge of AI and Deep Learning was acquired through online coursework [1] and the practical implementation of an MNIST classifier using the PyTorch library, a deeper understanding of the internal operations—specifically functions, gradient calculations, and parameter updates encapsulated within such libraries—was required.

Consequently, this study focuses on building a simple DNN from scratch using TypeScript to directly implement an MNIST classifier, thereby elucidating the underlying mechanisms.

2 Mathematical Background

2.1 Differentiation

2.1.1 Definition of Derivative

A derivative, or derived function, is a new function whose range consists of the limits of the ratio of the change in the function value to the change in the independent variable at each point in the domain [2]. From a geometric perspective, differentiation is the problem of finding the tangent line to a given curve.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

2.1.2 Proofs of Differentiation Rules

1. Scalar Multiplication Rule

$$\begin{aligned}(a \cdot f(x))' &= \lim_{h \to 0} \frac{af(x+h) - af(x)}{h} \\ &= a \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \\ &= a \cdot f'(x)\end{aligned}$$

2. Sum Rule

$$\begin{aligned}(f+g)' = \lim_{h \to 0} \frac{1}{h} \Big[&(f(x+h) + g(x+h)) \\ &- (f(x) + g(x)) \Big] \\ &= f'(x) + g'(x)\end{aligned}$$

3. Product Rule

To avoid overlapping text, we expand the fraction terms individually:

$$\begin{aligned}(fg)' &= \lim_{h \to 0} \frac{f(x+h)g(x+h) - f(x)g(x)}{h} \\ &= \lim_{h \to 0} \frac{1}{h} \Big[f(x+h)g(x+h) \\ &\quad - f(x+h)g(x) \\ &\quad + f(x+h)g(x) - f(x)g(x) \Big] \\ &= \lim_{h \to 0} \Big[f(x+h) \frac{g(x+h) - g(x)}{h} \\ &\quad + g(x) \frac{f(x+h) - f(x)}{h} \Big] \\ &= f(x)g'(x) + f'(x)g(x)\end{aligned}$$

4. Chain Rule

This rule handles composite functions.

$$\begin{aligned}(f(g(x)))' &= \lim_{h \to 0} \frac{f(g(x+h)) - f(g(x))}{h} \\ &= \lim_{h \to 0} \Big[\frac{f(g(x+h)) - f(g(x))}{g(x+h) - g(x)} \\ &\quad \times \frac{g(x+h) - g(x)}{h} \Big]\end{aligned}$$

Let $A = g(x)$ and $B = g(x+h)$. As $h \to 0$, $B \to A$.

$$= f'(g(x)) \times g'(x)$$

5. Quotient Rule

$$\left(\frac{f}{g}\right)' = \lim_{h\to 0} \frac{\frac{f(x+h)}{g(x+h)} - \frac{f(x)}{g(x)}}{h}$$
$$= \lim_{h\to 0} \frac{1}{h} \frac{f(x+h)g(x) - f(x)g(x+h)}{g(x+h)g(x)}$$

Subtract & add $f(x)g(x)$ in numerator:

$$= \frac{1}{g(x)^2} \lim_{h\to 0} \frac{1}{h} \Big[f(x+h)g(x) - f(x)g(x) + f(x)g(x) - f(x)g(x+h) \Big]$$
$$= \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

2.2 Definitions and Differentiation of Various Functions

2.2.1 Definition of e and Differentiation of e^x

The mathematical constant e can be defined as follows (approximate value 2.718):

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

This function has the characteristic that its slope (derivative) is equal to its function value at every moment.

Proof:

$$(e^x)' = \lim_{h\to 0} \frac{e^{x+h} - e^x}{h} = e^x \lim_{h\to 0} \frac{e^h - 1}{h}$$

Let $e^h - 1 = t$, then $h = \ln(1+t)$. As $h \to 0$, $t \to 0$:

$$= e^x \lim_{t\to 0} \frac{t}{\ln(1+t)}$$
$$= e^x \lim_{t\to 0} \frac{1}{\ln(1+t)^{1/t}}$$
$$= e^x \cdot \frac{1}{\ln e} = e^x$$

2.2.2 Natural Logarithm Function

The natural logarithm has the base e. It is often written as $\ln(x)$. Since it is the inverse function of the exponential function, its slope is $\frac{1}{x}$.

2.2.3 Sigmoid

Definition: The sigmoid function is a type of activation function [3, 4].

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

It is used in intermediate steps of deep learning and to determine the final result of logistic regression or binary classification.

Differentiation:

$$\sigma'(x) = ((1+e^{-x})^{-1})'$$
$$= -(1+e^{-x})^{-2} \cdot (e^{-x}) \cdot (-1)$$
$$= \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$
$$= \sigma(x)(1-\sigma(x))$$

2.2.4 Softmax

Softmax normalizes multiple input values into a range of 0 to 1. The sum of the output values is 1.

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

2.3 Partial Differentiation

2.3.1 Definition

Partial differentiation is the process of finding how the value of a multivariate function changes as one specific variable changes. Other variables are treated as constants. For example, if $z = f(x,y) = x^2 + xy + y^2$:

$$\frac{\partial f}{\partial x}(x,y) = 2x + y$$

2.3.2 Partial Differentiation of Softmax

Let $S_x = \frac{e^x}{D}$ and $S_y = \frac{e^y}{D}$ (where $D = \sum e^i$). See also derivation of Softmax Loss [5].

Case 1: Same Index (x)

$$\frac{\partial S_x}{\partial x} = \frac{e^x D - e^x e^x}{D^2}$$
$$= \frac{e^x}{D}\left(1 - \frac{e^x}{D}\right)$$
$$= S_x(1 - S_x)$$

Case 2: Different Index (y)

$$\frac{\partial S_x}{\partial y} = \frac{0 \cdot D - e^x e^y}{D^2}$$
$$= -\frac{e^x}{D}\frac{e^y}{D}$$
$$= -S_x S_y$$

3 Artificial Intelligence Background

3.1 Terminology

- **Artificial Intelligence (AI):** A field of science related to building computers and machines capable of reasoning, learning, and acting in ways that typically require human intelligence [6].
- **Machine Learning (ML):** A subset of AI where algorithms learn from data to produce results without being explicitly programmed [7].
- **Deep Learning (DL):** Artificial neural networks (Perceptrons) with multiple layers [8]. Models with more than three layers are called Deep Neural Networks [9].

3.2 Model

Broad Definition: An expression of an algorithm that discovers patterns or performs predictions from data [10]. This includes layer configuration, number/size of hidden layers, activation functions, regularization, etc.

Narrow Definition: The set of weights (W) and biases (b) between layers. Training updates these values to predict answers for new data [11].

3.3 Parameters vs Hyperparameters

- **Parameters:** Internal variables estimated from data (e.g., weights, biases) [12].
- **Hyperparameters:** External configurations set by the developer (e.g., learning rate, batch size, number of epochs).

3.4 Activation Functions

Activation functions allow the model to learn nonlinear data. Without them, stacking multiple linear layers is mathematically equivalent to a single linear layer [4].

- **Sigmoid:** Outputs 0 to 1. Used in logistic regression [3]. Vulnerable to gradient vanishing.
- **ReLU:** Solves gradient vanishing. Gradient is 0 or 1 [13].
- **LeakyReLU:** Allows small updates for negative inputs [14].

3.5 Loss Functions

Loss represents the error between the model's output and the actual target.

- **MSE (Mean Square Error):** Average of squared differences.
- **Cross Entropy (CE):** Uses log functions to penalize wrong probability predictions [15].

3.6 Gradient Descent

A method to adjust parameters to minimize error (Loss).

$$W_{new} = W_{old} - \eta \frac{\partial E}{\partial W}$$

where η is the learning rate.

4 Implementation of MNIST Classifier

4.1 Problem Definition

The goal is to classify 28x28 pixel images of handwritten digits (0-9) from the MNIST dataset [16, 17]. The input is a flattened 784-element array.

4.2 Model Configuration

- **Input Layer:** 784 nodes (+1 bias node).
- **Hidden Layer:** 100 nodes (+1 bias node) [18].
- **Output Layer:** 10 nodes.
- **Architecture:** Linear $\rightarrow$ Sigmoid $\rightarrow$ Linear $\rightarrow$ Softmax.

The bias is handled by appending a value of '1' to the input vector of each layer, simplifying matrix operations [11].

4.3 TypeScript Implementation

4.3.1 Model Class and Initialization

```
class Model {
    ih: number[][] = []; // Weights Input -> Hidden
    hz: number[][] = []; // Weights Hidden -> Output

    init() {
        // Input -> Hidden Weights Initialization
        for (let i = 0; i < INPUT_DIM + 1; ++i) {
            this.ih[i] = [];
            for (let hi = 0; hi < H_DIM; ++hi) {
                // Initialize weights between -0.5 and 0.5
                this.ih[i][hi] = Math.random() - 0.5;
            }
        }
        // Initialize Bias for Hidden Layer
        this.ih[INPUT_DIM] = [];
        for (let hi = 0; hi < H_DIM; ++hi) {
            this.ih[INPUT_DIM][hi] = 0;
        }

        // Hidden -> Output Weights Initialization
        this.hz[H_DIM] = [];
        for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
            this.hz[H_DIM][zi] = 0; // Bias initialization
        }
        for (let hi = 0; hi < H_DIM; ++hi) {
            this.hz[hi] = [];
            for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
                this.hz[hi][zi] = Math.random() - 0.5;
            }
        }
    }
}
```

Listing 1: Model Class Implementation

4.3.2 Forward Function

The forward pass calculates predictions. The bias handling (adding 1) is integrated.

```
forward(data: Data, from: number, to: number) {
    for (let di = from; di <= to; ++di) {
        // 1. Input -> Hidden
        for (let hi = 0; hi < H_DIM; ++hi) {
            data.h[di][hi] = data.x[di].reduce(
                (prev, cur, i) => prev + cur * this.model.ih[i][hi], 0
            );
            data.hs[di][hi] = sigmoid(data.h[di][hi]); // Activation
        }

        // 2. Hidden -> Output
        for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
            data.z[di][zi] = data.hs[di].reduce(
                (prev, cur, hsi) => prev + cur * this.model.hz[hsi][zi], 0
            );
        }

        // 3. Apply Softmax
        const denominator = denominatorOfSoftmax(data.z[di]);
        for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
            data.out[di][zi] = softmax(data.z[di][zi], denominator);
        }
    }
}
```

Listing 2: Forward Propagation

4.3.3 Backward Function

Calculates gradients. The derivative of Softmax + Cross Entropy simplifies to ‘output - target‘.

```
// 1. Calculate Error at Output Layer (Softmax + CE derivative)
for (let di = from; di <= to; ++di) {
    const answer = this.trainingData.y[di];
    for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
        const outI = this.trainingData.out[di][zi];
        const yI = zi === answer ? 1 : 0; // One-hot encoding logic
        this.trainingData.pepz[di][zi] = outI - yI; // Simplified derivative
    }
}

// 2. Backpropagate to Hidden Layer
for (let di = from; di <= to; ++di) {
    for (let hsi = 0; hsi < H_DIM; ++hsi) {
        let temp = 0;
        for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
            temp += this.trainingData.pepz[di][zi] * this.model.hz[hsi][zi];
        }
        this.trainingData.pephs[di][hsi] = temp;
        // Multiply by derivative of Sigmoid: h * (1 - h)
        this.trainingData.peph[di][hsi] =
            this.trainingData.pephs[di][hsi] * this.trainingData.hs[di][hsi] * (1 -
    this.trainingData.hs[di][hsi]);
    }
}
```

Listing 3: Backward Propagation

4.3.4 Step Function (Parameter Update)

Updates parameters using Gradient Descent with a fixed learning rate.

```
step(from: number, to: number) {
    const dataSize = to - from + 1;

    // Update Hidden -> Output Weights
    for (let hsi = 0; hsi < H_DIM + 1; ++hsi) { // +1 for Bias
        for (let zi = 0; zi < OUTPUT_DIM; ++zi) {
            let gradient = 0;
            for (let di = from; di <= to; ++di) {
                gradient += this.trainingData.pepz[di][zi] * this.trainingData.hs[di
    ][hsi];
            }
            // Update weight using Learning Rate
            this.model.hz[hsi][zi] -= LEARNING_RATE * (gradient / dataSize);
        }
    }

    // Update Input -> Hidden Weights
    for (let i = 0; i < INPUT_DIM + 1; ++i) { // +1 for Bias
        for (let hi = 0; hi < H_DIM; ++hi) {
            let gradient = 0;
            for (let di = from; di <= to; ++di) {
                gradient += this.trainingData.peph[di][hi] * this.trainingData.x[di][
    i];
            }
            this.model.ih[i][hi] -= LEARNING_RATE * (gradient / dataSize);
        }
    }
}
```

Listing 4: Gradient Descent Step

5 Results

5.1 Hyperparameters

Parameter	Value
INPUT_DIM	784
H_DIM	100
OUTPUT_DIM	10
BATCH_SIZE	200
N_EPOCHS	1000
LEARNING_RATE	0.1

5.2 Performance Analysis

Loss and Accuracy: As epochs increased, the Training Loss decreased consistently. However, the Validation Loss reached its minimum at epoch 513 and then began to rise, indicating **Overfitting**.

The final accuracy stabilized at approximately **97%**. From 200 epochs (96.34%) onwards, the improvement in accuracy was minimal.

6 Conclusion

This study successfully implemented a Deep Neural Network (DNN) with high accuracy (97%) using TypeScript, without reliance on high-level libraries such as PyTorch [19]. The process provided significant insight into the mathematical underpinnings and layer management of AI models, which are typically abstracted by standard libraries.

Additionally, the research served to verify the principles utilized in previous projects, such as the MediaPipe-based interface. Having established a verified low-level implementation, future work will involve leveraging standard AI libraries to investigate more complex architectures, including CNNs, RNNs, and GANs, to address real-world problems.

References

[1] Fast Campus, "Deep learning article." https://fastcampus.co.kr/story_article_dl. Accessed: 2023.

[2] Wikipedia, "Derivative." https://ko.wikipedia.org/wiki/. Accessed: 2023.

[3] E. W. Weisstein, "Sigmoid function." https://mathworld.wolfram.com/SigmoidFunction.html. Accessed: 2023.

[4] V7 Labs, "Neural networks activation functions." https://www.v7labs.com/blog/neural-networks-activation-functions. Accessed: 2023.

[5] "Derivation of softmax loss (multi-class classification differentiation)." https://www.youtube.com/watch?v=znqbtL0fRA0. Accessed: 2023.

[6] Google Cloud, "What is artificial intelligence?." https://cloud.google.com/learn/what-is-artificial-intelligence?hl=ko. Accessed: 2023.

[7] Wikipedia, "Machine learning." https://ko.wikipedia.org/wiki/_¯. Accessed: 2023.

[8] NVIDIA Blog, "Difference between ai, machine learning, and deep learning." https://blogs.nvidia.co.kr/2016/08/03/difference_ai_learning_machinelearning/, 2016. Accessed: 2023.

[9] Simplilearn, "Deep learning algorithm." https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm. Accessed: 2023.

[10] NVIDIA Blog, "What is a machine learning model?." https://blogs.nvidia.co.kr/2021/08/24/what-is-a-machine-learning-model/, 2021. Accessed: 2023.

[11] IncludeHelp, "Uni-layer neural network." https://www.includehelp.com/python/uni-layer-neural-network.aspx. Accessed: 2023.

[12] J. Brownlee, "Difference between a parameter and a hyperparameter." https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/, 2017. Accessed: 2023.

[13] InsideAIML, "Relu activation function." https://insideaiml.com/blog/ReLUActivation-Function-1026. Accessed: 2023.

[14] Deep Learning University, "Leaky relu as an activation function." https://deeplearninguniversity.com/leaky-relu-as-an-activation-function-in-neural-networks/. Accessed: 2023.

[15] AndroidKT, "Choose cross entropy loss function in keras." https://androidkt.com/choose-cross-entropy-loss-function-in-keras/. Accessed: 2023.

[16] Kaggle, "Mnist dataset." https://www.kaggle.com/. Accessed: 2023.

[17] ResearchGate, "Architecture of convolutional neural network trained for mnist." https://www.researchgate.net/figure/Architecture-of-our-Convolutional-Neural-Network-trained-for-the-MNIST-dataset-The-upper_fig2_321376573. Accessed: 2023.

[18] WikiDocs, "Perceptron." https://wikidocs.net/24958. Accessed: 2023.

[19] Analytics Vidhya, "Understanding rnn step by step with pytorch." https://www.analyticsvidhya.com/blog/2021/07/understanding-rnn-step-by-step-with-pytorch/, 2021. Accessed: 2023.