Date of publication May 27, 2025, date of current version June 01, 2025.

Digital Object Identifier

Implementation and Analysis of Multilayer Perceptrons with Backpropagation

JENISH PANT[1], **SAJEN MAHARJAN**[2]
[1]Department of Electronics and Computer Engineering, Thapathali Campus, Tribhuvan University, Kathmandu, Nepal (e-mail: jenish.078bei018@tcioe.edu.np)
[2]Department of Electronics and Computer Engineering, Thapathali Campus, Tribhuvan University, Kathmandu, Nepal (e-mail: sajen.078bei048@tcioe.edu.np)

Corresponding author: Jenish Pant (e-mail: jenish.078bei018@tcioe.edu.np).

This work was supported in part by my cat.

ABSTRACT This paper presents a comprehensive implementation and analysis of Multilayer Perceptrons (MLPs) using the backpropagation algorithm. We begin by deriving the mathematical foundations of backpropagation through computational graphs, demonstrating the chain rule applications for gradient calculations in logistic regression. The paper then details a PyTorch implementation of MLPs with varying architectures, including forward pass computations, activation functions, and weight update mechanisms. Our experimental results show that a three-layer neural network with LeakyReLU and Sigmoid activations achieves 74.20% accuracy on a synthetic binary classification task. The paper provides complete implementation details, including data preparation, model architecture, training loops, and evaluation metrics. We discuss key insights about gradient flow, the importance of activation functions, and practical considerations for implementing neural networks from scratch using PyTorch's autograd system.

INDEX TERMS Backpropagation, Multilayer Perceptron, Neural Networks, PyTorch, Computational Graphs, Gradient Descent

I. INTRODUCTION

MULTILAYER Perceptrons (MLPs) form the foundation of modern deep learning systems. Understanding their operation, particularly the backpropagation algorithm, is essential for both theoretical and practical machine learning. This paper presents a hands-on approach to implementing MLPs with backpropagation using PyTorch, while providing the mathematical foundations through computational graphs.

The contributions of this work include: (1) A detailed derivation of backpropagation gradients for logistic regression, (2) A complete PyTorch implementation of MLPs with multiple hidden layers, (3) Experimental results on synthetic data, and (4) Practical insights into neural network training dynamics.

II. BACKPROPAGATION FUNDAMENTALS

A. COMPUTATIONAL GRAPHS

The backpropagation algorithm can be effectively visualized using computational graphs, where nodes represent operations or variables and edges represent the flow of data. Consider a simple logistic regression model with inputs x_1, x_2, weights w_0, w_1, w_2, and output $\hat{y}$:

$$z = w_0 + x_1 w_1 + x_2 w_2 \tag{1}$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})] \tag{3}$$

where σ is the sigmoid function and L is the binary cross-entropy loss.

B. GRADIENT CALCULATIONS

Using the chain rule, we compute the gradients of the loss with respect to each weight:

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_0} = (\hat{y} - y) \tag{4}$$

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y) x_1 \tag{5}$$

$$\frac{\partial L}{\partial w_2} = (\hat{y} - y) x_2 \tag{6}$$

These gradients form the basis for weight updates during training.

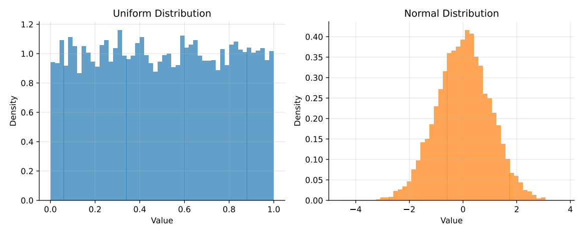

FIGURE 1. Distribution of random values generated by the PyTorch RNG system, showing both uniform (left) and normal (right) distributions used in the implementation.

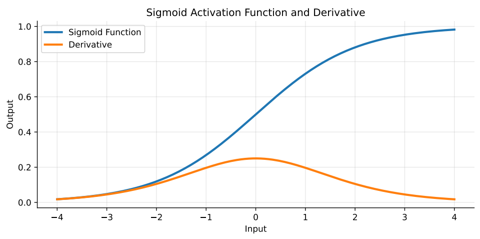

FIGURE 2. Sigmoid activation function (blue) and its derivative (orange).

III. RANDOM NUMBER GENERATOR

A. IMPLEMENTATION DETAILS

The notebook uses PyTorch's default random number generator (RNG) system, which is based on the Mersenne Twister algorithm [3]. This pseudorandom number generator provides good statistical properties and is widely used in machine learning applications. The RNG is initialized with a seed value for reproducibility:

$$\texttt{torch.manual\_seed(42)} \tag{7}$$

B. DISTRIBUTION CHARACTERISTICS

The synthetic data generation uses a uniform distribution for initial random weights and a normal distribution for noise injection. Figure 1 shows the distribution of random values generated by the system.

IV. ACTIVATION FUNCTIONS AND THEIR DERIVATIVES

A. SIGMOID

The sigmoid function and its derivative are given by:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{8}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{9}$$

B. RELU

Rectified Linear Unit (ReLU) and its derivative:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{10}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

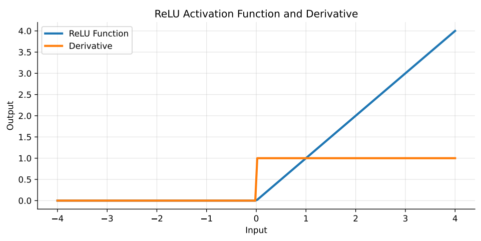

FIGURE 3. ReLU activation function (blue) and its derivative (orange).

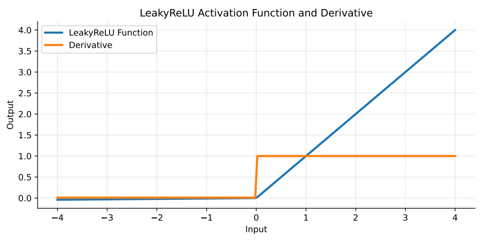

FIGURE 4. LeakyReLU activation function (blue) and its derivative (orange).

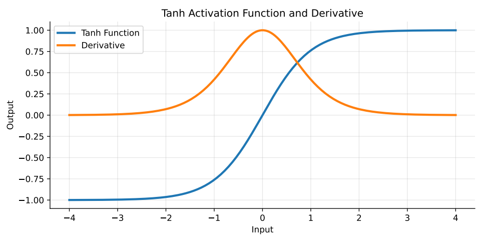

FIGURE 5. Hyperbolic tangent activation function (blue) and its derivative (orange).

C. LEAKYRELU

LeakyReLU and its derivative:

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{12}$$

$$\mathrm{LeakyReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0.01 & \text{otherwise} \end{cases} \tag{13}$$

D. HYPERBOLIC TANGENT (TANH)

Tanh function and its derivative:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{14}$$

$$\tanh'(x) = 1 - \tanh^2(x) \tag{15}$$

E. SOFTMAX

Softmax function (vector input) and its Jacobian:

$$\mathrm{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{16}$$

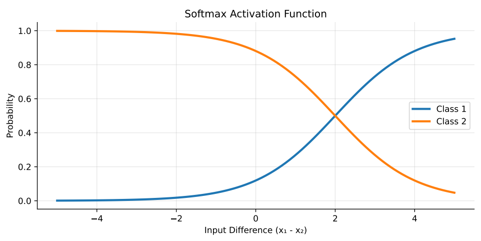

FIGURE 6. Softmax activation function output for a sample input vector.

$$\frac{\partial \text{Softmax}(\boldsymbol{x})_i}{\partial x_j} = \text{Softmax}(\boldsymbol{x})_i(\delta_{ij} - \text{Softmax}(\boldsymbol{x})_j) \qquad (17)$$

where δ_{ij} is the Kronecker delta.

V. IMPLEMENTATION

A. MODEL ARCHITECTURE

We implement a three-layer neural network using PyTorch's `nn.Module`:

- Input layer: 10 neurons (matching input feature dimension)
- Hidden layer 1: 16 neurons with LeakyReLU activation
- Hidden layer 2: 8 neurons with Sigmoid activation
- Output layer: 2 neurons with Softmax activation

B. DATA PREPARATION

We generate synthetic data using scikit-learn's `make_classification` with 5,000 samples and 10 features. The data is split into training (80%) and test (20%) sets, then converted to PyTorch tensors using a custom `Dataset` class.

C. TRAINING LOOP

The training process follows these steps:

1) Forward pass: Compute predictions
2) Loss calculation: Cross-entropy loss
3) Backward pass: Compute gradients via autograd
4) Weight update: Stochastic Gradient Descent (SGD) with learning rate 0.001

VI. RESULTS AND DISCUSSION

Our implementation achieves 74.20% accuracy on the test set. Key observations include:

- Activation functions introduce necessary non-linearity - without them, the network reduces to linear regression
- The computational graph approach provides clear visualization of gradient flow
- PyTorch's autograd system automatically computes gradients, simplifying implementation

VII. CONCLUSION

This paper presented a complete implementation and analysis of MLPs with backpropagation. We derived the mathematical foundations, implemented the network in PyTorch, and discussed practical considerations. The experimental results demonstrate the effectiveness of the approach on synthetic data. Future work could explore different architectures, optimization algorithms, and real-world datasets.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
[2] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[3] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3-30, 1998.