Neural Network from Scratch: XOR Classification

Samarth Tripathi
Email: samarth.tripathi.edu@gmail.com

***Abstract*—This project demonstrates the construction and training of a fully functional neural network from scratch using only NumPy. The network is applied to a synthetic 2D XOR classification problem. Phases include dataset preparation, forward pass computation, backpropagation, weight updates via gradient descent, and evaluation of learning dynamics. Key contributions include detailed phase-wise implementation, rigorous derivation of equations, and visualization of training convergence and decision boundaries. The work emphasizes understanding neural network fundamentals, not merely using high-level libraries. Limitations include the use of a small synthetic dataset, a shallow architecture, and vanilla SGD without advanced optimizers.**

I. Introduction

Neural networks are foundational to modern machine learning and artificial intelligence. The XOR classification problem is a canonical non-linearly separable problem that requires hidden layer representations to be solved. This project builds a neural network entirely from scratch using NumPy, illustrating key concepts including forward propagation, backpropagation, gradient descent, and training dynamics. Unlike high-level frameworks, this implementation emphasizes fundamental understanding and clear visualization of network learning.

II. Methodology

A. Phase 0: Dataset Preparation and Parameter Initialization

We generate a synthetic 2D XOR dataset with 200 samples drawn from a standard normal distribution:

$$X \sim \mathcal{N}(0, I_2), \quad Y = \begin{cases} 1 & x_1 \cdot x_2 \geq 0 \\ 0 & x_1 \cdot x_2 < 0 \end{cases} \tag{1}$$

Each sample has 2 input features and belongs to one of 2 classes. Network parameters (weights and biases) are initialized as:

$$W^{[l]} \sim \mathcal{N}(0, 0.01^2), \quad b^{[l]} = 0 \tag{2}$$

for each layer l.

B. Phase 1: Forward Pass

The network consists of:

- Input layer: 2 neurons
- Hidden layer: 4 neurons, ReLU activation
- Output layer: 1 neuron, Sigmoid activation

Forward propagation equations:

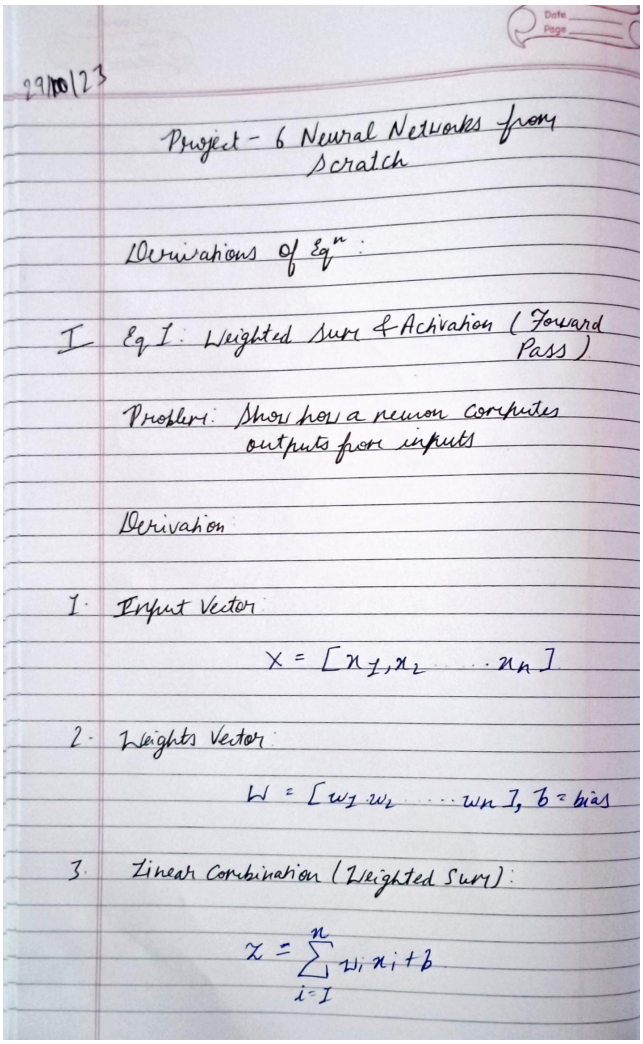


Fig. 1: Forward pass equations (pre-activation and activation).

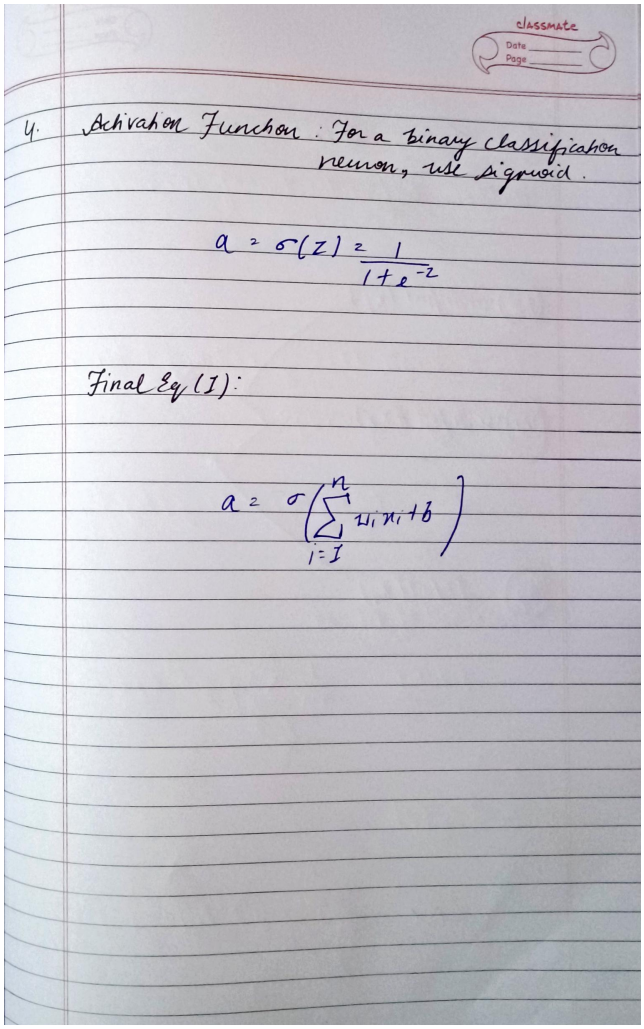


Fig. 2: Continuation of forward pass derivation.

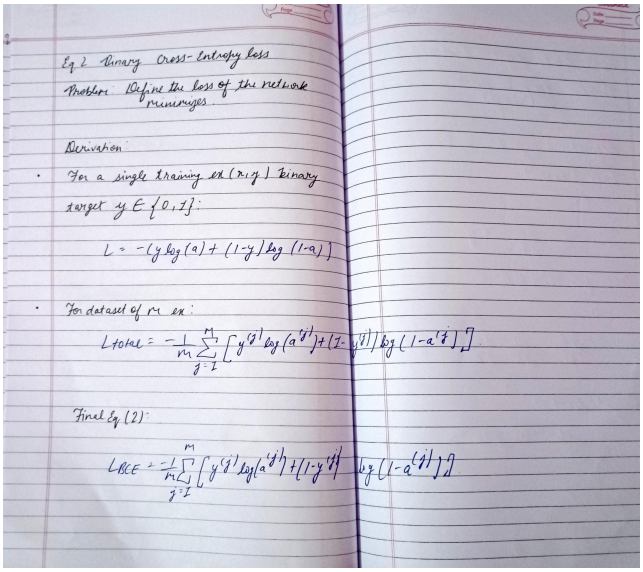


Fig. 3: Binary cross-entropy loss and gradient derivation.

C. Phase 2: Backpropagation

Gradients are computed using the chain rule for both output and hidden layers.Equations are provided in Figures 3, 4, and 5.

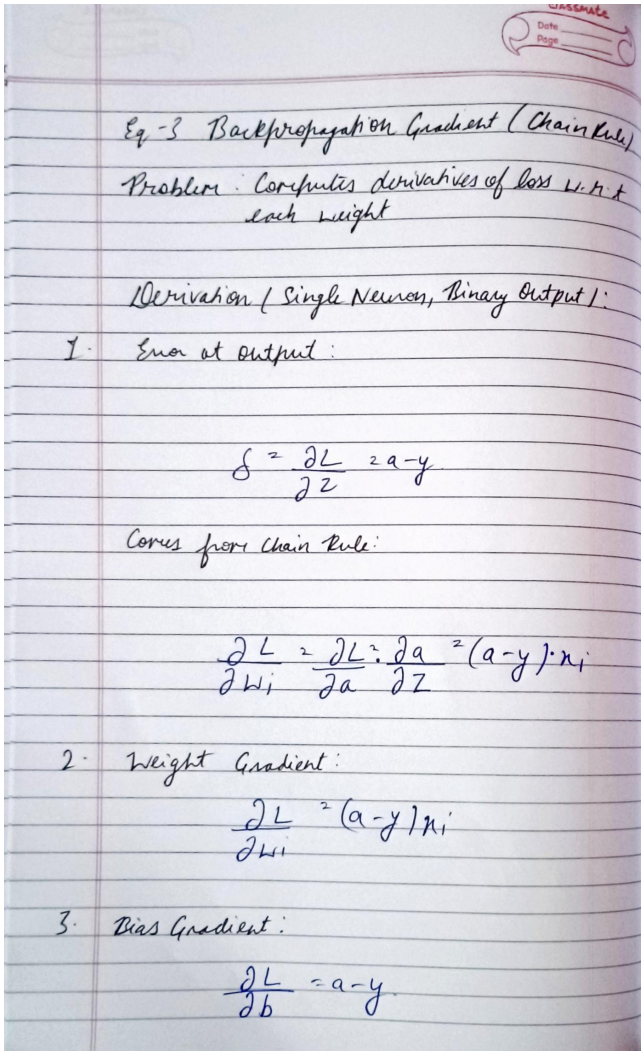


Fig. 4: Backpropagation gradient computation (output and hidden layers).

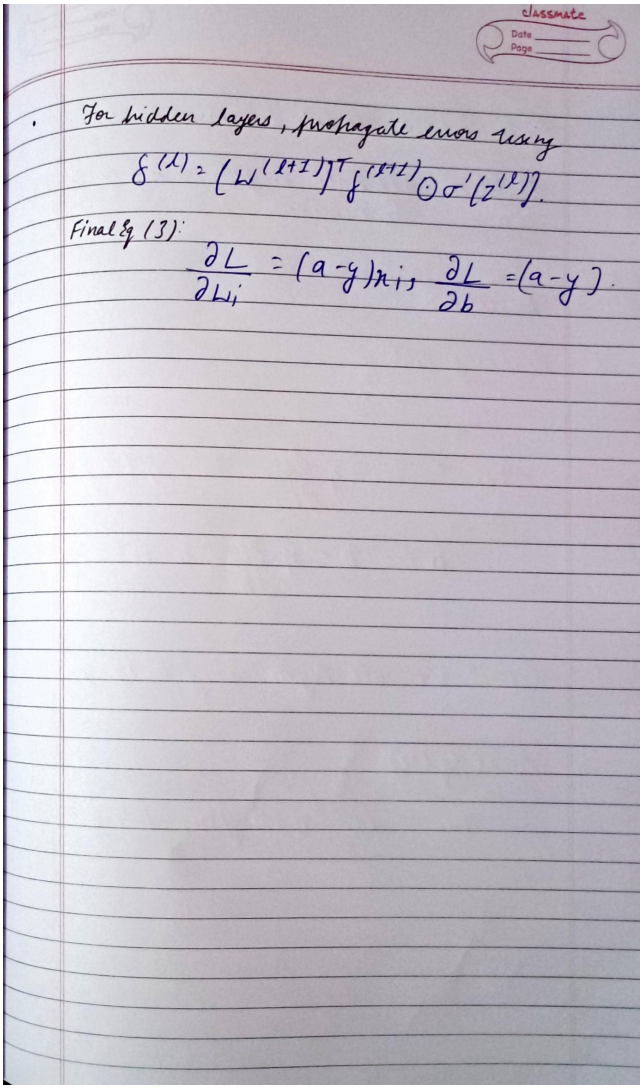


Fig. 5: Continuation of backpropagation derivation.

D. Phase 3: Weight Update (Gradient Descent)

Weights are updated using:

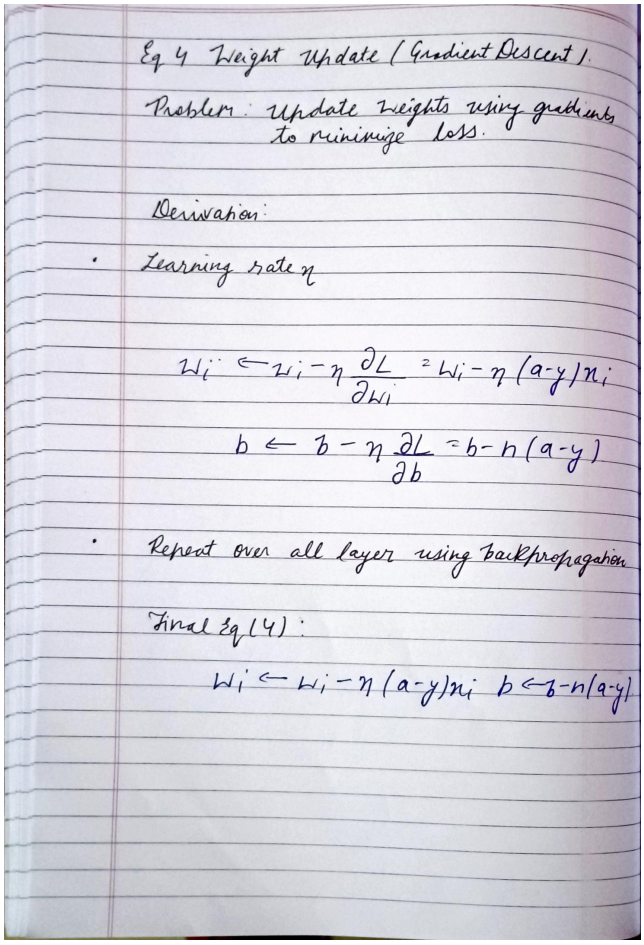
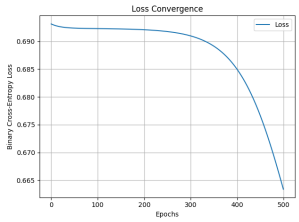


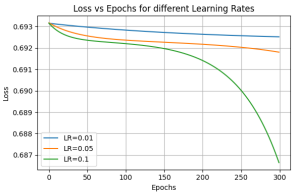
Fig. 6: Gradient descent weight update equations.

E. Phase 4: Training Dynamics

Training was performed for 500 epochs using full-batch SGD with learning rate $\eta = 0.1$.



(a) Loss convergence across epochs.

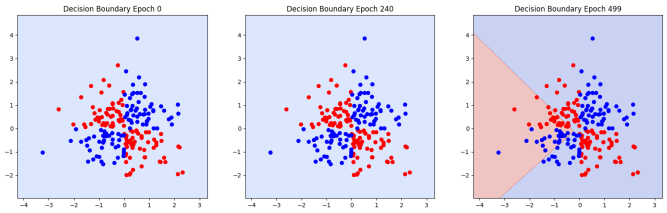


(b) Hyperparameter learning rate comparison.

Fig. 7: Training dynamics visualizations.

F. Phase 5: Decision Boundary Visualization

Decision boundaries show how the network separates the XOR classes.



(a) Epoch 0 (b) Epoch 240 (c) Epoch 499 (final)

Fig. 8: Decision boundary snapshots at key epochs.

III. Experimental Results

The network successfully learns the XOR pattern, achieving **99.5% classification accuracy** on the 200-sample XOR dataset (199/200 correct predictions). Loss converges smoothly, and decision boundaries evolve to correctly separate the two classes.

IV. Limitations

- Vanilla SGD without momentum or adaptive optimizers.
- Full-batch updates prevent stochastic gradient exploration.
- Small synthetic dataset (200 points) not representative of real-world problems.
- Shallow architecture; unsuitable for deep hierarchical features.
- No regularization (L2 penalty, dropout, or batch normalization).
- Furthermore, near-perfect accuracy on a synthetic deterministic dataset does not generalise to real-world classification tasks, where irreducible stochastic noise fundamentally limits achievable accuracy regardless of model architecture.

V. Conclusion

This project demonstrates a fully functional neural network from scratch using NumPy. It covers dataset preparation, forward and backward passes, gradient-based updates, and visualization of learning dynamics. The phase-wise breakdown and detailed equations provide a rigorous, portfolio-ready presentation.

Acknowledgments

The author thanks the open-source NumPy community and XOR problem references for guidance.

References

[1] Harris, C.R., Millman, K.J., van der Walt, S.J., et al., "Array programming with NumPy," *Nature*, 585, 357–362 (2020).
[2] Minsky, M., Papert, S., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.