Neural Network Implementation on Medical Appointment No-Show Dataset

U. Ravi Kiran, 23074031

June 1, 2025

1 Abstract

This report is a study of NeuralNetwork implementations using both NumPy and PyTorch on the Medical Appointment No-Show dataset. The models are evaluated based on training time, classification accuracy, F1 Score, PR-AUC, memory consumption and confusion matrix. The purpose is to analyze efficiency, resource usage and prediction quality especially in the context of imbalanced datasets.As it is a highly imbalanced data and told that we should not use any methods which balances data.

2 Introduction

This study focuses on predicting whether patients will miss their medical appointments using a real world dataset. We implement a NeuralNetwork using NumPy and compare its performance on a PyTorch model as it uses tensors. Our goal is to find out learning behavior and computational efficiency and practical usability.

3 Dataset and Preprocessing

The Medical Appointment No-Show dataset contains patient attributes like Age,Gender,Appointment date,Scheduled date and medical conditions and indicating whether a patient missed their appointment or not.It seems that message only sent if gaps between the Scheduled and Appointed is more than 3.

Preprocessing steps:

- Encoded categorical columns using label encoder.
- For getting DaysGap we subtracted Appointed form scheduled but converting object to date field.
- For Scholarship column if eligible then 1 if not we made it as -1.
- And in Handcap column also we have values greater than 1 we made them to 1.
- I left Age = -1 columns as it may possible for pre bookings as mentioned in the discussions in kaggle about this data set.
- And I did standard normalization which we subtract mean and divide by standard deviation.
- Applied identical preprocessing for both implementations.

4 Methodology

4.1 NumPy based NeuralNetwork

Implemented all neural network components from scratch like weight initialization, forward propagation, backpropagation, loss calculation (binary cross entropy) and gradient descent updates. This version offers full transparency into the training process but is complex and uses more memory.

4.2 PyTorch based NeuralNetwork

Built a functionally equivalent model using PyTorch with the same architecture and optimization strategy .Took advantage of PyTorch's "Autograd" for efficient backpropagation and tensor operations which reduces both training time and memory consumption.

5 Results

Evaluation Metrics Comparison

Table 1: Comparison Between NumPy and PyTorch Models

Metric	**NumPy Model**	**PyTorch Model**
Training Time (s)	709.69	363.41
Accuracy	0.7993	0.7982
F1 Score	0.0000	0.0317
PR-AUC	0.2833	0.3459
Memory Usage (MB)	103.07	11.09
Confusion Matrix	$\begin{bmatrix} 17669 & 0 \\ 4437 & 0 \end{bmatrix}$	$\begin{bmatrix} 17573 & 96 \\ 4364 & 73 \end{bmatrix}$

6 Analysis and Discussion

1. Accuracy and Predictive Quality:
While both models yield similar accuracy ∼79.9%, the PyTorch model demonstrates a Non zero F1 Score and better PR-AUC (0.3459 v/s 0.2833). This means it could identify at least a small subset of patients who missed their appointments, whereas the NumPy model predicted only the majority class which is showed up means No-show no resulting in true positives or it is only giving output 0.

2. Confusion Matrix Interpretation:
The NumPy model is completely biased towards predicting "No-show = No" for all instances as it is a highly imbalanced dataset leading to the above confusion matrix. The PyTorch model however detected a few positive samples "No-show = Yes" indicating slightly better generalization.

3. Memory Usage Difference:
A significant gap is observed in memory usage:

- **NumPy:** 103.07 MB
- **PyTorch:** 11.09 MB

This difference arises from several factors:

- NumPy operations were manually implemented and often used arrays which consume more memory.
- Lack of inplace operations and no memory reuse in manual implementation.
- PyTorch benefits from efficient memory management, inplace updates (using '.data' or '.detach()'), and avoids storing unnecessary tensors.
- PyTorch automatically frees memory used in previous computations through its Computation Graph and Autograd engine.

4. Time Efficiency:
The PyTorch model is roughly 2x faster. This speed gain is attributed to:

- Vectorized and optimized low-level CUDA operations.

- Efficient data batching and parallel computation.
- PyTorch's computational graph avoids redundant calculations.

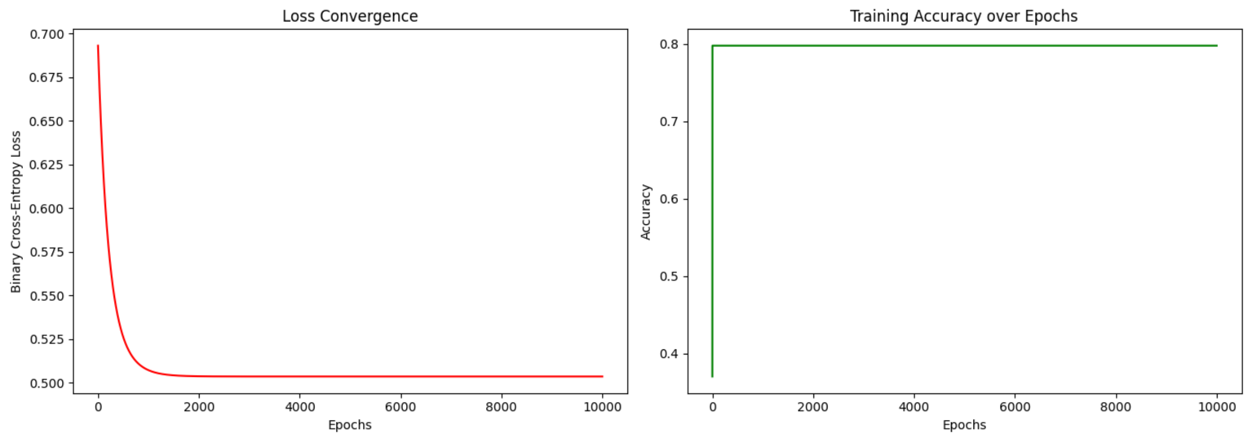

Figure 1: loss convergence and accuracy over epochs for Numpy implementation

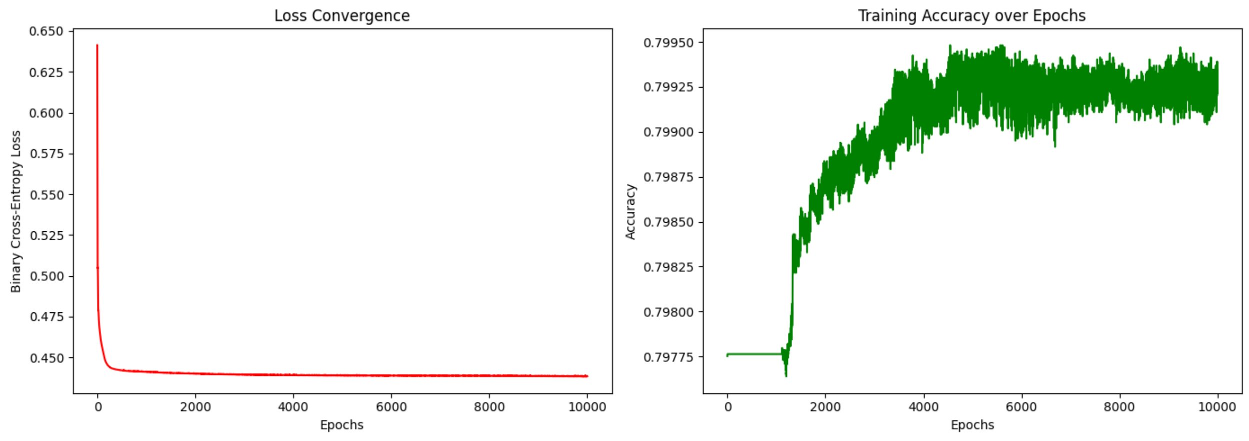

Figure 2: loss convergence and accuracy over epochs for PyTorch implementation

7 Conclusion

This comparative study gives that While NumPy implementations good for foundational understanding they are neither efficient nor easy. PyTorch on the other hand offers better performance both in terms of memory and time and allows for more complex data. The primary challenge remains the extreme class imbalance which must be addressed to achieve reliable prediction outcomes. Future directions include oversampling and undersampling or use of weighted loss functions.