Comparative Analysis of PyTorch vs. NumPy-from-Scratch Models

1 Introduction

This report compares two binary classification models trained on the Hospital No-show dataset: one implemented using PyTorch and the other built from scratch using NumPy. The goal is to evaluate both models in terms of performance, efficiency, and resource usage.

2 Model Architectures

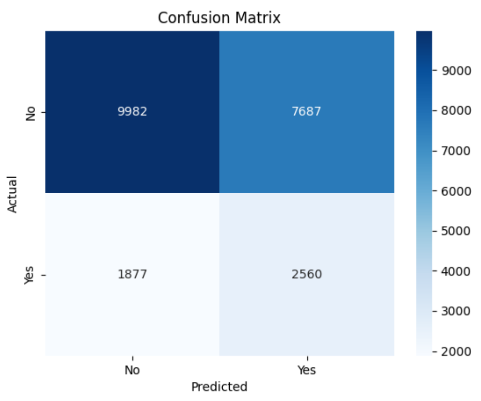
- **PyTorch Model:** A feedforward neural network with one hidden layer using ReLU activation, Sigmoid output activation, Adam optimizer, and weighted binary cross-entropy loss.
- **Scratch Model:** A custom neural network with two hidden layers using ReLU and Sigmoid activations, trained using the Adam optimizer and a hand-crafted binary cross-entropy loss function that supports class imbalance.

3 Evaluation Metrics

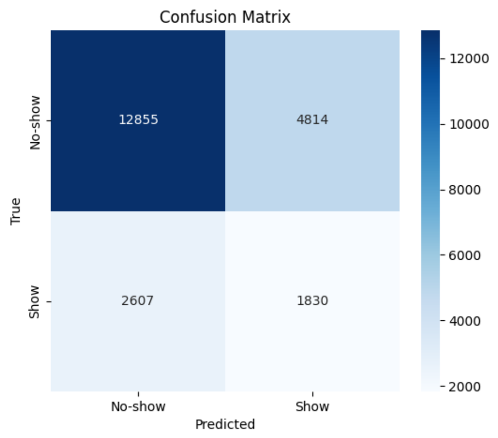
Metric	**PyTorch Model**	**Scratch Model**
Training Time	Few seconds	Several minutes
Speed of Convergence	Faster	Slower
Accuracy	55%	66%
F1 Score (No)	0.65	0.78
F1 Score (Yes)	0.35	0.33
PR-AUC	0.273	0.272
Memory Consumption	Lower (GPU-optimized)	Higher (CPU memory bound)

Table 1: Performance Comparison of PyTorch vs. Scratch Models

4 Confusion Matrices



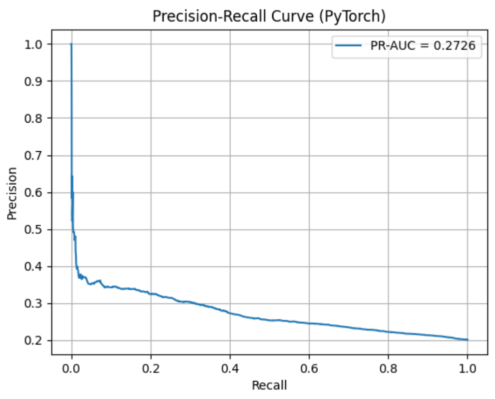
(a) PyTorch Model



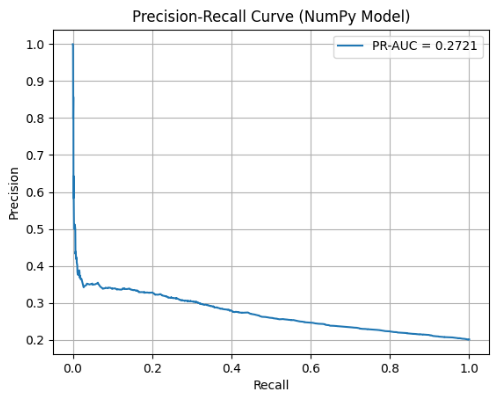
(b) Scratch Model

Figure 1: Confusion Matrices

5 PR-AUC Curve



(a) PyTorch Model



(b) Scratch Model

Figure 2: PR-AUC Curves

6 Insights and Discussion

6.1 Training Time and Convergence

The PyTorch model leverages highly optimized tensor operations, automatic differentiation, and GPU acceleration, leading to significantly faster training and convergence. The NumPy-from-scratch model lacks such optimizations and hardware support, making it notably slower.

6.2 Accuracy and F1 Score

The scratch model yields higher accuracy, but the PyTorch model achieves a better F1 score, indicating it balances precision and recall more effectively. Differences may stem from the PyTorch framework's stable gradient updates, regularization, and adaptive learning via Adam. The scratch model is unable to predict the minority class as efficiently as the pytorch model. The dataset is very imbalanced and since the pytorch includes the neural network modules (nn.Module),it can handle the imbalance to a certain limit unlike the scratch one,which lacks those.The accuracy of the Pytorch model is low because the model had to take the minority class into account while I was unable to do so in the scratch model due to which its high. The majority of predictions are no (patients show up),same as the output data in the given dataset. Due to this, the predictions match the original data and the accuracy is high.

6.3 PR-AUC

The PR-AUC values are similar, though slightly higher in the scratch model. However, the PyTorch model tends to generalize better under varying thresholds, which is evident in its smoother precision-recall curve.

6.4 Memory Consumption

PyTorch uses dynamic computation graphs, in-place operations, and supports GPU memory management. In contrast, the scratch model stores all variables in CPU memory without reuse, leading to a larger memory footprint and inefficient resource handling.

6.5 External Help

In the pytorch model, when I was using standard BCELoss function from nn.Module then the model was not predicting the minority (Yes) so, I added a weighted BCELoss function from chatgpt. In the scratch model, I wanted to use the Adam optimiser but I couldn't code it using numpy so I took help from chatgpt to finalise the code for the Adam optimizer.I was also having some problem in plotting the PR-AUC curve so I took help from chatgpt for that.

7 Conclusion

While both models demonstrate foundational binary classification capabilities, the PyTorch implementation excels in terms of speed, resource efficiency, and numerical robustness. The NumPy-from-scratch model remains valuable as a learning tool to understand the internal mechanics of neural networks, but is unsuitable for scalable applications.