COPS Summer of Code (CSOC) 2025

Artificial Neural Networks

Neural Network Comparison

Yash Bankar
IIT (BHU) Varanasi
Roll Number: 24034017

June 1, 2025

Objective

The aim of this project was to implement and compare two neural network models for predicting patient no-shows in medical appointments:

- A Vanilla Neural Network implemented from scratch using NumPy.
- A Neural Network built using PyTorch.

Both models were trained and evaluated on the same dataset and analyzed using metrics such as Accuracy, F1 Score, PR-AUC, and Confusion Matrix.

Dataset

We used the **Medical Appointment No-show** dataset from Kaggle. The target variable `No-show` was binary encoded as:

- 0 = Patient showed up
- 1 = Patient did not show up

The dataset is highly imbalanced: approximately 80% show-ups and 20% no-shows.

Preprocessing

- Removed irrelevant columns such as `PatientId`, `AppointmentID`, and original `Neighbourhood`.
- Computed `waiting_days` as the difference between `AppointmentDay` and `ScheduledDay`.
- Encoded categorical variables (e.g., Gender, Neighbourhood).
- Normalized numerical features using `StandardScaler`.
- No oversampling or data augmentation was used, as per constraints.

Vanilla Neural Network (NumPy)

Architecture

- Input Layer: 10 features
- Hidden Layers: 16 neurons $\rightarrow$ 8 neurons (Leaky ReLU)
- Output Layer: 1 neuron (Sigmoid)

Key Improvements

- Replaced ReLU with Leaky ReLU.
- Xavier Initialization for better gradient flow.
- Lowered prediction threshold to 0.3 to boost recall.

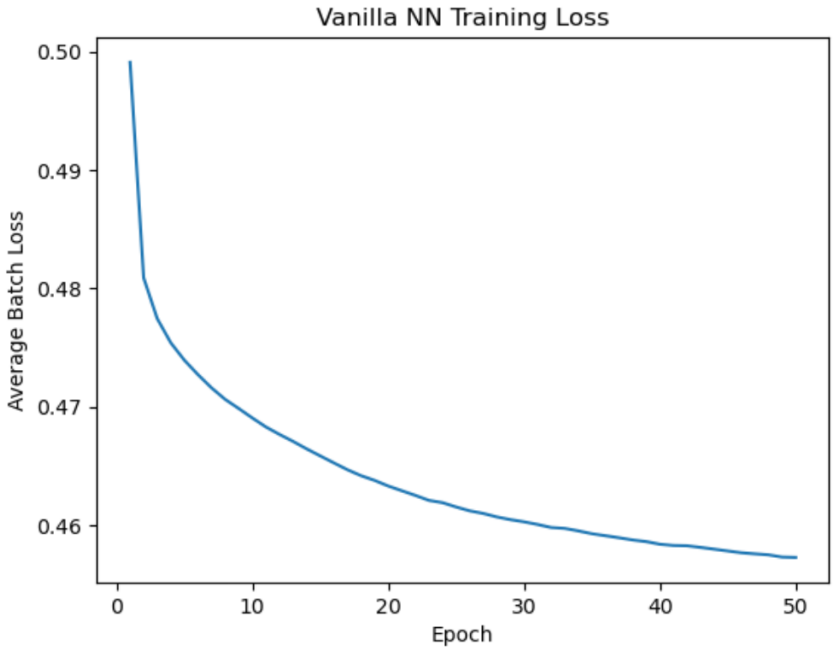

Figure 1: NumPy NN Training Loss over 50 epochs

PyTorch Neural Network

Architecture

Same as the NumPy model, but implemented using PyTorch modules.

Enhancements

- Used `BCEWithLogitsLoss` with `pos_weight = 3.95` to handle class imbalance.
- Removed Sigmoid from final layer (since `BCEWithLogitsLoss` includes it internally).
- Lowered threshold to 0.3 during evaluation.

Final Results Comparison

Model	Accuracy	F1 Score	PR-AUC
NumPy NN	0.721	0.399	0.346
PyTorch NN	0.511	**0.426**	**0.349**

Table 1: Performance comparison of the two models

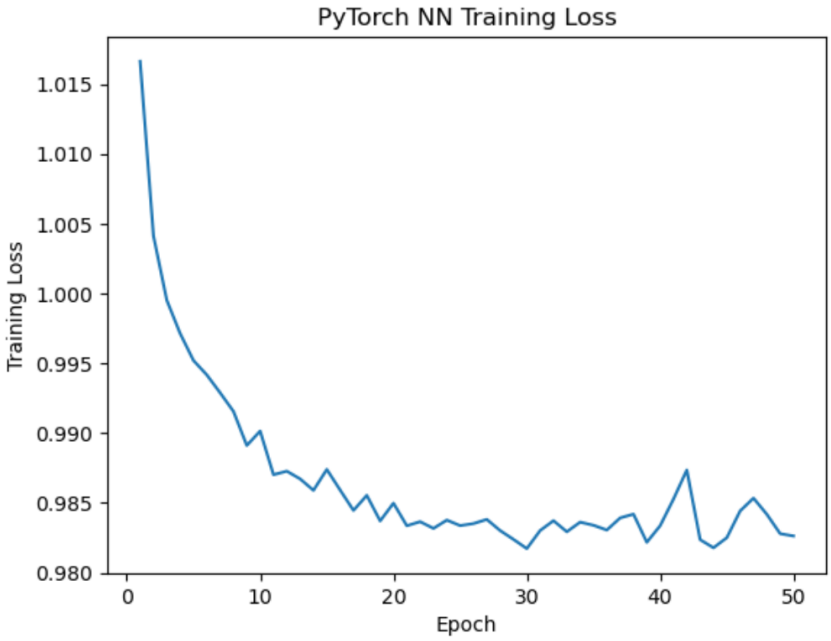

Figure 2: PyTorch NN Training Loss over 50 epochs

Confusion Matrix Summary

NumPy Model

```
[[13903  3739]
 [ 2420  2044]]
```

PyTorch Model

```
[[ 5779 11863]
 [  338  4126]]
```

Analysis and Discussion

Convergence Time

The PyTorch model demonstrated smoother and faster convergence, as shown in the training loss plots (Figure 1 and Figure 2). This is due to its use of advanced optimizers (e.g., Adam), better initialization, and optimized tensor operations. The NumPy model converged slowly, requiring more epochs to reduce the loss and showed slower learning in early epochs.

Memory Usage

The NumPy model relies on Python lists and NumPy arrays on CPU, which may consume more memory for manual operations. The PyTorch model uses efficient tensor computations and supports GPU acceleration, making it more memory-efficient and suitable for scaling.

Model Behavior

- **NumPy Model:** Initially predicted only class 0 due to imbalance. With threshold adjustment and better activation/init, it achieved a balanced F1 score.
- **PyTorch Model:** Initially also biased. After applying weighted loss and threshold tuning, it achieved better recall and F1.

Conclusion

The PyTorch model, with class-weighted loss and threshold tuning, outperformed the NumPy model in terms of balanced performance. It is better suited for healthcare scenarios where identifying no-shows is more important than simply maximizing accuracy.