# Evaluation and Analysis Report

Neural Network Implementation on the Medical Appointment No-Show Dataset

Aniket Mandal
IIT Varanasi

## Model 1: From-Scratch Implementation

### Training Metrics

- Training Time: 10.40 seconds
- Best Threshold: 0.16
- Best F1 Score: 0.3406
- Accuracy: 30.67%
- PR-AUC: 0.2328

### Confusion Matrix

$$\begin{bmatrix} 2823 & 14846 \\ 479 & 3958 \end{bmatrix}$$

## Model 2: PyTorch Implementation

### Training Metrics

- Training Time: 2.13 seconds
- Best Threshold: 0.17
- Best F1 Score: 0.3498
- Accuracy: 45.50%
- PR-AUC: 0.2724

**Confusion Matrix**

$$\begin{bmatrix} 6817 & 10852 \\ 1196 & 3241 \end{bmatrix}$$

# 1. Convergence Time

| | NumPy Model | PyTorch Model |
|---|---|---|
| Training Time | 10.40 s | 2.13 s |

**Speedup Calculation:**

$$\text{Speedup Factor} = \frac{10.40}{2.13} \approx 4.88$$

**Analysis:** The PyTorch model converges 4.88 times faster due to optimized backend libraries (e.g., cuBLAS, MKL), efficient memory management, and possible parallelism. NumPy lacks such low-level optimizations.

# 2. Performance Metrics

| Metric | NumPy Model | PyTorch Model |
|---|---|---|
| Accuracy | 30.67% | 45.50% |
| F1 Score | 0.3406 | 0.3498 |
| PR-AUC | 0.2328 | 0.2724 |

# 3. Memory Usage

- NumPy model: $\sim$9 MB
- PyTorch model: $\sim$30 MB

**Memory Calculation:**

- Parameters: $14 \times 64 + 64 \times 1 + 64 + 1 = 1025$
- Memory: $1025 \times 4 = 4100$ bytes $\approx 4$ KB

Additional memory includes activations, gradients, and optimizer states. PyTorch uses $3\times$ the memory for optimizers like Adam: $3075 \times 4 = 12,300$ bytes $\approx 12$ KB. Metadata and runtime buffers contribute to 30 MB usage.

**Conclusion:** PyTorch's flexibility and automation increase memory needs; NumPy is leaner but requires manual coding.

# 4. Confusion Matrix and Inference

## NumPy Model

- True Negatives (TN): 2823
- False Positives (FP): 14846
- False Negatives (FN): 479
- True Positives (TP): 3958

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{3958}{3958 + 14846} \approx 0.210$$
$$\text{Recall} = \frac{TP}{TP + FN} = \frac{3958}{3958 + 479} \approx 0.892$$

**Interpretation:** High false positives indicate a bias towards predicting "No-show".

## PyTorch Model

- True Negatives (TN): 6817
- False Positives (FP): 10852
- False Negatives (FN): 1196
- True Positives (TP): 3241

$$\text{Precision} = \frac{3241}{3241 + 10852} \approx 0.230$$
$$\text{Recall} = \frac{3241}{3241 + 1196} \approx 0.730$$

**Interpretation:** PyTorch offers a better trade-off between false positives and false negatives.

# 5. Final Analysis and Reflection

**Why PyTorch performs better:**

- Uses optimized C++/CUDA backends
- Supports GPU (though CPU used here)
- Better numerical stability
- Reduces implementation bugs and improves productivity

# 6. Summary Table

| Aspect | NumPy Model | PyTorch Model |
|---|---|---|
| Training Time | 10.40 s | 2.13 s |
| Accuracy | 30.67% | 45.50% |
| F1 Score | 0.3406 | 0.3498 |
| PR-AUC | 0.2328 | 0.2724 |
| Memory Usage | $\sim$9 MB | $\sim$30 MB |
| Precision | $\sim$21.0% | $\sim$23.0% |
| Recall | $\sim$89.2% | $\sim$73.0% |