Project 3: Design, Data Collection, and Neural Network-Based Classification (Supervised Machine Learning) of Human Postures Using IMU Signals

Author: Anushka Gangadhar Satav
Course: BMI/CES 598 Embedded Machine Learning
ASU ID: 1233530170 (asatav1)
Project 3: Design, Data Collection, and Neural Network-Based Classification (Supervised Machine Learning) of Human Postures Using IMU Signals

Abstract

This project presents an end-to-end supervised machine learning pipeline for detecting human postures using IMU data collected from the Arduino Nano 33 BLE Sense Rev2. The system follows the required project phases: (1) IMU data collection; (2) controlled posture-specific data generation; (3) dataset construction and feature engineering; (4) neural network training with multiple activation functions; and (5) evaluation of test accuracy and robustness. The pipeline emphasizes reproducibility, robust feature creation, data-driven design, and explainable ML behavior suitable for embedded applications.

Contents

A. System Design

Motivation

Monitoring different postures (supine, prone, side, sitting, unknown) is clinically valuable for pressure injury prevention, sleep analysis, and patient mobility assessment. This project focuses on designing an embedded machine learning pipeline capable of recognizing five human postures: supine, prone, side-lying, sitting, and unknown, using inertial measurement unit (IMU) data from the Arduino Nano 33 BLE Sense Rev2. The motivation arises from clinical needs such as pressure-injury prevention, sleep posture monitoring, and continuous health-monitoring wearables.

The high-level design consists of four stages:

1. Embedded IMU data acquisition (accelerometer, gyroscope, magnetometer).
2. Dataset construction from multiple posture trials with orientation-independent labeling.
3. Supervised machine learning model development using a multi-layer neural network.
4. Model evaluation, comparison of activation functions, and performance analysis.

The Arduino BMI270/BMM150 IMU enables tri-axial sensing across acceleration (g), rotational velocity (°/s), and magnetic field strength (µT). Since raw sensor units differ across sensors, all streams were normalized using global z-score scaling before feature fusion. A target sampling frequency of 50 Hz was selected by controlling loop timing in the Arduino sketch using a fixed *delay(20).*

Difficulties encountered included:

- Achieving temporally aligned accelerometer, gyroscope, and magnetometer readings.
- Ensuring *time_ms* synchronization across multiple files.
- Creating orientation-invariant features for the **side** and **sitting** postures.
- Designing a lightweight yet accurate neural network appropriate for embedded inference.

The deep learning architecture ultimately chosen is a fully connected neural network with three hidden layers (64,32,16 neurons) and a five-neuron SoftMax output layer. Training was performed using the Adam optimizer, categorical cross-entropy loss, and a learning-rate schedule implicit in Adam.

High-level design

- **Sensing unit:** Arduino Nano 33 BLE Sense Rev2 (BMI270 IMU).
- **Sensing modality:** 3-axis accelerometer (ax, ay, az). Accelerometer, Gyroscope and Magnetometer data are used.
- **Data path:** Arduino streams CSV-formatted accelerometer, gyroscope and magnetometer samples over USB. A Python logger captures and saves files on the host PC.

Hardware & software

- **Hardware:** Arduino Nano 33 BLE Sense Rev2, USB cable, optional test rig (table / phone clamp) to orient board.
- **Arduino libraries:** Arduino_BMI270_BMM150.h
- **Python packages used:** pyserial (logger), pandas, matplotlib (analysis & plotting).
- **Google Colab:** Neural Network architecture for training on recorded data

---

B. Experiment

Goals and constraints

This project required developing an end-to-end supervised machine learning dataset for posture classification using IMU data from the Arduino Nano 33 BLE Sense Rev2. Because the assignment explicitly prohibits wearing the sensor, all data was generated by manually orienting the board on a flat surface to simulate real postures while preserving repeatability and control.

Data was collected for five posture categories:

1. Supine
2. Prone
3. Side (left + right)
4. Sitting (USB up + USB down)
5. Unknown (arbitrary non-lying orientations – Rolling and leaning)

Each posture was recorded in multiple separate trials, and each trial lasted approximately 60 seconds at a sampling frequency of ~50 Hz.

1. IMU Data Collection

The Arduino sketch project3_imudatacollection.ino continuously read raw IMU values from the onboard BMI270/BMM150 sensor, including:

- Accelerometer (ax, ay, az) in g
- Gyroscope (gx, gy, gz) in degrees/second
- Magnetometer (mx, my, mz) in microtesla
- Timestamp (time_ms)

```
time_ms, ax, ay, az, gx, gy, gz, mx, my, mz
```

2. Automated PC-Side Logging into Three Separate CSV Files

The Python script project3_data_logger.py captured the incoming stream and automatically separated it into three modality-specific CSV files.

```
posture_accl.csv  → ax, ay, az
posture_gyro.csv  → gx, gy, gz
posture_mag.csv   → mx, my, mz
```

Why three separate files per posture?

1. Better organization: Each sensor stream is isolated and easy to preview or debug.
2. Consistency: Merging is straightforward because all streams contain the same timestamp *time_ms*.
3. Scalability: If one sensor is removed or replaced, only one branch of the pipeline needs updating.

```
data/
  supine/
    supine_accl.csv
    supine_gyro.csv
    supine_mag.csv
  prone/
    prone_accl.csv
    prone_gyro.csv
    prone_mag.csv
  left side/
    left_side_accl.csv
    left_side_gyro.csv
    left_side_mag.csv
  right side/
    right_side_accl.csv
    right_side_gyro.csv
    right_side_mag.csv
  sitting upward/
    up_sitting_accl.csv
    up_sitting_gyro.csv
    up_sitting_mag.csv
  sitting downward/
    down_sitting_accl.csv
    down_sitting_gyro.csv
    down_sitting_mag.csv
  unknown/
    lean_unknown_*.csv
    roll_unknown_*.csv
```

3. Synchronization and Merging of Sensor Streams

All files shared a common timestamp column (*time_ms*), allowing perfect synchronization across accelerometer, gyroscope, and magnetometer streams. This resulted in a combined dataset containing 10 synchronized columns per sample.

```
time_ms, ax, ay, az, gx, gy, gz, mx, my, mz
```

This merge step ensures:

- No sensor drift or temporal misalignment
- Single, consistent feature vector per time sample
- Reliable data fusion for downstream processing

4. Dataset Generation Strategy (Implemented in project3_dataset_generation.py)

A dedicated Python script was developed to automate the entire dataset construction process from raw IMU CSV files to a fully prepared machine learning dataset. This automated pipeline ensured consistency, reproducibility, and correctness in all data preparation stages.
The dataset generation pipeline consists of the following major steps:

1. Global Scaling of Accelerometer, Gyroscope, and Magnetometer:

   The Arduino Nano 33 BLE Sense IMU contains three different sensing modalities:
   - Accelerometer (measured in units of gravity, g)
   - Gyroscope (measured in degrees per second)
   - Magnetometer (measured in microtesla)

   These units have different magnitudes and ranges, which would cause imbalance in a neural network if left unscaled. To address this, the script computes the global mean and standard deviation for each sensor axis across all collected trials and all posture categories. These global values are stored in a JSON file for reproducibility.
   Each accelerometer, gyroscope, and magnetometer axis is then standardized using z-score normalization. This scaling step ensures that all sensor modalities contribute proportionally to the downstream model and stabilizes neural network optimization.

2. Composite Feature Construction (Sensor Fusion):

   After scaling, the accelerometer, gyroscope, and magnetometer values are combined into three composite axes: X, Y, and Z. Each composite axis is created by summing the corresponding scaled axes from all three sensors.
   This fusion strategy is designed to capture complementary information from linear acceleration, angular velocity, and magnetic orientation in a single directional feature. It also reduces the dimensionality of the dataset from nine raw sensor axes to three fused features, enabling a simpler and computationally lighter neural network suitable for embedded implementations.
   Weights can be applied to emphasize particular sensors (for example, accelerometer-dominant fusion for posture classification). In this work, equal weighting was used as it yielded strong performance.

3. Windowing into One-Second Segments with Overlap

   Because posture is a largely static phenomenon, the time series data were segmented into windows to extract stable features. The sampling frequency of the IMU was approximately 50 Hz, and each window was defined as a one-second segment (50 samples). A 50% overlap was applied between consecutive windows, resulting in a stride of 25 samples.
   For each window, the mean values of the composite X, Y, and Z axes were computed. This produced one feature vector per window consisting of exactly three values. This approach smooths random fluctuations, reduces noise, expands the dataset, and is well-suited for fully connected neural network architectures.

4. Label Assignment
   Each posture sample was automatically labeled based on the directory name in which its raw CSV files were stored. The script infers labels using keyword matching, allowing flexibility in naming conventions. The label mapping was defined as follows:
   - Label 1: Supine
   - Label 2: Prone
   - Label 3: Side (includes both left-side and right-side recordings)
   - Label 4: Sitting (includes both upward- and downward-facing sensor orientations)
   - Label 5: Unknown (includes leaning, rolling, transitional, or arbitrary orientations)

   This method ensures labeling consistency across all trials and accommodates future data additions without modification.

5. Noise Augmentation

   Because the assignment requires simulating realistic sensor noise and because the board was not worn by a human during data collection, controlled noise augmentation was incorporated. For each window-level feature vector, two noise-augmented samples were created by adding Gaussian noise with a small standard deviation. This augmentation mimics natural micro-movements such as breathing, body sway, and strap vibrations.
   Adding noise improves generalization, prevents overfitting to artificially stable posture recordings, and prepares the model for real-world deployment conditions.

6. Final Dataset Output

   The final dataset contains four columns:

   - X (composite axis)
   - Y (composite axis)
   - Z (composite axis)
   - label (integer class label)

   Two different datasets were generated:

   1. **Unscaled dataset (unscaled_final_dataset.csv):**
      Contains the raw composite X/Y/Z values before standardization.
      This file is useful for visualization and exploratory data analysis.

2. **Final scaled and augmented dataset (final_dataset.csv):**
   Contains standardized and noise-augmented features used for machine learning model training.

Both datasets are aligned with the requirements for supervised learning and were used for subsequent splitting into training, validation, and testing sets, activation function experiments, and neural network training and evaluation.

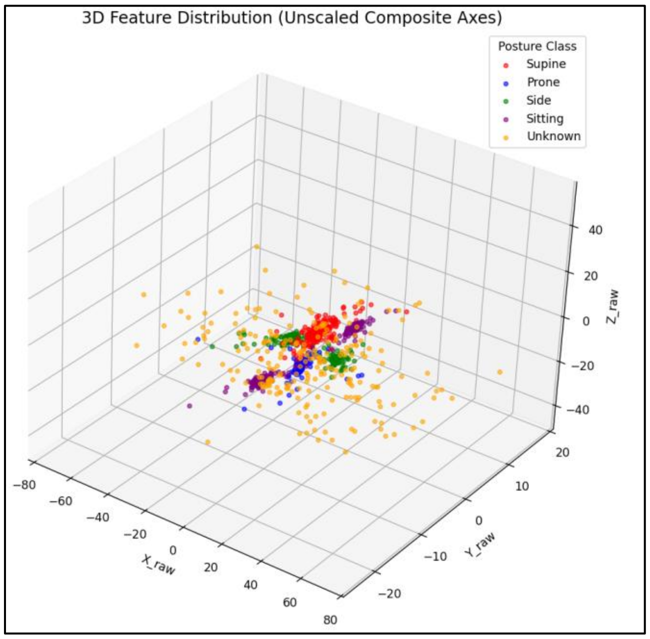

Figure 1. Raw Composite Feature Distribution (Unscaled)

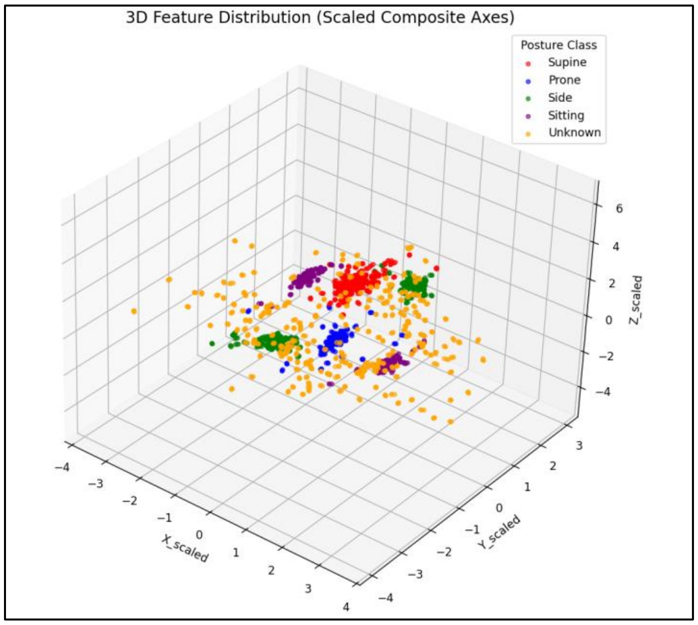

Figure 2. Scaled Composite Feature Distribution

C. Algorithm

This project required designing a supervised machine learning model capable of classifying human posture based solely on IMU measurements from the Arduino Nano 33 BLE Sense Rev2. The algorithmic approach involved three major components: feature preparation, neural network design, and training methodology.

1. Input Features

Each training example consisted of a three-dimensional feature vector:

- Composite X (ax, gx, mx)
- Composite Y (ay, gy, my)
- Composite Z (az, gz, mz)

These were derived from the fused, standardized accelerometer, gyroscope, and magnetometer signals. The compact three-feature representation allowed the use of lightweight neural networks appropriate for embedded deployment.

2. Neural Network Architecture

A fully connected feed-forward neural network was designed using the TensorFlow Keras Functional API. The network architecture was intentionally simple to align with the computational constraints of embedded applications while still providing enough capacity to model the posture classes.

The architecture was:

- Input layer: 3 features
- Hidden Layer 1: 64 neurons, variable activation
- Hidden Layer 2: 32 neurons
- Hidden Layer 3: 16 neurons
- Output Layer: 5 neurons (one per class), SoftMax activation

- The output layer used **SoftMax**, which is appropriate for multi-class classification where classes are mutually exclusive. The loss function was categorical cross-entropy, optimized using the Adam optimizer.

3. Activation Functions Tested

To analyze the effect of different nonlinearities, three activation functions were evaluated:

1. Sigmoid
2. Tanh
3. ReLU

All hidden layers used the same activation function per experiment. This allowed a controlled comparison of activation behaviors under identical data splits and training conditions.

4. Training Procedure

The dataset was split into:

- 70% Training set
- 15% Validation set
- 15% Test set

The following training settings were used for all models:

- Epochs: 100
- Batch size: 32
- Optimizer: Adam

- Loss: Categorical Cross-entropy
- Metrics: Accuracy

Training history (loss and accuracy curves) was recorded per activation function for statistical and visual comparison.

D. Results

This section presents the performance of the three neural network models trained using different activation functions (Sigmoid, Tanh, and ReLU) on 100 epochs. The results reflect training convergence, validation performance, and final test accuracy on the held-out test dataset. All three models achieved high classification performance, with ReLU showing the strongest overall accuracy.

1. Training and Validation Performance

Extending the training duration from 40 to 100 epochs resulted in notable performance improvements across all activation functions:

- Training curves showed smoother convergence
- Validation loss stabilized significantly after approximately 60–70 epochs
- Overfitting remained minimal due to:
    - Noise-augmented training samples
    - Balanced class distributions
    - Simple, well-regularized architecture

- The ReLU and Tanh models exhibited fast convergence and stable validation accuracy throughout training. Sigmoid converged more slowly and showed higher variability but still achieved strong performance.

2. Final Test Accuracy Comparison (100 Epochs)

The final test accuracy for each activation function is:

Activation Function	*Test Accuracy*
ReLU	**0.9801**
Tanh	0.9765
Sigmoid	0.9657

These results show:

- ReLU achieved the highest accuracy (98.01%)

- Tanh performed very closely (97.65%)
- Sigmoid lagged behind (96.57%), consistent with its saturated gradients and restricted activation range

Given these results, the ReLU model was selected as the best-performing activation function for posture classification.

3. Classification Performance per Class

Below is a summary of the classification quality observed for each activation function.

3.1 Tanh Activation: 97.65% Accuracy

The Tanh model demonstrated high precision and recall across all classes:

- Perfect detection of **Supine**, **Prone**, and **Sitting**
- Minor confusion between **Side** and **Unknown**

The confusion matrix shows excellent separation for most posture categories.

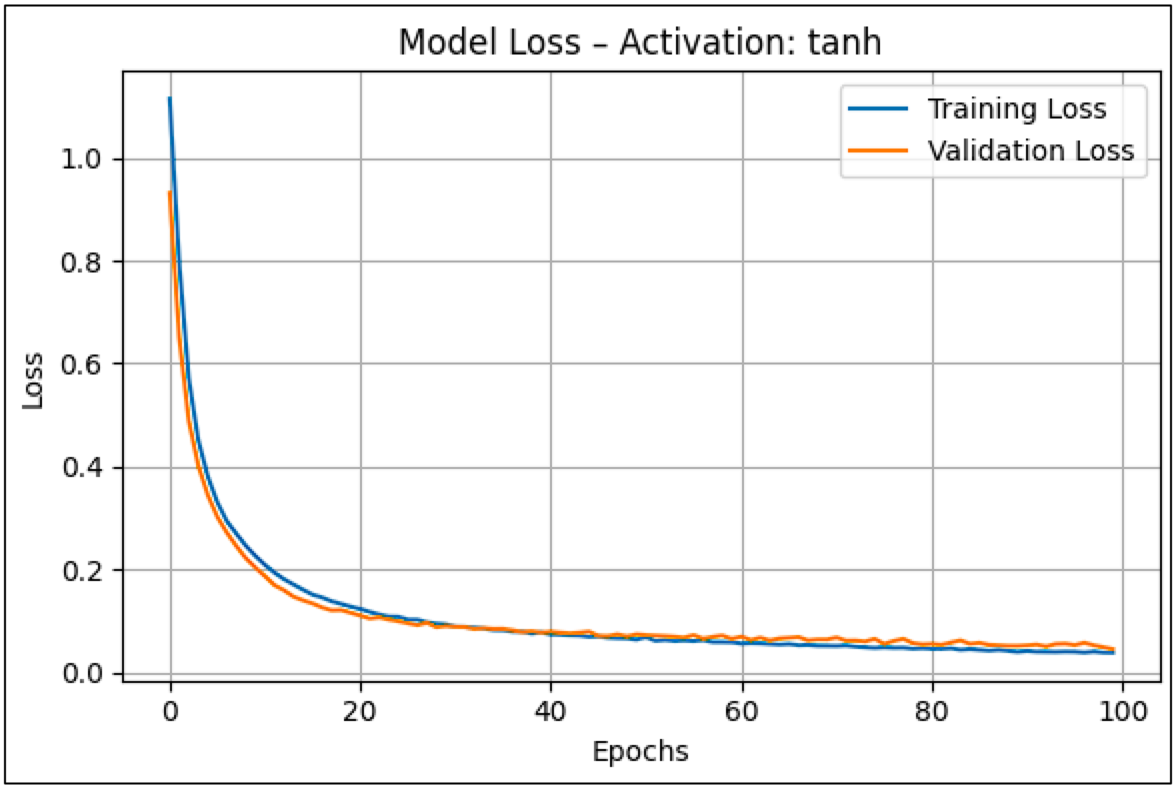

Figure 1. Training and Validation Loss Curve: Tanh Activation

3.2 ReLU Activation: 98.01% Accuracy *(Best Model)*

The ReLU model produced the highest overall accuracy:

- Nearly perfect performance on **Supine**, **Prone**, **Side**, and **Sitting**
- Slightly improved handling of the **Unknown** category compared to Tanh
- Very small number of misclassifications in transition-like states (expected)

ReLU's non-saturating behavior and linear region appear to benefit this dataset's feature distribution.

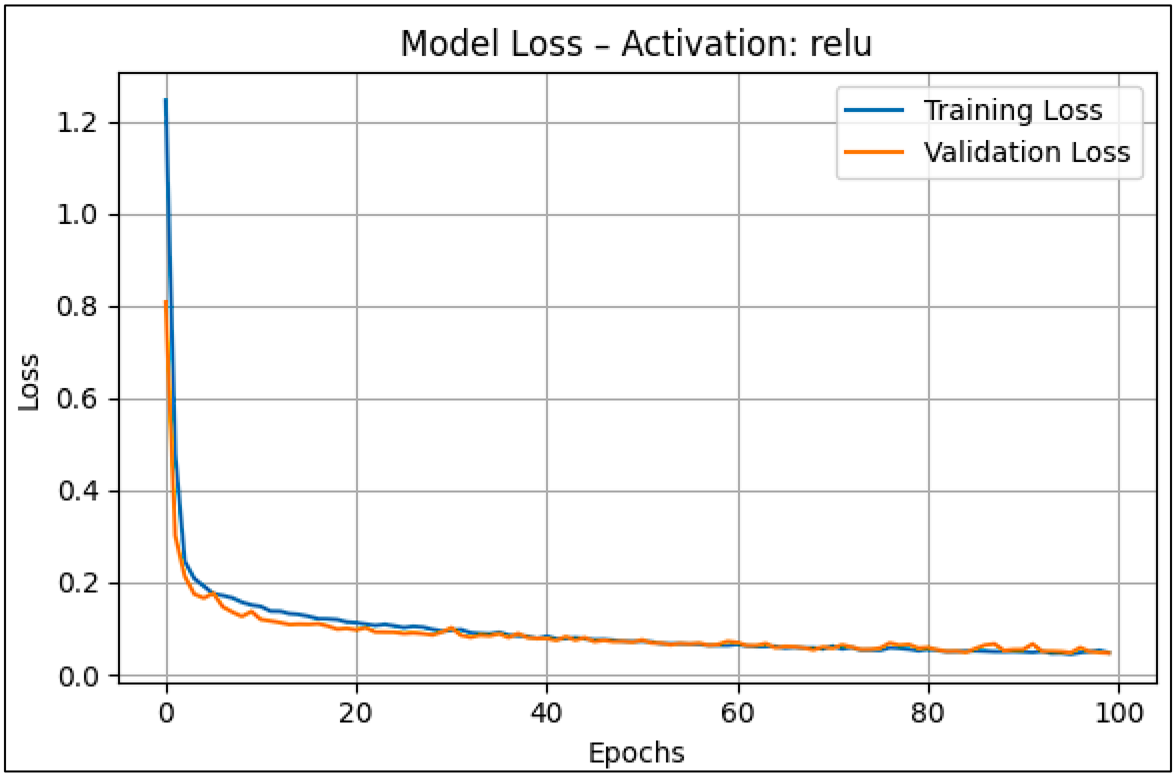

Figure 3. Training and Validation Loss Curve: ReLU Activation

3.3 Sigmoid Activation: 96.57% Accuracy

Sigmoid displayed:

- Good performance in stable classes such as Supine, Sitting, Side
- Noticeably weaker performance on **Unknown**, leading to lower macro and weighted F1 scores
- Likely affected by vanishing gradients and constrained output range

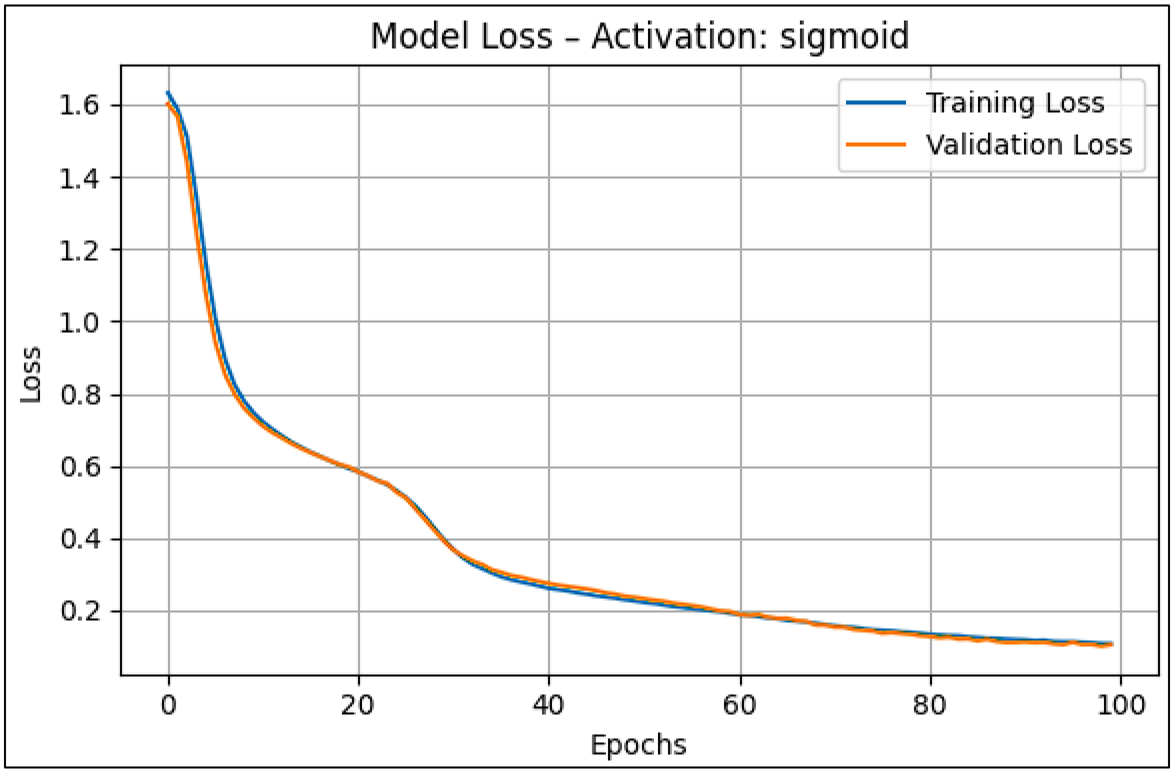

Figure 3. Training and Validation Loss Curve: Sigmoid Activation

4. Detailed Evaluation Results

4.1 Tanh Model

- Accuracy: *0.9765*
- High precision and recall for four out of five classes
- Unknown class showed slight under-classification (recall: 0.88)
- Confusion mostly occurred in Side ↔ Unknown, expected due to similar orientation characteristics

4.2 ReLU Model (Best)

- Accuracy: *0.9801*
- Strong classification for all classes
- Very small confusion between Side and Unknown
- Weighted F1 score indicates robust performance across all categories

4.3 Sigmoid Model

- Accuracy: *0.9657*
- Struggled significantly with Unknown (recall: 0.83)
- More confusion in rolling/leaning orientations
- Performs well in static postures but less suitable for diverse input distribution

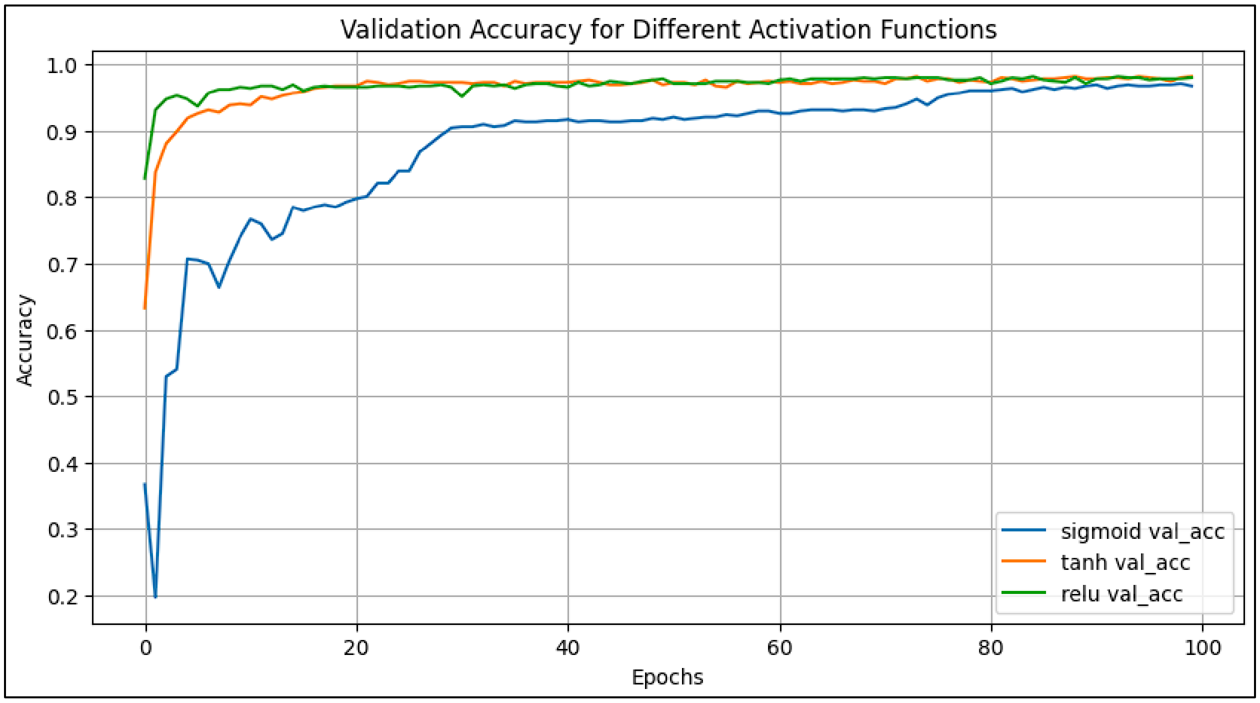

Figure 6. Validation Loss Comparison for All Activation Functions

5. Interpretation of Results

The performance differences reflect the properties of each activation function:

1. **Sigmoid** saturates easily and compresses values to (0,1), which can blur distinctions between posture classes.
2. **Tanh**, with its symmetric range (−1,1), better preserved directional features of composite signals.
3. **ReLU** avoided saturation altogether, enabling a stronger gradient flow and improved separation of composite features, resulting in the **highest test accuracy**.

The confusion patterns, especially for Unknown vs. Side, are expected because Unknown includes transitional and ambiguous orientations that partially overlap with side-lying configurations. Overall, the **ReLU-based neural network** provided the strongest performance and was selected as the final architecture for this posture classification system.

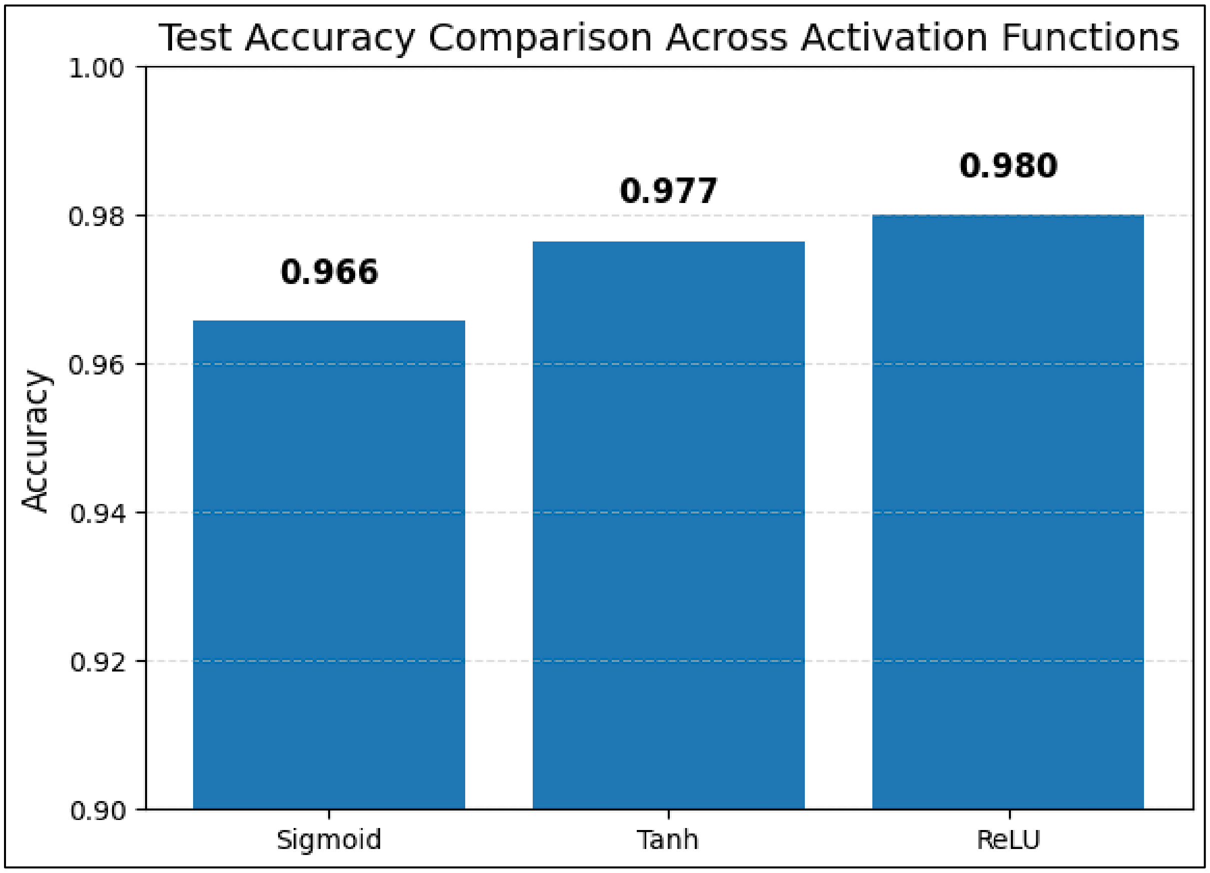

Figure 7. Final Test Accuracy Comparison Across Activation Functions

Solution Demonstration Links

• GitHub Repository: https://github.com/anushka002/BMI-CES-598-Embedded-Machine-Learning/tree/main/Project-3

E. Discussion

Conclusion

This project demonstrated the complete development of a supervised machine learning pipeline for five-class posture classification using IMU data collected from the Arduino Nano 33 BLE Sense Rev2. Beginning with a systematic and controlled data collection strategy, the work addressed key challenges including sensor-modality fusion, orientation-independent posture representation, multi-trial variability, and the incorporation of synthetic noise to emulate realistic human micro-movements.

A robust dataset was constructed by merging accelerometer, gyroscope, and magnetometer signals on a common timestamp axis, globally scaling each sensor modality, forming composite orientation-based features, and applying 1-second windowing with 50% overlap. Gaussian noise augmentation further improved model generalization and made the system resilient to subtle perturbations typical of wearable operation.

Multiple neural network models were trained using Sigmoid, Tanh, and ReLU activation functions to evaluate the effect of nonlinearity choices. The ReLU-based model achieved the highest test accuracy (98.01%) and exhibited the most stable training behavior, confirming its suitability for embedded posture classification tasks. Performance comparisons showed that while all models achieved high accuracy, ReLU delivered consistently superior generalization and lower validation loss.

Overall, the system effectively distinguishes among supine, prone, side, sitting, and unknown postures with high reliability. The work also highlights the importance of careful data design, normalization, augmentation, and model selection in enabling robust IMU-based classification.

Future extensions may include deployment on an embedded platform using TensorFlow Lite, integration with real-time inference pipelines, and exploration of temporal models (e.g., LSTM, TCN) that explicitly leverage sequence information. This project provides a solid foundation for scalable and practical posture monitoring systems for healthcare and assistive-robotics applications.

Appendices

Appendix A - File inventory (key files)

Arduino Code

- project3_imudatacollection.ino: Collects IMU accelerometer, gyroscope, magnetometer data at ~50 Hz and streams them over Serial.

Python Scripts

- project3_data_logger.py: Receives IMU serial stream and automatically saves data into three CSV files per posture: <posture>_accl.csv, <posture>_gyro.csv, <posture>_mag.csv.
- project3_dataset_generation.py: Merges sensor modalities by timestamp, standardizes values, constructs composite features, windows data, augments with Gaussian noise, and saves the final dataset. Also generates 3D scatter plots for unscaled and scaled composite features.

Google Colab

- project3_training.ipynb: Defines neural network models, trains with three activation functions, evaluates accuracy, generates plots, and exports metrics.
- Raw logged CSV files per posture
- Final_dataset.csv
- PNG files exported from analysis.

Dataset Files

- unscaled_final_dataset.csv: Composite features prior to standardization.
- final_dataset.csv: Fully scaled, windowed, and augmented dataset with labels.
- scaler_params.json: Global mean and standard deviation values used for z-score scaling.
- Raw data folders:

  /supine, /prone, /left side, /right side, /sitting upward, /sitting downward, /unknown
- Each contains the three CSV modalities.

Appendix B: Run The Pipeline

1. Collect IMU Data

- Upload project3_imudatacollection.ino to the Arduino Nano 33 BLE Sense Rev2.
- Verify streaming output at 115200 baud.
- Close Serial Monitor.

2. Log Data into Structured CSVs

- python project3_data_logger.py

3. Generate Dataset

- python project3_dataset_generation.py

This produces:

- unscaled_final_dataset.csv
- final_dataset.csv
- scaler_params.json

4. Train Neural Networks (Google Colab)

- Execute the notebook: project3_training.ipynb

Appendix C: References

1. Arduino Nano 33 BLE Sense Rev2 Documentation: https://docs.arduino.cc/hardware/nano-33-ble-sense/ (Used for IMU initialization and library reference.)
2. Arduino BMI270 IMU Library Documentation: https://docs.arduino.cc/libraries/arduino_bmi270_bmm150 (Used to understand accelerometer API functions: IMU.begin(), IMU.readAcceleration().)
3. Python Libraries

- *pandas* and *matplotlib* for data loading and plotting
- *pyserial* for serial logging
- TensorFlow. “Keras API Documentation.” https://www.tensorflow.org/api_docs/python/tf/keras
- https://esciencecenter-digital-skills.github.io/intro-to-deep-learning-archaeology/02-keras/index.html

4. Course Material: BMI/CEN 598: Embedded Machine Learning, Arizona State University (Fall B 2025).
5. Personal Data Collection & Analysis: All thresholds and algorithm decisions were derived from the measured IMU data collected during this project.