Designing DNN for IMU sensor data

Name: Erik Barraza Cordova

System Design

In the collection of sensor data, I wanted to be able to train a model with a significantly larger data set than when I had completed the operational collection for Project 02. As a result, I decided it was crucial to expedite the sensor reading process by developing a collection program for sensor data that collected data at a higher frequency than prior. Below you can see how the code for each differentiated, as I hoped to enhance the abilities of my sensor reading abilities by increasing the readings per second and expediting the baud rate. Although there would be duplications of data per second, I saw it as an opportunity to make it easier to collect larger amount of data with smaller step sizes, allowing for the ability to collect batch samples of code for each posture & record them in the Serial Monitor.

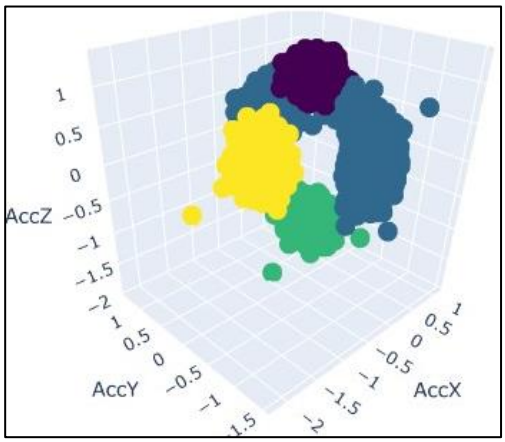
This data was recorded through the Serial Monitor and then saved into a .csv file through this script written in Python deployed for collection.

```python
import serial
import csv

# Open a serial connection to the Arduino
serial_port = 'COM6'  # Replace with the appropriate serial port
baud_rate = 115200
ser = serial.Serial(serial_port, baud_rate)

# Create and open a CSV file for writing
csv_file = open('sensor_data.csv', 'w', newline='')
csv_writer = csv.writer(csv_file)

# Write the header row to the CSV file
csv_writer.writerow(["AccX", "AccY", "AccZ", "GyroX", "GyroY", "GyroZ"])

try:
    while True:
        # Read a line of data from the Arduino
        data = ser.readline().decode().strip()

        # Split the data into individual values
        values = data.split(",")

        # Extract numeric values (skip labels)
        numeric_values = [float(val.split(":")[1]) for val in values if ":" in val]

        # Ensure that the received data contains the expected number of values
        if len(numeric_values) == 6:
            # Write the data to the CSV file
            csv_writer.writerow(numeric_values)

except KeyboardInterrupt:
    pass

# Close the CSV file and the serial connection
csv_file.close()
ser.close()
```

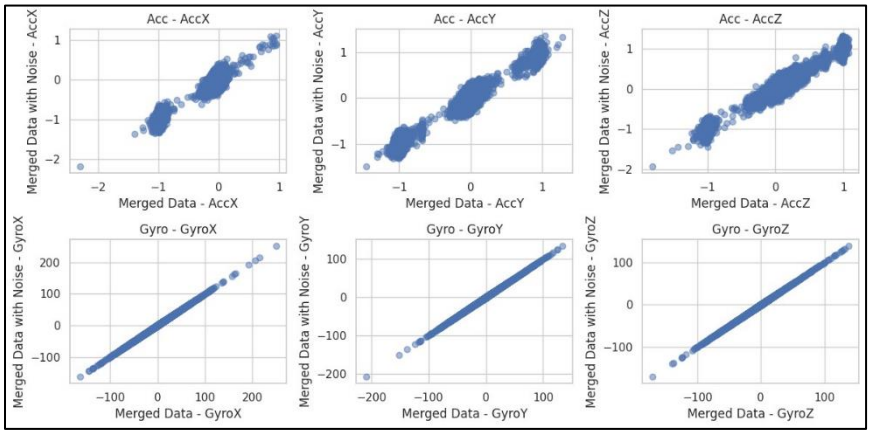
Data was then parsed and put into classifications.

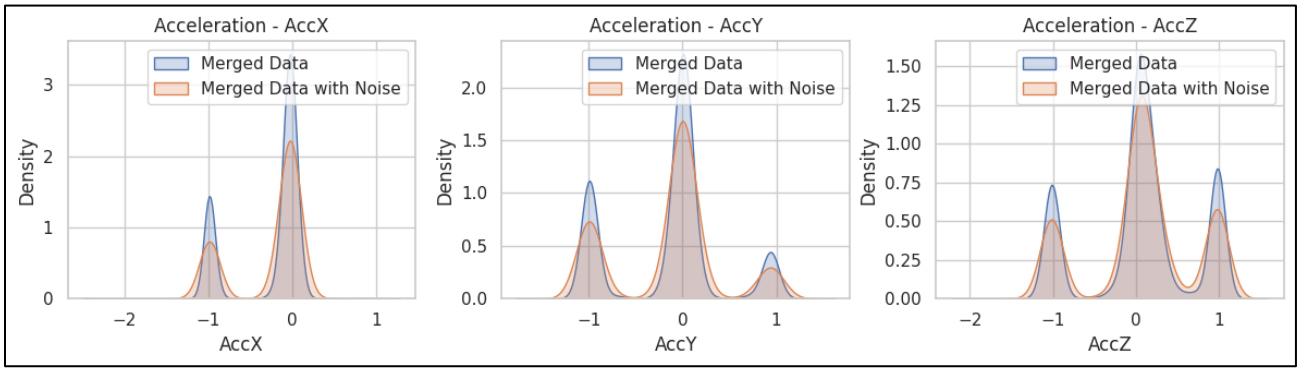
The motivation behind my development was to create large batch files of posture data for each position to capture the discreet differences in positioning that may exist in lying face up, down, on your side, or sitting. My motivation for this project was to create a higher-level machine learning model with a large amount of data that could be reliability for sampling in the event of a future parsing/optimization of data for implementation in an embedded system. As such, it required a lot of initial data to accomplish high-level deduction now so whatever smaller model is developed from it would have a reliability dataset to rely on. My higher-level design goes through a DNN model, a dense neural network due to the sampling method resembling batches rather than sequential data. This enabled me the ability to render the data with a merged file with the states (postures) being classified numerically. 1: Supine, 2: Side, 3: Prone, 4: Prone, 5: Unknown.

Initially, I saw the possibility of using CNN to provide the ability to classify data on a 3D space, so I can use a more dimensional portrayal of the data for interpretation. This would enable me the ability to use the axis to develop the interpretation of data. Unfortunately, the use of CNN would require some material I was unfamiliar with & unable to render past a few layering pools. I chose to use a DNN architecture for the development of a functional neural network instead, which allows for the interpretation of data on a 1D vector. It runs through activation functions to model the complex relationships of the data. The SoftMax activation function is recommended for taking an output layer in a multi-class classification deciding on which is the most likely to be the data point's class.



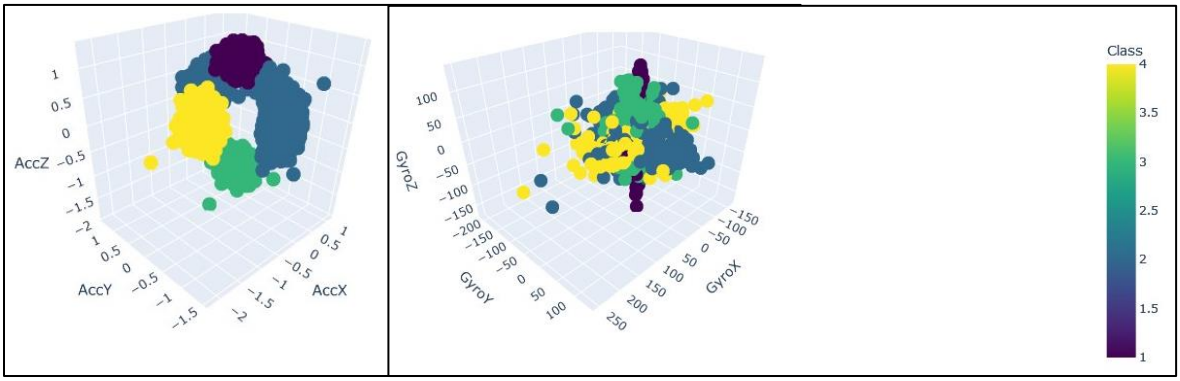
Upon collecting the data in batches for each posture, I merged the files with class identification at the beginning in the first column, 1 for supine, 2 for side, 3 for prone, 4 for sitting, 5 for unknown. Using Python, I split the data set into 60% training data, 20% validation data, and 20% test data for development of the model. Prior to splitting the data, I added noise to the files to introduce some larger variation that would not have occurred in the simulated environment as it would in a real-life sensor on someone's chest.





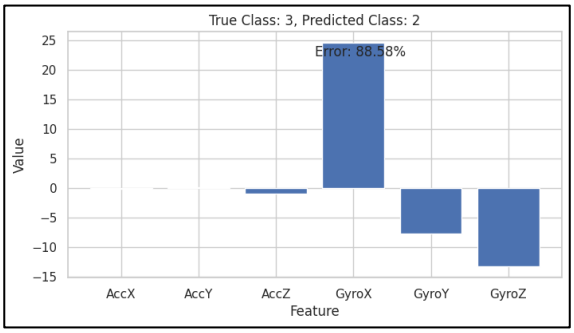
A few specific observations I have come across are that the noise presented into data expanded the bounds of the data's range. This enables us to capture variations in the data to appear in a real-life scenario where the margins are not as clear or defined as in our simulation. A difficulty I ran into was the implementation of the 5th posture, "unknown", as it created the difficulty of taking in data that was undefined in patterns but may be well defined on a point-by-point basis. This overlap creates difficulty with data point interpretation. Therefore, the data collection for this section required random movement in the sequencing. Upon designing the system, I considered the visual aspects of the work I did to make sure I presented a wide range of data points. The sampling frequency for this was 20 Hz, which is well above the recommended 0.5 Hz. The reasoning for this is to be able to develop a stronger modeling of data for a DNN without an activation function. More data will help inform it, but counterbalances are needed to make sure it does not experience overfitting, as underfitting is taken care of with more data to inform our model.

Experiment



As you can see in the data, there is a lot of nonlinearity in these clusterings. Although it is more well defined in accelerometer due to my extensive data collection from the sensors at high speeds for each posture's batch, the gyroscopic data has some variability due to my interest in incorporating data that made the orientation of the usb port irrelevant. This results in some planing of the data across the y,x plane & the z axis so as to make the sensor able to detect the posture irrespective of the orientation of the Arduino board across an individual's chest. That is why for gyroscopic data there is elongation of where the data is with a few outliers.

The experiment was simulated in each position for batch data collection. I always began at supine (1) and then move into the corresponding posture then started collecting data. This methodology coupled with 20 data points per second enabled a great return on data. A lot of data can lead to overfitting though, so I looked or methods to counter it for my non activation function DNN, which could work due to a large data set informing the options available. For the unknown category, I moved it around in large accelerative stages around the air, so as to create a dataset that when batched together would show no relational consistency & would output unknown. In a similar way, I had some cases that had large errors for certain data points in my array that ended up with wrong predictions, such as the one above from my test dataset.



Algorithm/Architecture Development

My DNN architecture comprises an input layer with 6 nodes, representing the sensor data features (AccX, AccY, AccZ, GyroX, GyroY, GyroZ). This is interpreted through class identification, which is developed through incorporated two hidden layers, the first with 64 nodes and the second with 128 nodes, utilizing specific activation functions such as Sigmoid, Tanh, or ReLU based on experimental comparisons on how they interpret the same training dataset.

The output layer consists of 5 nodes, one for each classification class, employing the SoftMax activation function for converting raw model outputs into class probabilities, this was employed from documentation that had a similar structure which was used as a reference (1).

Using references & documentation, I employed Adam optimizer for training, which changes the learning rate during training so there is mitigation of overfitting with large datasets. In multi-class classification, an ML reference shared that **'sparse_categorical_crossentropy'** provided a way to minimize loss. Regularization in the form of L2 regularization with a lambda value of 0.01 is applied to the weights of the dense layers to mitigate overfitting. The training process utilizes a batch size of 32 (standardized) , and training for 10 epochs, representing entire training of the dataset. In experimentation, I've explored the impact of 3 activation functions (Sigmoid, Tanh, and ReLU) on model performance in validation & test data randomized & proportionally selected. The training process incorporates backpropagation to adjust the model's weights to increase generalization capabilities.

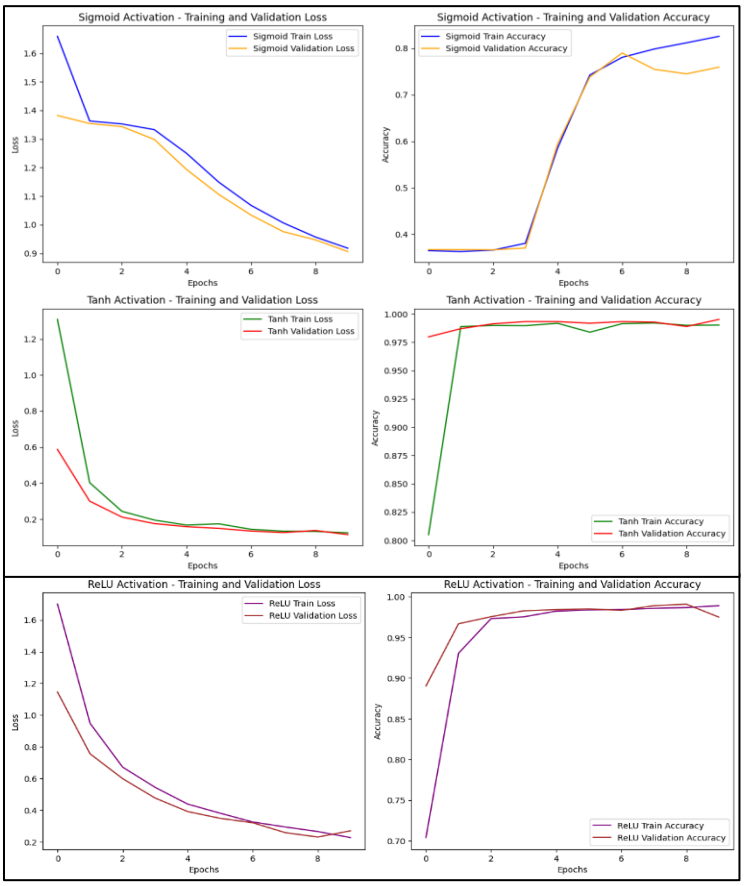
Key Components:

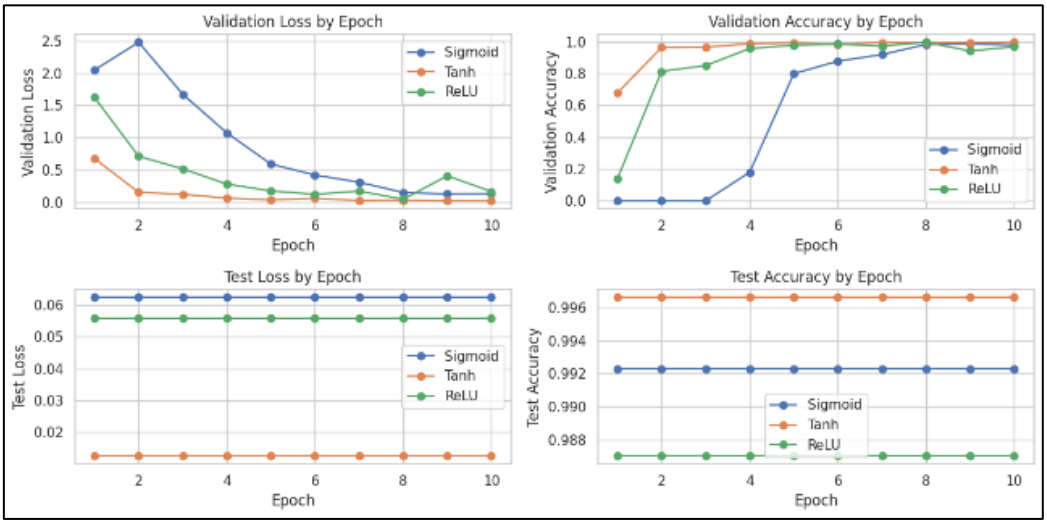
- **Activation Functions:** The choice of activation function (e.g., Sigmoid, Tanh, ReLU) affects the model's capacity to capture complex patterns in the data.
- **Optimizer (Adam):** The optimizer algorithm (e.g., Adam) is responsible for updating the model's weights during training.
- **Loss Function (Sparse Categorical Cross-Entropy):** The loss function quantifies the difference between the predicted class probabilities and the true class labels.
- **Regularization (L2 Regularization):** L2 regularization is applied to the model's weights to prevent overfitting. It adds a penalty term to the loss function based on the magnitude of weights.
- **Validation Data:** Using a validation dataset helps monitor the model's performance and early stopping to prevent overfitting.
- **Batch Size:** Batch size determines how many data samples are processed together before updating the model's weights (set at 1000).
- **Epochs:** The number of epochs controls how many times the entire training dataset is used during training.

Results

As you can denote in the graphs, there was some overfitting that I fixed with the introduction of regularization of L2. This lead to large changes in the graphs that were able to change the way each activation function interpreted the data & weren't overfitting as much. The validation process teaches so much about how the code uses activation functions and neural networks to help modify and cover the development of weights and functions for decision making.

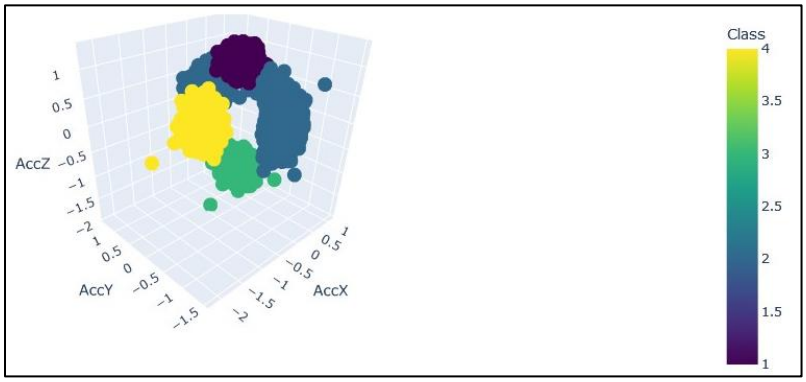
The performance for each section can be seen below, and as you can see variability exists at each section and stage on how it performed in its development:





Discussions

A large difficulty in designing a system for posture detection is finding the necessary balance for underfitting and overfitting the system. You need to find good counterbalances so as to not experience an exact duplication of training data or veer too far out that you interpret **unknown** data into all the categories. A future resolution would be to use the 3D imaging of data for acceleration to help train a CNN. By using the visual abilities of interpretation for image processing in a CNN, you can use it as a 'map' for features in gyroscopic and accelerometer data. Ultimately, this will lead to using it as a medium for interpreting if someone is in one of the 5 postures. Because it is spatial learning, you will be able to easily visualize any anomalies in incoming sensor data because they will visually be outside the bounds of the 3D modelling parameters set through training by existing data. The part of the project that was most difficult to accomplish was finding the right counter balances for overfitting & a large dataset. I believe a CNN can help provide a more visual understanding of data, although I would also attempt to adding more dynamic noise as to increase variability bounds for classification would improve my approach & help limit the amount of gathered data needed as it will span the same lengths with larger 'arms' of variation.



References

- https://machinelearningmastery.com/softmax-activation-function-with-python/
- https://www.turing.com/kb/softmax-activation-function-with-python
- https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f
- https://ieeexplore.ieee.org/document/9445263
- https://www.mdpi.com/1424-8220/21/15/5236
- https://myweb.uiowa.edu/pbreheny/uk/764/notes/11-1.pdf
- https://www.sciencedirect.com/science/article/pii/S0378779622003790
- http://faculty.washington.edu/yenchic/18W_425/Lec12_class.pdf
- https://online.stat.psu.edu/stat508/lesson/9/9.2/9.2.1
- https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
- https://deeplizard.com/learn/video/U4WB9p6ODjM
- https://saturncloud.io/blog/how-to-change-kerastensorflow-version-in-google-colab/
- https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/
- https://pytorch.org/docs/stable/generated/torch.nn.functional.softmax.html
- https://www.tensorflow.org/api_docs/python/tf/nn/softmax
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9117456
- https://arxiv.org/abs/2109.01118
- https://arxiv.org/ftp/arxiv/papers/2202/2202.03274.pdf
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7727224
- https://datascience.stackexchange.com/questions/41921/sparse-categorical-crossentropy-vs-categorical-crossentropy-keras-accuracy
- https://keras.io/api/losses/
- https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy