Project 3: Sequence Classification with Deep Neural Networks

1. Introduction

The analysis of trajectory data has gained significant attention due to advancements in deep learning techniques. This type of data captures both temporal and spatial characteristics, which require specialized preprocessing to effectively identify patterns. One intriguing challenge is the classification of moving objects based on their movement behaviors. Understanding these movement patterns and the ability to classify the objects will facilitate the assessment of their efficiency and optimize their utility. This project focuses on classifying taxi drivers based on their movements over a specific timeframe. To achieve high accuracy in classifying these drivers, thorough preprocessing of the trajectory data is essential.

To tackle the classification challenge, this project will utilize deep learning algorithms. These algorithms are particularly well-suited for handling trajectory data due to their capacity to process input through multiple layers of a neural network, ultimately generating the desired output. In essence, the modeling process involves complex matrix multiplications among the input data and various network layers, culminating in the final output.

For this project, both Neural Networks and Recurrent Neural Networks (RNNs) will serve as the foundational models for classification. A comparative evaluation of their performance will be conducted based on accuracy metrics derived from the training, validation, and test datasets.

2. Methodology

The dataset is organized in CSV files, with each file corresponding to a specific day between July 1, 2016, and December 26, 2016. Each file contains five columns: plate, longitude, latitude, time, and status. The plate column represents the unique identifier of the taxi driver, denoted by numerical values (0, 1, 2, 3, 4). The longitude and latitude columns provide the GPS coordinates of the taxis. `time` is the timestamp when the taxi is moving. `status` is two states where 1 when the taxi is serving passengers and 0 when the taxi is looking for passengers.

a. Data Processing

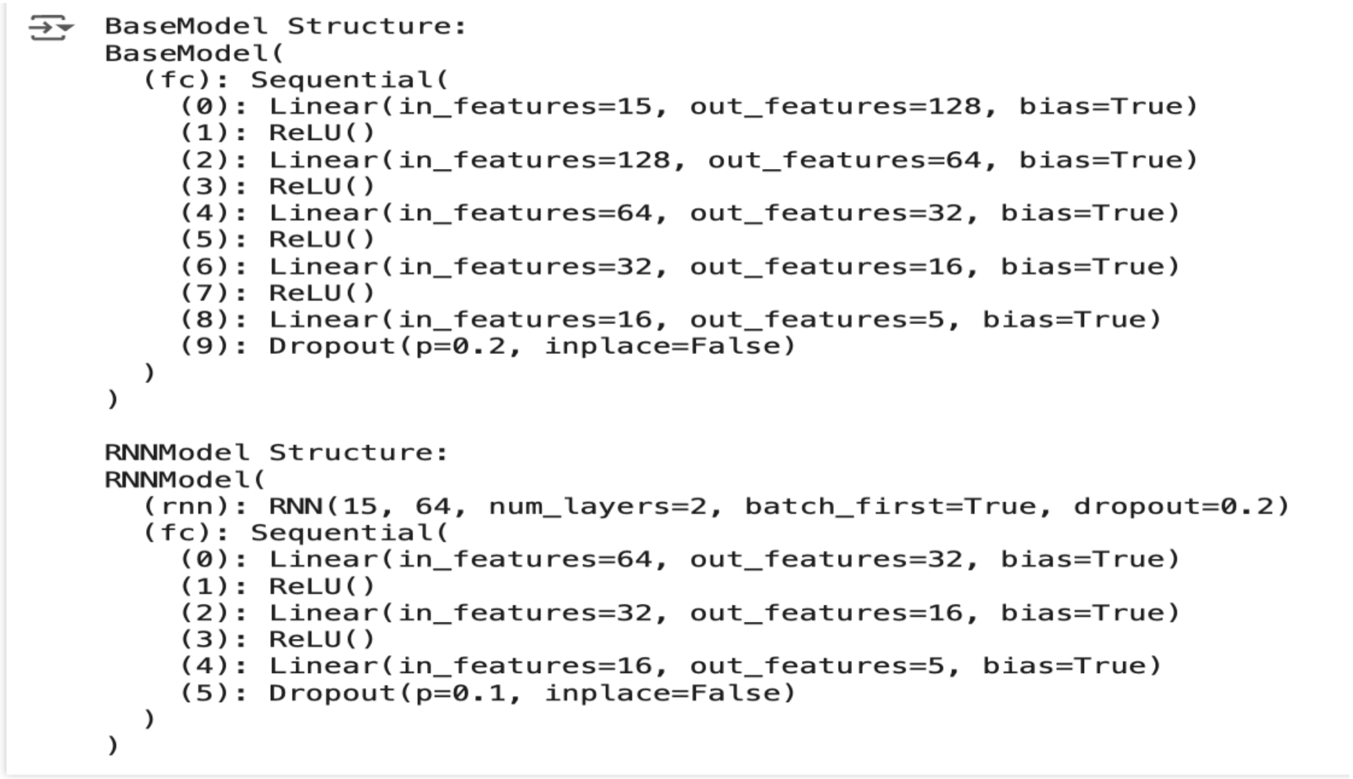
Initially, all the CSV files are consolidated into a single dataset. The next step involves determining the maximum length of the trajectory lists for each driver, which sets a limit based on this maximum value. This ensures that every driver has a uniform length for their trajectory lists. Following this, the labels and features are separated to facilitate the feature engineering process, allowing for the creation of additional features.

b. Feature Generation

To capture the variations in taxi drivers' movement patterns, it is essential to generate additional features that reflect both the GPS coordinates and the time aspects. For the longitude and latitude data, Z-standardization is applied for normalization. The time column undergoes cyclical encoding, where its components—month, day, week, hour, minute, and second—are extracted. Sin and cos transformations are then applied to these components. This encoding method aims to account for the influence of time on taxi drivers' movement behaviors, effectively bridging the gap between the end of one day and the start of the next. For instance, the movement patterns at 11 PM and 1 AM are transformed into a continuous circular pattern, minimizing their differences. Ultimately, the feature engineering process will yield fifteen features, which are expected to enhance the accuracy of the subsequent classification modeling.

c. Network Structure

The project utilizes two distinct network structures: the base model and the RNN model. In the base model, a series of linear layers are combined with non-linear layers, and some layers are dropped out to reduce the number of features as the data flows through them. In contrast, the RNN model retains both linear and non-linear layers but also incorporates an additional RNN layer to capture sequential dependencies.



```
BaseModel Structure:
BaseModel(
  (fc): Sequential(
    (0): Linear(in_features=15, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=32, bias=True)
    (5): ReLU()
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU()
    (8): Linear(in_features=16, out_features=5, bias=True)
    (9): Dropout(p=0.2, inplace=False)
  )
)

RNNModel Structure:
RNNModel(
  (rnn): RNN(15, 64, num_layers=2, batch_first=True, dropout=0.2)
  (fc): Sequential(
    (0): Linear(in_features=64, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=16, bias=True)
    (3): ReLU()
    (4): Linear(in_features=16, out_features=5, bias=True)
    (5): Dropout(p=0.1, inplace=False)
  )
)
```

d. Training and Validation Process

The dataset is split into training and validation sets using PyTorch's built-in function SubsetRandomSampler. A random selection process is employed, with the validation set accounting for 20% of the total dataset.

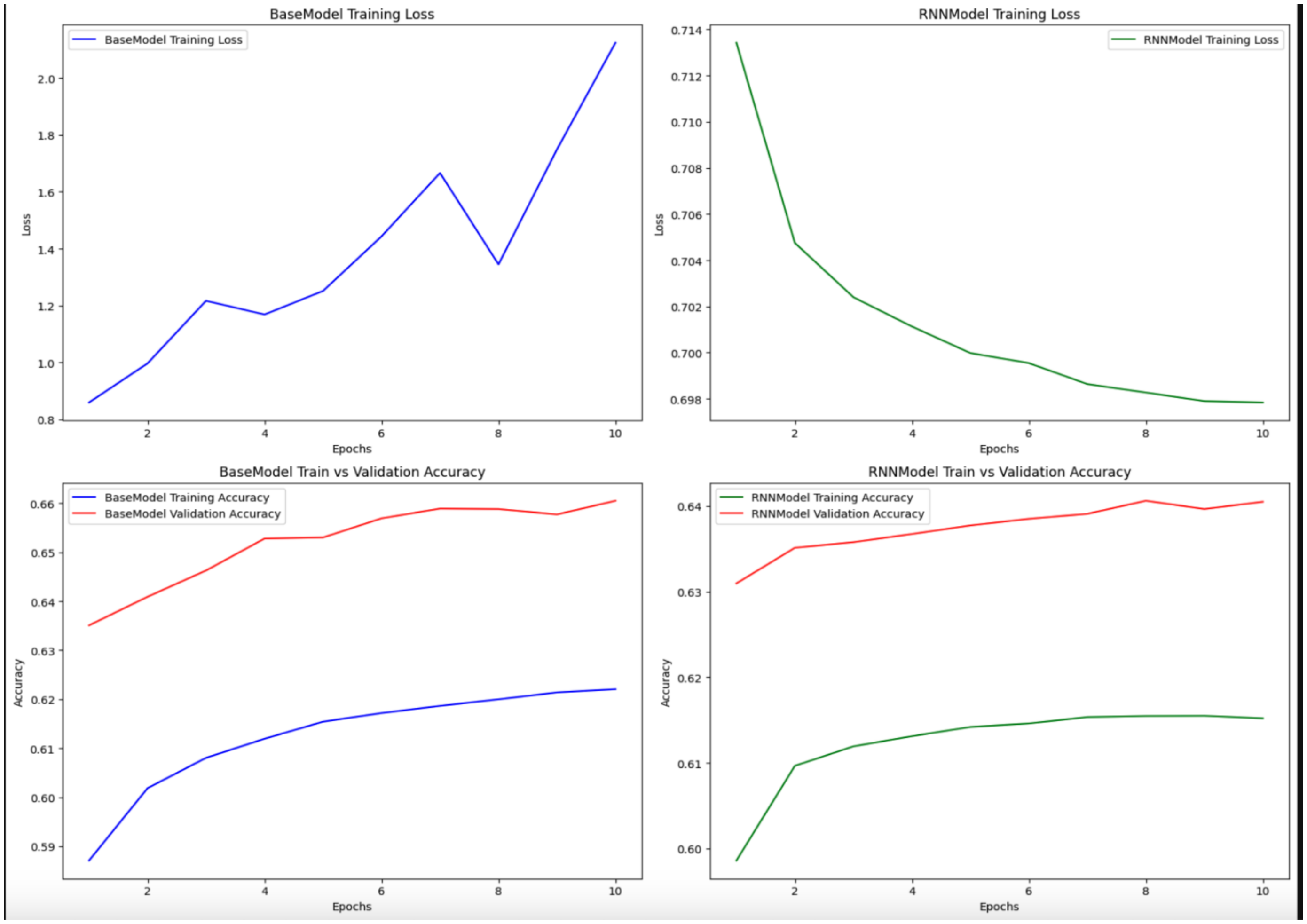
3. Evaluation and Results

a. Training and Validation Results

Plot Explanations

The results are presented in two sets of plots: individual model performance (Image 1) and comparative performance (Image 2).

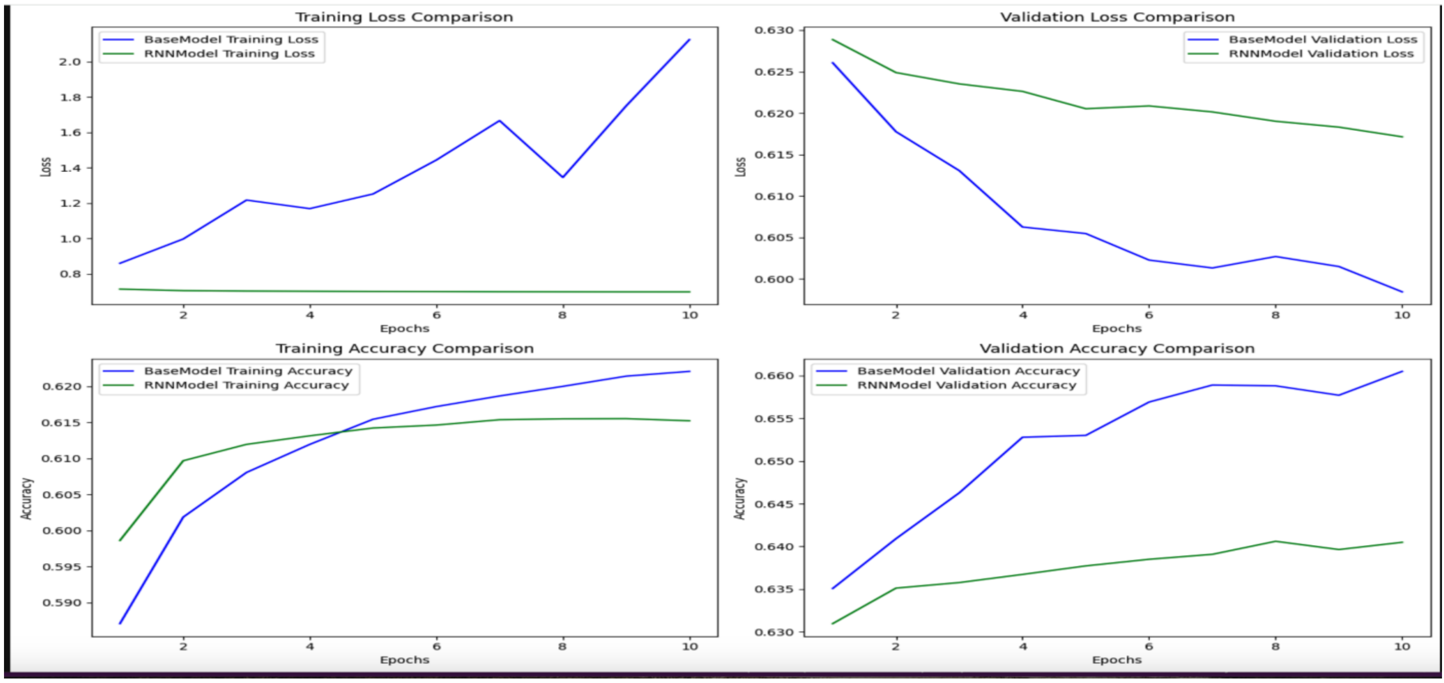
Image 1: Individual Model Performance



This image contains four plots, two for each model (BaseModel and RNNModel):

1. **BaseModel Training Loss**:
   - Blue line graph showing an increasing trend from ~0.8 to ~2.2 over 10 epochs.
   - The upward trend suggests potential overfitting to the training data.
2. **BaseModel Train vs Validation Accuracy**:
   - Blue line: Training accuracy, increasing from ~59% to ~62%.
   - Red line: Validation accuracy, increasing more significantly from ~63.5% to ~66%.
   - The gap between training and validation accuracy narrows, indicating good generalization.
3. **RNNModel Training Loss**:
   - Green line graph showing a decreasing trend from ~0.714 to ~0.698 over 10 epochs.
   - The downward trend indicates effective learning and potential for further improvement.
4. **RNNModel Train vs Validation Accuracy**:
   - Green line: Training accuracy, increasing from ~59.8% to ~61.5%.
   - Red line: Validation accuracy, slightly increasing from ~63% to ~64%.
   - The consistent gap between training and validation accuracy suggests room for improvement in model generalization.

Image 2: Comparative Performance



This image contains four plots comparing both models:

1. **Training Loss Comparison**:
    - Blue line (BaseModel): Shows an increasing trend from ~0.8 to ~2.2.
    - Green line (RNNModel): Shows a stable, slightly decreasing trend from ~0.714 to ~0.698.
    - The contrasting trends highlight the different learning dynamics of the two models.
2. **Validation Loss Comparison**:
    - Blue line (BaseModel): Shows a decreasing trend from ~0.625 to ~0.6.
    - Green line (RNNModel): Shows a slightly decreasing trend from ~0.628 to ~0.617.
    - Both models show improvement, with the BaseModel having a slight edge in final validation loss.
3. **Training Accuracy Comparison**:
    - Blue line (BaseModel): Increases from ~58.7% to ~62.3%.
    - Green line (RNNModel): Increases from ~59.8% to ~61.5%.
    - The BaseModel shows slightly better improvement in training accuracy.
4. **Validation Accuracy Comparison**:
    - Blue line (BaseModel): Increases significantly from ~63.5% to ~66%.
    - Green line (RNNModel): Increases modestly from ~63% to ~64%.
    - The BaseModel demonstrates superior improvement in validation accuracy.

BaseModel

1. **Training Loss**:
    - Started at ~0.8 and increased to ~2.2 over 10 epochs.
    - The increasing trend suggests potential overfitting to the training data.
2. **Validation Loss**:
    - Began at ~0.625 and decreased to ~0.6 over 10 epochs.
    - The decreasing trend indicates good generalization to unseen data.
3. **Training Accuracy**:
    - Improved from ~59% to ~62% over 10 epochs.
    - Shows modest but consistent improvement.
4. **Validation Accuracy**:
    - Increased significantly from ~63.5% to ~66% over 10 epochs.
    - Demonstrates effective learning and generalization.

RNNModel

1. **Training Loss**:
    - Decreased steadily from ~0.714 to ~0.698 over 10 epochs.
    - Indicates an effective learning process with potential for further improvement.

2. **Validation Loss**:
   - Slightly decreased from ~0.628 to ~0.617 over 10 epochs.
   - Suggests some generalization to unseen data, but less pronounced than the BaseModel.
3. **Training Accuracy**:
   - Improved from ~59.8% to ~61.5% over 10 epochs.
   - Shows steady and significant improvement.
4. **Validation Accuracy**:
   - Increased slightly from ~63% to ~64% over 10 epochs.
   - Indicates positive learning, though less pronounced than the BaseModel.

b. Performance Comparison

1. **Validation Accuracy**:
   - BaseModel: Achieved higher final accuracy at 66%.
   - RNNModel: Reached 64% accuracy.
   - The BaseModel outperformed the RNNModel in terms of final validation accuracy.
2. **Training Stability**:
   - BaseModel: Showed increasing training loss, potentially indicating overfitting.
   - RNNModel: Demonstrated more stable training with consistently decreasing loss.
3. **Generalization**:
   - Both models showed improvements in validation accuracy, indicating successful learning and generalization to unseen data.
   - The BaseModel exhibited a more pronounced improvement in validation accuracy compared to the RNNModel.
4. **Learning Rate**:
   - The RNNModel showed a more gradual and consistent improvement in both training and validation metrics.
   - The BaseModel had more dramatic changes, particularly in its training loss.
5. **Loss Dynamics**:
   - The contrasting trends in training loss (increasing for BaseModel, decreasing for RNNModel) suggest different learning dynamics and potential areas for optimization in each model.

c. Hyperparameters

BaseModel Hyperparameters

1. **Architecture**:
    - Input size: 15
    - Hidden layers: 4 (128 -> 64 -> 32 -> 16 neurons)
    - Output size: 5 (for 5-class classification)
2. **Activation function**: ReLU
3. **Dropout rate**: 0.2 (20%)
4. **Loss function**: CrossEntropyLoss
5. **Optimizer**: Adam
6. **Learning rate**: 0.001
7. **Batch size**: 20
8. **Number of epochs**: 10

RNNModel Hyperparameters

1. **Architecture**:
    - Input size: 15
    - RNN hidden size: 64
    - Number of RNN layers: 2
    - Fully connected layers: 2 (32 -> 16 neurons)
    - Output size: 5 (for 5-class classification)
2. **Activation function**: ReLU (in fully connected layers)
3. **Dropout**: 0.2 in RNN, 0.1 in fully connected layer
4. **Loss function**: CrossEntropyLoss
5. **Optimizer**: Adam
6. **Learning rate**: 0.001
7. **Batch size**: 20
8. **Number of epochs**: 10

4. Conclusion

Based on the evaluation, results, and plot analysis, we can draw the following conclusions:

1. **Model Performance**:
    - The BaseModel demonstrated superior performance in terms of final validation accuracy (66% vs 64% for the RNNModel).
    - However, the RNNModel showed more stable training characteristics with a consistently decreasing loss.
2. **Generalization**:
    - Both models showed improvements in validation accuracy over the training period, indicating successful learning and generalization to unseen data.

   - The BaseModel exhibited a more significant improvement in validation accuracy, suggesting it may have a slight edge in generalization capability for this dataset and task.
3. **Training Dynamics**:
   - The increasing training loss of the BaseModel raises concerns about potential overfitting, despite its higher validation accuracy. This suggests that the model might benefit from additional regularization techniques or architecture modifications.
   - The RNNModel's stable training process, with decreasing training and validation loss, indicates a more controlled learning process. However, its slower improvement in validation accuracy suggests there might be room for increased model capacity or learning rate adjustments.
4. **Loss-Accuracy Relationship**:
   - The BaseModel shows an interesting dynamic where training loss increases while both training and validation accuracy improve. This suggests that while the model is becoming more confident in its predictions (higher loss), it's also becoming more accurate.
   - The RNNModel displays a more traditional learning curve, with decreasing loss corresponding to increasing accuracy. This suggests a more stable learning process but potentially slower convergence.
5. **Hyperparameter Considerations**:
   - The similar hyperparameters used for both models allow for a fair comparison of the architectural differences.
   - The differing dropout rates between the models might partially explain the performance differences and could be an area for further tuning.
6. **Future Improvements**:
   - For the BaseModel, exploring techniques to mitigate potential overfitting could lead to even better performance. This could include stronger regularization, early stopping, or architecture simplification.
   - The RNNModel might benefit from increased model capacity, a higher learning rate, or a longer training period to potentially match or exceed the BaseModel's performance.
   - For both models, experimenting with different learning rates, batch sizes, and potentially using learning rate scheduling could lead to improved performance.

7. **Application Considerations**:
    - The choice between these models would depend on the specific requirements of the application. If higher accuracy is the primary concern, the BaseModel appears to be the better choice.
    - However, if training stability and potentially better generalization to very different datasets are important, the RNNModel's characteristics might be preferable.
8. **Extended Training**:
    - Given the trends observed in the plots, it would be interesting to extend the training beyond 10 epochs to see if the BaseModel's validation accuracy continues to improve or if it plateaus due to the increasing training loss.
    - For the RNNModel, extended training might allow it to catch up to or surpass the BaseModel's performance, given its more stable learning trajectory.