Battery State of Charge (SoC) Prediction Using GRU

This project aims to predict the State of Charge (SoC) of a battery using a Gated Recurrent Unit (GRU) neural network. The model is trained and evaluated on a simulated dataset.

Steps

1. Libraries

The project utilizes various libraries for data manipulation, preprocessing, and model training. Key libraries include:
- `pandas` and `numpy` for data manipulation and numerical operations.
- `torch` for building and training the neural network.
- `sklearn` for preprocessing and model evaluation.
- `matplotlib` for plotting training and validation loss curves.

2. Dataset Generation

A simulated dataset is generated to mimic battery readings over 24 hours. The dataset includes voltage, current, concentration, and SoC values.
- **Voltage and Current**: Generated using normal distribution to simulate real battery behavior.
- **Concentration and SoC**: Updated every 15 minutes using the Coulomb counting method, which calculates the total charge over intervals and updates the SoC accordingly.

3. Preprocessing

Preprocessing steps ensure the data is ready for training:
- **Handling Missing Values**: Fill missing SoC values with the mean and drop rows with missing voltage, current, or concentration values.
- **Feature Scaling**: Normalize features using `StandardScaler` to ensure they have a mean of 0 and standard deviation of 1, which helps improve model training performance.
- **Data Conversion**: Convert the preprocessed data into PyTorch tensors, making them suitable for model training.

4. GRU Model Definition

Define a GRU model with dropout to prevent overfitting:
- **GRU Layers**: Capture temporal dependencies in the data.
- **Dropout Layer**: Regularize the model to prevent overfitting.
- **Fully Connected Layer**: Map GRU outputs to the final prediction.

5. Hyperparameter Tuning

Perform grid search with cross-validation to find the best combination of hyperparameters:
- **Hyperparameters**: Include learning rate, batch size, hidden size, number of layers, dropout rate, and weight decay.
- **K-Fold Cross-Validation**: Split the data into training and validation sets multiple times to evaluate the model's performance and ensure it generalizes well.
- **Early Stopping**: Monitor validation loss to stop training if it does not improve for a specified number of epochs, preventing overfitting.

6. Training

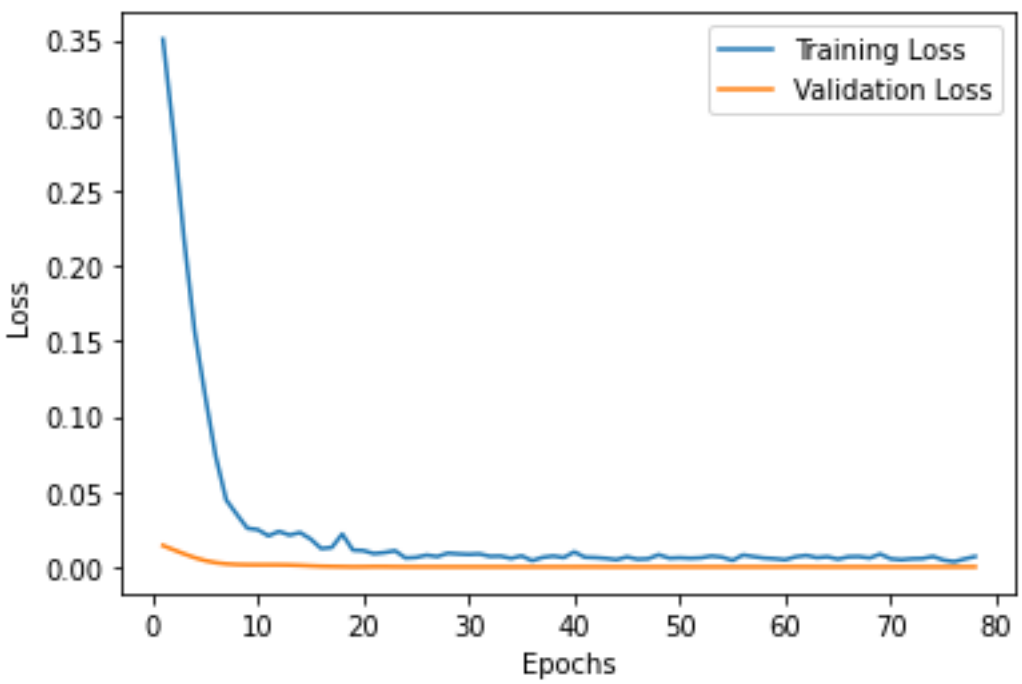
Train the final model using the best hyperparameters identified in the tuning phase:
- **Data Splitting**: Split the data into training and test sets.
- **Model Training**: Train the model on the training set using the optimal hyperparameters. Track training and validation losses to monitor model performance.
- **Early Stopping**: Implement early stopping to prevent overfitting by stopping the training when the validation loss stops improving.

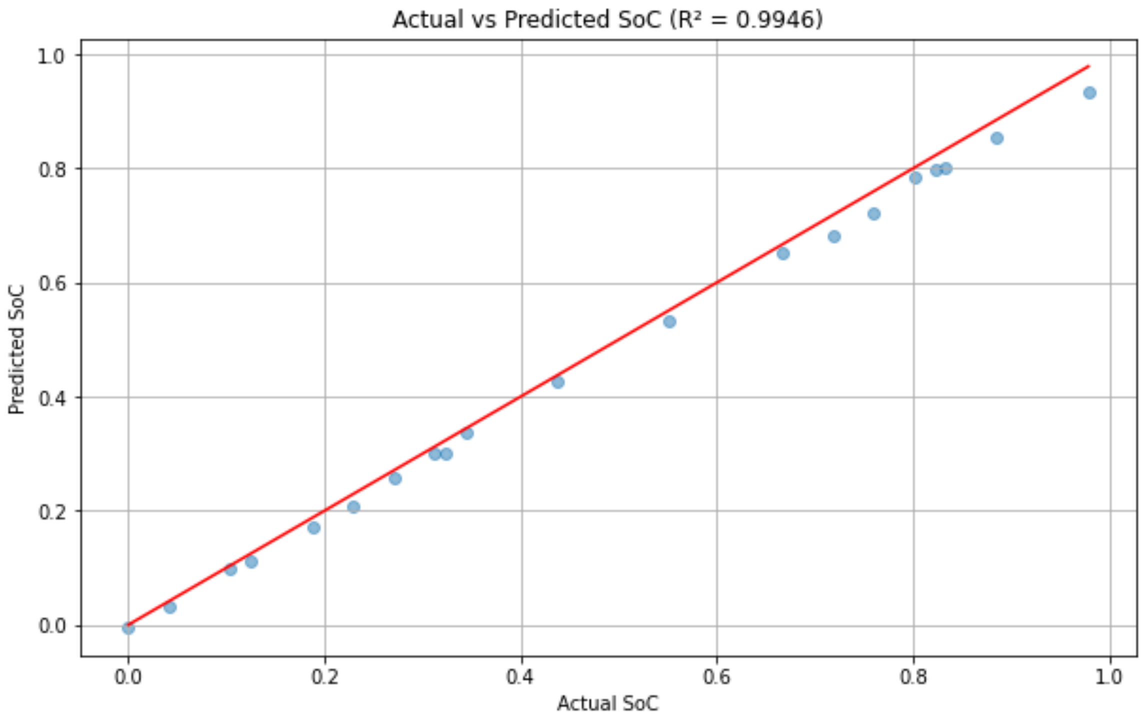
7. Plotting and Evaluation

Evaluate the trained model on the test set and visualize the training process:
- **Loss Curves**: Plot training and validation losses to visualize the model's learning process.
- **Model Evaluation**: Evaluate the model on the test set using Mean Absolute Error (MAE) and R-squared (R^2) metrics to assess its accuracy and goodness of fit.

Conclusion

By following these steps, the project demonstrates how to generate a simulated battery dataset, preprocess the data, define and train a GRU model, and evaluate its performance. The use of hyperparameter tuning and early stopping ensures that the model is well-optimized and generalizes well to new data.




Actual vs Predicted SoC (R² = 0.9946)
1.0
0.8
0.6
0.4
0.2
0.0
Predicted SoC
0.0
0.2
0.4
0.6
0.8
1.0
Actual SoC