Task 3 : RNN

Time series forecasting plays a vital role in many domains, particularly in finance, where anticipating future trends based on historical patterns is crucial for informed decision-making. In this project, you will use a Recurrent Neural Network (RNN) to forecast future Bitcoin prices. RNNs are specifically designed to handle sequential data, allowing them to capture temporal dependencies in financial time series.

You will begin by exploring the historical OHLCV (Open, High, Low, Close, Volume) data and selecting a target feature (specific definition of target mentioned in the data processing section) that serves as an indicator of potential profit or loss. This selected feature will then be used as the prediction target. The goal is to train an RNN model that learns from past sequences and can forecast the target value for future time steps, helping to identify favorable or risky market movements.

<u>Attention: You must analyze each plot in the EDA and Visualization sections.</u>

1- Dataset Loading

Start by loading historical Bitcoin data from the dataset file:

- Dataset name: `BTC-USD.csv`
- Source: The file contains daily Bitcoin prices with columns: Open, High, Low, Close, and Volume.
- Format: Load the data using Pandas or an equivalent tool.
- Explain what each feature is and how calculated.

2- Exploratory Data Analysis (EDA) on OHLCV:

Before training your model, perform exploratory analysis on the full OHLCV dataset to understand trends and relationships.

- Visualize Price Movement: Plot each column (Open, High, Low, Close, Volume) over time.
- Check Correlations: Calculate correlations between columns to see how they influence one another.
- Statistical Summary: Review metrics like min, max, mean, and standard deviation for these features.

3- Data Processing:

Prepare the data from the `BTC-USD.csv` file , focusing on designing a meaningful prediction target and constructing time-aware input sequences.

- Handle Missing Data: Identify and resolve missing or anomalous values.
- Normalization: Normalize features at your discretion.
- Feature Engineering: Using a combination of the OHLCV features , define a custom target that serves as a strong indicator of profit or loss over time. This target could represent a future change or a signal reflecting expected market movement, and it may be a mathematical combination of the extracted features. Your chosen target should be clearly justified based on your initial analysis.
- Sequence Creation: Transform the time series into a supervised format by creating sequences of historical values. For each sample, use a sequence of past data to predict the target at the next time step. This prepares the data for RNN training, where each input sequence represents past market behavior.
- Lookback Period Tuning: The lookback period—how many previous time steps the model considers—greatly affects performance. Try different values (30, 60, 90 days) and evaluate the impact by training models with each setting. This helps identify the optimal window size that captures the right amount of historical context.
- Split the dataset into training, validation, and test sets: Use the validation set for hyperparameter tuning and model selection, and use the test set for final evaluation. In your opinion, why using both a validation and test set is better than relying on just a test set?

4- Model Architecture:

Design a Recurrent Neural Network (RNN) to predict the future value of your target.

- Base Structure: Start with standard RNN layers to model the temporal patterns in your input sequences.
- Output Design: Add one or more fully connected layers after the recurrent layers to produce a single numeric prediction as the model's output.
- Dropout: Incorporating dropout and other regularization techniques is recommended to mitigate overfitting and improve the model's generalization performance.
- Loss and Optimization: Select an appropriate loss function for regression tasks and an efficient optimizer to guide the learning process.

5- Model Training:

Train your RNN model using the prepared training data and evaluate its ability to forecast the selected target.You can use TensorFlow or Torch for implementing your model.

- Training Process: Feed the sequential input data into the model and let it learn to minimize the prediction error over time.
- Monitoring: Track the loss throughout the training process to ensure stable convergence. Watch for signs of underfitting or overfitting.
- Hyperparameter Tuning: Experiment with different model configurations—such as the number of recurrent units, number of layers, batch size, and learning rate—to improve performance. These adjustments can significantly influence the model's ability to generalize to unseen data.

6- Evaluation and Visualization:

Once your model is trained and tested, analyze its performance both numerically and visually to draw meaningful conclusions about its predictive ability.

- Metric reporting: Report key regression metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) on the test set. Discuss the use case of each.
  Next, report the Mean Absolute Percentage Error (MAPE), which expresses the error as a percentage of the actual values, making it especially useful when the target variable's scale varies.
  Finally, report the Cumulative Error (CE), defined as the sum of the differences between actual and predicted values across all test samples. This metric provides insight into the total prediction bias.
- Prediction vs Actual Plot: Visualize the predicted values against the actual target values over time for train and test sets. This comparison helps identify how well the model captures the general trend and turning points in the data.
- Error Trends: Plot the prediction error over time to highlight specific intervals where the model struggles. Analyzing these moments can lead to valuable insights for future refinement.

7- Implementing an LSTM-Based Model (10% Bonus):

As an extension to the standard RNN approach, you may optionally implement an LSTM (Long Short-Term Memory) network. LSTMs are a more advanced type of recurrent neural network designed to capture long-term dependencies and mitigate issues like vanishing gradients.

- Model Architecture: Replace the base RNN layers with one or more LSTM layers. You can adjust the number of units, stack multiple layers, or add dropout for regularization.
- Training and Evaluation: Train your LSTM model using the same data preparation and sequence setup. Like RNN do Evaluation and Visualization part and then compare its performance—both in terms of metrics and visualizations—with your standard RNN model.
- Insight: Reflect on whether the LSTM improved predictive accuracy or generalization. Discuss possible reasons based on the structure of the data and your model design.