GTU Department of Computer Engineering

CSE 655

Homework 3 Report

Emirkan Burak Yılmaz

254201001054

Contents

Dataset Overview

Two datasets were used in this study to train and evaluate stock price prediction models:

Tesla Stock Price Dataset

- Source: henryshan/tesla-stock-price on Kaggle.
- Contents: Daily Tesla stock data including **Open**, **High**, **Low**, **Close**, and **Volume**.
- Usage: The **Close** price was used as the target variable for prediction.

Gold Price Dataset

- Source: franciscogcc/financial-data on Kaggle.
- Contents: Contains multiple financial time series including the gold closing price.
- Usage: Only the **gold close** column was retained, renamed to Gold.

```
import pandas as pd

tesla_df = pd.read_csv(tesla_data_path, parse_dates=["Date"])
print(tesla_df.head())

gold_df = pd.read_csv(financial_data_path, parse_dates=["date"])

print(gold_df.head())
```

```
        Date      Open      High       Low     Close  Adj Close     Volume
0 2010-06-29  1.266667  1.666667  1.169333  1.592667   1.592667  281494500
1 2010-06-30  1.719333  2.028000  1.553333  1.588667   1.588667  257806500
2 2010-07-01  1.666667  1.728000  1.351333  1.464000   1.464000  123282000
3 2010-07-02  1.533333  1.540000  1.247333  1.280000   1.280000   77097000
4 2010-07-06  1.333333  1.333333  1.055333  1.074000   1.074000  103003500
        date  sp500 open  sp500 high  sp500 low  sp500 close  sp500 volume  \
0 2010-01-14      114.49      115.14     114.42       114.93   115646960.0
1 2010-01-15      114.73      114.84     113.20       113.64   212252769.0
2 2010-01-18         NaN         NaN        NaN          NaN           NaN
3 2010-01-19      113.62      115.13     113.59       115.06   138671890.0
4 2010-01-20      114.28      114.45     112.98       113.89   216330645.0

   sp500 high-low  nasdaq open  nasdaq high  nasdaq low  ...  palladium high  \
0            0.72        46.26       46.520       46.22  ...           45.02
1            1.64        46.46       46.550       45.65  ...           45.76
2             NaN          NaN          NaN         NaN  ...             NaN
3            1.54        45.96       46.640       45.95  ...           47.08
4            1.47        46.27       46.604       45.43  ...           47.31

   palladium low  palladium close  palladium volume  palladium high-low  \
0          43.86            44.84          364528.0                1.16
1          44.40            45.76          442210.0                1.36
2            NaN              NaN               NaN                 NaN
3          45.70            46.94          629150.0                1.38
4          45.17            47.05          643198.0                2.14

   gold open  gold high  gold low  gold close  gold volume
0     111.51     112.37    110.79      112.03   18305238.0
1     111.35     112.01    110.38      110.86   18000724.0
2        NaN        NaN       NaN         NaN          NaN
3     110.95     111.75    110.83      111.52   10467927.0
4     109.97     110.05    108.46      108.94   17534231.0
```

Figure 1: Tesla and Gold dataset overview

Preprocessing Steps

- **Date parsing:** Dates were parsed and aligned in both datasets.
- **Interpolation:** Missing gold prices were **linearly interpolated** to ensure continuity.
- **Merging:** The Tesla and gold datasets were merged on the Date column using an inner join to align records.
- **Final Dataset:** The resulting dataframe was indexed by date and used as input for model training.

```python
# Keep only required columns
gold_df = gold_df[['date', 'gold close']]
gold_df.columns = ['Date', 'Gold']
print(gold_df.head())

# Sort by date to prepare for interpolation
gold_df.sort_values('Date', inplace=True)

# Interpolate missing gold prices linearly
gold_df['Gold'] = gold_df['Gold'].interpolate(method='linear')

# Fill any remaining NaNs at the start or end of the series
gold_df['Gold'] = gold_df['Gold'].bfill().ffill()

# Merge on Date
tesla_gold_df = pd.merge(tesla_df, gold_df, on='Date', how='inner')
tesla_gold_df.set_index('Date', inplace=True)

print(tesla_gold_df.head())
```

Figure 2: Merging Tesla and Gold datasets

This preprocessing ensured consistent temporal alignment and completeness of the input features for both LSTM and Transformer models.

Part 1: Prediction with LSTM

Overview

This part implements a Long Short-Term Memory (LSTM) model to forecast Tesla's stock closing prices based on historical time-series data. The dataset includes multiple financial features such as Open, High, Low, Close, and Volume, with an optional inclusion of the Gold price as an external economic indicator. The LSTM model is constructed using two stacked LSTM layers followed by dense layers. A sliding window approach with a **look-back period of 60 days** is used to generate sequential input samples. Data is normalized using MinMax scaling, and time-based splitting is employed to ensure the temporal integrity of the training and testing sets. The training process incorporates early stopping, learning rate scheduling, and model checkpointing to enhance performance and prevent overfitting. Predictions are evaluated using **MSE**, **MAE**, and **RMSE**, and the predicted prices are inverse transformed for interpretability.

```python
def create_sequences(data, target_index, look_back=60):
    X, y = [], []
    for i in range(look_back, len(data)):
        X.append(data[i - look_back:i])
        y.append(data[i, target_index])
    return np.array(X), np.array(y)

def preprocess_data(df, feature_cols, target_col, look_back=60, test_ratio=0.3):
    # Step 1: Normalize all features
    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(df[feature_cols])
    target_index = feature_cols.index(target_col)

    # Step 2: Create sequences
    X, y = create_sequences(scaled, target_index=target_index, look_back=60)

    # Step 3: Train/Test split
    # In time-series forecasting, we do NOT shuffle or random split the data because
    # shuffling breaks the temporal order, making predictions meaningless
    split = int(len(X) * (1 - test_ratio))
    X_train, X_test = X[:split], X[split:]
    y_train, y_test = y[:split], y[split:]

    return X_train, X_test, y_train, y_test, scaler, target_index

def build_LSTM_model(input_shape, lstm_units=64):
    model = Sequential([
        Input(shape=input_shape),
        LSTM(lstm_units, return_sequences=True),
        LSTM(lstm_units),
        Dense(25),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    return model
```

Figure 3: Data preprocessing and LSTM model construction pipeline

Results

Tesla-Only LSTM Model

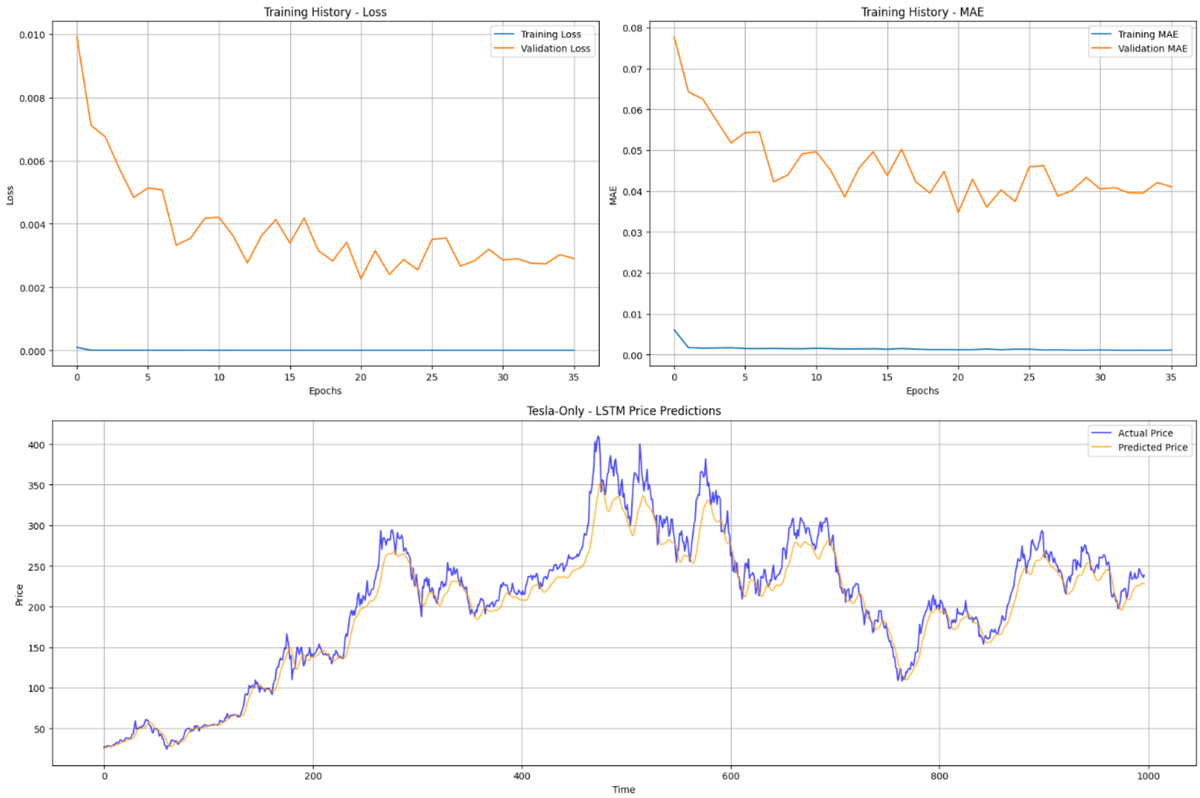

Figure 4: Training and validation loss curves for Tesla-only LSTM predictions

- Mean Squared Error (MSE): **378.8101**
- Mean Absolute Error (MAE): **14.2168**
- Root Mean Squared Error (RMSE): **19.4630**

The model exhibited rapid convergence in training loss, indicating efficient learning of the training set's patterns. However, the validation loss demonstrated persistent fluctuations, suggesting potential challenges in generalizing to unseen data.

Tesla + Gold LSTM Model

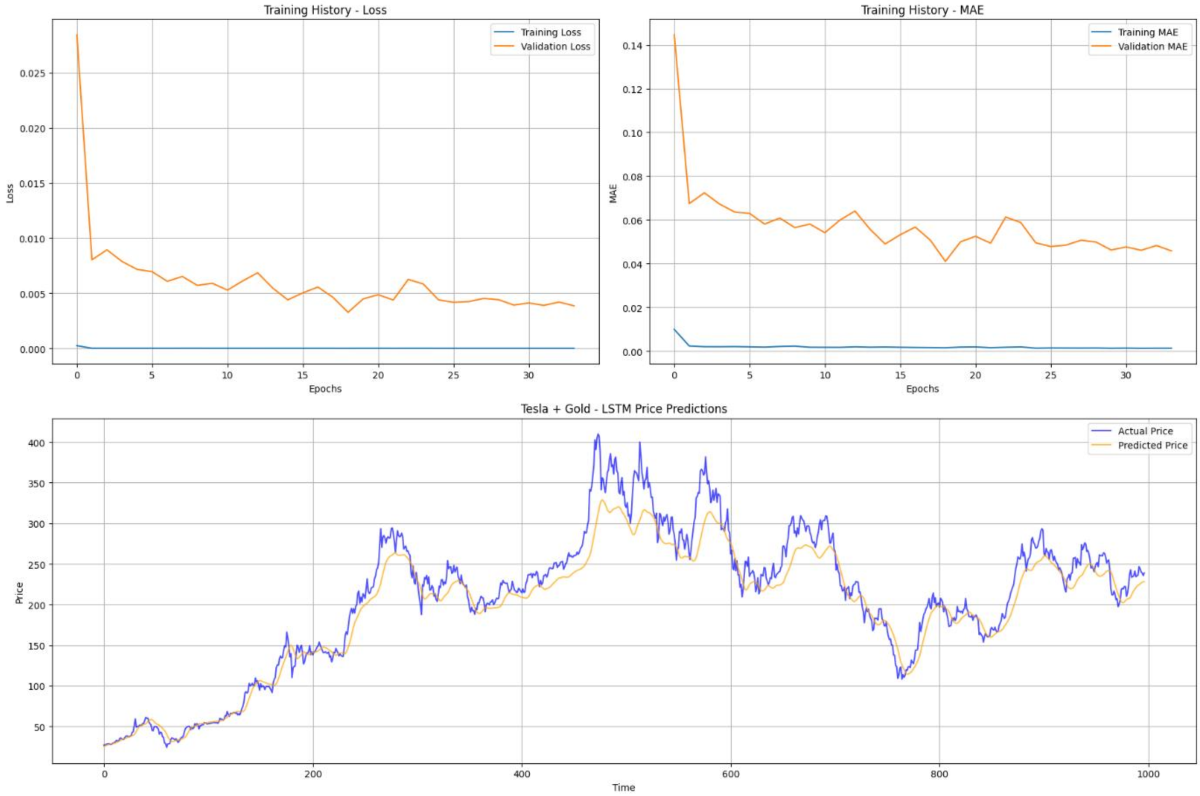

Figure 5: Training and validation loss curves for Tesla + Gold LSTM predictions

- Mean Squared Error (MSE): **545.9234**
- Mean Absolute Error (MAE): **16.7577**
- Root Mean Squared Error (RMSE): **23.3650**

Although training loss remained low, the validation loss was consistently higher than in the Tesla-only model. The predicted price curve appears to lag behind actual values more frequently and shows slightly worse alignment during volatile periods.

Comments

Adding gold price as an additional input did **not improve** the LSTM model's performance—in fact, it slightly degraded it. This is evident from the increase in all error metrics (MSE, MAE, RMSE) compared to the Tesla-only model. Possible reasons include:

- Weak correlation: Gold prices might not have a strong direct relationship with Tesla's stock movements.
- Noise introduction**:** The gold data may have introduced unrelated variance, confusing the model.

Overall, the Tesla-only LSTM model produced better and more stable predictions. It suggests that, for this task, Tesla stock prices alone provide sufficient temporal structure for LSTM-based forecasting without needing auxiliary features like gold prices.

Part 2: Prediction with Transformers

Overview

This part applies a Transformer-based neural network to the same time-series forecasting task, aiming to predict Tesla's closing stock prices. The model leverages **self-attention mechanisms** through a custom Transformer encoder block, which captures long-range temporal dependencies more efficiently than traditional recurrent structures. The architecture includes **multi-head attention**, **layer normalization**, **feed-forward networks**, and **dropout** for regularization. Global average pooling is used to reduce the temporal dimension before the output layers. Similar to the LSTM setup, the dataset is normalized and sequential inputs are generated with a **look-back window of 60**. Two versions of the model are trained: one using only Tesla-related features and the other including the gold price as an additional input. The model is trained with callbacks such as early stopping and learning rate reduction, and its performance is assessed using standard error metrics.

```python
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    x = LayerNormalization(epsilon=1e-6)(inputs)
    x = MultiHeadAttention(key_dim=head_size, num_heads=num_heads, dropout=dropout)(x, x)
    x = Dropout(dropout)(x)
    res = x + inputs

    x = LayerNormalization(epsilon=1e-6)(res)
    x = Dense(ff_dim, activation="relu")(x)
    x = Dropout(dropout)(x)
    x = Dense(inputs.shape[-1])(x)
    return x + res


def build_transformer_model(input_shape, head_size=256, num_heads=4, ff_dim=4, dropout=0.2):
    inputs = Input(shape=input_shape)
    x = transformer_encoder(inputs, head_size=head_size, num_heads=num_heads, ff_dim=ff_dim, dropout=dropout)
    x = GlobalAveragePooling1D(data_format='channels_first')(x)
    x = Dropout(dropout)(x)
    x = Dense(20, activation="relu")(x)
    outputs = Dense(1, activation="linear")(x)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(
        optimizer='adam',
        loss="mean_squared_error",
        metrics=["mae"]
    )
    model.summary()
    return model
```

Figure 6: Transformer model architecture with attention mechanisms

Results

Tesla-Only Transformer Model

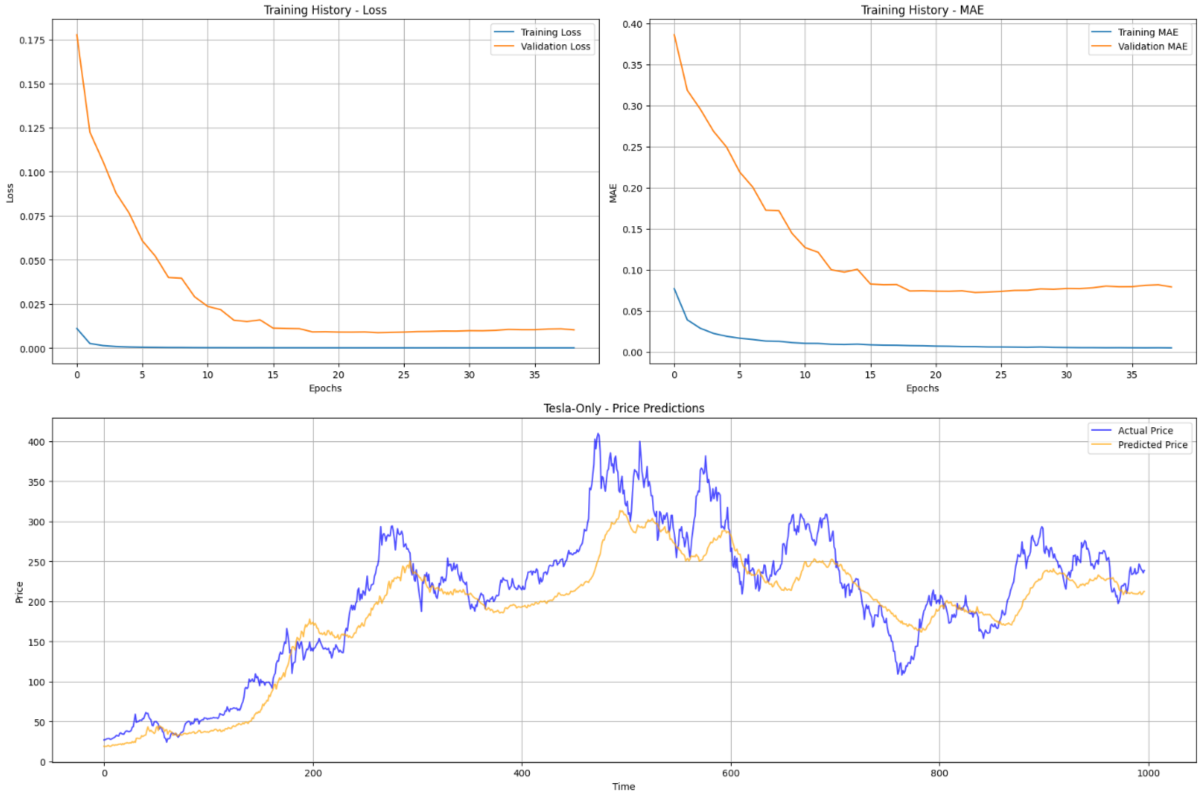

Figure 7: Training and validation loss curves for Tesla-only Transformer predictions

- Mean Squared Error (MSE): **1448.8523**
- Mean Absolute Error (MAE): **29.4764**
- Root Mean Squared Error (RMSE): **38.0638**

The training and validation loss curves showed steady convergence, though with higher overall error compared to the LSTM models. The predicted prices follow the general trend of the actual Tesla stock but with more pronounced lag and smoothing, especially during rapid price movements.

Tesla + Gold Transformer Model

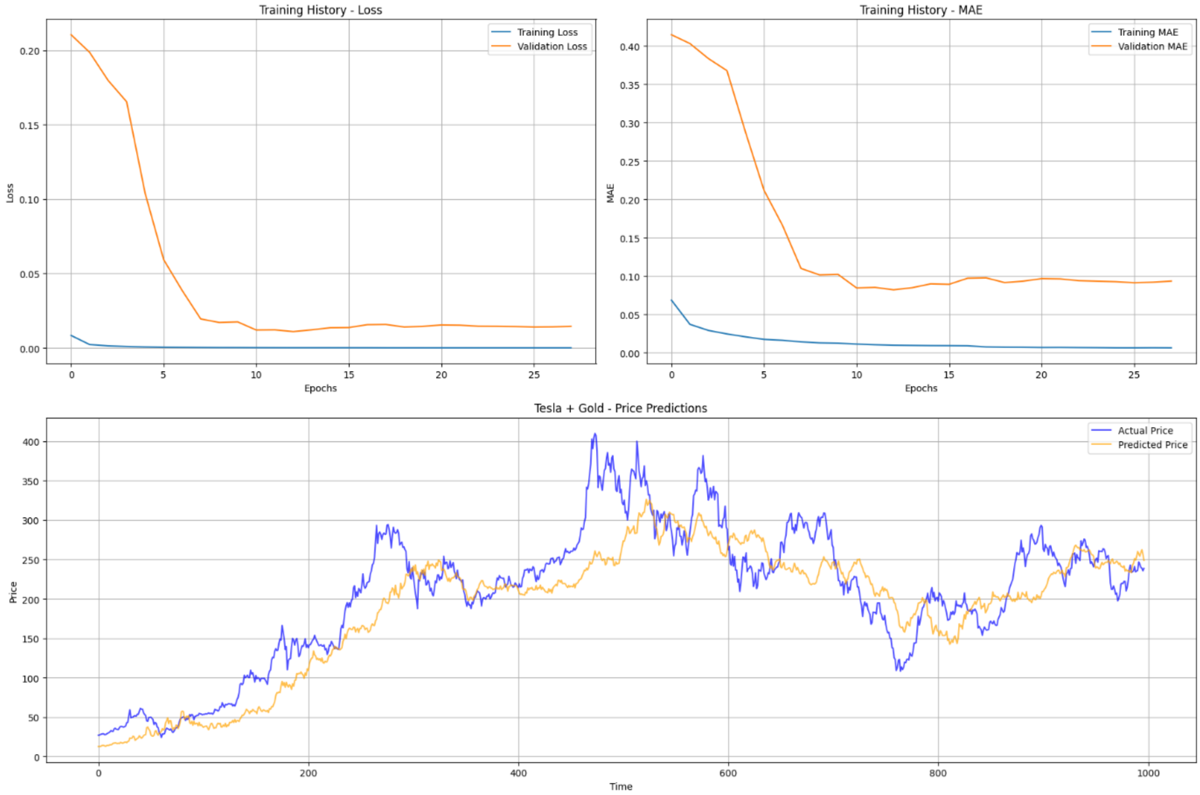

Figure 8: Training and validation loss curves for Tesla + Gold Transformer predictions

- Mean Squared Error (MSE): **1837.0724**
- Mean Absolute Error (MAE): **33.6029**
- Root Mean Squared Error (RMSE): **42.8611**

Adding the gold price feature did **not lead to improvement**. Validation loss plateaued early, and the predicted price trajectory became more biased toward the mean, further missing high volatility regions.

Comments

The Transformer models underperformed compared to the LSTM models in this stock prediction task. Both the Tesla-only and Tesla+Gold Transformer configurations yielded significantly higher MSE, MAE, and RMSE values than their LSTM counterparts. Key observations:

- Temporal modeling challenge: Transformers may require more data and longer training to effectively model fine-grained temporal dependencies in financial data.
- No benefit from gold feature: Similar to Part 1, incorporating gold prices again worsened performance. This reaffirms that gold prices might not hold useful predictive power for Tesla's short-term stock movement.

While Transformers offer strong modeling capacity for long-range dependencies, their use in short time series with **limited features** and **data volume**—like in this assignment—may not be optimal without additional tuning, data augmentation, or architectural enhancements.

Final Comments

Model Performance Summary

Across all tested configurations, **the LSTM models consistently outperformed the Transformer models** in predicting Tesla stock prices. This held true for both input settings—using only Tesla stock prices and using Tesla prices combined with gold prices.

Tesla-only Models

The LSTM model yielded substantially lower error metrics compared to the Transformer, with RMSE nearly halved (19.46 vs 38.06).

This suggests that the LSTM architecture is better suited to capturing the short-term temporal dependencies present in the stock price data, particularly given the size and structure of the dataset.

Tesla + Gold Models

Similarly, the LSTM model with gold input also outperformed the Transformer variant (RMSE: 23.37 vs 42.86).

However, adding gold prices led to degraded performance for both LSTM and Transformer models. This implies that gold, in this context, did not provide additional predictive value and may have introduced noise.

Concluding Insights

- **LSTM is the preferred model** for this task due to its superior handling of short-to-medium term time series patterns in smaller datasets.
- **Transformer models may require more data, more features, or additional tuning** (e.g., positional encoding adjustments, longer input sequences) to match or exceed LSTM performance.
- **Adding external features like gold prices** should be approached with caution, and only after validating their correlation or predictive utility.