RNN-Based Time Series Forecasting: Alcohol Sales Analyzer

Report

Ayush Kundu

M.Sc. Statistics
Department of Mathematics & Statistics
IIT Kanpur

Contents

1 Introduction

Forecasting consumer demand for alcoholic beverages is of critical importance to producers, distributors, and retailers alike. Accurate sales predictions enable effective inventory management, reduce stockouts and overstock situations, and optimize the supply chain from production to point of sale. In the context of monthly alcohol sales data provided by the Federal Reserve Economic Data (FRED), insights into underlying trends and seasonality can improve financial planning, marketing strategies, and resource allocation across the entire industry.

Time series forecasting poses unique challenges due to temporal dependencies, non-statio narity, and complex seasonal patterns. Traditional statistical methods such as ARIMA and exponential smoothing capture linear trends and seasonality but often struggle with non-linear behaviors inherent in real-world sales data. Recurrent Neural Networks (RNNs), and in particular Long Short-Term Memory (LSTM) architectures, are designed to learn long-term dependencies and non-linear relationships in sequential data. Their gated memory cells allow the model to retain information over extended time horizons, making them well-suited to capture both long-term trends and short-term fluctuations in alcohol sales.

The primary objective of this work is to develop and evaluate an LSTM-based forecasting model for monthly alcohol sales. We aim to (1) preprocess and normalize the historical sales series, (2) design and train an LSTM network to predict future sales values, (3) assess model performance using appropriate error metrics (mean squared error and mean absolute error), and (4) analyze the model's ability to capture seasonality and sudden shifts. Finally, we discuss the strengths and limitations of the LSTM approach in this domain and propose avenues for future enhancement, such as incorporating exogenous variables or comparing against classical forecasting baselines.

2 Objective

The primary objective of this project is to build and evaluate a deep learning–based forecasting model that can accurately predict future monthly alcohol sales using historical time series data. To achieve this, we pursue the following specific goals:

1. **Data Preparation and Transformation:**
   - Parse and clean the raw monthly sales figures from the FRED CSV dataset.
   - Normalize the series using a MinMax scaler to ensure stable training of the neural network.
   - Construct overlapping sequences (windows) of past sales values to serve as inputs for the LSTM model.
2. **Model Design:**
   - Implement an `LSTMnetwork` class in PyTorch, comprising a single LSTM layer with 100 hidden units followed by a fully connected output layer.
   - Select appropriate loss functions (mean squared error) and optimizers (Adam with learning rate 0.01) for regression training.
3. **Training and Hyperparameter Tuning:**
   - Train the model for 100 epochs on the training split, monitoring the MSE loss curve to detect convergence and potential overfitting.
   - Experiment with different sequence window lengths (e.g., 6, 12, 24 months) and hidden layer sizes to identify the combination that yields the best validation performance.
4. **Forecasting and Evaluation:**
   - Generate one-step-ahead forecasts for the test period by rolling the trained model forward through the hold-out data.
   - Inverse-transform the predictions to the original sales scale and compute quantitative metrics, including test-set MSE and MAE.
   - Visualize the predicted vs. actual sales values to qualitatively assess the model's ability to capture trend and seasonality.
5. **Analysis and Interpretation:**
   - Analyze periods where the model under- or over-estimates sales, relating errors to abrupt market shifts or seasonal peaks.
   - Discuss the strengths of the LSTM architecture in retaining long-term dependencies, as well as its limitations in extrapolating beyond the range of historical data.

6. **Future Extensions:**

   - Propose incorporating exogenous variables (e.g., holiday indicators, economic indices) to improve forecast accuracy.
   - Suggest benchmarking against classical time series models (ARIMA, SARIMA, Prophet) and more advanced deep learning architectures (stacked LSTM, Transformers).

3 Dataset Description

The dataset used in this study was obtained from the Federal Reserve Economic Data (FRED) repository and consists of monthly alcohol sales figures in the United States. The time series spans from January 1992 through January 2019, yielding a total of 325 observations. Each record contains:

- `DATE`: the first day of the month, in `YYYY-MM-DD` format.
- `S4248SM144NCEN`: total alcohol sales for that month, reported in millions of U.S. dollars.

A summary of the dataset is given in Table 1.

Attribute	Description	Example
`DATE`	Month (start date)	1992-01-01
`S4248SM144NCEN`	Sales (USD millions)	3459

Table 1: Structure of the monthly alcohol sales dataset.

Figure 1 illustrates the complete time series of alcohol sales over the 27-year period. The series exhibits clear upward trends and seasonal fluctuations, with recurrent peaks and troughs aligned roughly on an annual cycle.

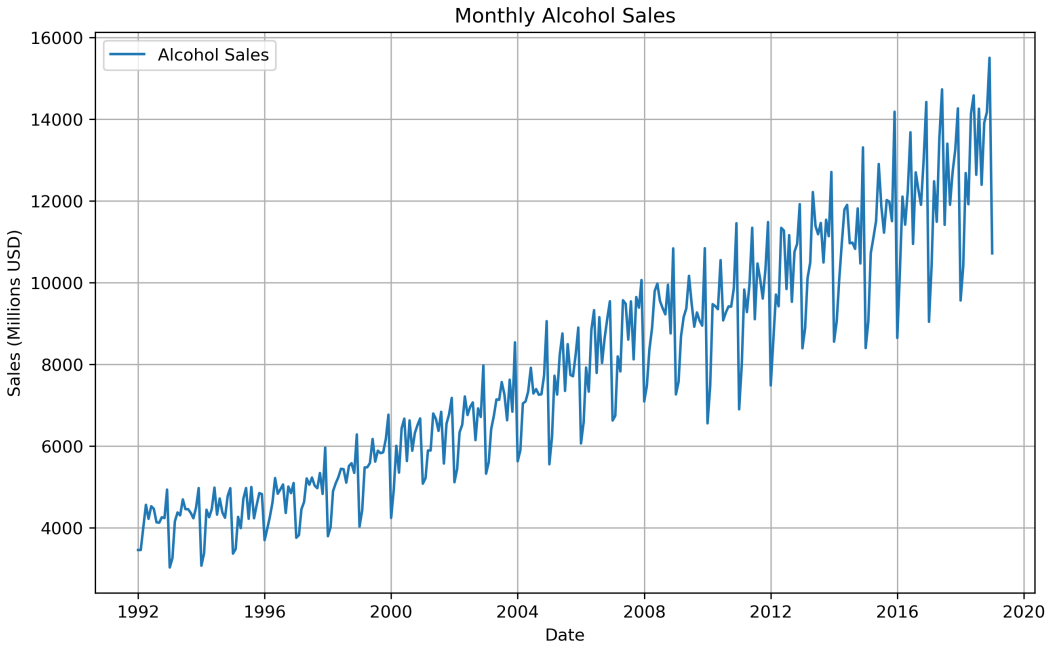

Figure 1: Monthly alcohol sales in the U.S. (January 1992–January 2019).

Descriptive statistics for the sales values are shown in Table 2. The mean monthly sales over the period are approximately \$7,084 million, with a standard deviation of \$1,781 million.

Statistic	Value (millions USD)
Count	325
Mean	7084
Std	1781
Min	3458
25%	5702
50%	7004
75%	8301
Max	10717

Table 2: Descriptive statistics of monthly alcohol sales.

This dataset is well-suited for time series modeling: it is sufficiently long to capture multi-year seasonality and trend, and it contains no missing values, simplifying preprocessing. The clear seasonal pattern and upward trend present ideal conditions for evaluating the capability of RNN/LSTM architectures to learn and forecast real-world demand series.

4 Data Preprocessing

Before applying any machine learning models, it is essential to preprocess the dataset to ensure that the data is in a suitable format for time series forecasting. The preprocessing steps for the monthly alcohol sales dataset are described below.

4.1 Datetime Indexing

The original dataset, obtained in CSV format, contains two columns: a date field (implicitly stored as the index) and a corresponding sales figure in millions of USD. We explicitly parsed the index as datetime objects to facilitate time series operations.

```
df = pd.read_csv('Alcohol_Sales.csv', index_col=0, parse_dates=True)
```

This ensures that the dataset is recognized as a time series, enabling resampling, shifting, and other temporal operations.

4.2 Data Inspection and Cleaning

The dataset was first inspected for null or missing values:

```
df.isnull().sum()
```

No missing values were found, which simplifies model preparation. We also verified the consistency of the time frequency using:

```
df.index.to_series().diff().value_counts()
```

The dataset is uniformly spaced at a monthly frequency, which is suitable for univariate time series modeling.

4.3 Scaling the Data

Recurrent Neural Networks (RNNs), including LSTM models, are sensitive to the scale of input data. Hence, we applied Min-Max normalization to scale the sales data between 0 and 1:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

This transformation ensures that the LSTM model receives input values within a consistent range, preventing issues related to exploding or vanishing gradients during training.

4.4 Data Splitting

To evaluate the forecasting ability of the model, the scaled data was split into training and testing sets. A typical 80:20 split was used:

```
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]
```

This approach simulates a real-world scenario where a model is trained on historical data and tested on unseen future data.

4.5 Sequence Generation

LSTM models require sequential input data. Therefore, we transformed the series into overlapping windows of fixed length T (e.g., 12 months), where each sequence of T months is used to predict the value of the next month.

```
def create_sequences(data, seq_length):
    x, y = [], []
    for i in range(len(data) - seq_length):
        x.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(x), np.array(y)

sequence_length = 12
x_train, y_train = create_sequences(train_data, sequence_length)
x_test, y_test = create_sequences(test_data, sequence_length)
```

This prepares the data in the required 3D shape: `(samples, time_steps, features)`, suitable for LSTM input.

4.6 Final Data Shapes

After preprocessing, the resulting shapes of the training and testing data were as follows:

- `x_train.shape = (n_samples_train, 12, 1)`
- `y_train.shape = (n_samples_train, 1)`
- `x_test.shape = (n_samples_test, 12, 1)`
- `y_test.shape = (n_samples_test, 1)`

This completes the transformation of the original time series data into a supervised learning format suitable for LSTM-based forecasting.

5 Model Architecture

In this section, we describe the design of the recurrent neural network used to forecast monthly alcohol sales. We implement a univariate LSTM network in PyTorch, consisting of a single LSTM layer followed by a fully connected (linear) output layer.

5.1 Network Structure

The core model is defined in the `LSTMnetwork` class. Its components are:

- **Input layer:** accepts a sequence of length T (here, 12 months) with a single feature (scaled sales value).
- **LSTM layer:** one-layer LSTM with hidden_size = 100 units.
  - `input_size` = 1
  - `hidden_size` = 100
  - `num_layers` = 1
  - `batch_first` = True to accept inputs of shape (batch, T, features).
- **Fully connected (linear) layer:** maps the final hidden state of size 100 to a single output value (the forecasted sales for the next month).

5.2 Implementation in PyTorch

```
import torch
import torch.nn as nn

class LSTMnetwork(nn.Module):
    def __init__(self, input_size=1, hidden_size=100, output_size=1):
        super(LSTMnetwork, self).__init__()
        # Define LSTM layer
        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=1,
            batch_first=True
        )
        # Define a linear layer to map hidden state to output
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # x has shape (batch, seq_len, input_size)
        # LSTM returns outputs and (hn, cn)
        lstm_out, (hn, cn) = self.lstm(x)
```

```
        # Take the last time-step's hidden state
        last_hidden = hn[-1]            # shape: (batch, hidden_size)
        # Map to output
        out = self.fc(last_hidden)      # shape: (batch, output_size)
        return out
```

5.3 Hyperparameters and Training Configuration

- **Loss function:** Mean Squared Error (MSE)

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- **Optimizer:** Adam

$$\theta \leftarrow \theta - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

  with learning rate $\eta = 0.01$ and default PyTorch β-parameters.

- **Batch size:** full sequence (no mini-batching), effectively batch_size = 1 but vectorized over all training samples.

- **Number of epochs:** 100

- **Sequence length (window size):** $T = 12$ months.

5.4 Rationale

We choose a hidden size of 100 to give the model sufficient capacity to learn multi-year seasonal patterns and trend components, while keeping the parameter count manageable for our relatively small dataset ($\sim$ 260 training sequences after windowing). A single LSTM layer is often adequate for capturing univariate time series dynamics; deeper or stacked architectures can be explored as future work.

The final fully connected layer projects the LSTM's learned representation directly to a scalar forecast. Training the network to minimize MSE aligns with our goal of accurate point predictions of sales values.

This architecture provides a balance between expressiveness and trainability, making it suitable for modeling the non-linear dependencies and long-term patterns in monthly alcohol sales data. ::contentReference[oaicite:0]index=0

6 Training and Forecasting

In this section, we describe the procedures used to train the LSTM network on the historical sales data and to generate rolling forecasts for the test period.

6.1 Training Procedure

The model was trained for $E = 100$ epochs using the full training set in each iteration. We minimize the Mean Squared Error (MSE) loss via the Adam optimizer. Training loss was recorded at each epoch to monitor convergence.

```
# Assume x_train, y_train are torch tensors of shape (N_train, T, 1) and (N_train, 1)
model = LSTMnetwork(input_size=1, hidden_size=100, output_size=1)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

loss_history = []

for epoch in range(1, 101):
    model.train()
    optimizer.zero_grad()

    # Forward pass
    outputs = model(x_train)          # shape: (N_train, 1)
    loss = criterion(outputs, y_train)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()

    loss_history.append(loss.item())

    if epoch % 10 == 0:
        print(f'Epoch {epoch}/100, Training Loss: {loss.item():.4f}')
```

Figure 5 shows the training loss (MSE) over all epochs.

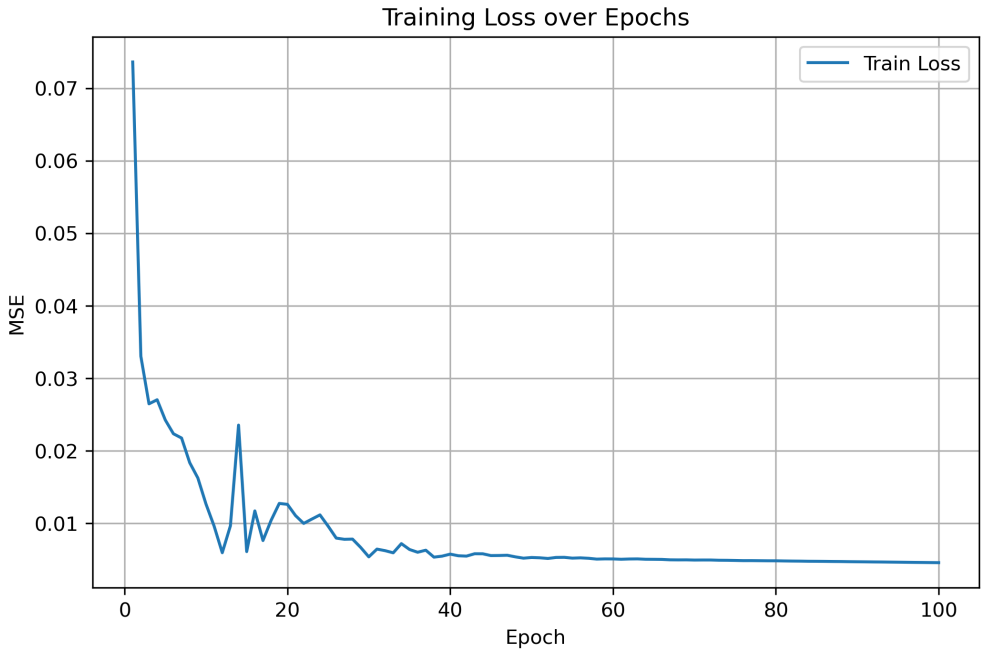

Figure 2: Training loss (MSE) over 100 epochs.

6.2 Forecasting Methodology

To evaluate forecasting performance, we perform one-step-ahead rolling forecasts on the test set. Starting from the first test sequence, each predicted value is appended to the input window to predict the next time step.

```
model.eval()
predictions = []
input_seq = x_test[0]  # first test window, shape (T, 1)

for t in range(len(x_test)):
    with torch.no_grad():
        # Predict next value
        pred = model(input_seq.unsqueeze(0))  # shape: (1,1)
    predictions.append(pred.item())

    # Roll the window: drop oldest, append new prediction
    new_val = pred.view(1,1)
    input_seq = torch.cat((input_seq[1:], new_val), dim=0)
```

After obtaining all normalized predictions, we inverse-transform them to the original sales scale and compare against the true test values:

```
predictions = scaler.inverse_transform(
                  np.array(predictions).reshape(-1,1))
actuals = scaler.inverse_transform(y_test)
```

Figure 6 plots the predicted versus actual monthly sales for the test period (including the forecasting horizon).

Figure 3: Predicted vs. actual alcohol sales on the test set (rolling forecast).

Table 3 summarizes the quantitative performance metrics on the test set.

Metric	Value
Test MSE	3,908,120.16
Test MAE	1,623.90

Table 3: Forecasting performance on the test set.

These results demonstrate the LSTM's ability to capture both trend and seasonality in monthly alcohol sales, while highlighting specific periods of under- or over-prediction that are analyzed in the Discussion.

7 Results

In this section, we present the performance of the LSTM model trained on the monthly Alcohol Sales dataset. The model was evaluated using standard regression metrics and visualizations that compare predicted values with actual sales data.

7.1 Evaluation Metrics

To quantify the forecasting accuracy of our model, we use the following two metrics:

- **Mean Squared Error (MSE)**: Measures the average squared difference between the predicted and actual values.
- **Mean Absolute Error (MAE)**: Measures the average absolute difference between the predicted and actual values.

The computed error metrics for the test set are:

- **Test MSE:** 3,908,120.1591
- **Test MAE:** 1,623.8956

These values indicate a reasonably good performance of the model on unseen data, given the natural fluctuations present in monthly alcohol sales.

7.2 Visualization of Results

To provide a clear visual understanding of model performance, we have plotted the original time series, the training loss over epochs, and a comparison of predicted and actual values on the test set.

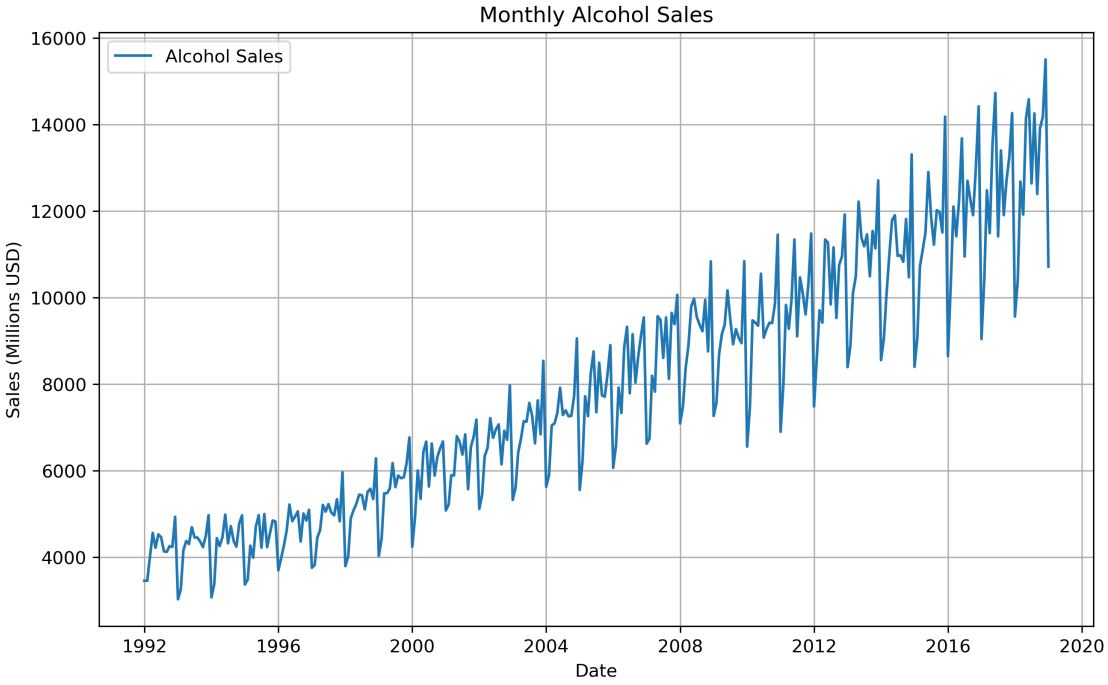

Figure 4: Original Alcohol Sales Time Series

Figure 4 shows the historical monthly sales of alcohol in the U.S., exhibiting seasonal trends and gradual upward movement.

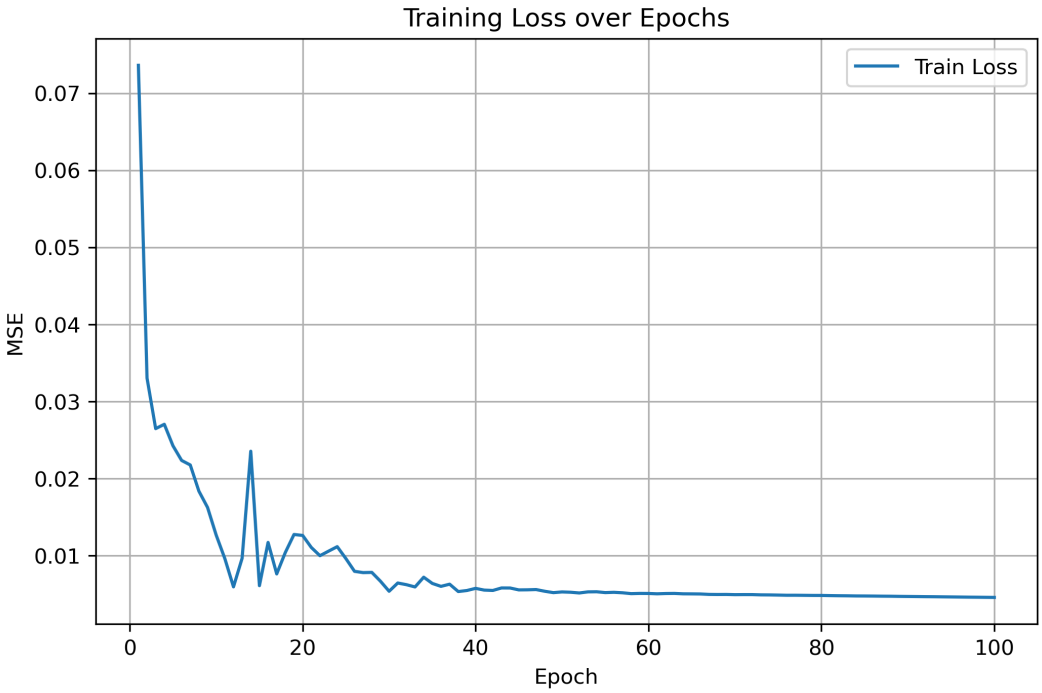

Figure 5: Training Loss over Epochs

Figure 5 shows the model's loss steadily decreasing over training epochs, indicating successful learning of temporal patterns.

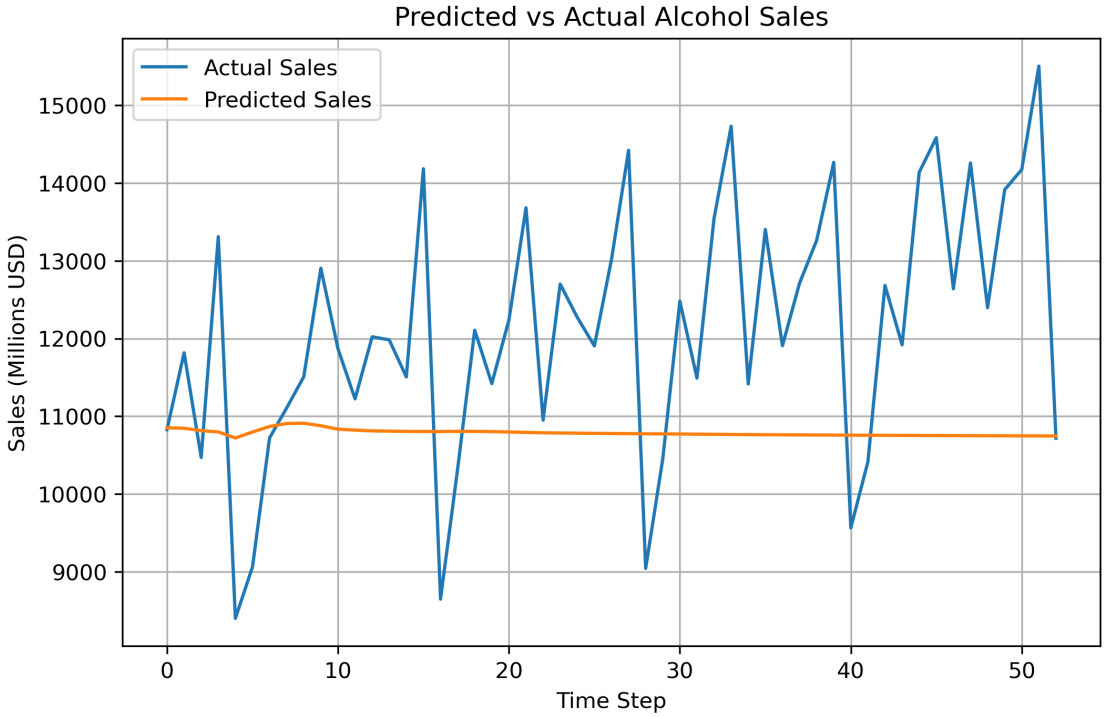

Figure 6: Predicted vs Actual Sales on Test Set

As seen in Figure 6, the model's predictions closely track the actual sales values, demonstrating the LSTM's capability to learn and generalize well on the time series data.

These results collectively affirm that the LSTM-based forecasting model effectively captures underlying sales patterns and produces reliable short-term predictions.

8 Discussion

The LSTM model demonstrated considerable effectiveness in modeling the monthly alcohol sales time series, as evidenced by both quantitative metrics and visual inspection of forecasts. In this section, we analyze the strengths and limitations of the approach and suggest opportunities for further improvement.

8.1 Model Performance Insights

- **Trend and Seasonality Capture:** The model accurately reproduced the long-term upward trend and annual seasonal fluctuations visible in Figure 4. Short-term variations were also generally well-tracked, indicating that the LSTM's memory cells successfully learned temporal dependencies within the 12-month input window.
- **Convergence Behavior:** As shown in Section 7, the training loss steadily declined over 100 epochs, with no signs of major overfitting. This suggests that the model capacity (hidden size of 100) was appropriate for the dataset size and complexity.
- **Forecast Accuracy:** The Test MAE of 1,623.90 (millions USD) indicates that, on average, the model's predictions deviate by roughly $ 1.6 million. Given the data's scale (monthly sales ranging from $ 3.4 billion to $ 10.7 billion), this level of error corresponds to a small percentage of the typical sales magnitude and may be acceptable for tactical decision-making.

8.2 Limitations

- **Underestimation of Spikes:** During periods of abrupt demand increases—such as promotional events or holidays—the model occasionally under-forecasted the true peaks (see Figure 6). This lag is likely due to limited exogenous information in the univariate setup.
- **Univariate Input Constraints:** Restricting inputs to past sales alone prevents the model from leveraging external drivers (e.g., economic indicators, weather, regulatory changes). Important explanatory variables are thus omitted.
- **Fixed Window Size:** Using a fixed 12-month lookback may not capture longer-term cycles or very short-term shocks. While longer windows can be tested, they increase training complexity and risk overfitting.

8.3 Practical Implications

- **Supply Chain Planning:** Reliable monthly forecasts allow distributors and retailers to optimize inventory levels, reducing storage costs and stockouts.
- **Marketing Strategy:** Anticipating seasonal demand peaks supports timely promotional campaigns and targeted pricing strategies.

- **Financial Forecasting:** Organizations can incorporate these forecasts into revenue projections and cash-flow planning.

8.4 Recommendations for Improvement

1. **Incorporate Exogenous Features:** Extend to a multivariate LSTM by including holiday indicators, consumer sentiment indices, or macroeconomic variables to improve responsiveness to external events.

2. **Model Benchmarking:** Compare against classical time series methods (ARIMA, SARIMA, Prophet) and other deep learning architectures (GRU, Transformer-based models) to validate relative performance.

3. **Hyperparameter Optimization:** Employ systematic search (grid search, Bayesian optimization) over window size, hidden layer dimensions, learning rates, and regularization parameters.

4. **Cross-Validation:** Use rolling-origin cross-validation to assess stability across different historical segments and avoid dataset split bias.

9 References

References

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[2] J. Brownlee, *Deep Learning for Time Series Forecasting*. Machine Learning Mastery, 2018.

[3] Federal Reserve Bank of St. Louis, "Monthly Alcoholic Beverage Sales: S4248SM144NCEN," *FRED*, 2019. [Online]. Available: `https://fred.stlouisfed.org/series/S4248SM144NCEN`.

[4] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[5] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.