19CSE456 Neural Network and Deep Learning Laboratory

List of Experiments

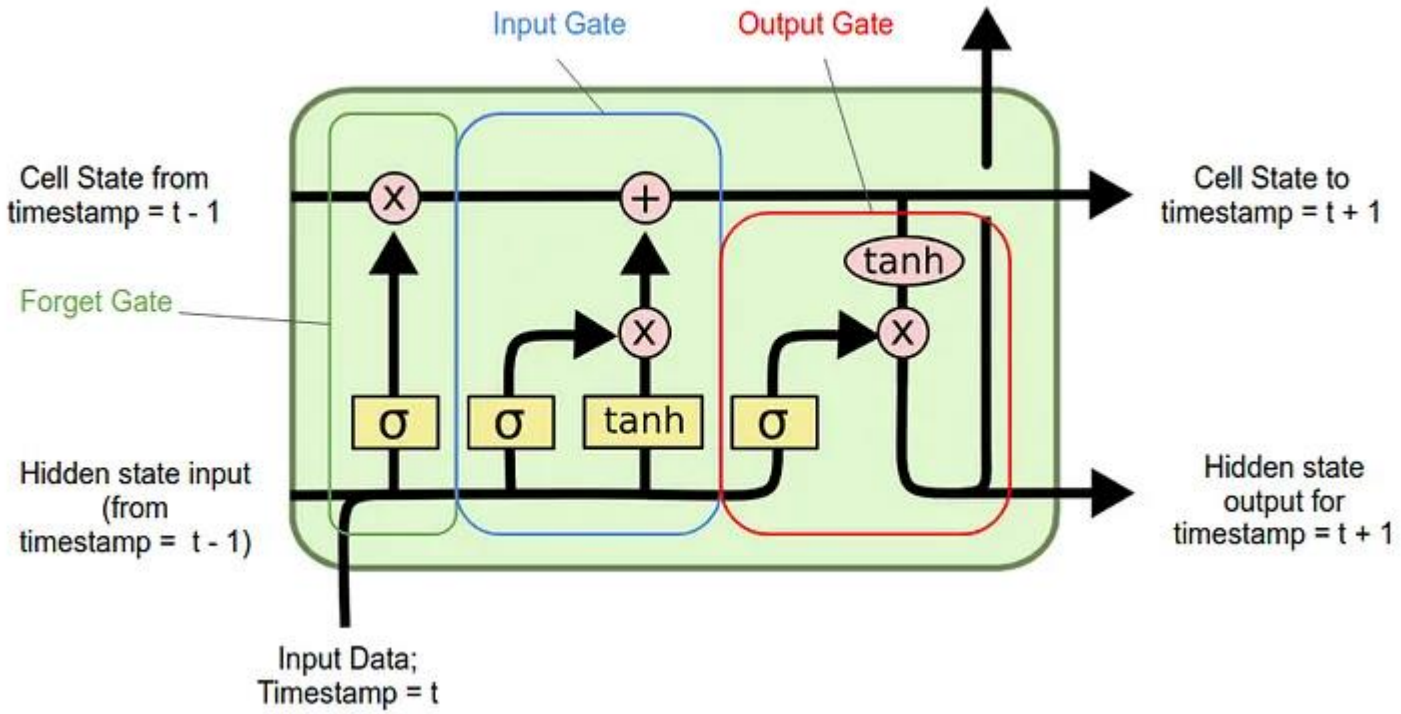
Week #	Experiment Title
1	Introduction to the lab and Implementation of a simple Perceptron (Hardcoding)
2	Implementation of Perceptron for Logic Gates (Hardcoding, Sklearn, TF)
3	Implementation of Multilayer Perceptron for XOR Gate and other classification problems with ML toy datasets (Hardcoding & TF)
4	Implementation of MLP for Image Classification with MNIST dataset (Hardcoding & TF)
5	Activation Functions, Loss Functions, Optimizers (Hardcoding & TF)
6	Lab Evaluation 1 (based on topics covered from w1 to w5)
7	Convolution Neural Networks for Toy Datasets (MNIST & CIFAR)
8	Convolution Neural Networks for Image Classification (Oxford Pets, Tiny ImageNet, etc.)
9	Recurrent Neural Networks for Sentiment Analysis with IMDB Movie Reviews
10	Long Short Term Memory for Stock Prices (Yahoo Finance API)

List of Experiments contd.

Week #	Experiment Title
11	Implementation of Autoencoders and Denoising Autoencoders (MNIST/CIFAR)
12	Boltzmann Machines (MNIST/CIFAR)
13	Restricted Boltzmann Machines (MNIST/CIFAR)
14	Hopfield Neural Networks (MNIST/CIFAR)
15	Lab Evaluation 2 (based on CNN, RNN, LSTM, and AEs)
16	Case Study Review (Phase 1)
17	Case Study Review (Phase 1)

Long Short Term Memory (LSTM)

- It's a type of artificial neural network architecture, specifically designed for processing and making predictions based on sequential data, like time series or natural language
- LSTM networks are a special kind of RNNs, capable of learning long-term dependencies and patterns in data



Cell State: This is the key component of LSTMs, acting as a "memory" that carries information across different time steps in the sequence

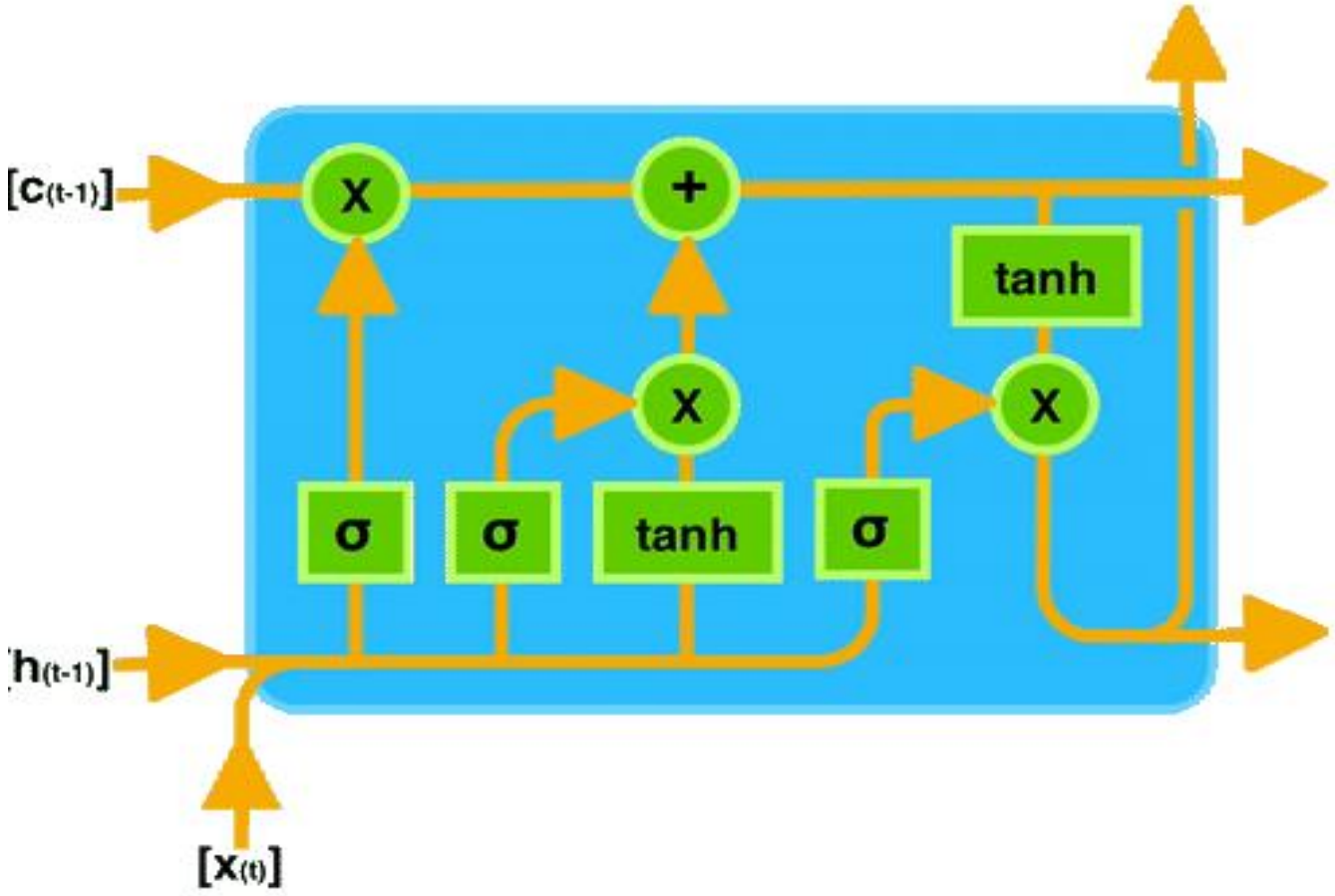
Forget Gate: Decides what information should be discarded from the cell state

Input Gate: Determines what new information should be added to the cell state

Output Gate: Controls the output based on the cell state and the current input

Long Short Term Memory (LSTM)

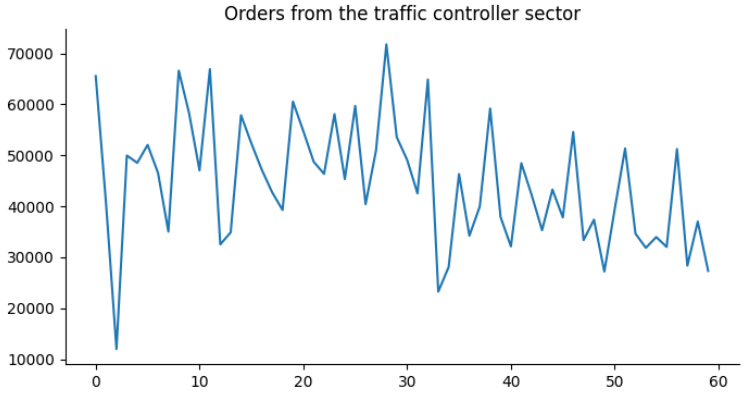
- By utilizing these gates, LSTMs can retain important information over long sequences and ignore irrelevant data, making them effective for tasks like language modeling, machine translation, and more

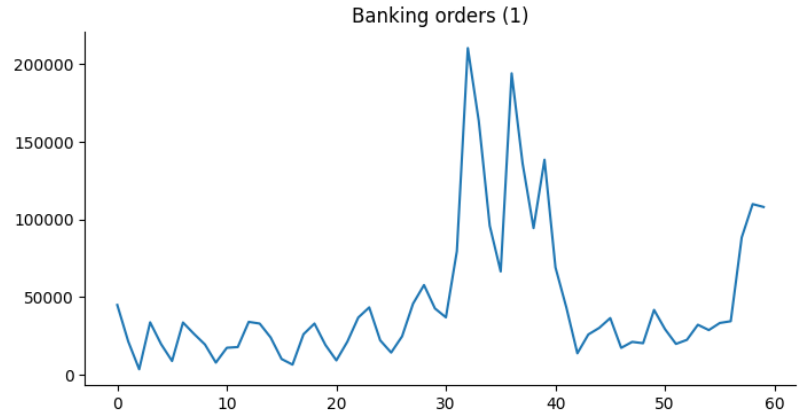


Daily Demand Forecasting Orders Dataset

This dataset was collected over 60 days from a Brazilian logistics company. It contains 12 predictive attributes and a target attribute, which is the total number of orders for each day. This dataset is used for regression tasks and is valuable for studying and developing models for demand forecasting in logistics.

	Week of the month (first week, second, third, fourth or fifth week	Day of the week (Monday to Friday)	Non-urgent order	Urgent order	Order type A	Order type B	Order type C	Fiscal sector orders	Orders from the traffic controller sector	Banking orders (1)	Banking orders (2)	Banking orders (3)	Target (Total orders)
0	1	4	316.307	223.270	61.543	175.586	302.448	0	65556	44914	188411	14793	539.577
1	1	5	128.633	96.042	38.058	56.037	130.580	0	40419	21399	89461	7679	224.675
2	1	6	43.651	84.375	21.826	25.125	82.461	1.386	11992	3452	21305	14947	129.412
3	2	2	171.297	127.667	41.542	113.294	162.284	18.156	49971	33703	69054	18423	317.120
4	2	3	90.532	113.526	37.679	56.618	116.220	6.459	48534	19646	16411	20257	210.517





Long Short Term Memory (LSTM)


```
def load_demand_dataset():
    # URL for the UCI dataset
    dataset_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00409/Daily_Demand_Forecasting_Orders.csv"
    local_filename = "Daily_Demand_Forecasting_Orders.csv"

    # Download the file if it doesn't exist
    if not os.path.exists(local_filename):
        print("Downloading dataset...")
        urlretrieve(dataset_url, local_filename)

    # Load the dataset
    df = pd.read_csv(local_filename, delimiter=';', decimal=',')
    print(f"Dataset loaded with shape: {df.shape}")
    return df
```

This function checks if the dataset file already exists locally, downloads it if it doesn't, and then loads the dataset into a pandas DataFrame

Long Short Term Memory (LSTM)


```
# Create sequences for time series prediction
def create_sequences(X, y, seq_length):
    X_seq, y_seq = [], []
    for i in range(len(X) - seq_length):
        X_seq.append(X[i:i + seq_length])
        y_seq.append(y[i + seq_length])
    return np.array(X_seq), np.array(y_seq)

# Parameters
sequence_length = 7  # One week lookback period
train_size = int(len(X_scaled) * 0.8)
```

This function generates sequences of a specified length from the input data for time series prediction

Loop through the data, creating sequences of seq_length from X and the corresponding single target value from y

Convert the lists to NumPy arrays and return them

Creating sequences helps models understand the relationship between data points over time, which is essential for making accurate predictions in tasks involving sequential data

Long Short Term Memory (LSTM)


```
# Split data into train and test sets
X_train_data = X_scaled[:train_size]
X_test_data = X_scaled[train_size - sequence_length:]
y_train_data = y_scaled[:train_size]
y_test_data = y_scaled[train_size - sequence_length:]

# Create sequences
X_train, y_train = create_sequences(X_train_data, y_train_data, sequence_length)
X_test, y_test = create_sequences(X_test_data, y_test_data, sequence_length)

print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")
```

Splits Data into Train and Test Sets

Uses the create_sequences() function to generate sequences for the training and test sets.

Long Short Term Memory (LSTM)


```
# Build the LSTM model
def build_lstm_model(sequence_length, n_features):
    model = Sequential([
        # First LSTM layer with return sequences
        LSTM(64, return_sequences=True, input_shape=(sequence_length, n_features),
             recurrent_dropout=0.1),
        # Second LSTM layer with return sequences
        LSTM(64, return_sequences=True, recurrent_dropout=0.1),
        # Third LSTM layer
        LSTM(64, recurrent_dropout=0.1),
        # Dropout to prevent overfitting
        Dropout(0.2),
        # Output layer
        Dense(1)
    ])
```

Creates a sequential model, which is a linear stack of layers

- Adds an LSTM layer with 64 units
- Ensures that the layer outputs the full sequence instead of just the last output
- Specifies the shape of the input
- Applies dropout to the recurrent state with a 10% dropout

Similar to the first LSTM layer, but without specifying the input shape

This LSTM layer does not return sequences, meaning it only outputs the last time step's result

Long Short Term Memory (LSTM)


```
# Compile model
model.compile(optimizer='adam', loss='mean_squared_error')
print(model.summary())
return model
```

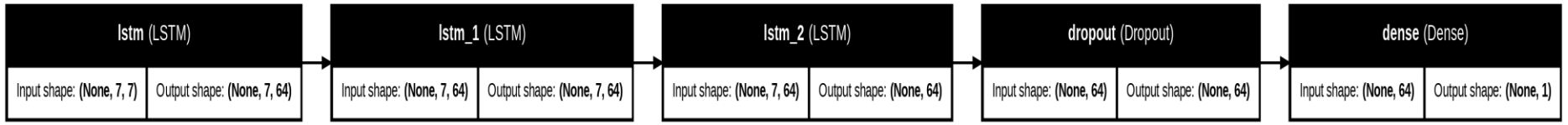
Compiles the model with the Adam optimizer and mean squared error loss function

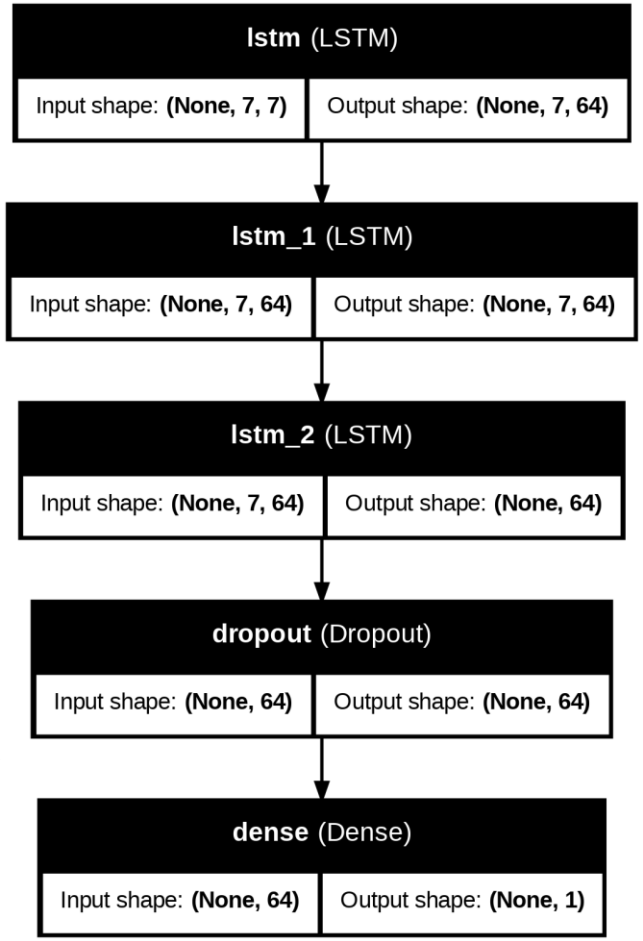
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 7, 64)	18,432
lstm_1 (LSTM)	(None, 7, 64)	33,024
lstm_2 (LSTM)	(None, 64)	33,024
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 1)	65

Total params: 84,545 (330.25 KB)
Trainable params: 84,545 (330.25 KB)
Non-trainable params: 0 (0.00 B)

Long Short Term Memory (LSTM)


```
# Visualize the model
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='lstm_model.png', show_shapes=True, show_layer_names=True, rankdir='LR')
```





Long Short Term Memory (LSTM)


```
# Early stopping to prevent overfitting
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
)

# Learning rate scheduler
lr_scheduler = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=0.0001
)
```

Early stopping is a technique used to prevent overfitting during training

Specifies that the callback will monitor the validation loss

The training will stop if the validation loss does not improve for 15 consecutive epochs

A learning rate scheduler adjusts the learning rate during training to improve convergence

The learning rate will be reduced by a factor of 0.5, if the validation loss does not improve

The learning rate will not be reduced below 0.0001

Long Short Term Memory (LSTM)


```
# Train the model
history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop, lr_scheduler],
    verbose=1
)

# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
```

```
Epoch 14/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.0455 - val_loss: 0.0218 - learning_rate: 5.0000e-04
Epoch 15/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - loss: 0.0466 - val_loss: 0.0234 - learning_rate: 5.0000e-04
Epoch 16/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0449 - val_loss: 0.0244 - learning_rate: 2.5000e-04
Epoch 17/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.0440 - val_loss: 0.0254 - learning_rate: 2.5000e-04
Epoch 18/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - loss: 0.0474 - val_loss: 0.0263 - learning_rate: 2.5000e-04
Epoch 19/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - loss: 0.0441 - val_loss: 0.0269 - learning_rate: 2.5000e-04
Epoch 20/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - loss: 0.0401 - val_loss: 0.0273 - learning_rate: 2.5000e-04
Epoch 21/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0385 - val_loss: 0.0274 - learning_rate: 1.2500e-04
Epoch 22/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - loss: 0.0408 - val_loss: 0.0274 - learning_rate: 1.2500e-04
Epoch 23/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0396 - val_loss: 0.0273 - learning_rate: 1.2500e-04
Epoch 24/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.0424 - val_loss: 0.0271 - learning_rate: 1.2500e-04
Epoch 25/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.0433 - val_loss: 0.0268 - learning_rate: 1.2500e-04
Epoch 26/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - loss: 0.0386 - val_loss: 0.0265 - learning_rate: 1.0000e-04
```

Long Short Term Memory (LSTM)


```
# Invert predictions back to original scale
train_predictions = y_scaler.inverse_transform(train_predictions)
y_train_inv = y_scaler.inverse_transform(y_train)
test_predictions = y_scaler.inverse_transform(test_predictions)
y_test_inv = y_scaler.inverse_transform(y_test)

# Calculate performance metrics
def calculate_metrics(actual, predicted):
    metrics = {
        'RMSE': math.sqrt(mean_squared_error(actual, predicted)),
        'MAE': mean_absolute_error(actual, predicted),
        'R²': r2_score(actual, predicted),
        'MAPE': np.mean(np.abs((actual - predicted) / (actual + 1e-5))) * 100 }
    return metrics
```

Reverse the scaling applied to the data, converting it back to the original scale.

Measures the square root of the average of the squared differences between actual and predicted values

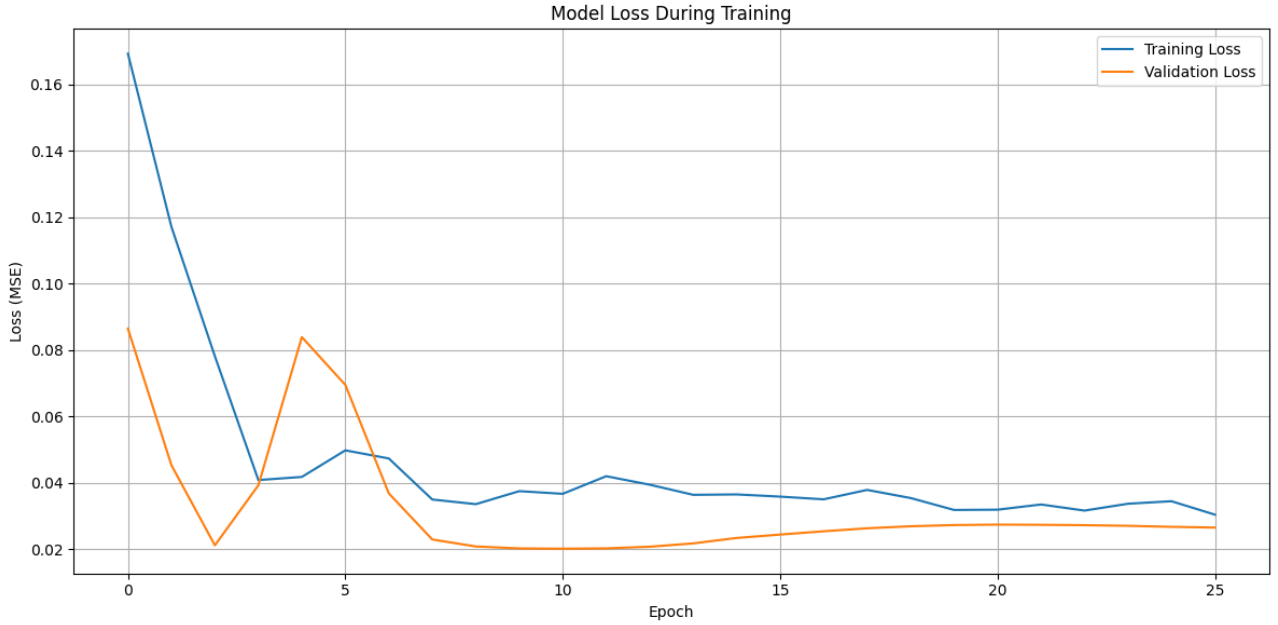
Measures the average of the absolute differences between actual and predicted values

Measures the proportion of the variance in the dependent variable that is predictable from the independent variables

Measures the average of the absolute percentage errors between actual and predicted values

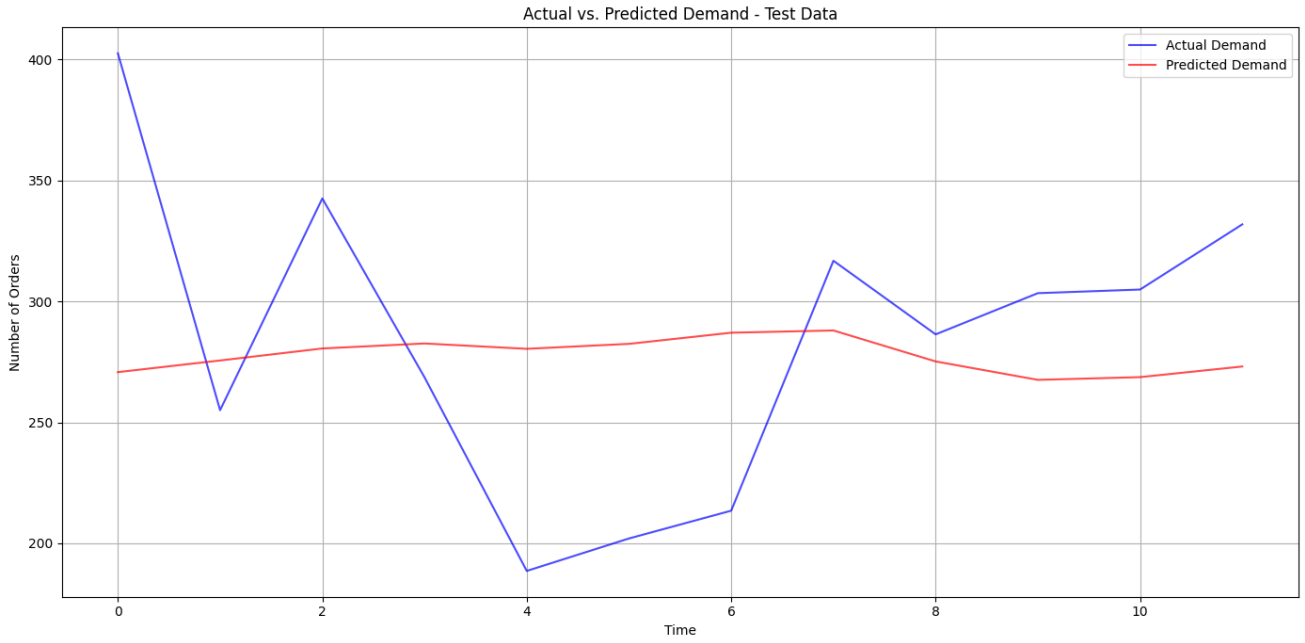
Training-Validation Loss


```
# Training and Validation Loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss During Training')
plt.ylabel('Loss (MSE)')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.savefig('training_loss.png')
plt.show()
```



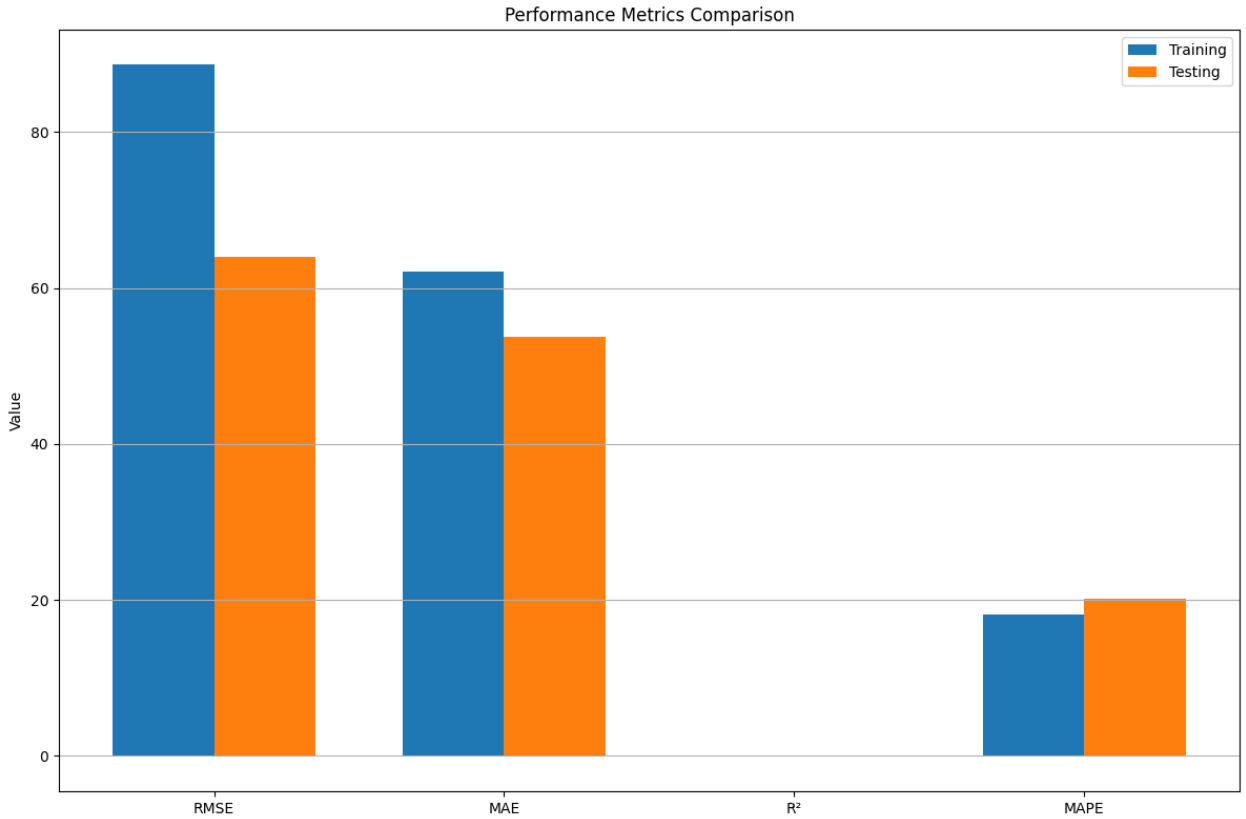
Training-Validation Loss

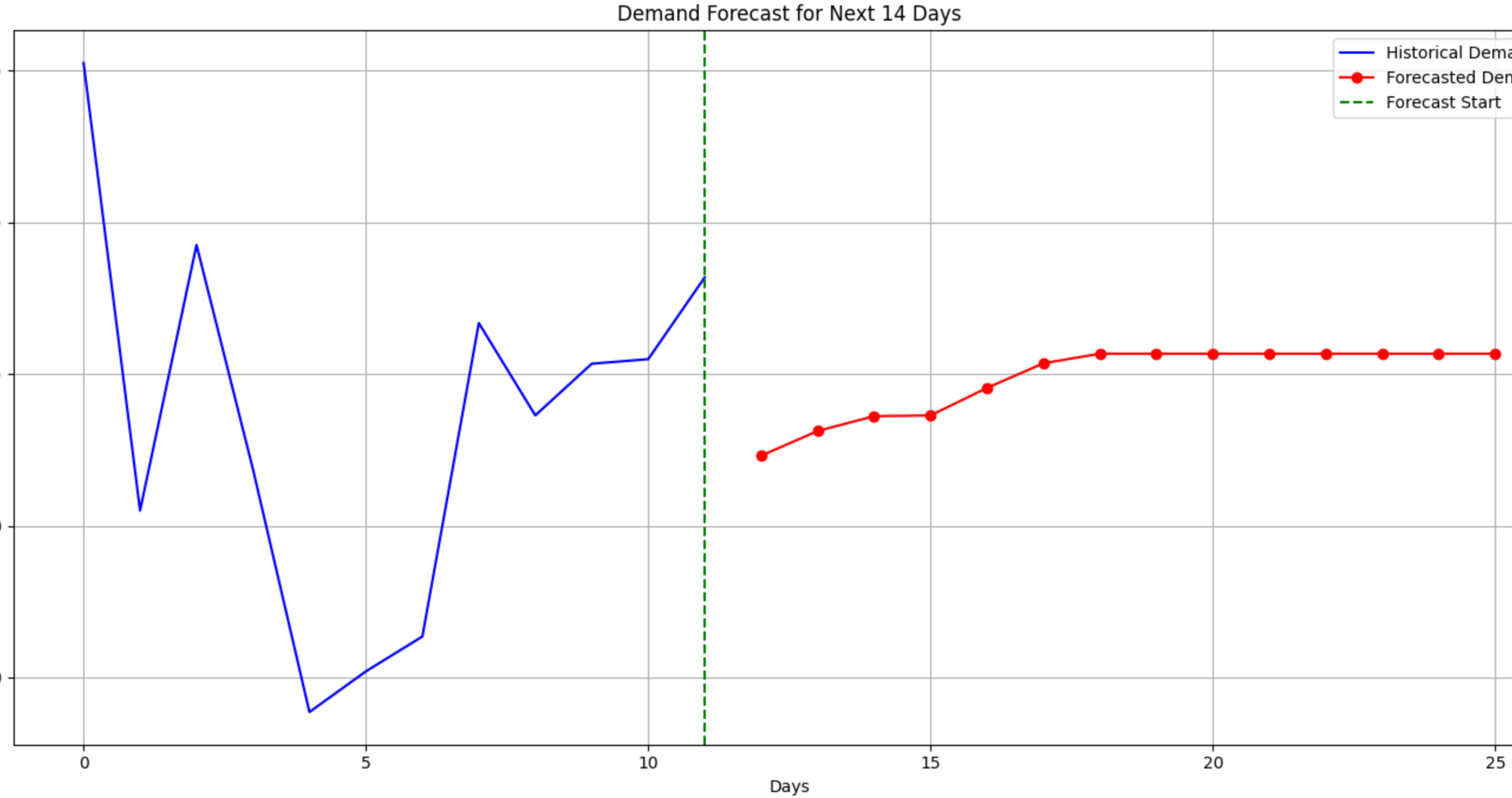

```
# 3. Actual vs. Predicted - Test Data
plt.figure(figsize=(14, 7))
plt.plot(y_test_inv, label='Actual Demand', color='blue', alpha=0.7)
plt.plot(test_predictions, label='Predicted Demand', color='red', alpha=0.7)
plt.title('Actual vs. Predicted Demand - Test Data')
plt.ylabel('Number of Orders')
plt.xlabel('Time')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('test_predictions.png')
plt.show()
```



Long Short Term Memory (LSTM)


```
# Performance Metrics Visualization
metrics_names = list(train_metrics.keys())
train_values = [train_metrics[m] for m in metrics_names]
test_values = [test_metrics[m] for m in metrics_names]
plt.figure(figsize=(12, 8))
x = np.arange(len(metrics_names))
width = 0.35
plt.bar(x - width/2, train_values, width, label='Training')
plt.bar(x + width/2, test_values, width, label='Testing')
plt.title('Performance Metrics Comparison')
plt.xticks(x, metrics_names)
plt.ylabel('Value')
plt.legend()
plt.grid(True, axis='y')
plt.tight_layout()
plt.savefig('performance_metrics.png')
plt.show()
```




Demand Forecast for Next 14 Days
Historical Dema
Forecasted Dem
Forecast Start
0
5
10
15
20
25
Days

Week 9 Exercises

1.Time Series Forecasting with LSTM Using the Istanbul Stock Exchange Dataset

Objective: Build an LSTM (Long Short-Term Memory) model to forecast the Istanbul Stock Exchange (ISE) index returns based on historical data and the returns of other international indices.

Dataset: Istanbul Stock Exchange dataset from the UCI Machine Learning Repository

Tasks:

1. Data Loading and Exploration
2. Sequence Creation
3. Build an LSTM Model
4. Train the Model
5. Evaluate the Model
6. Make Predictions

Experiment with different hyperparameters and model architectures to improve the forecasting accuracy