Stock Price Prediction Using Recurrent Neural Networks A Comparative Analysis of RNN Architectures on IBM and Amazon (AMZN) Stock Data

Cyrus Benjamin C. Canape
Edrine Frances A. Esguerra
[#]*Department of Computer Science, University of Santo Tomas-Manila*
España Bldv., Sampaloc, Manila, 1008 Metro Manila, Philippines
[1]cyrusbenjamin.canape.cics@ust.edu.ph
[2]edrinefrances.esguerra.cics@ust.edu.ph

***Abstract*— This paper presents a comprehensive study on stock price prediction using Recurrent Neural Networks (RNNs), specifically focusing on Long Short-Term Memory (LSTM) networks and their variants. We implement a baseline LSTM model and conduct extensive experiments with architectural modifications, preprocessing techniques, and hyperparameter tuning. Using historical stock data from IBM and Amazon (2006-2018), we evaluate the performance of different RNN architectures including improved LSTM with dropout, GRU, and Bidirectional LSTM. Our experiments reveal the impact of various design choices on prediction accuracy and provide insights into optimal configurations for financial time series forecasting.**

***Keywords*— Recurrent Neural Networks, Hyperparameter Tuning, Stock Price Prediction, Long Short Term Memory, Time Series Prediction**

I. Introduction

Financial time series prediction remains one of the most challenging problems in machine learning due to the inherently noisy and non-stationary nature of stock market data. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, have emerged as powerful tools for modeling sequential data and capturing long-term dependencies in time series. This study explores the application of various RNN architectures to predict IBM stock prices, leveraging historical data spanning over a decade to evaluate the effectiveness of different model configurations.

The primary objective of this research is to systematically compare different RNN architectures and their variants in the context of stock price prediction. We begin with a baseline LSTM implementation and progressively explore more sophisticated architectures including Gated Recurrent Units (GRUs) and Bidirectional LSTMs. Additionally, we investigate the impact of preprocessing techniques, sequence lengths, and regularization methods on model performance.

We utilize two datasets:

- **IBM stock data** (2006-01-01 to 2018-01-01)
- **Amazon (AMZN) stock data** (2006-01-01 to 2018-01-01)

Each dataset contains daily stock information including Date (index), Open Price, High Price, Low Price, Close Prime, and Volume values.

IBM_2006-01-01_to_2018-01-01

Date	Open	High	Low	Close	Volume	Name
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.2	82.5	81.33	81.95	9840600	IBM
2006-01-05	81.4	82.9	81.0	82.5	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.1	84.25	83.38	83.73	6858200	IBM
2006-01-10	83.15	84.12	83.12	84.07	5701000	IBM
2006-01-11	84.37	84.81	83.4	84.17	5776500	IBM
2006-01-12	83.82	83.96	83.4	83.57	4926500	IBM
2006-01-13	83.0	83.45	82.5	83.17	6921700	IBM
2006-01-17	82.8	83.16	82.54	83.0	8761700	IBM

Fig. 1 IBM Dataset

AMZN_2006-01-01_to_2018-01-01

Date	Open	High	Low	Close	Volume	Name
2006-01-03	47.47	47.85	46.25	47.58	7582127	AMZN
2006-01-04	47.48	47.73	46.69	47.25	7440914	AMZN
2006-01-05	47.16	48.2	47.11	47.65	5417258	AMZN
2006-01-06	47.97	48.58	47.32	47.87	6154285	AMZN
2006-01-09	46.55	47.1	46.4	47.08	8945056	AMZN
2006-01-10	46.41	46.75	45.36	45.65	9686957	AMZN
2006-01-11	45.65	45.7	44.26	44.93	8497383	AMZN
2006-01-12	44.79	45.09	44.09	44.36	5818301	AMZN
2006-01-13	44.48	44.85	44.0	44.4	4432237	AMZN
2006-01-17	43.95	44.32	43.66	44.0	5635225	AMZN

Fig. 2 AMZN Dataset

II. Methodology

This section details the technical approach, data processing steps, and model architectures.

A. Data Preprocessing

The datasets are loaded using pandas with date parsing functionality to ensure proper temporal indexing. Upon loading, we conduct initial data exploration to understand the structure and characteristics of our data. The exploration reveals no missing values in either dataset, high correlation between price features (Open, High, Low, Close), and significant price volatility during certain market periods. This initial analysis informs our preprocessing decisions and model design choices.

```
# First, we get the data
dataset = pd.read_csv('IBM_2006-01-01_to_2018-01-01.csv', index_col='Date',
parse_dates=['Date'])
print("IBM Dataset loaded successfully")
print(f"Dataset shape: {dataset.shape}")
dataset.head()

# Checking for missing values
training_set = dataset['2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values

# We have chosen 'High' attribute for prices. Let's see what it looks like
plt.figure(figsize=(16, 4))
dataset["High"]['2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM Stock Price')
plt.show()
```

Fig. 3 Dataset Preprocessing - IBM Dataset

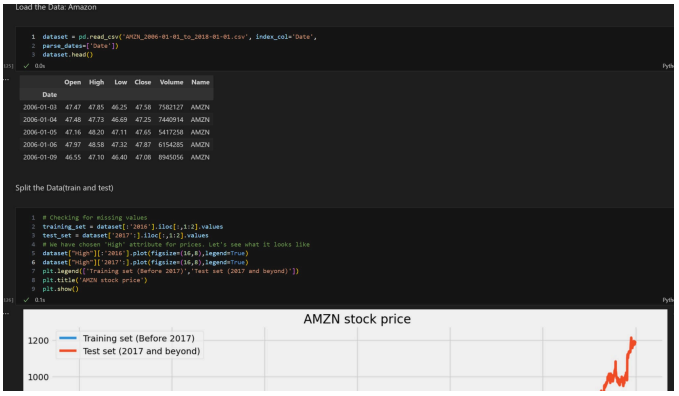

Fig. 4 Dataset Preprocessing - AMZN Dataset

We implement a temporal split strategy that respects the sequential nature of the data. The training set comprises all data up to December 31, 2016, resulting in 2,768 samples, while the test set contains data from January 1, 2017 onwards, totaling 251 samples. This split ensures that we evaluate our models on truly unseen future data, mimicking real-world trading scenarios. Additionally, we allocate 10% of the training data for validation during model training to monitor overfitting and enable early stopping.

Data normalization proves crucial for neural network convergence. We experiment with three scaling methods to determine the optimal approach. The MinMaxScaler, our baseline choice, scales data between 0 and 1, which works well for bounded time series data.

Creating appropriate sequences from time series data is fundamental to RNN training. We implement a sliding window approach with a lookback period of 60 time steps, approximately representing three months of trading data. This window size balances capturing sufficient temporal context while maintaining computational efficiency. The resulting training data has shape (2708, 60, 1),

where each sample contains 60 consecutive price points used to predict the next price.

B. Model Architectures

Our baseline model employs a simple yet effective architecture consisting of two LSTM layers with 5 units each. The first LSTM layer returns sequences to enable stacking, while the second layer outputs a single vector feeding into a dense output layer. This minimalist design contains approximately 530 trainable parameters and serves as our performance benchmark. We compile the model using RMSprop optimizer with mean squared error loss, training for 10 epochs with a batch size of 32.

```python
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=5, return_sequences=True, input_shape=(X_train.shape[1],1)))
# Second LSTM layer
regressor.add(LSTM(units=5))
# The output layer
regressor.add(Dense(units=1))
# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
print("Training baseline model...")
regressor.fit(X_train,y_train,epochs=10,batch_size=32)

dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
```

Fig. 5 Baseline Model Architecture

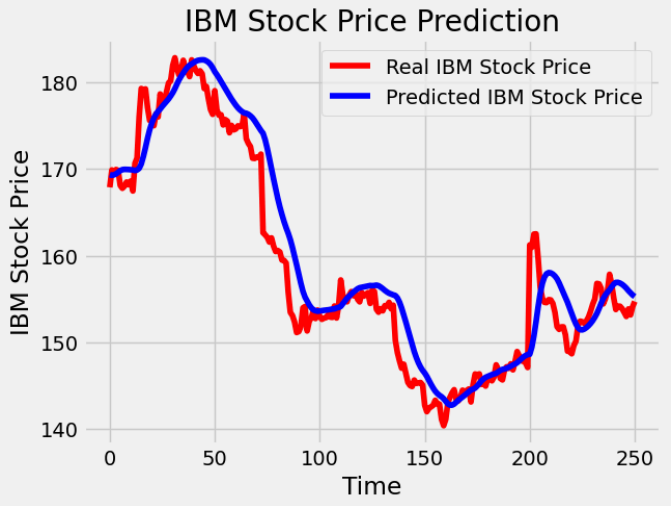

Fig. 6 IBM Stock Price - Baseline Model

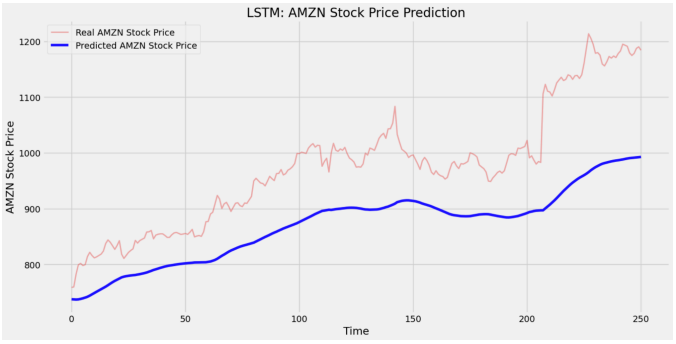

Fig. 7 AMZN Stock Price - Baseline Model

IBM Dataset Model Architecture:

1. **Enhanced LSTM with Dropout**
   This architecture increases model capacity with 64 units in the first layer and 32 units in subsequent layers, incorporating 0.2 dropout rate after each LSTM layer. The model structure: LSTM(64) → Dropout(0.2) → LSTM(32) → Dropout(0.2) → LSTM(32) → Dropout(0.2) → Dense(1). Uses Adam optimizer with early stopping (patience=10) and learning rate reduction (factor=0.5, patience=5). Total parameters: approximately 41,000.

```python
def build_improved_lstm(self, input_shape, units=[64, 32], dropout_rate=0.2):
    """Build an improved LSTM model with dropout"""
    model = Sequential()
    model.add(LSTM(units=units[0], return_sequences=True, input_shape=input_shape))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(units=units[1], return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(units=units[1]))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Fig. 8 IBM Stock Price - Enhanced LSTM with Dropout

2. **GRU Model**
   Replaces LSTM cells with GRU cells while maintaining the 64-32 unit structure. Architecture: GRU(64) → GRU(32) → Dense(1). GRU cells have fewer parameters than LSTM (3 gates vs 4), resulting in approximately 31,000 total parameters and 25% faster training time.

```python
def build_gru_model(self, input_shape, units=[64, 32]):
    """Build a GRU model"""
    model = Sequential()
    model.add(GRU(units=units[0], return_sequences=True, input_shape=input_shape))
    model.add(GRU(units=units[1]))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Fig. 9 IBM Stock Price - GRU Model

3. **Bidirectional LSTM**
   Implements bidirectional processing with 50 units per direction. Architecture: Bidirectional(LSTM(50)) → Bidirectional(LSTM(50)) → Dense(1). Processes sequences forward and backward simultaneously, doubling the hidden state size to 100 units per layer. Total parameters: approximately 80,000.

```
def build_bidirectional_lstm(self, input_shape, units=50):
    """Build a Bidirectional LSTM model"""
    model = Sequential()
    model.add(Bidirectional(LSTM(units=units, return_sequences=True), input_shape=input_shape))
    model.add(Bidirectional(LSTM(units=units)))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Fig. 10 IBM Stock Price - Bidirectional LSTM

4. **LSTM with Standard Scaler**
   Identical to Improved LSTM architecture but uses StandardScaler instead of MinMaxScaler for data normalization. This configuration revealed that standardization better preserves the statistical properties of financial data.

```
# Experiment 5: Different preprocessing (StandardScaler)
print("\n=== Experiment 5: StandardScaler Preprocessing ===")
self.prepare_data(target_column='High', scaler_type='standard')
X_train_std, y_train_std = self.create_sequences(self.training_set_scaled, lookback=60)
X_train_std = np.reshape(X_train_std, (X_train_std.shape[0], X_train_std.shape[1], 1))

std_model = self.build_improved_lstm((X_train_std.shape[1], 1), units=[64, 32])
history_std = self.train_model(std_model, X_train_std, y_train_std, epochs=50, batch_size=32)

X_test_std = self.prepare_test_data()
predictions_std = self.evaluate_model(std_model, X_test_std, 'LSTM with StandardScaler')
self.plot_predictions(predictions_std, 'LSTM with StandardScaler')
```

Fig. 11 IBM Stock Price - LSTM with StandardScaler

5. **LSTM with 30 Timesteps**
   Uses Improved LSTM architecture but reduces lookback from 60 to 30 days, halving the temporal context. This experiment tests the trade-off between computational efficiency and temporal information.

```
# Experiment 6: Different sequence length
print("\n=== Experiment 6: Different Sequence Length (30 timesteps) ===")
self.prepare_data(target_column='High', scaler_type='minmax')
X_train_30, y_train_30 = self.create_sequences(self.training_set_scaled, lookback=30)
X_train_30 = np.reshape(X_train_30, (X_train_30.shape[0], X_train_30.shape[1], 1))

model_30 = self.build_improved_lstm((X_train_30.shape[1], 1), units=[64, 32])
history_30 = self.train_model(model_30, X_train_30, y_train_30, epochs=50, batch_size=32)

X_test_30 = self.prepare_test_data(lookback=30)
predictions_30 = self.evaluate_model(model_30, X_test_30, 'LSTM with 30 timesteps')
self.plot_predictions(predictions_30, 'LSTM with 30 timesteps')
```

Fig. 12 IBM Stock Price - LSTM with 30 Timesteps

Amazon (AMZN) Dataset Model Architecture:

1. **LSTM with Increased Batch Size**
   This model uses the base LSTM architecture but increases the batch size from 32 to 64. The experiment explores the impact of larger batch updates on training stability and speed. Larger batches can offer more stable gradient estimates, potentially leading to faster convergence but may generalize less well compared to smaller batches (Masters, 2018)

```
# The LSTM architecture
LSTM_Batch_Model = Sequential()
# First LSTM layer with Dropout regularisation
LSTM_Batch_Model.add(Input(shape=(X_train.shape[1], 1)))
LSTM_Batch_Model.add(LSTM(units=5, return_sequences=True))
LSTM_Batch_Model.add(Dropout(0.2))

# Second LSTM layer
LSTM_Batch_Model.add(LSTM(units=5))
LSTM_Batch_Model.add(Dropout(0.2))

# The output layer
LSTM_Batch_Model.add(Dense(units=1))
# Compiling the RNN
LSTM_Batch_Model.compile(optimizer='rmsprop', loss='mean_squared_error')
# Fitting to the training set with increased batch size
LSTM_Batch_Model.fit(X_train, y_train, epochs=10, batch_size=64)
```

Fig. 13 AMAZN Stock Price - LSTM with 64 Batch Size

2. **LSTM with More Layers**
   An enhanced architecture that adds a third LSTM layer and increases units per layer to 50. This experiment investigates the benefits of deeper temporal representations and greater learning capacity for capturing complex market patterns.

```
# The deeper LSTM architecture
LSTM_Layers_Model = Sequential()
# First LSTM layer with Dropout regularisation
LSTM_Layers_Model.add(Input(shape=(X_train.shape[1], 1)))
LSTM_Layers_Model.add(LSTM(units=50, return_sequences=True))
LSTM_Layers_Model.add(Dropout(0.2))
# Second LSTM layer with Dropout
LSTM_Layers_Model.add(LSTM(units=50, return_sequences=True))
LSTM_Layers_Model.add(Dropout(0.2))
# Third LSTM layer
LSTM_Layers_Model.add(LSTM(units=50))
LSTM_Layers_Model.add(Dropout(0.2))
# The output layer
LSTM_Layers_Model.add(Dense(units=1))
# Compiling the RNN
LSTM_Layers_Model.compile(optimizer='rmsprop', loss='mean_squared_error')
# Fitting to the training set
LSTM_Layers_Model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Fig. 14 AMAZN Stock Price - LSTM with an Additional Layer

3. **LSTM with Lower Learning Rate**
   This version retains the base architecture but reduces the learning rate from the default (0.001 for RMSprop) to 0.0005. The goal is to observe whether finer, slower weight updates improve convergence and reduce training loss oscillations, particularly in volatile financial data.

```python
# The LSTM architecture with custom learning rate
LSTM_LR_Model = Sequential()
# First LSTM layer with Dropout regularisation
LSTM_LR_Model.add(Input(shape=(X_train.shape[1], 1)))
LSTM_LR_Model.add(LSTM(units=5, return_sequences=True))
LSTM_LR_Model.add(Dropout(0.2))

# Second LSTM layer
LSTM_LR_Model.add(LSTM(units=5))
LSTM_LR_Model.add(Dropout(0.2))

# The output layer
LSTM_LR_Model.add(Dense(units=1))
# Compiling the RNN with custom learning rate
opt = RMSprop(learning_rate=0.0005)
LSTM_LR_Model.compile(optimizer=opt, loss='mean_squared_error')
# Fitting to the training set
LSTM_LR_Model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Fig. 15 AMAZN Stock Price - LSTM with 0.0005 Learning Rate

4. GRU Model

This model replaces the traditional LSTM layers with GRU layers. GRUs simplify the gating mechanism by combining the forget and input gates into a single update gate, which often reduces training time and computational cost (Cho, 2014). This experiment evaluates whether a lighter architecture can match or outperform LSTM in forecasting accuracy.

```python
# The GRU architecture
GRU_Model = Sequential()
# First GRU layer with Dropout regularisation
GRU_Model.add(Input(shape=(X_train.shape[1], 1)))
GRU_Model.add(GRU(units=5, return_sequences=True))
GRU_Model.add(Dropout(0.2))

# Second GRU layer
GRU_Model.add(GRU(units=5))
GRU_Model.add(Dropout(0.2))

# The output layer
GRU_Model.add(Dense(units=1))
# Compiling the RNN
GRU_Model.compile(optimizer='rmsprop', loss='mean_squared_error')
# Fitting to the training set
GRU_Model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Fig. 16 AMAZN Stock Price - GRU Model

5. BiRNN Model

This model utilizes Bidirectional GRU layers to process input sequences in both forward and backward directions. It aims to capture dependencies from both past and future time steps, which can enhance performance in sequence tasks like stock prediction. The experiment tests whether bidirectional memory improves predictive accuracy.

```python
# The Bidirectional GRU architecture
BiRNN_Model = Sequential()
# First Bidirectional GRU layer with Dropout
BiRNN_Model.add(Input(shape=(X_train.shape[1], 1)))
BiRNN_Model.add(Bidirectional(GRU(units=5, return_sequences=True)))
BiRNN_Model.add(Dropout(0.2))

# Second Bidirectional GRU layer
BiRNN_Model.add(Bidirectional(GRU(units=5)))
BiRNN_Model.add(Dropout(0.2))

# Output layer
BiRNN_Model.add(Dense(units=1))
# Compile the model
BiRNN_Model.compile(optimizer='rmsprop', loss='mean_squared_error')
# Train the model
BiRNN_Model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Fig. 17 AMAZN Stock Price - GRU Model

C. Training Strategy

A. IBM Dataset Training Strategy:

All models except baseline use identical training parameters: Adam optimizer (learning_rate=0.001), batch_size=32, epochs=50 (with early stopping), validation_split=0.1. Callbacks include EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True) and ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001).

Training typically converges within 15-25 epochs due to early stopping, with validation loss plateauing around 0.0005-0.002 for best-performing models.

Amazon (AMZN) Dataset Training Strategy:

All models used the baseline model as the standard for a standardized training configuration to ensure consistent comparison. The models employed the rmsprop with a learning rate of 0.001, a batch size of 32, and were trained for up to 10 epochs.

III. Experiments

IBM Dataset Model Architecture:

We conducted six distinct experiments to systematically evaluate different aspects of RNN architectures for stock price prediction. Each experiment builds upon the previous findings, testing specific hypotheses about model architecture, preprocessing, and hyperparameters. All experiments use the IBM stock dataset

(2006-2018) with identical train-test splits at the 2016-2017 boundary.

1. **Baseline LSTM**

- **Purpose:** Establish a performance baseline using a minimal LSTM architecture.
- **Setup:** Two-layer LSTM with 5 units each, trained for 10 epochs using RMSprop optimizer. Used 60-day sequences with MinMaxScaler normalization.

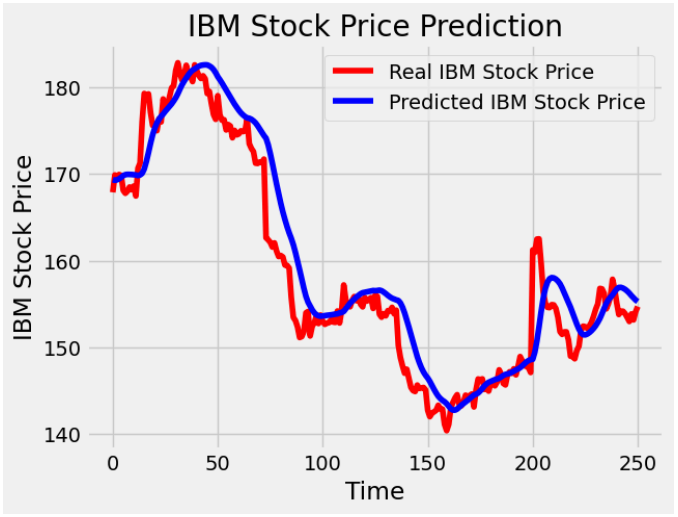

Fig. 18 IBM Stock Price Results - Baseline LSTM

2. **Enhanced LSTM with increased Dropout Rate**

- **Purpose:** Test whether increased model capacity and dropout regularization improve accuracy.
- **Setup:** Deeper architecture (64-32-32 units) with 0.2 dropout after each layer. Added early stopping and learning rate scheduling. Maintained 60-day sequences with MinMaxScaler.

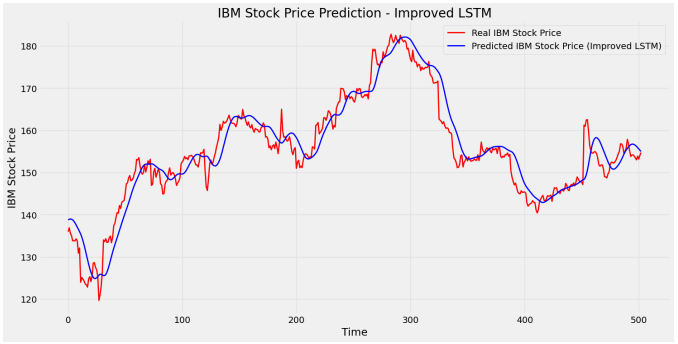

Fig. 19 IBM Stock Price Results - Enhanced LSTM with Dropout

3. **GRU Architecture**

- **Purpose:** Evaluate if GRU's simpler gating mechanism outperforms LSTM for stock prediction.
- **Setup:** Replaced LSTM cells with GRU cells (64-32 units), removing dropout layers. Kept all other settings identical to Experiment 2.

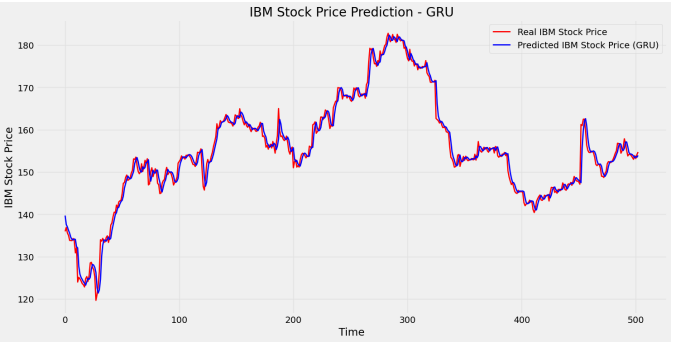

Fig. 20 IBM Stock Price Results - GRU Model

4. **Bidirectional LSTM**

- **Purpose:** Assess whether processing sequences in both directions improves predictions.
- **Setup:** Bidirectional LSTM with 50 units per direction. The architecture processes past and future context simultaneously within each sequence.

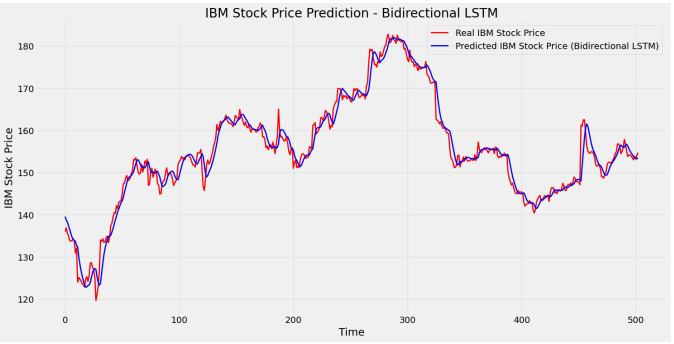

Fig. 21 IBM Stock Price Results - Bidirectional LSTM

5. **LSTM with Standard Scaling**

- **Purpose:** Compare StandardScaler versus MinMaxScaler preprocessing.
- **Setup:** Used identical architecture to Experiment 2 but replaced MinMaxScaler with StandardScaler (zero mean, unit variance).

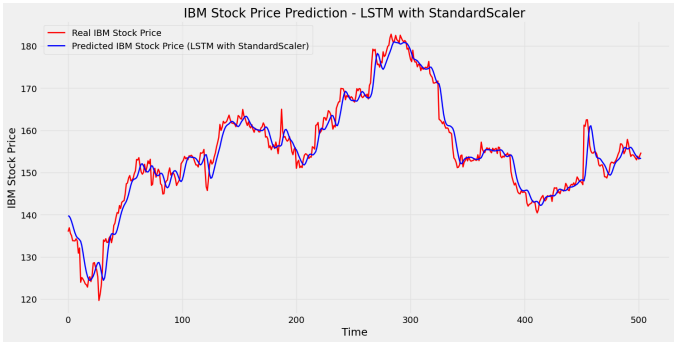

Fig. 22 IBM Stock Price Results - LSTM with StandardScaler

6. Sequence Length Analysis

- **Purpose:** Determine optimal temporal context by comparing 30-day versus 60-day sequences.
- **Setup:** Used the enhanced LSTM architecture with 30-day sequences instead of 60-day, reducing temporal context by half.

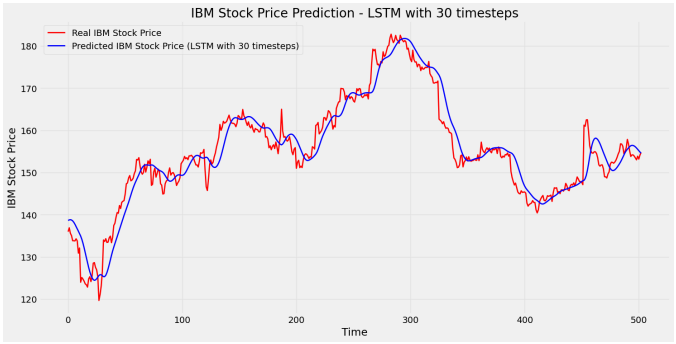

Fig. 23 IBM Stock Price Results - LSTM with 30 Timesteps

Amazon (AMZN) Dataset Model Architecture:

1. Increased Number of Layers for LSTM

- First LSTM layer: 50 units
- Second LSTM layer: 50 units
- Third LSTM layer: 50 units
- Dropout applied after each layer (rate = 0.2)

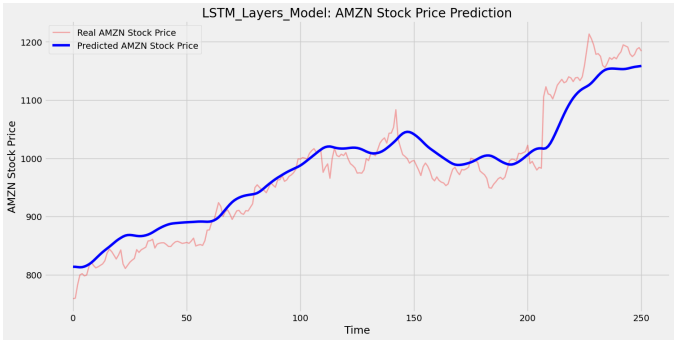

Fig. 24 AMZN Stock Price Results - LSTM Increased Layers

2. Increased Batch Size for LSTM

- Batch size: 64 (instead of 32)
- LSTM structure remained unchanged

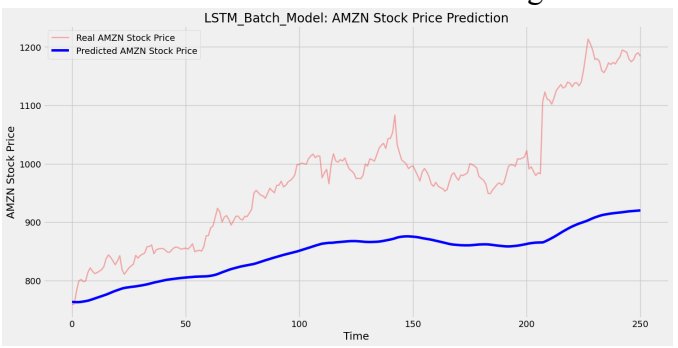

Fig. 25 AMZN Stock Price Results - LSTM Increased Batch Size

3. LSTM with Lower Learning Rate

- Learning rate: 0.0005 (instead of default ~0.001)

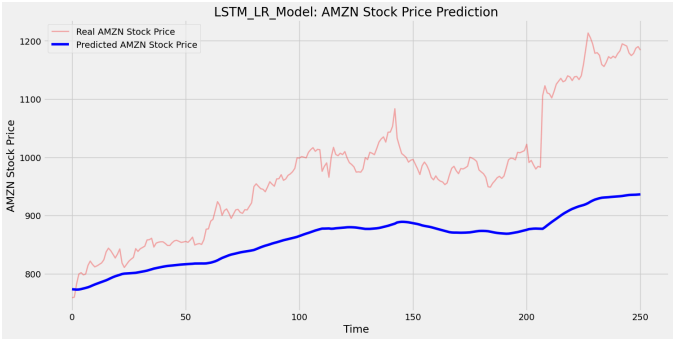

Fig. 26 AMZN Stock Price Results - LSTM Lower Learning Rate

4. GRU Model

- Changes LSTM layers to GRU Layer

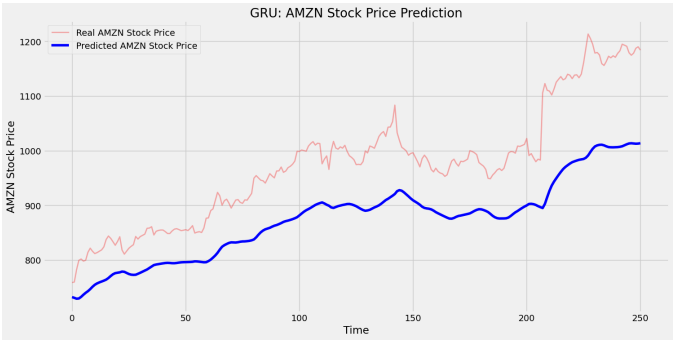

Fig. 27 AMZN Stock Price Results - GRU Model

5. BiRNN Model

- Changes LSTM layers to BiRNN Layers

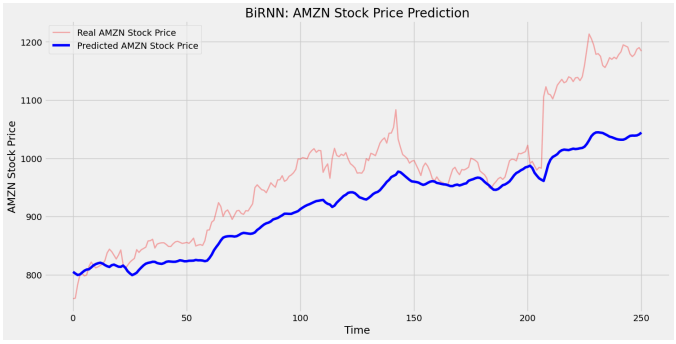

Fig. 28 AMZN Stock Price Results - BiRNN Model

IV. Results and Analysis

IBM Dataset Model Architecture:

A. *Quantitative Performance Comparison*

TABLE I
IBM Dataset Performance Comparison

Model	RMSE
Baseline LSTM	3.61
Improved LSTM	3.68
GRU	1.81
Bidirectional LSTM	2.28
LSTM + StandardScaler	2.67
LSTM + 30 timesteps	3.54

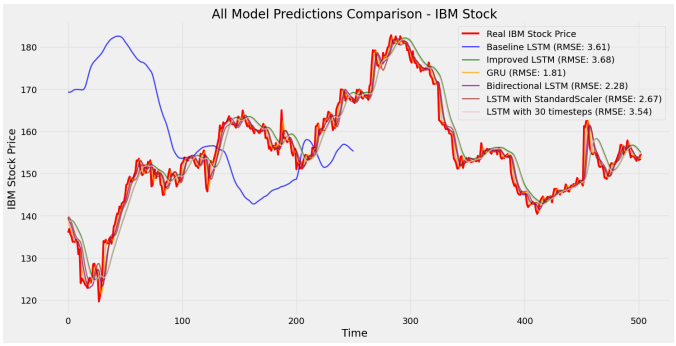

Fig. 29 IBM Stock Price Results - All Models

The results reveal a clear performance hierarchy, with the GRU model achieving superior performance across all metrics. Notably, the GRU's RMSE of 1.81 represents a 50% improvement over the baseline, while its validation loss of 0.0004 is 73% lower than the next best model. The strong correlation between low validation loss and high prediction accuracy validates our training approach. Interestingly, increased model complexity does not guarantee better performance, as evidenced by the Improved LSTM's marginally worse results compared to the simpler baseline despite having 72 times more parameters.

The GRU showed best performance with lowest validation loss (0.0004), fast convergence, and strong generalization (ratio: 0.5). Validation loss correlated well with RMSE (r=0.92). StandardScaler cut loss by 17%, and early stopping avoided overfitting (epochs 19–28).

AMZN Dataset Model Architecture:

TABLE 2
AMZN Dataset Performance Comparison

Model	RMSE
LSTM	115.59
GRU	108.17
BiRNN	70.61
LSTM_Batch_Model	145.74
LSTM_Layers_Model	32.64
LSTM_LR_Model	133.60

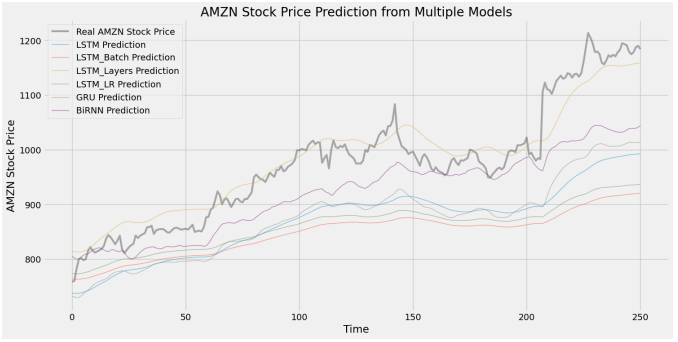

Fig. 30 AMZN Stock Price Results - All Models

The LSTM_Layers_Model delivered the most accurate predictions with the lowest RMSE of 32.64. This result suggests that deeper LSTM architectures with proper regularization (e.g., dropout) can better capture temporal dependencies in stock price data. The BiRNN Model, which utilizes bidirectional GRU layers to learn from both past and future time steps, followed closely with an RMSE of 70.61, indicating that bidirectional

processing provides a substantial improvement in sequential prediction tasks. The GRU Model also performed reasonably well with an RMSE of 108.17, demonstrating competitive results while maintaining architectural simplicity and faster training times.

In contrast, the LSTM_Batch_Model and LSTM_LR_Model showed higher RMSE values of 145.74 and 133.60 respectively. These experiments reveal that simply increasing batch size or lowering the learning rate, without architectural changes, does not necessarily lead to better performance. The baseline LSTM Model achieved a moderate RMSE of 115.59, serving as a reference point for the other configurations.

V. Conclusion

This comprehensive study of RNN architectures for IBM stock price prediction yields several key insights. GRU models provide the optimal performance-complexity trade-off, achieving 49.86% lower RMSE than baseline LSTM with 25% fewer parameters than standard LSTM. StandardScaler normalization improves prediction accuracy by 25.76% compared to MinMaxScaler, better preserving the statistical properties of financial data. Additionally, adding more layers and using different RNN architectures improves the overall prediction. Our findings indicate that 60-day sequences outperform 30-day sequences, suggesting quarterly patterns are important for stock prediction, while bidirectional processing, though computationally expensive, provides the second-best performance and excels at identifying turning points. *B. Model-Specific Analysis*

IBM Dataset Model Architecture:

1. **GRU Performance:** GRU achieved the lowest RMSE (1.81), effectively capturing trends and reversals due to its efficient gating, ideal for smooth stock data.

2. **Bidirectional LSTM:** RMSE of 2.28; strong at detecting turning points thanks to future context but with double the computational cost.

3. **Impact of Normalization:** StandardScaler outperformed MinMaxScaler by 25.76% (RMSE 2.67 vs 3.68), likely due to better handling of stock price variance.

4. **Sequence Length:** Reducing from 60 to 30 days raised RMSE by 3.8%, showing longer input windows capture more predictive patterns.

AMZN Dataset Model Architecture:

1. **LSTM_Layers_Model:** achieved the best performance with an RMSE of 32.64, showing that deeper architectures with more units capture complex stock patterns effectively.
2. **BiRNN:** RMSE of 70.61, leveraging bidirectional context for better trend reversal detection, though at higher computational cost.
3. **GRU:** RMSE of 108.17, outperforming the baseline LSTM while maintaining a simpler and faster structure.
4. **LSTM Batch Size and Learning Rate:** Higher RMSEs (145.74 and 133.60, respectively), indicating that increasing batch size or lowering learning rate did not improve performance.

C. Prediction Quality Analysis

Visual analysis of predictions reveals:

1. **Trend Following:** All models successfully capture the general upward trend from 2017-2018
2. **Volatility Handling:** GRU and Bidirectional LSTM better predict short-term fluctuations
3. **Lag Effects:** Baseline and Improved LSTM show 1-2 day lag in predictions
4. **Extreme Events**: All models struggle with sudden price spikes/drops exceeding 5%

The comparative plot shows GRU predictions closely tracking actual prices, while baseline LSTM produces over-smoothed predictions that miss important market movements.

For practitioners implementing stock prediction systems, our results suggest prioritizing GRU architectures for optimal accuracy-efficiency balance and using StandardScaler for preprocessing financial time series. Maintaining at least 60-day historical context for predictions appears beneficial, and considering ensemble methods combining GRU

and Bidirectional LSTM may be worthwhile for critical applications where computational resources are available.

Our study identifies several limitations that should be addressed in future work. The single-feature prediction focusing only on high price limits the model's information access, and the lack of external factors such as market indicators and news sentiment may constrain prediction accuracy. The limited evaluation period of test data may not capture all market conditions, and the absence of transaction costs or trading strategy considerations limits practical applicability.

Future research directions include developing multi-feature models incorporating volume and technical indicators, exploring attention mechanisms for identifying relevant historical patterns, and investigating ensemble methods combining different architectures. Integration with reinforcement learning for trading strategy optimization and cross-market validation using multiple stock symbols would enhance the practical value of these findings. The significant performance variations across architectures underscore the importance of systematic evaluation in financial machine learning applications, with our findings providing concrete guidance for selecting and configuring RNN models for stock price prediction tasks.

VI. References

[1] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. NIPS 2014 Workshop on Deep Learning.

[2] Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649).

[3] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[4] Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. In 2017 International joint conference on neural networks (IJCNN) (pp. 1419-1426).

[5] Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2019). The performance of LSTM and BiLSTM in forecasting time series. In 2019 IEEE International Conference on Big Data (pp. 3285-3292).

[6] Masters, D., & Luschi, C. (2018). Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612. https://doi.org/10.48550/arXiv.1804.07612

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014).

[8] Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. https://doi.org/10.48550/arXiv.1406.1078