فهرست مطالب

فهرست تصاویر

توضیح قالب گزارش

گروه تدریسیاری محترم؛ با عرض سلام و خداقوت. این گزارش در قالب لاتک (LaTeX) آماده شده است. قالب آن را عیناً مشابه و دارای مولفه‌های قالبی که در سامانه قرار داده بودید طراحی و آماده کردیم. هم‌چنین آن را بر بستر برخط Overleaf هم بارگذاری کرده‌ایم که از طریق این لینک در دسترس است.

پرسش ۱. تخمین قیمت رمزارزها

۱ پاسخ پرسش اول

توضیح پوشهٔ کدهای تخمین قیمت رمزارزها

کدهای مربوط به این قسمت، علاوه بر پوشهٔ محلی کدها در این لینک گوگل‌کولب آورده شده است.

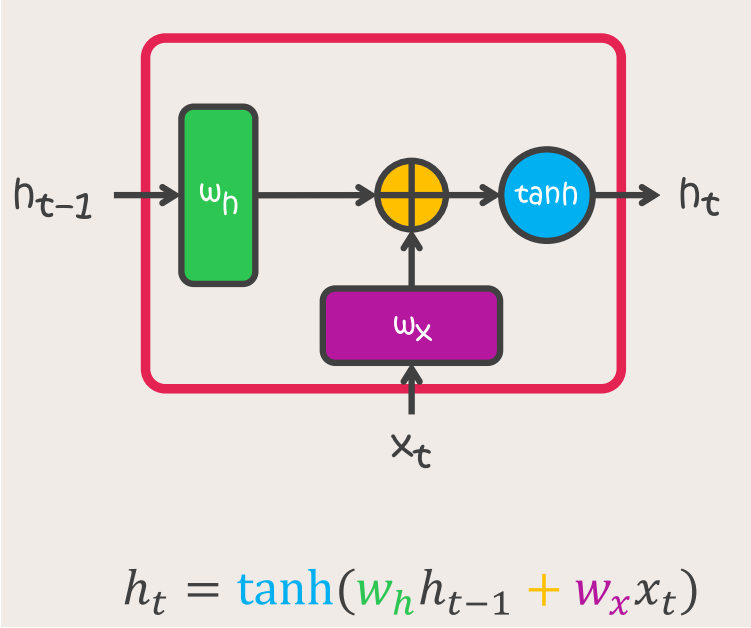
۱.۱ پاسخ قسمت ۱ - توضیحات مدل‌ها

مقاله دو مدل پیشنهادی برای پیش‌بینی قیمت ارزهای دیجیتال لایتکوین و مونرو معرفی می‌کند. مدل اول از یک ساختار مبتنی بر GRU و LSTM استفاده می‌کند. شبکه‌های RNN برای مدل‌کردن داده‌های دنباله‌ای طراحی شده‌اند و دارای حافظهٔ دنباله‌ای هستند. ساختار این شبکه‌ها به گونه‌ای تعریف شده‌اند که خروجی آن‌ها ترکیبی از ورودی‌های کنونی و اطلاعاتی است که از ورودی‌ها و خروجی‌های قبلی یادگرفته‌اند. این خاصیت آن‌ها را در کارهای مربوط به دنباله‌ها، مانند پیش‌بینی سری زمانی، کارآمد می‌کند. اما چالشِ کم‌شدن گرادیان‌ها در RNN باعث می‌شود که آن‌ها برای آموزش سخت‌تر باشند. LSTM و GRU دو نوع مختلف از RNN هستند که به گونه‌ای طراحی شده‌اند که مشکل کم‌شدن گرادیان را برطرف کنند. بسیاری از تحقیقات قبلی، نشان داده‌اند که LSTM و GRU در پیش‌بینی سری‌های زمانی برتری دارند. بنابراین، مقاله یک روش پیشنهادی برای استفاده از هر دو مدل LSTM و GRU با هم پیشنهاد می‌دهد تا از مزایای هر دو مدل بهره‌مند شود.

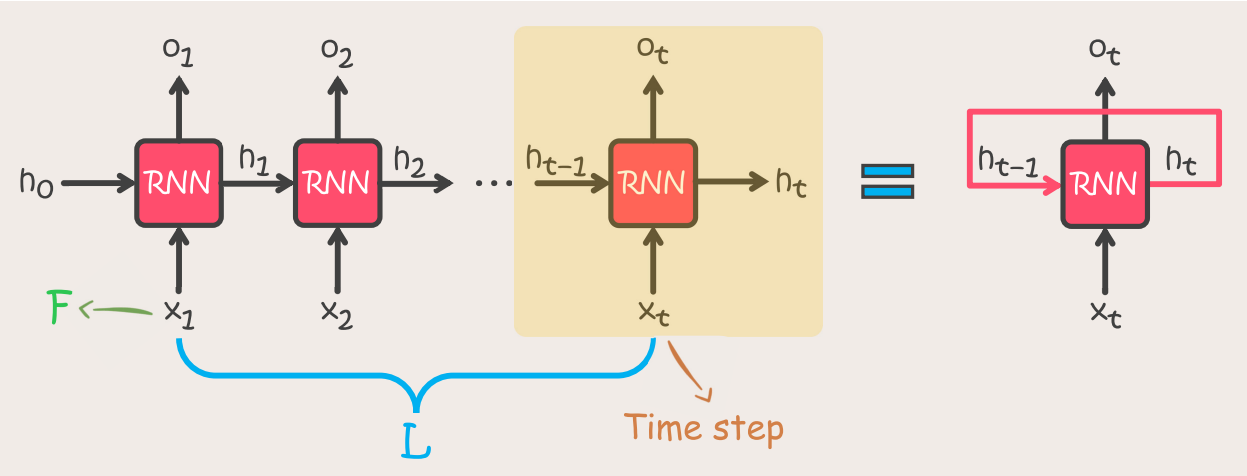
مدل پیشنهادی مقاله یک ترکیب از شبکهٔ GRU و LSTM است. هر دوی این مدل‌ها یک ورودی مشترک دارند و هر دو شبکه به یکدیگر ادغام شده و از طریق یک لایهٔ تماماً‌متصل (چگال)، خروجی نهایی به دست می‌آید. شبکهٔ GRU شامل یک لایه GRU با ۳۰ نورون است. پس از آن، یک لایه dropout برای جلوگیری از بیش‌برازش اضافه شده است. خروجی dropout به یک لایهٔ چگال وارد می‌شود. در حالی که شبکهٔ LSTM یک لایه LSTM با ۳۰ نورون دارد. این لایه پس از یک لایه dropout برای جلوگیری از بیش‌برازش اضافه شده است. سپس یک لایه LSTM دیگر با ۵۰ نورون اضافه می‌شود. خروجی این لایه به یک لایهٔ چگال وارد می‌شود. خروجی از هر دو شبکه ترکیب شده و از طریق یک لایهٔ چگال که خروجی قیمت پیش‌بینی‌شده را ارائه می‌دهد، گذر کرده است. تابع فعال‌سازی استفاده شده ReLU و بهینه‌ساز مورداستفاده Adam است. این مدل برای ۱۰۰ دوره آموزش داده شده است.

پس از آموزش، پیش‌بینی‌ها انجام می‌شوند. برای پیش‌بینی، آخرین n مشاهده به‌عنوان ورودی دریافت می‌شوند، که n طول دنبالهٔ ورودی مدل است. با استفاده از این ورودی، مقدار بعدی پیش‌بینی می‌شود. پس از دریافت این مقدار، ورودی بعدی با شامل آخرین $n-1$ مقدار و مقدار پیش‌بینی‌شده آماده می‌شود. این فرآیند به تعداد k بار انجام می‌شود، که k اندازهٔ پنجرهٔ پیش‌بینی است. این فرآیند به عنوان الگوریتم ۲ معرفی شده است. (شکل ۶)

مدل استفاده شده برای مقایسه هم، یک شبکهٔ LSTM با لایه‌های LSTM دارای ۵۰ نورون و یک لایهٔ چگال است. این مدل با استفاده از بهینه‌ساز Adam و فعال‌ساز ReLU در ۱۰۰ دوره آموزش داده می‌شود.



شکل ۱: نمایی مرتبط با شبکه‌های بازگشتی.

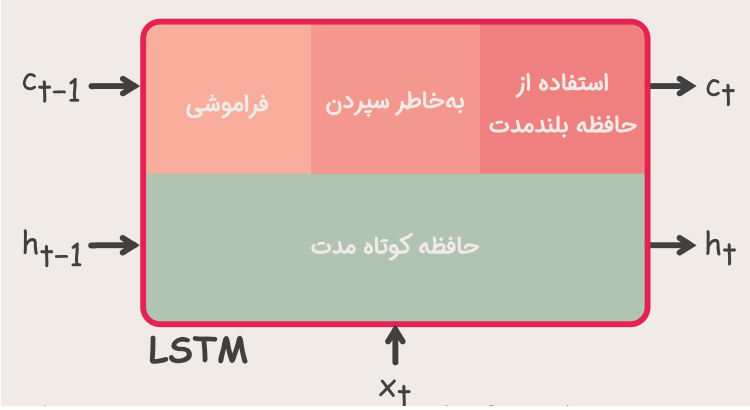


شکل ۲: نمایی مرتبط با شبکه‌های بازگشتی.

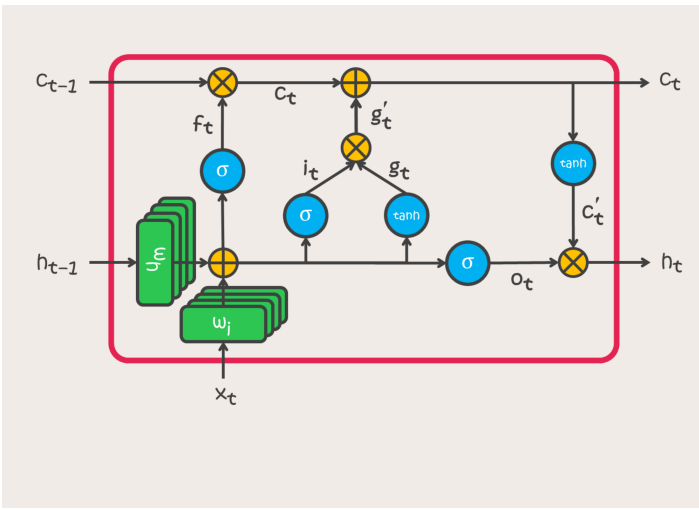
۲.۱ پاسخ قسمت ۲ - مجموعه‌دادگان و پیش‌پردازش آن‌ها

همانطور که در مقاله ذکر شده است، داده‌های مورد استفاده در این پژوهش از سایت Investing.com جمع‌آوری و دریافت شده‌اند. این سایت یک پرتال جهانی است که تحلیل و اخباری درباره بازارهای مالی جهان ارائه می‌دهد. داده‌ها برای دو ارز دیجیتال (کریپتوکارنسی)، لایت‌کوین و مونرو، انتخاب و جمع‌آوری شد. داده‌های جمع‌آوری‌شده دارای پنج ویژگی هستند که به شرح زیر هستند:

- قیمت (`Price`): میانگین قیمت رمزارز در طول روز.

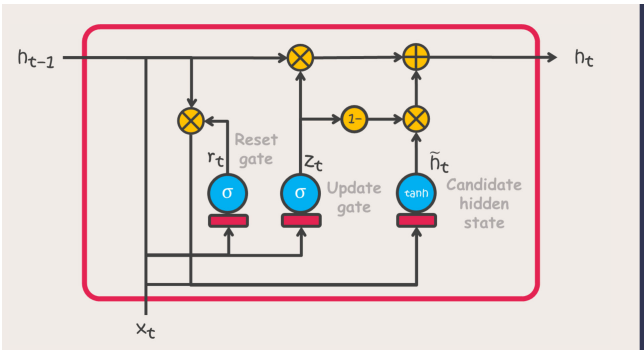


شکل ۳: نمایی مرتبط با شبکه‌های LSTM (سه ویژگی حافظهٔ بلندمدت).



- $f_t = \sigma(w_{hf}h_{t-1} + w_{if}x_t + b_{hf} + b_{if})$
  - $c_t = c_{t-1} \odot f_t$
- $g_t = \tanh(w_{hg}h_{t-1} + w_{ig}x_t + b_{hg} + b_{ig})$
- $i_t = \sigma(w_{hi}h_{t-1} + w_{ii}x_t + b_{hi} + b_{ii})$
  - $g'_t = g_t \odot i_t$
  - $c_t = c_t + g'_t$
- $o_t = \sigma(w_{ho}h_{t-1} + w_{io}x_t + b_{ho} + b_{io})$
  - $h_t = o_t \odot \tanh(c_t)$

شکل ۴: نمایی مرتبط با شبکه‌های LSTM.



- $r_t = \sigma(w_{hr}h_{t-1} + w_{xr}x_t + b_{hr} + b_{xr})$
- $z_t = \sigma(w_{hz}h_{t-1} + w_{xz}x_t + b_{hz} + b_{xz})$
- $\tilde{h}_t = tanh(w_{h\tilde{h}}(r_t \odot h_{t-1}) + w_{x\tilde{h}}x_t + b_{h\tilde{h}} + b_{x\tilde{h}})$
- $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$

شکل ۵: نمایی مرتبط با شبکه‌های GRU.

- باز (Open): قیمت بازشدن رمزارز در طول روز (قیمت اول روز).
- بسته (Close): قیمت بستن رمزارز در طول روز (قیمت آخر روز).

Input: $D_x \in features, D_y \in target, \mathcal{P}_{Window} \in prediction\ window\ length$
Output: $\mathcal{P}_{Values} \in predicted\ prices$

procedure PREDICT_PRICE($D_x, D_y, \mathcal{P}_{Window}$)
  $\mathfrak{R} \leftarrow \xi(D_x, D_y)$ ▷ $\mathfrak{R}$ is trained model
  $\mathcal{P}_{Values} \leftarrow \emptyset$ ▷ $\mathcal{P}$ is the prediction
  $\iota \leftarrow append(D_{x \to lastValue})$
  DELETE($\iota, D_{x \to FirstValue}$)
  $\iota \to append(D_{y \to LastValue})$
  for $\lambda = 1, 2, \ldots, \mathcal{P}_{Window}$ **do**
    $\mathcal{P}_{Values} \to append(\mathfrak{R}.\mathcal{P}_{\iota})$
    DELETE($\iota, D_{x \to FirstValue}$)
    $\iota \to append(\mathcal{P}_{Values}.D_{y \to LastValue})$
  end for
  $\mathcal{R}(\mathcal{P}_{Values})$
end procedure

شکل ۶: الگوریتم پیش‌بینی ارزدیجیتال.

- بالا (`High`): بیشترین قیمت رمزارز در طول روز.
- پایین (`Low`): کمترین قیمت رمزارز در طول روز.
- حجم (`Volume`): حجم معاملات رمزارز در طول روز

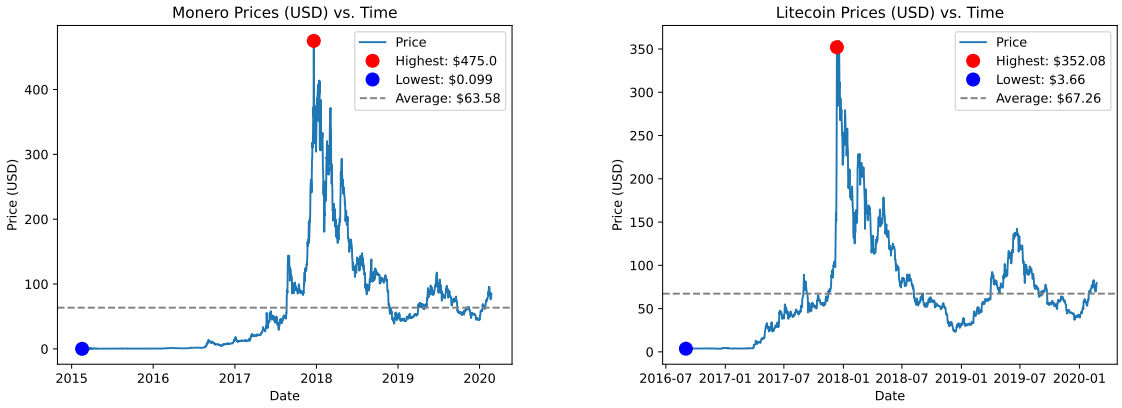
داده ها به صورت روزانه در دسترس هستند. جزئیات هر مجموعه داده به شرح زیر است:

- لایت‌کوین: از ۲۴ اوت ۲۰۱۶ تا ۲۳ فوریه ۲۰۲۰ (۱۲۷۹ نقطه‌داده).
- مونرو: از ۳۰ ژانویه ۲۰۱۵ تا ۲۳ فوریه ۲۰۲۰ (۱۸۵۱ نقطه‌داده).

مقاله دو نمودار قیمت‌های موجود در فایل‌های مجموعه‌داده کشیده که ما برای نمایش بهتر، از دستورات پایتون زیر استفاده کردیم و نمودارهای مربوطه را در ؟؟ نمایش دادیم.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load data from CSV file
data = pd.read_csv('/content/Litecoin.csv')

# Convert 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Plot prices vs. time
plt.plot(data['Date'], data['Price'], label='Price')

# Highlight highest and lowest prices with red and blue points
highest_price = data.loc[data['High'].idxmax(), 'Price']
lowest_price = data.loc[data['Low'].idxmin(), 'Price']
plt.plot(data.loc[data['High'].idxmax(), 'Date'], highest_price, 'ro', markersize=10, label='
    Highest: $' + str(highest_price))
```

```
plt.plot(data.loc[data['Low'].idxmin(), 'Date'], lowest_price, 'bo', markersize=10, label='Lowest
    : $' + str(lowest_price))

# Plot average with dash-line
average_price = data['Price'].mean()
plt.axhline(average_price, linestyle='--', color='gray', label='Average: $' + str(round(
    average_price, 2)))

# Format plot
plt.title('Litecoin Prices (USD) vs. Time')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()

# Save plot as PDF
plt.savefig('Litecoin Prices.pdf')
```



Monero (ب)　　Litecoin (آ)

شکل ۷: نمودار توزیعی داده‌ها.

در مورد پیش‌پردازش، مقاله این‌گونه عنوان می‌کند که محدودهٔ مقادیر داده‌ها به طور گسترده‌ای متفاوت است و نمی‌توان به طور مستقیم از آن‌ها استفاده کرد. برای مقیاس‌کردن مقادیر داده‌ها به محدوده‌ای خاص از مقادیر، عملیات نرمال‌سازی انجام می‌شود. در این مطالعه، از نرمال‌سازی maxmin- برای آماده‌سازی داده‌ها استفاده شده است که مقادیر داده‌ها را به محدودهٔ ۰ تا ۱ می‌برد:

$$x_{\text{normalized}} = \frac{x_{\text{original}} - x_{\min}}{x_{\max} - x_{\min}} \qquad (۱)$$

این مقاله هم‌چنین، قیمت را به عنوان ویژگی اصلی خود انتخاب کرده است و مدل را با داده‌های ۳۰ روز گذشته آموزش می‌دهد. پس از نرمال‌سازی، داده‌ها به شکلی مناسب برای ورودی به مدل تبدیل شده است. داده‌ها به نمونه‌های ورودی/خروجی تقسیم می‌شوند. ورودی دارای ۳۰ مرحلهٔ زمانی است، به عبارت دیگر داده‌های ۳۰ روز گذشته و ۱ مرحله زمانی به عنوان خروجی. این کار با استفاده از الگوریتم آورده‌شده در شکل ۸ انجام می‌شود: پس از انجام نرمال سازی، داده‌ها به دو مجموعه به نام‌های

Input: $D \in \{normalized\ prices\ of\ cryptocurrency\}$
Output: $D_x \in features, D_y \in target$

procedure PROCESS_DATA($D, \varpi^{Count}_{Features}$)
  $D_x \leftarrow \emptyset, \forall D_x \in D_{Training \rightarrow Features}$
  $D_y \leftarrow \emptyset, \forall D_y \in D_{Target}$
  $n \leftarrow \ell$ (D) ▷ ℓ is the input data length
  for $\lambda = 1, 2, \ldots, n$ **do**
    $\zeta \leftarrow \lambda + \varpi^{Count}_{Features}$
    if $\zeta > (n-1)$ **then**
      break
    end if
    $\mathcal{T}_x \leftarrow$ data$[\lambda : \zeta]$ ▷ $\mathcal{T}_x$ is temporary variable for x
    $\mathcal{T}_y \leftarrow$ data$[\zeta]$ ▷ $\mathcal{T}_y$ is temporary variable for y
    $D_{x \rightarrow append}(\mathcal{T}_x)$
    $D_{y \rightarrow append}(\mathcal{T}_y)$
  end for
  $\mathcal{R}(D_x, D_y)$ ▷ $\mathcal{R}$ is return the D_x, D_y values
end procedure

شکل ۸: الگوریتم آماده‌سازی داده‌ها.

مجموعهٔ آموزش و آزمون تقسیم می شوند. مدل هیبریدی پیشنهادی با استفاده از داده‌های آموزشی آموزش داده می‌شود و برای پیش‌بینی، آخرین w (طول پنجرهٔ ورودی) به مدل وارد می‌شود که قیمت روز بعد را پیش‌بینی کند. این قیمت پیش‌بینی‌شده دوباره به مدل وارد می‌شود تا قیمت روز بعدی پیش‌بینی شود. این فرایند برابر با طول پنجرهٔ پیش‌بینی تکرار می‌شود. مجموعه‌داده‌های آزمون برای ارزیابی عملکرد پس از پیش بینی قیمت‌ها استفاده می‌شود.

برای رسیده به اهداف پیش‌پردازشی این قسمت از سوال، ابتدا کتابخانه‌های لازم را فراخوانی می‌کنیم. سپس، مجموعه‌داده را بارگذاری می‌کنیم. سپس علامت‌های غیرعددی آن را حذف می‌کنیم. مثلاً علامت درصد را حذف و به‌جای علامت میلیون و هزار عدد داده را در ۱۰۰۰۰۰۰ و ۱۰۰۰ ضرب می‌کنیم. در ادامه، ویژگی اصلی آموزش؛ یعنی قیمت را مشخص و انتخاب می‌کنیم. و داده‌ها را به‌صورت یک آرایهٔ دوبُعدی یک‌ستونه درمی‌آوریم. داده‌های آموزش را با `MinMaxScaler` بین صفر و یک نرمال می‌کنیم. طول پنجرهٔ ورودی که تعداد روزهای گذشته برای پیش‌بینی است را تعیین می‌کنیم (۳۰). یک تابع مخصوص پیش‌پردازش می‌نویسیم که دو پارامتر داده و طول پنجرهٔ ورودی ر می‌گیرد و با استفاده از الگوریتم آورده‌شده در شکل ۸ گار می‌کند و نمونه‌های ورودی-خروجی را برمی‌گرداند. داخل این تابع، n طول آرایهٔ داده‌های ورودی است. سپس تابع برای هر اندیس i آرایهٔ داده‌های ورودی، اندیس پایانی را با اضافه‌کردن طول پنجره (گام زمانی) به آن محاسبه می‌کند. اگر این عدد بزرگ‌تر یا مساوی با طول آرایهٔ داده منهای یک باشد،تابع از حلقه خارج می‌کشود. در غیر این صورت، تابع یک زیرآرایه از آرایهٔ داده‌های ورودی با شروع از اندیس i و پایان در اندیس پایانی استخراج و ایجاد می‌کند. در ادامه، زیرآرایهٔ استخراج‌شده در یک متغیر موقت به نام `temp_X` ذخیره می‌شود. مقدار هدف برای این دنبالهٔ ورودی، مقدار در اندیس پایانی است که در یک متغیر دیگر به نام `temp_Y` ذخیره می‌شود. متغیرهای موقت ورودی و خروجی به ترتیب به لیست‌های `Dx` و `Dy` اضافه می‌شوند. در نهایت، تابع یک تاپل از لیست‌های `Dx` و `Dy` را برمی‌گرداند که حاوی جفت ورودی/خروجی مناسب برای آموزش مدل هستند. در نهایت هم تقسیم‌بندی داده‌ها به دو مجموعهٔ آموزش و ارزیابی صورت می‌پذیرد. با ارائهٔ این توضیحات دستورات نوشته‌شده به شرح زیر است:

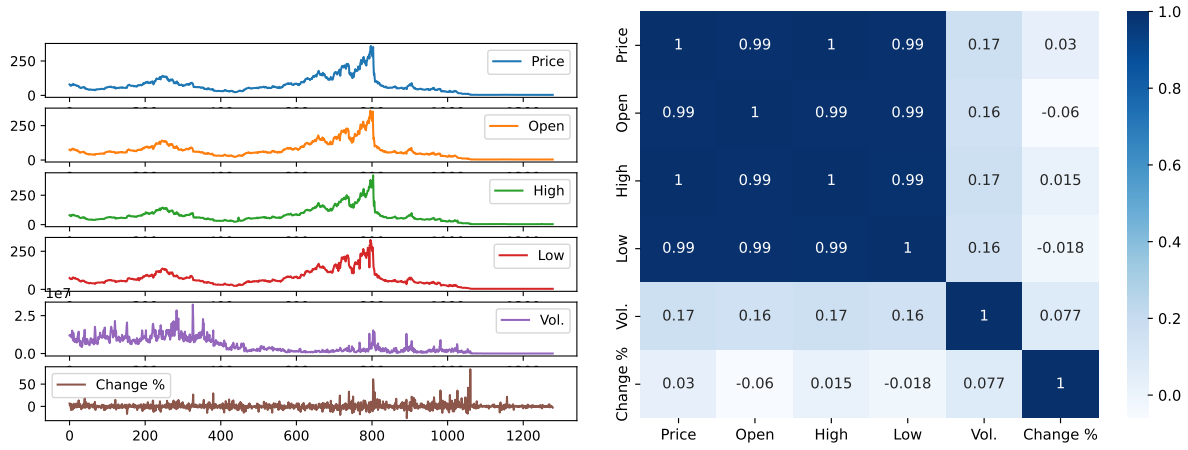
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

```

```
# Load data from CSV file
data = pd.read_csv('/content/Litecoin.csv')

# Convert string values to numeric values in the 'Vol.' column.
def convert_vol(vol):
    if vol.endswith('M'):
        return int(float(vol[:-1]) * 1000000)
    elif vol.endswith('K'):
        return int(float(vol[:-1]) * 1000)
    else:
        return int(vol)

data['Vol.'] = data['Vol.'].apply(convert_vol)
data['Change %'] = data['Change %'].str.replace('%', '').astype(float)

# Select the primary feature for training
training_data = data['Price'].values.reshape(-1, 1)

# Normalize the training data to scale values between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
training_data_normalized = scaler.fit_transform(training_data)

# Define the input window length (number of past days to use for prediction)
input_window_length = 30

# Use the PROCESS_DATA algorithm to create input/output samples
def process_data(data, input_window_length):
    Dx = []
    Dy = []
    n = len(data)
    for i in range(n):
        end_index = i + input_window_length
        if end_index > n-1:
            break
        temp_X = data[i:end_index, :]
        temp_Y = data[end_index, :]
        Dx.append(temp_X)
        Dy.append(temp_Y)
    return (Dx, Dy)

# Create input/output samples using the training data
train_X, train_Y = process_data(training_data_normalized, input_window_length)

# Split the data into training and testing sets (use 80/20 split)
split_index = int(len(train_X)*0.8)
```

```
train_X, test_X = train_X[:split_index], train_X[split_index:]
train_Y, test_Y = train_Y[:split_index], train_Y[split_index:]
```

در ادامه و با استفاده از دستورات زیر، نمودار هم‌بستگی میان ویژگی‌ها و هم‌چنین روند تغییرات آن‌ها را در شکل ۹ نشان می‌دهیم. مشاهده می‌شود که هم‌بستگی میان داده‌های مربوط به میزان قیمت (شروع، پایان، بالاترین و کم‌ترین) با توجه به روند تغییرات آن‌ها کاملاً منطقی است.

```
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files

plt.figure(figsize=(7,5))
sns.heatmap(df.corr(),annot=True,cmap=plt.cm.Blues)

plt.savefig('heatmap.pdf', format='pdf', bbox_inches='tight')
files.download('heatmap.pdf')

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(7,5))
df[['Price','Open','High','Low','Vol.','Change %']].plot(ax=ax, subplots=True)
plt.savefig('plot.pdf', format='pdf', dpi=300, bbox_inches='tight')
```



(ب) روند تغییرات (آ) هم‌بستگی

شکل ۹: نمودار هم‌بستگی میان ویژگی‌ها و روند تغییرات آن‌ها.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load data from CSV file
data = pd.read_csv('/content/Litecoin.csv')

```

```python
# Convert string values to numeric values in the 'Vol.' column.
def convert_vol(vol):
    if vol.endswith('M'):
        return int(float(vol[:-1]) * 1000000)
    elif vol.endswith('K'):
        return int(float(vol[:-1]) * 1000)
    else:
        return int(vol)

data['Vol.'] = data['Vol.'].apply(convert_vol)
data['Change %'] = data['Change %'].str.replace('%', '').astype(float)

# Select the primary feature for training
training_data = data['Price'].values.reshape(-1, 1)

# Normalize the training data to scale values between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
training_data_normalized = scaler.fit_transform(training_data)

# Define the input window length (number of past days to use for prediction)
input_window_length = 30

# Use the PROCESS_DATA algorithm to create input/output samples
def process_data(data, input_window_length):
    Dx = []
    Dy = []
    n = len(data)
    for i in range(n):
        end_index = i + input_window_length
        if end_index > n-1:
            break
        temp_X = data[i:end_index, :]
        temp_Y = data[end_index, :]
        Dx.append(temp_X)
        Dy.append(temp_Y)
    return (Dx, Dy)

# Create input/output samples using the training data
train_X, train_Y = process_data(training_data_normalized, input_window_length)

# Split the data into training and testing sets (use 80/20 split)
split_index = int(len(train_X)*0.8)
train_X, test_X = train_X[:split_index], train_X[split_index:]
train_Y, test_Y = train_Y[:split_index], train_Y[split_index:]
```

۳.۱ پاسخ قسمت ۳ - آموزش مدل‌ها

گام اول آماده‌سازی داده و تبدیل آن به یک ورودی مناسب برای مدل است. داده به چندین جفت ورودی-خروجی تقسیم می‌شود. ورودی یک دنباله از مقادیر یا مشاهدات گذشته است که با یک مقدار خروجی عمل می‌کند. طول این دنباله (n) یک هایپرپارامتر است. برای ایجاد جفت ورودی-خروجی، $[x_0, x_1, \cdots, x_{n-1}]$ را به‌عنوان یک ورودی و x_n را به عنوان خروجی برای این ورودی انتخاب می‌کنیم. برای ورودی بعدی، $[x_0, x_1, \cdots, x_n]$ را و خروجی را x_{n+1} انتخاب می‌کنیم. داده‌ها به این شیوه آماده می‌شوند. الگوریتم ایجاد جفت ورودی-خروجی به صورت الگوریتم ۱ در بخش ۲.۱ بحث شده است.

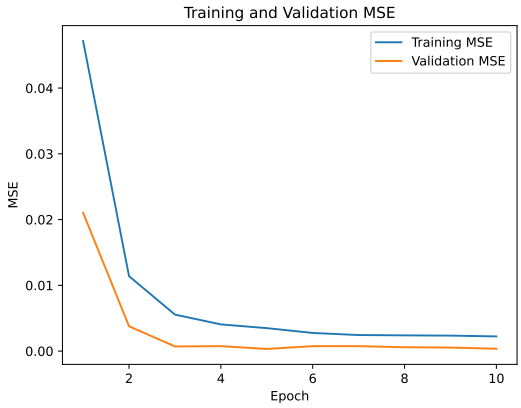
گام بعدی، آموزش مدل با توجه به داده‌هاست. برای این منظور و در راهبرد اول، دستوراتی می‌نویسیم که ا شبکه‌های عصبی LSTM و GRU برای پیش‌بینی قیمت ارز دیجیتال استفاده می‌کند. در ابتدا کتاب‌خانه‌های مورد نیاز برای اجرای کد فراخوانی شده و مجموعه‌داده‌های مربوط به لایت‌کوین و مونرو بارگذاری می‌شوند. سپس داده‌های آموزشی انتخاب می‌شوند و به صورت نرمال‌شده با استفاده از `MinMaxScaler` مقیاس‌بندی می‌شوند تا بین ۰ و ۱ باشند. سپس از تابع الگوریتمی `PROCESS_DATA` برای ایجاد نمونه‌های ورودی-خروجی استفاده می‌شود. این الگوریتم برای هر روز، یک دنباله‌ای از اطلاعات قبلی با طول تعیین شده توسط `input_window_length` (طول پنجره ورودی) را به‌عنوان ورودی و مقدار قیمت روز بعد را به عنوان خروجی برای مدل ایجاد می‌کند. در ادامه، داده‌های آموزشی به دو بخش آموزشی و آزمون با نسبت ۸۰:۲۰ تقسیم می‌شوند و مدل طراحی می‌شود. مدل شامل یک لایه ورودی با طول `input_window_length`، دو شبکهٔ GRU و LSTM با اندازه‌های ۳۰ و ۵۰ و لایه‌های چگال با اندازهٔ ۳۲ و لایه‌های دورریزی است. خروجی هر دو شبکه در نهایت وارد یک لایهٔ چگال می‌شود. مدل با تابع هزینهٔ MSE و بهینه‌ساز Adam آموزش داده می‌شود و برای جلوگیری از بیش‌برازش، از EarlyStopping استفاده می‌شود. برای نمایش بهتر نتایج حالت بدون استفاده از EarlyStopping را هم امتحان کرده و نتایج آن را نشان داده‌ایم. درنهایت، نمودارهای خواسته‌شده در صورت سوال نمایش داده شده و ذخیره می‌شوند. دستورات به شرح زیر است و نتایج به‌صورتی است که در شکل ۱۰ تا شکل ۱۷ نشان داده شده است. همان‌طور که نشان داده شده است نتایج در حالت استفاده از شبکهٔ پیشنهادی تقریباً حالت مطلوبی به خود گرفته است و در حالت استفاده از مدل LSTM ساده، نتایج قابل قبولی حاصل نشده است. لازم به ذکر است که اضافه‌کردن مفهوم دادهٔ اعتبارسنجی به کد برای نشان‌دادن همین موضوع بوده است. وگرنه می‌شد صرفاً با داده‌های آموزشی مدل‌ها را آموزش و نتایج را نشان داده که این کار را هم انجام دادیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Input, GRU, LSTM, Dense, Dropout, concatenate
from keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, mean_absolute_error,
    mean_absolute_percentage_error
from matplotlib.backends.backend_pdf import PdfPages
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, GRU, Dropout, Dense, concatenate
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

```

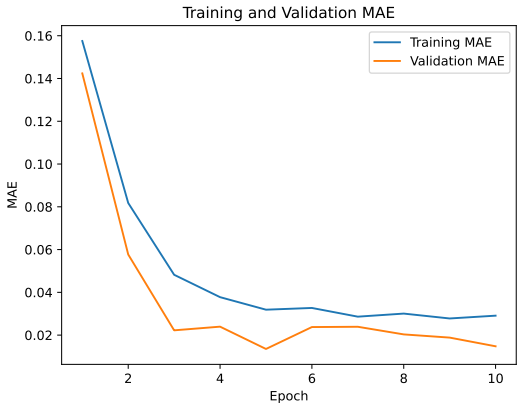
```python
# Load data from CSV file
data = pd.read_csv('/content/Litecoin.csv')

# Convert string values to numeric values in the 'Vol.' column.
def convert_vol(vol):
    if vol.endswith('M'):
        return int(float(vol[:-1]) * 1000000)
    elif vol.endswith('K'):
        return int(float(vol[:-1]) * 1000)
    else:
        return int(vol)

data['Vol.'] = data['Vol.'].apply(convert_vol)
data['Change %'] = data['Change %'].str.replace('%', '').astype(float)

# Select the primary feature for training
training_data = data['Price'].values.reshape(-1, 1)

# Normalize the training data to scale values between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
training_data_normalized = scaler.fit_transform(training_data)

# Define the input window length (number of past days to use for prediction)
input_window_length = 30

# Use the PROCESS_DATA algorithm to create input/output samples
def process_data(data, input_window_length):
    Dx = []
    Dy = []
    n = len(data)
    for i in range(n):
        end_index = i + input_window_length
        if end_index > n-1:
            break
        temp_X = data[i:end_index, :]
        temp_Y = data[end_index, 0]
        Dx.append(temp_X)
        Dy.append(temp_Y)
    Dx = np.array(Dx)
    Dy = np.array(Dy)
    return (Dx, Dy)

# Create input/output samples using the training data
train_X, train_Y = process_data(training_data_normalized, input_window_length)

```

```
# Split the data into training and testing sets (use 80/20 split)
split_index = int(len(train_X)*0.8)
train_X, test_X = train_X[:split_index], train_X[split_index:]
train_Y, test_Y = train_Y[:split_index], train_Y[split_index:]

# Define the model
inputs = Input(shape=(input_window_length, 1))

# GRU Network
gru = GRU(30, activation='relu', return_sequences=False)(inputs)
gru = Dropout(0.2)(gru)
gru = Dense(32, activation='relu')(gru)

# LSTM Network
lstm = LSTM(30, activation='relu', return_sequences=True)(inputs)
lstm = Dropout(0.2)(lstm)
lstm = LSTM(50, activation='relu')(lstm)
lstm = Dense(32, activation='relu')(lstm)

# Combine GRU and LSTM networks
combined = concatenate([gru, lstm])
outputs = Dense(1)(combined)

# LSTM Network
# lstm = LSTM(50, activation='relu', return_sequences=True)(inputs)
# outputs = Dense(1)(lstm)

model = Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss='mse', metrics=['mae','mse'])

# Train the model
early_stop = EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(train_X, train_Y, epochs=100, validation_split=0.2, batch_size=32, callbacks
    =[early_stop])

# Plot MAE over epochs
mae_history = history.history['mae']
val_mae_history = history.history['val_mae']
plt.plot(range(1, len(mae_history)+1), mae_history, label='Training MAE')
plt.plot(range(1, len(val_mae_history)+1), val_mae_history, label='Validation MAE')
plt.title('Training and Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.savefig('maeplot.pdf')
```

```
plt.show()

# Plot MSE over epochs
mse_history = history.history['loss']
val_mse_history = history.history['val_loss']
plt.plot(range(1, len(mse_history)+1), mse_history, label='Training MSE')
plt.plot(range(1, len(val_mse_history)+1), val_mse_history, label='Validation MSE')
plt.title('Training and Validation MSE')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()
plt.savefig('mseplot.pdf')
plt.show()
```



MSE (ب)



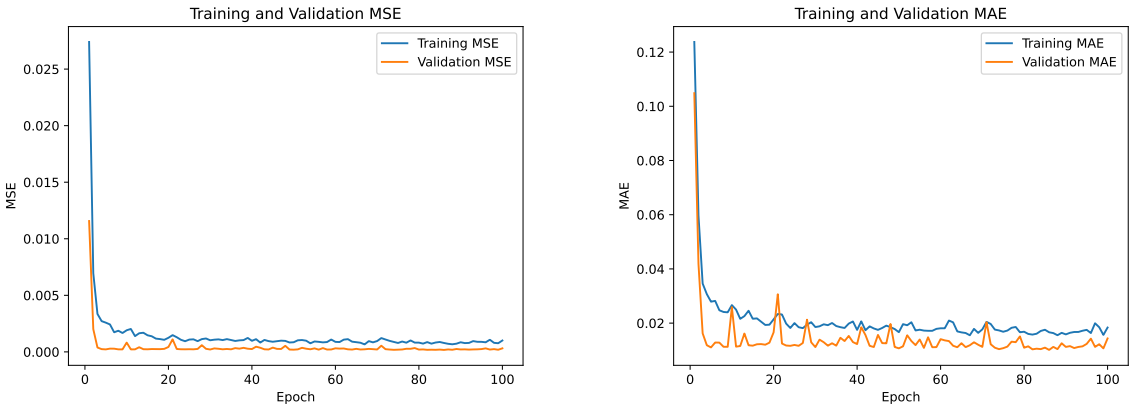
MAE (آ)

شکل ۱۰: نتایج نمودارهای MAE و MSE در حالت استفاده از EarlyStopping (لایت‌کوین - مدل ترکیبی).

به‌عنوان راه‌حل دوم مطابق آموزش قرارداده‌شده توسط گروه محترم تدریسیاری پیش می‌رویم.

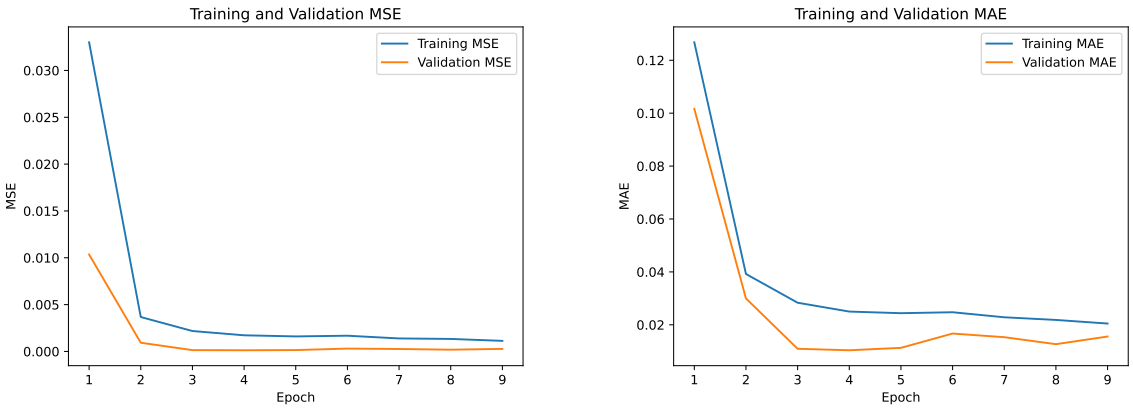
۴.۱ پاسخ قسمت ۴ - ارزیابی و تحلیل نتایج

برای ارزیابی روش پیشنهادی از خطای میانگین مطلق (MAE)، خطای میانگین مربعات (MSE)، خطای میانگین مطلق درصدی (MAPE) و خطای میانگین مربعات مطلق جذر (RMSE) استفاده می‌شود. روابط به صورتی است که در رابطهٔ ۲



MAE (آ) MSE (ب)

شکل ۱۱: نتایج نمودارهای MAE و MSE در حالت بدون استفاده از EarlyStopping (لایت‌کوین – مدل ترکیبی).

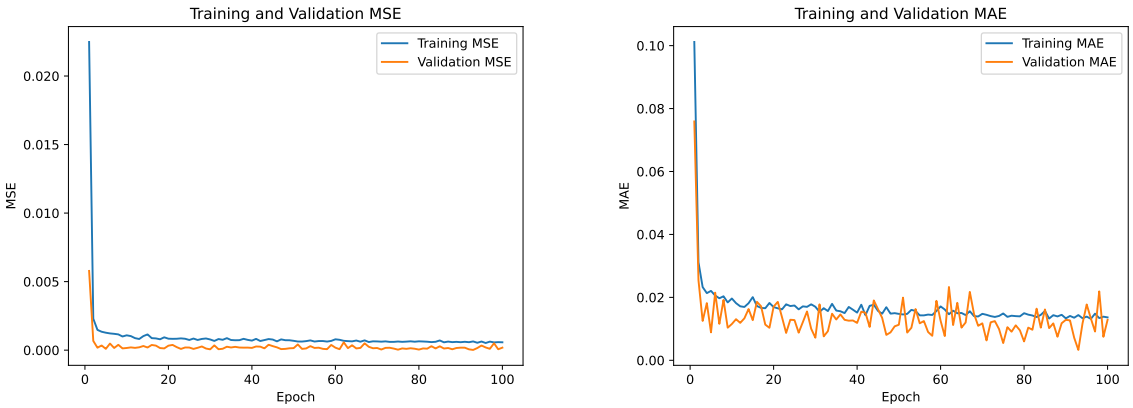


MAE (آ) MSE (ب)

شکل ۱۲: نتایج نمودارهای MAE و MSE در حالت استفاده از EarlyStopping (مونرو – مدل ترکیبی).

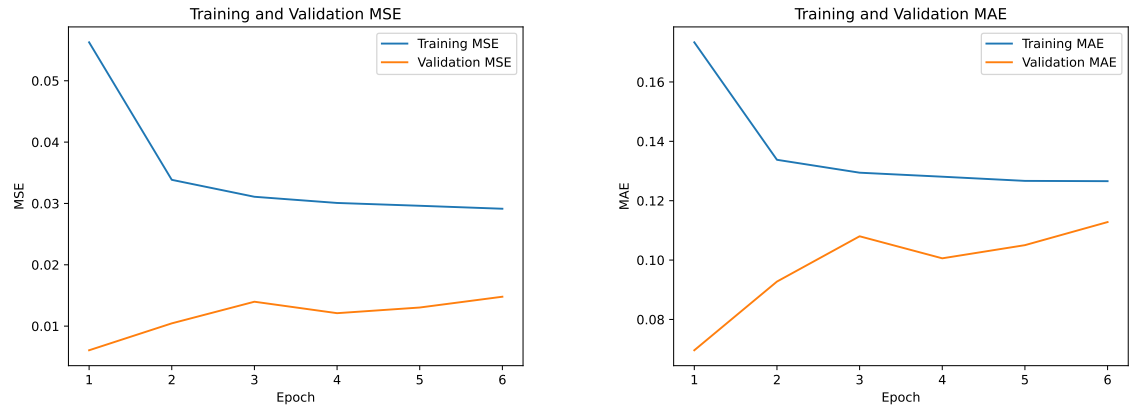
آورده شده که در در آن، $\hat{p}_i$ نشان‌دهندهٔ قیمت پیش‌بینی‌شده، p_i نشان‌دهندهٔ قیمت واقعی و N تعداد کل مشاهدات است.

$$
\begin{aligned}
\text{MSE} &= \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - p_i)^2 \\
\text{RMSE} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - p_i)^2} \\
\text{MAE} &= \frac{1}{N}\sum_{i=1}^{N}|\hat{p}_i - p_i| \\
\text{MAPE} &= \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{p}_i - p_i}{p_i}\right|
\end{aligned}
\qquad (۲)
$$



MSE (ب) MAE (آ)

شکل ۱۳: نتایج نمودارهای MAE و MSE در حالت بدون استفاده از EarlyStopping (مونرو – مدل ترکیبی).

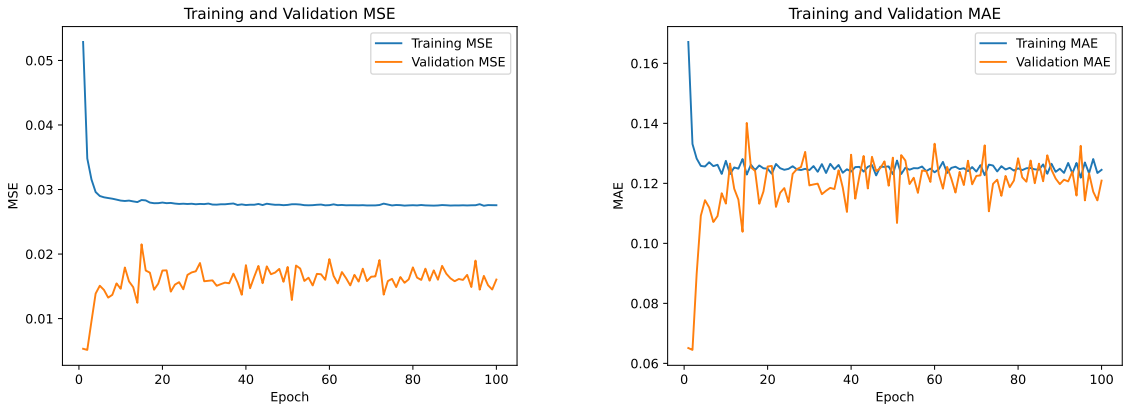


MSE (ب) MAE (آ)

شکل ۱۴: نتایج نمودارهای MAE و MSE در حالت استفاده از EarlyStopping (لایت‌کوین – مدل LSTM ساده).

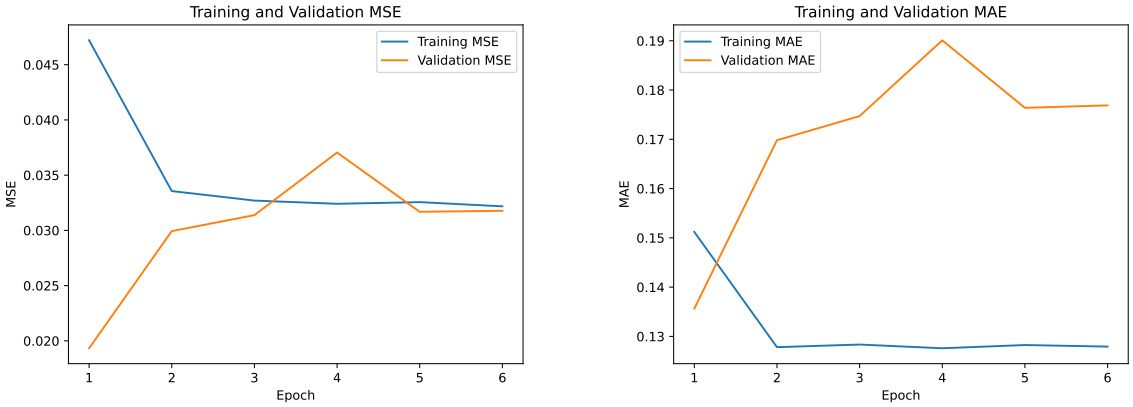
پیش‌بینی قیمت برای طول‌های پنجره‌ای مختلف انجام شده است: ۱ روز، ۳ روز و ۷ روز. برای ارزیابی کارایی واقعی روش پیشنهادی، این پیش‌بینی‌ها بر اساس داده‌های نمونه خارج از نمونه‌ای محاسبه شده‌اند که برای آموزش مدل استفاده نشده است.

برای انجام سناریوهای مختلف ارزیابی ابتدا از دستورات زیر استفاده می‌کنیم. در این قطعه کد، ابتدا داده‌ها با استفاده از `MinMaxScaler` به مقیاس $(0, 1)$ مقیاس می‌شوند. در ادامه، یک تابع به نام `fun` تعریف شده است. این تابع، دو ورودی دارد: داده و مقدار لاگ. خروجی این تابع دو آرایه با نام‌های x و y است. x یک آرایه با ابعاد (k, lag, n) است که در آن k تعداد نمونه‌ها و n تعداد ویژگی‌ها است. هر نمونه در x، یک دنباله از lag عدد مشاهدات پشت سر هم است. هم‌چنین y یک آرایه با ابعاد $(k, 1)$ است که در آن k تعداد نمونه‌ها است. هر عضو از y، مشاهدهٔ بعدی پس از دنبالهٔ lagعدد مشاهدات مربوط به آن درx است. برای پیش‌بینی مقادیر آینده، یک پنجرهٔ پیش‌بینی با طول `prediction_window` تعریف شده است. هم‌چنین با استفاده از پارامتر lag، تعداد مشاهدات قبلی که برای پیش‌بینی استفاده می‌شوند، تعیین می‌شود. سپس داده‌ها به دو قسمت آموزش و ارزیابی تقسیم می‌شوند. داده‌های آموزشی برای آموزش مدل استفاده می‌شوند. و داده‌های ارزیابی نیز برای ارزیابی مدل استفاده



MAE (آ) MSE (ب)

شکل ۱۵: نتایج نمودارهای MAE و MSE در حالت بدون استفاده از EarlyStopping (لایت‌کوین – مدل LSTM ساده).

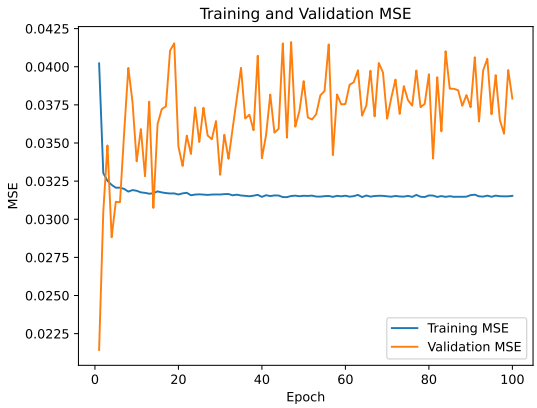


MAE (آ) MSE (ب)

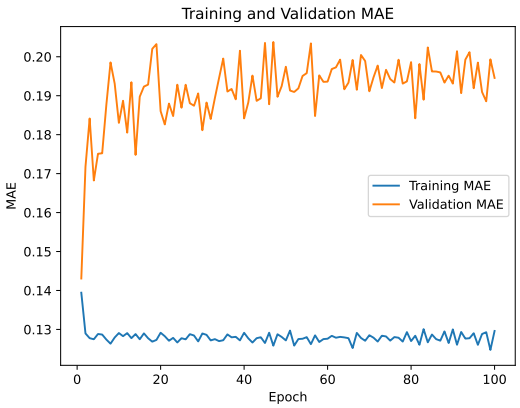
شکل ۱۶: نتایج نمودارهای MAE و MSE در حالت استفاده از EarlyStopping (مونرو – مدل LSTM ساده).

می‌شوند. دستورات به شرح زیر است:

```
# Scale the data using MinMaxScaler with range (0,1)
scaler = MinMaxScaler(feature_range=(0,1))
df = pd.DataFrame(scaler.fit_transform(df),columns=df.columns)
df.head()

def fun(data, lag):
    """
    A function that takes in data and a lag value and returns two arrays: x and y.
    x is an array of shape (k, lag, n), where k is the number of samples and n is the number of features.
    y is an array of shape (k, 1), where k is the number of samples.
```



MSE (ب)



MAE (آ)

شکل ۱۷: نتایج نمودارهای MAE و MSE در حالت بدون استفاده از EarlyStopping (مونرو – مدل LSTM ساده).

```
    Each sample in x is a sequence of lag consecutive observations from data, and the
    corresponding value in y is the next observation.
    """
    k = len(data) - lag
    x = []
    y = []
    for i in range(k):
        x.append(data[i:i+lag])
        y.append(data[i+lag,:1])
    x = np.array(x)
    y = np.array(y)
    return x, y

# Reverse the order of the data array
data = np.array(df.iloc[::-1])

# Set the number of data points to predict
prediction_window = 1

# Set the number of previous data points to use for prediction
lag = 30

# Split the data into train and test sets
test_start_index = len(data) - prediction_window - lag
test = data[test_start_index:]
train = data[:test_start_index]

# Use the function fun to create x_train, y_train, x_test, and y_test arrays
x_train, y_train = fun(train, lag)
```

```
x_test, y_test = fun(test, lag)
```

در ادامه، از آن‌جا که مهم است مدل روی داده‌های درست سناریوهای مختلف آموزش ببیند و ارزیابی شود از دستورات زیر برای بازسازی و آموزش مدل روی داده‌های سناریوهای مختلف استفاده می‌کنیم. برای این منظور با استفاده از دستورات زیر مدل را در صورت عادی و ترکیبی تشکیل می‌دهیم و برخی پارامترها هم‌چون توقف زودهنگام را هم برای آن در نظر می‌گیریم. درنهایت هم مثل گذشته رسم نمودار را داریم.

```
from tensorflow.keras.layers import GRU, LSTM, Dropout, Dense, Input, concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

# Define the loss function
loss_fn = 'mse'

# Define the model
inputs = Input(shape=(30,1))
gru = GRU(30, activation='relu', return_sequences=False)(inputs)
gru = Dropout(0.1)(gru)
gru = Dense(1, activation='relu')(gru)
lstm = LSTM(30, activation='relu', return_sequences=True)(inputs)
lstm = Dropout(0.1)(lstm)
lstm = LSTM(50, activation='relu')(lstm)
lstm = Dense(1, activation='relu')(lstm)
combined = concatenate([gru, lstm])
outputs = Dense(1)(combined)
model = Model(inputs=inputs, outputs=outputs)

# # LSTM Network
# lstm = LSTM(50, activation='relu')(inputs)
# outputs = Dense(1)(lstm)

# Compile the model with Adam optimizer and defined loss function
model.compile(optimizer=Adam(learning_rate=0.001), loss=loss_fn, metrics=['mae','mse'])

# Fit the model with train data and validate with test data for 100 epochs
callback=EarlyStopping(min_delta=1e-5,patience=26)
history = model.fit(x_train, y_train, epochs=100, callbacks=[callback], validation_data=(x_test,
    y_test))

```

```
# Plot the MAE and MSE graphs
plt.plot(history.history['mae'], label='train')
plt.plot(history.history['val_mae'], label='test')
plt.title('Model MAE')
plt.ylabel('MAE')
plt.xlabel('Epoch')
plt.legend()
plt.savefig('lossmaeM.pdf')
plt.show()

plt.plot(history.history['mse'], label='train')
plt.plot(history.history['val_mse'], label='test')
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend()
plt.savefig('lossmseM.pdf')
plt.show()
```

درنهایت دستوراتی را برای محاسبهٔ شاخص‌های موردنیاز و نمایش بصری مقادیر پیش‌بینی‌شده و واقعی می‌نویسیم. ابتدا بحث مقیاس‌بندی داده‌ها را در نظر می‌گیریم تا مقادیر غیرمقیاس‌شده را بتوانیم به خوبی نشان دهیم. سپس توابعی و دستوراتی را برای به دست آوردن خطاها و پارامترهای موردنیاز می‌نویسیم که بر اساس مقدار واقعی و مقدار پیش‌بینی‌شده کار می‌کنند. درنهایت هم دستوراتی می‌نویسیم تا مقادیر (مقدار) پیش‌بینی‌شده و مقادیر (مقدار) واقعی را در دو حالت مقیاس‌داده‌شده و بدون مقیاس در قالب نمودار نشان داده و آن را ذخیره کند. دستورات این بخش به شرح زیر است و با انجام پیاده‌سازی نتایج مربوط به پیش‌بینی با پنجره‌های یک، سه و هفت‌روزه به‌ترتیب در جدول ۱، جدول ۲ و جدول ۳ و هم‌چنین از شکل ۱۸ تا شکل ۲۳ آورده شده است.

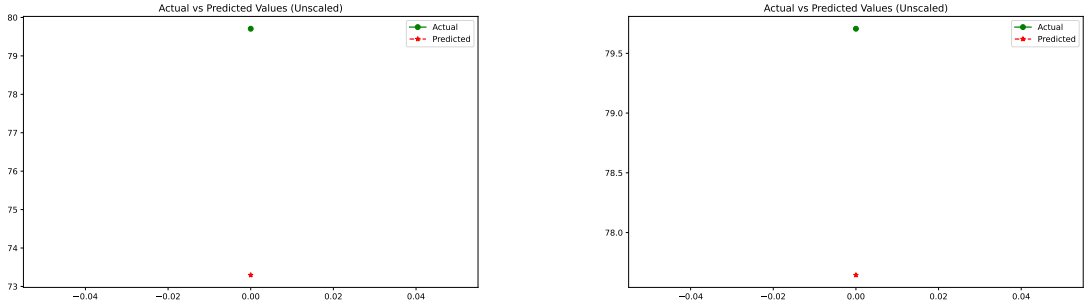
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Define and fit the scaler object on the training data
# scaler = MinMaxScaler()
# scaler.fit(x_train)

# Predict on the test data and unscale the predicted and test data
y_pred_scaled = model.predict(x_test)
y_pred_unscaled = scaler.inverse_transform(y_pred_scaled[::-1])
y_test_unscaled = scaler.inverse_transform(y_test)

# Print the evaluation metrics for the unscaled predictions
MAE_unscaled = mean_absolute_error(y_test_unscaled, y_pred_unscaled)
RMSE_unscaled = np.sqrt(mean_squared_error(y_test_unscaled, y_pred_unscaled))
R2_unscaled = r2_score(y_test_unscaled, y_pred_unscaled)
```

```
MAPE_unscaled = np.mean(np.abs((y_test_unscaled - y_pred_unscaled) / y_test_unscaled)) * 100
MSE_unscaled = mean_squared_error(y_test_unscaled, y_pred_unscaled)

print('Unscaled Metrics')
print('MSE : ', MSE_unscaled)
print('RMSE : ', RMSE_unscaled)
print('MAE : ', MAE_unscaled)
print('MAPE : ', MAPE_unscaled)
print('R2 : ', R2_unscaled)

# Print the evaluation metrics for the scaled predictions
y_pred_scaled = y_pred_scaled.flatten()[::-1]
y_test_scaled = y_test.flatten()
MAE_scaled = mean_absolute_error(y_test_scaled, y_pred_scaled)
RMSE_scaled = np.sqrt(mean_squared_error(y_test_scaled, y_pred_scaled))
R2_scaled = r2_score(y_test_scaled, y_pred_scaled)
MAPE_scaled = np.mean(np.abs((y_test_scaled - y_pred_scaled) / y_test_scaled)) * 100
MSE_scaled = mean_squared_error(y_test_scaled, y_pred_scaled)

print('Scaled Metrics')
print('MSE : ', MSE_scaled)
print('RMSE : ', RMSE_scaled)
print('MAE : ', MAE_scaled)
print('MAPE : ', MAPE_scaled)
print('R2 : ', R2_scaled)

print('y_test_unscaled:')
print(y_test_unscaled)

print('y_pred_unscaled:')
print(y_pred_unscaled)


# Plot the actual vs predicted values for the unscaled data
plt.figure(figsize=(10, 6))
plt.plot(y_test_unscaled, 'go-', label='Actual')
plt.plot(y_pred_unscaled, 'r*--', label='Predicted')
plt.title('Actual vs Predicted Values (Unscaled)')
plt.legend()
plt.savefig('ActualPredictedValuesUnscaledM.pdf')
plt.show()

# Plot the actual vs predicted values for the scaled data
plt.figure(figsize=(10, 6))
plt.plot(y_test_scaled, 'go-', label='Actual')
```

```
plt.plot(y_pred_scaled, 'r*--', label='Predicted')
plt.title('Actual vs Predicted Values (Scaled)')
plt.legend()
plt.savefig('ActualPredictedValuesScaledM.pdf')
plt.show()
```

جدول ۱: نتایج پیش‌بینی با پنجرهٔ یک‌روزه

Model	Currency	MSE	RMSE	MAE	MAPE
LSTM	Litecoin	41.0973	6.4107	6.4107	8.0429
	Monero	227.6432	15.0878	15.0878	17.6819
Proposed	Litecoin	4.2527	2.0622	2.0622	2.5872
	Monero	14.5054	3.8085	3.8085	4.4634

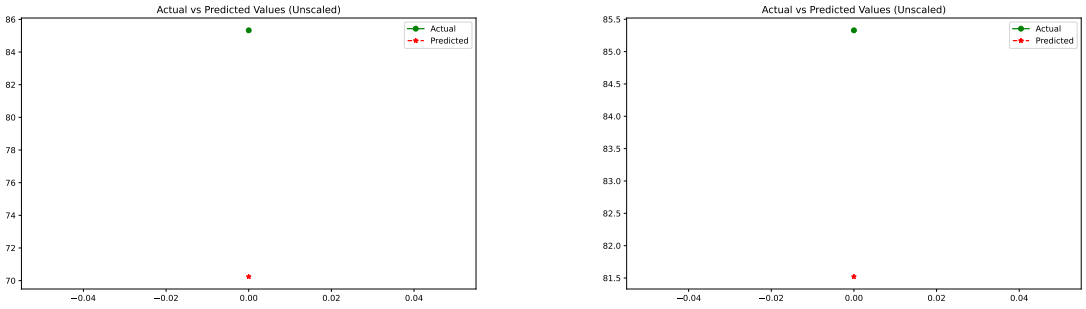


(ب) LSTM (آ) Hybrid

شکل ۱۸: نتایج پیش‌بینی با پنجرهٔ یک‌روزه (لایت‌کوین).

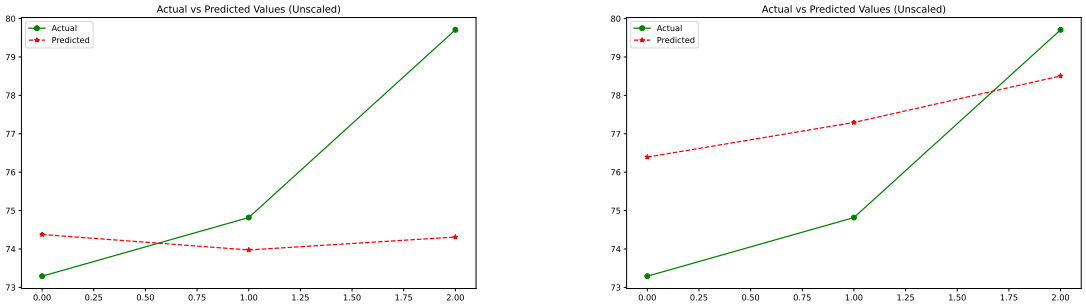
جدول ۲: نتایج پیش‌بینی با پنجرهٔ سه‌روزه

Model	Currency	MSE	RMSE	MAE	MAPE
LSTM	Litecoin	10.3325	3.2144	2.4414	3.12584
	Monero	21.544	4.6416	3.9112	4.9009
Proposed	Litecoin	5.7314	2.3940	2.25974	3.0162
	Monero	39.5641	6.2900	5.7367	6.9386



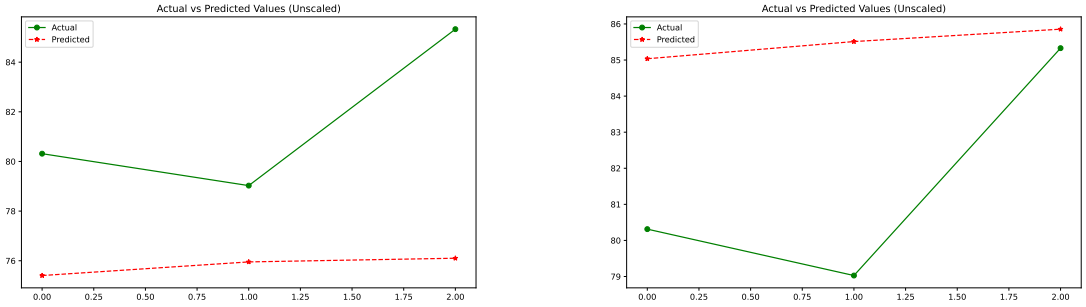
(آ) Hybrid (ب) LSTM

شکل ۱۹: نتایج پیش‌بینی با پنجرهٔ یک‌روزه (مونرو).



(آ) Hybrid (ب) LSTM

شکل ۲۰: نتایج پیش‌بینی با پنجرهٔ سه‌روزه (لایت‌کوین).

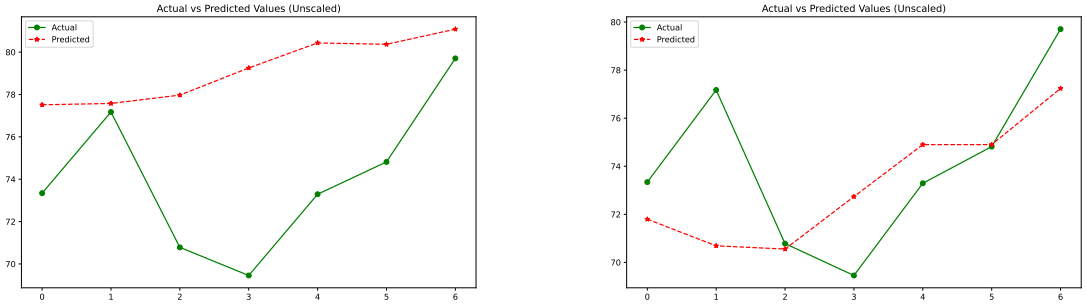


(آ) Hybrid (ب) LSTM

شکل ۲۱: نتایج پیش‌بینی با پنجرهٔ سه‌روزه (مونرو).

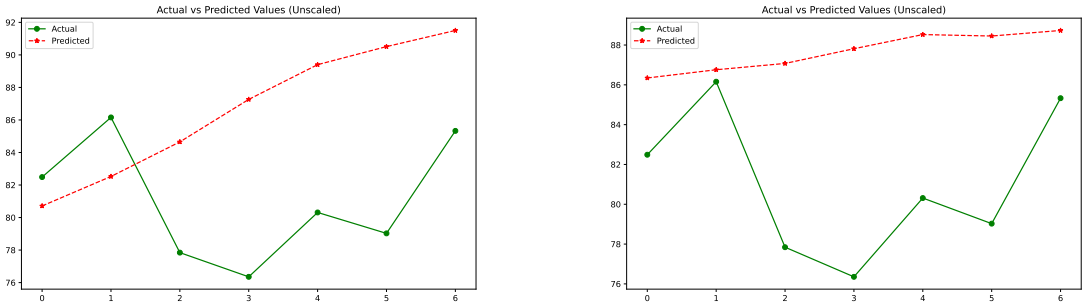
جدول ۳: نتایج پیش‌بینی با پنجرهٔ هفت‌روزه

Model	Currency	MSE	RMSE	MAE	MAPE
LSTM	Litecoin	35.5803	5.9649	5.0921	7.0546
	Monero	62.0430	7.8767	7.1245	8.9264
Proposed	Litecoin	9.1328	3.0220	2.2422	2.9934
	Monero	57.0888	7.5557	6.5983	8.3398



(ب) LSTM (آ) Hybrid

شکل ۲۲: نتایج پیش‌بینی با پنجرهٔ هفت‌روزه (لایت‌کوین).



(ب) LSTM (آ) Hybrid

شکل ۲۳: نتایج پیش‌بینی با پنجرهٔ هفت‌روزه (مونرو).