Hands-On

Hands-On ini digunakan pada kegiatan Microcredential Associate Data Scientist 2021

Tugas Mandiri Pertemuan 14

Pertemuan 14 (empatbelas) pada Microcredential Associate Data Scientist 2021 menyampaikan materi mengenai Membangun Model (RNN dan LSTM). silakan Anda kerjakan Latihan 1 s/d 5. Output yang anda lihat merupakan panduan yang dapat Anda ikuti dalam penulisan code :)

RNN

Jaringan saraf berulang atau recurrent neural network (RNN) adalah jenis arsitektur jaringan saraf tiruan yang pemrosesannya dipanggil berulang-ulang untuk memroses masukan yang biasanya adalah data sekuensial. RNN masuk dalam kategori deep learning karena data diproses melalui banyak lapis (layer). RNN telah mengalami kemajuan yang pesat dan telah merevolusi bidang-bidang seperti pemrosesan bahasa alami (NLP), pengenalan suara, sintesa musik, pemrosesan data finansial seri waktu, analisa deret DNA, analisa video, dan sebagainya.

RNN memroses input secara sekuensial, sampel per sampel. Dalam tiap pemrosesan, output yang dihasilkan tidak hanya merupakan fungsi dari sampel itu saja, tapi juga berdasarkan state internal yang merupakan hasil dari pemrosesan sampel-sampel sebelumnya (atau setelahnya, pada bidirectional RNN).

Berikut adalah ilustrasi bagaimana RNN bekerja. Misalnya kita membuat RNN untuk menerjemahkan bahasa Indonesia ke bahasa Inggris

Ilustrasi di atas kelihatan rumit, tapi sebenarnya cukup mudah dipahami.

- sumbu horizontal adalah waktu, direpresentasikan dengan simbol t. Dapat kita bayangkan pemrosesan berjalan dari kiri ke kanan. Selanjutnya kita sebut t adalah langkah waktu (time step).
- Keseluruhan input adalah kalimat, dalam hal ini:
  ```
  Budi pergi ke sekolah.
  ```
- Pemrosean input oleh RNN adalah kata demi kata. Input kata-kata ini disimbolkan dengan x1 ,x2 , … ,x5 , atau secara umum xt.
- Output adalah kalimat, dalam hal ini:
  ```
  Budi goes to school.
  ```
- RNN memberikan output kata demi kata, dan ini kita simbolkan dengan ŷ1, ŷ2, …, ŷ5 , atau secara umum ŷt .
- Dalam tiap pemrosesan, RNN akan menyimpan state internal yaitu st, yang diberikan dari satu langkah waktu ke langkah waktu berikutnya. Inilah "memori" dari RNN.

Dengan contoh di atas, kita bisa generalisasikan arsitektur RNN sebagai berikut:

Tambahan yang tidak terdapat di diagram sebelumnya adalah U, V, dan W, yang merupakan parameter-parameter yang dimiliki RNN. Kita akan bahas pemakaian parameter-parameter ini nanti.

Penting untuk dipahami bahwa walaupun ada empat kotak RNN di gambar di atas, empat kotak itu mencerminkan satu modul RNN yang sama (satu instans model dengan parameter-parameter U, V, dan W yang sama). Penggambaran di atas hanya agar aspek sekuensialnya lebih tergambar.

Alternatif representasinya adalah seperti ini, agar lebih jelas bahwa hanya ada satu modul RNN:

Inilah sebabnya kenapa arsitektur ini disebut RNN. Kata recurrent (berulang) dalam RNN timbul karena RNN melakukan perhitungan yang sama secara berulang-ulang atas input yang kita berikan.

Sering juga kedua ilustrasi di atas digabungkan jadi satu sbb:

Sesuai dengan gambar di atas, ilustrasi di sebelah kanan adalah penjabaran (unrolled) dari versi berulang di sebelah kiri.

Latihan (1)

Melakukan import library yang dibutuhkan

In [1]:

```python
# import library pandas
import pandas as pd

# Import library numpy
import numpy as np

# Import library matplotlib untuk visualisasi
import matplotlib.pyplot as plt

# import library for build model
from keras.layers import Dense, Dropout, SimpleRNN, LSTM
from keras.models import Sequential

# import library untuk data preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
```

Load Dataset

In [2]:

```python
#Panggil file (load file bernama Stock.csv) dan simpan dalam dataframe
dataset = "Stock.csv"
data = pd.read_csv(dataset)
```

In [3]:

```python
# tampilkan 5 baris data
data.head()
```

Out[3]:

	Date	Open	High	Low	Close	Volume	Name
0	2006-01-03	56.45	56.66	55.46	56.53	3716500	UTX
1	2006-01-04	56.80	56.80	55.84	56.19	3114500	UTX
2	2006-01-05	56.30	56.49	55.63	55.98	3118900	UTX
3	2006-01-06	56.45	56.67	56.10	56.16	2874300	UTX
4	2006-01-09	56.37	56.90	56.16	56.80	2467200	UTX

Review Data

In [4]:

```python
# Melihat Informasi lebih detail mengenai struktur DataFrame dapat dilihat menggunakan fungsi info()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3020 entries, 0 to 3019
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    3020 non-null   object 
 1   Open    3019 non-null   float64
 2   High    3020 non-null   float64
 3   Low     3020 non-null   float64
 4   Close   3020 non-null   float64
 5   Volume  3020 non-null   int64  
 6   Name    3020 non-null   object 
dtypes: float64(4), int64(1), object(2)
memory usage: 165.3+ KB
```

In [5]:

```python
# Kolom 'low' yang akan kita gunakan dalam membangun model
# Slice kolom 'low'

Low_data = data.iloc[:,3:4]
```

In [6]:

```python
# cek output low_data

Low_data
```

Out[6]:

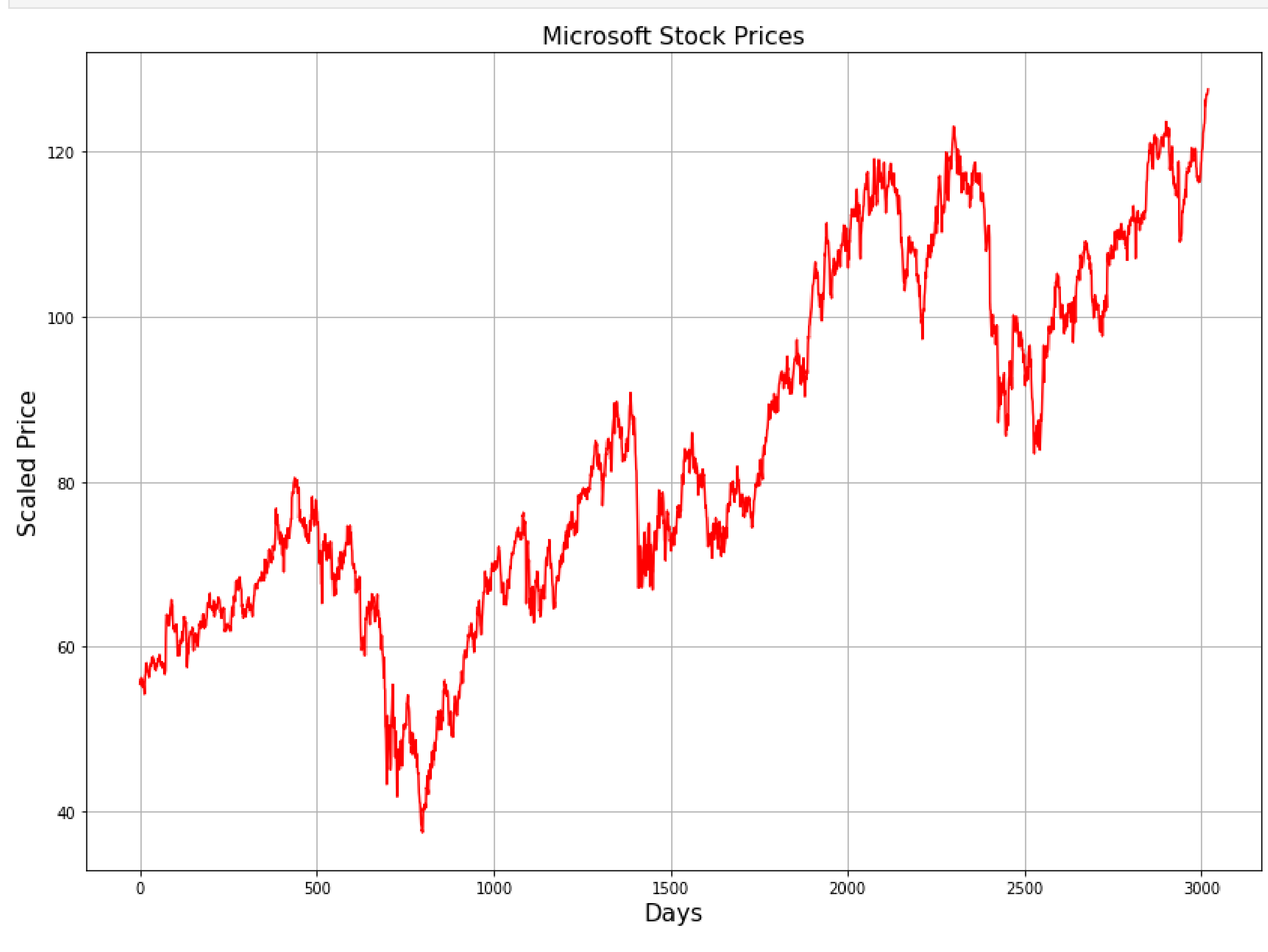
	Low
0	55.46
1	55.84
2	55.63
3	56.10
4	56.16
...	...
3015	126.95
3016	126.99
3017	126.92
3018	127.29
3019	127.57

3020 rows × 1 columns

In [7]:

```python
# Visualizing low_data

plt.figure(figsize=(14,10))
plt.plot(Low_data,c="red")
plt.title("Microsoft Stock Prices",fontsize=16)
plt.xlabel("Days",fontsize=16)
plt.ylabel("Scaled Price",fontsize=16)
plt.grid()
plt.show()
```



Latihan (2)

Data Preprocessing

In [8]:

```python
# Menskalakan data antara 1 dan 0 (scaling) pada low data
scaler = MinMaxScaler()
Low_scaled = scaler.fit_transform(Low_data)
```

In [9]:

```python
# definisikan variabel step dan train
step_size = 21

train_x = []
train_y = []
```

In [10]:

```python
# membuat fitur dan lists label
for i in range(step_size, 3019):
    train_x.append(Low_scaled[i-step_size:i, 0])
    train_y.append(Low_scaled[i, 0])
```

In [11]:

```python
# mengonversi list yang telah dibuat sebelumnya ke array
train_x = np.array(train_x)
train_y = np.array(train_y)
```

In [12]:

```python
# cek dimensi data dengan function .shape
print(train_x.shape)
```

```
(2998, 21)
```

In [13]:

```python
# 498 hari terakhir akan digunakan dalam pengujian
# 2500 hari pertama akan digunakan dalam pelatihan

test_x = train_x[2500:]
train_x = train_x[:2500]
test_y = train_y[2500:]
train_y = train_y[:2500]
```

In [14]:

```python
# reshape data untuk dimasukkan kedalam Keras model

train_x = np.reshape(train_x, (2500, step_size, 1))
test_x = np.reshape(test_x, (498, step_size, 1))
```

In [15]:

```python
# cek kembali dimensi data yang telah di reshape dengan function .shape

print(train_x.shape)
print(test_x.shape)
```

```
(2500, 21, 1)
(498, 21, 1)
```

Sekarang kita bisa mulai membuat model kita, dimulai dengan RNN

Latihan (3)

Build Model - RNN

In [16]:

```python
# buat varibel penampung model RNN
rnn_model = Sequential()
```

In [17]:

```python
# Output dari SimpleRNN akan menjadi bentuk tensor 2D (batch_size, 40) dengan Dropout sebesar 0.15

rnn_model.add(SimpleRNN(40, activation="tanh", return_sequences=True, input_shape=(train_x.shape[1], 1)))
rnn_model.add(Dropout(0.15))

rnn_model.add(SimpleRNN(40, activation="tanh", return_sequences=True))
rnn_model.add(Dropout(0.15))

rnn_model.add(SimpleRNN(40, activation="tanh", return_sequences=False))
rnn_model.add(Dropout(0.15))

# Add a Dense layer with 1 units.
rnn_model.add(Dense(1))
```

In [18]:

```python
# menambahkan loss function kedalam model RNN dengan tipe MSE
rnn_model.compile(optimizer="adam", loss="MSE")
```

In [19]:

```python
# fit the model RNN, dengan epoch 20 dan batch size 25
rnn_model.fit(train_x, train_y, epochs=20, batch_size=25)
```

```
Epoch 1/20
100/100 [==============================] - 3s 8ms/step - loss: 0.1872
Epoch 2/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0479
Epoch 3/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0245
Epoch 4/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0167
Epoch 5/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0126
Epoch 6/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0100
Epoch 7/20
100/100 [==============================] - 1s 9ms/step - loss: 0.0078
Epoch 8/20
100/100 [==============================] - 1s 9ms/step - loss: 0.0074
Epoch 9/20
100/100 [==============================] - 1s 7ms/step - loss: 0.0061
Epoch 10/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0052
Epoch 11/20
100/100 [==============================] - 1s 7ms/step - loss: 0.0049
Epoch 12/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0039
Epoch 13/20
100/100 [==============================] - 1s 7ms/step - loss: 0.0037
Epoch 14/20
100/100 [==============================] - 1s 7ms/step - loss: 0.0035
Epoch 15/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0033
Epoch 16/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0030
Epoch 17/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0028
Epoch 18/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0028
Epoch 19/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0024
Epoch 20/20
100/100 [==============================] - 1s 8ms/step - loss: 0.0023
```

Out[19]:

```
<keras.callbacks.History at 0x1ed65c235e0>
```

In [20]:

```python
# Prediksi Model RNN
rnn_predictions = rnn_model.predict(test_x)
rnn_score = r2_score(test_y, rnn_predictions)
```

In [21]:

```python
rnn_score
```

Out[21]:

```
0.9808178375534509
```

Latihan (4)

Build Model - LSTM

In [22]:

```python
# buat varibel penampung model LSTM
lstm_model = Sequential()
```

In [23]:

```python
# Add a LSTM layer with 40 internal units. dengan Dropout sebesar 0.15
lstm_model.add(LSTM(40, activation="tanh", return_sequences=True, input_shape=(train_x.shape[1], 1)))
lstm_model.add(Dropout(0.15))

lstm_model.add(LSTM(40, activation="tanh", return_sequences=True))
lstm_model.add(Dropout(0.15))

lstm_model.add(LSTM(40, activation="tanh", return_sequences=False))
lstm_model.add(Dropout(0.15))

# Add a Dense layer with 1 units.
lstm_model.add(Dense(1))
```

In [24]:

```python
# menambahkan loss function kedalam model lstm dengan tipe MSE
lstm_model.compile(optimizer="adam", loss="MSE")
```

In [25]:

```python
# fit lstm model, dengan epoch 20 dan batch size 25
lstm_model.fit(train_x, train_y, epochs=20, batch_size=25)
```

```
Epoch 1/20
100/100 [==============================] - 6s 14ms/step - loss: 0.0186
Epoch 2/20
100/100 [==============================] - 1s 13ms/step - loss: 0.0037
Epoch 3/20
100/100 [==============================] - 1s 15ms/step - loss: 0.0031
Epoch 4/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0028
Epoch 5/20
100/100 [==============================] - 2s 15ms/step - loss: 0.0031
Epoch 6/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0026
Epoch 7/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0027
Epoch 8/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0023
Epoch 9/20
100/100 [==============================] - 1s 15ms/step - loss: 0.0024
Epoch 10/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0021
Epoch 11/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0022
Epoch 12/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0019
Epoch 13/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0019
Epoch 14/20
100/100 [==============================] - 1s 14ms/step - loss: 0.0017
Epoch 15/20
100/100 [==============================] - 2s 16ms/step - loss: 0.0018
Epoch 16/20
100/100 [==============================] - 1s 15ms/step - loss: 0.0017
Epoch 17/20
100/100 [==============================] - 2s 15ms/step - loss: 0.0018
Epoch 18/20
100/100 [==============================] - 1s 13ms/step - loss: 0.0017
Epoch 19/20
100/100 [==============================] - 1s 13ms/step - loss: 0.0018
Epoch 20/20
100/100 [==============================] - 1s 13ms/step - loss: 0.0016
```

Out[25]:

```
<keras.callbacks.History at 0x1ed70832490>
```

In [26]:

```python
# Prediksi Model LSTM
lstm_predictions = lstm_model.predict(test_x)
lstm_score = r2_score(test_y, lstm_predictions)
```

In [27]:

```python
lstm_score
```

Out[27]:

```
0.9549785674878875
```

Latihan (5)

Evaluation

In [28]:

```python
# Cetak nilai prediksi masing-masing model dengan menggunakan r^2 square

print("R^2 Score of RNN", rnn_score)
print("R^2 Score of LSTM", lstm_score)
```

```
R^2 Score of RNN 0.9808178375534509
R^2 Score of LSTM 0.9549785674878875
```

Visualisasi Perbandingan Hasil Model prediksi dengan data original

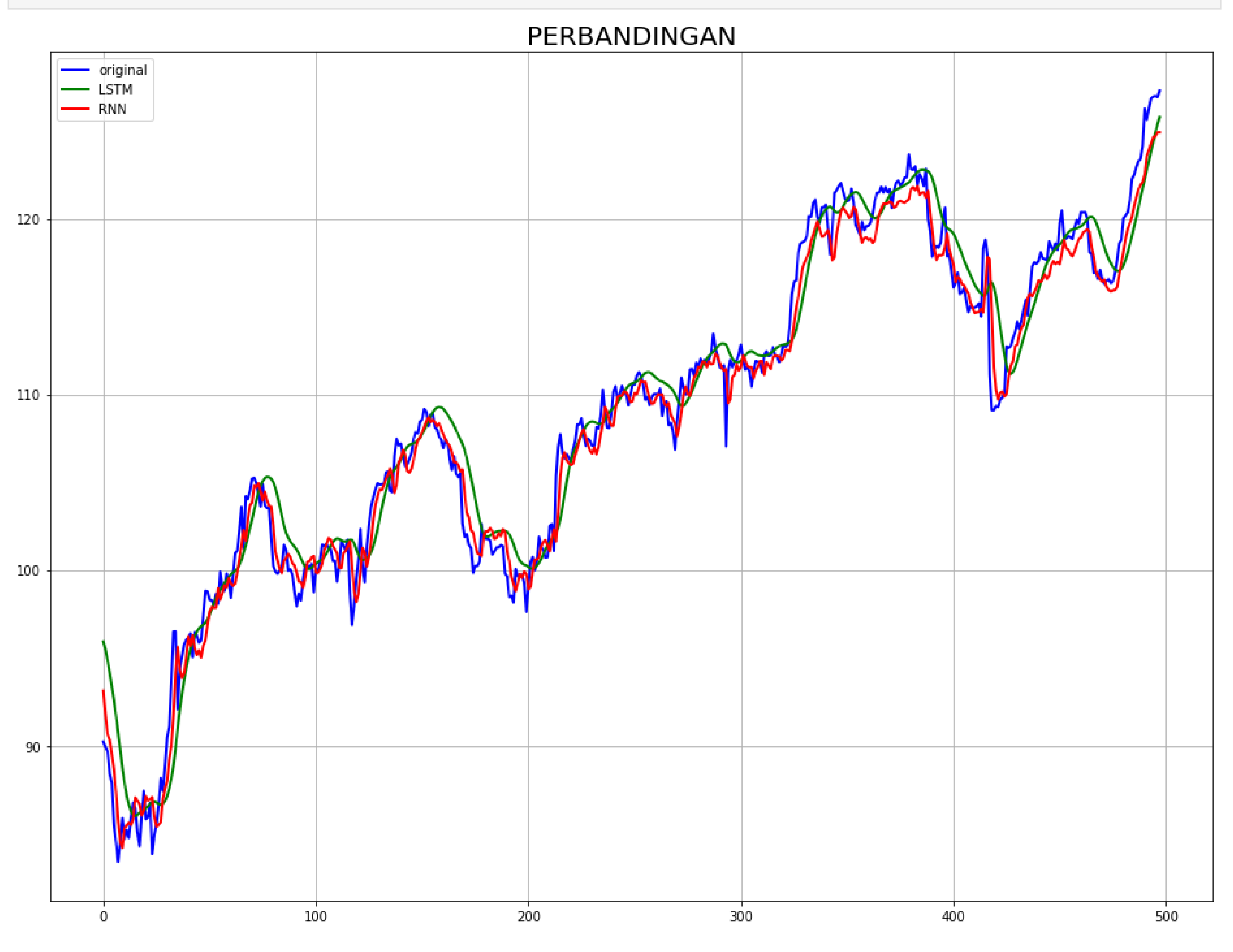
In [29]:

```python
lstm_predictions = scaler.inverse_transform(lstm_predictions)
rnn_predictions = scaler.inverse_transform(rnn_predictions)
test_y = scaler.inverse_transform(test_y.reshape(-1, 1))
```

In [30]:

```python
plt.figure(figsize=(16, 12))

plt.plot(test_y, c="blue", linewidth=2, label="original")
plt.plot(lstm_predictions, c="green", linewidth=2, label="LSTM")
plt.plot(rnn_predictions, c="red", linewidth=2, label="RNN")
plt.legend()
plt.title("PERBANDINGAN", fontsize=20)
plt.grid()
plt.show()
```



Berikan Kesimpulan Anda!

In [31]:

```python
# Kesimpulan yang dapat di peroleh adalah model RNN memiliki akurasi yang lebih tinggi dari LSTM
# dapat dilihat dari perbandingan hasil R2 Score, sehingga pada kasus ini model RNN lebih baik dari LSTM.
```