



Hands-On

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

Tugas Mandiri Pertemuan 14

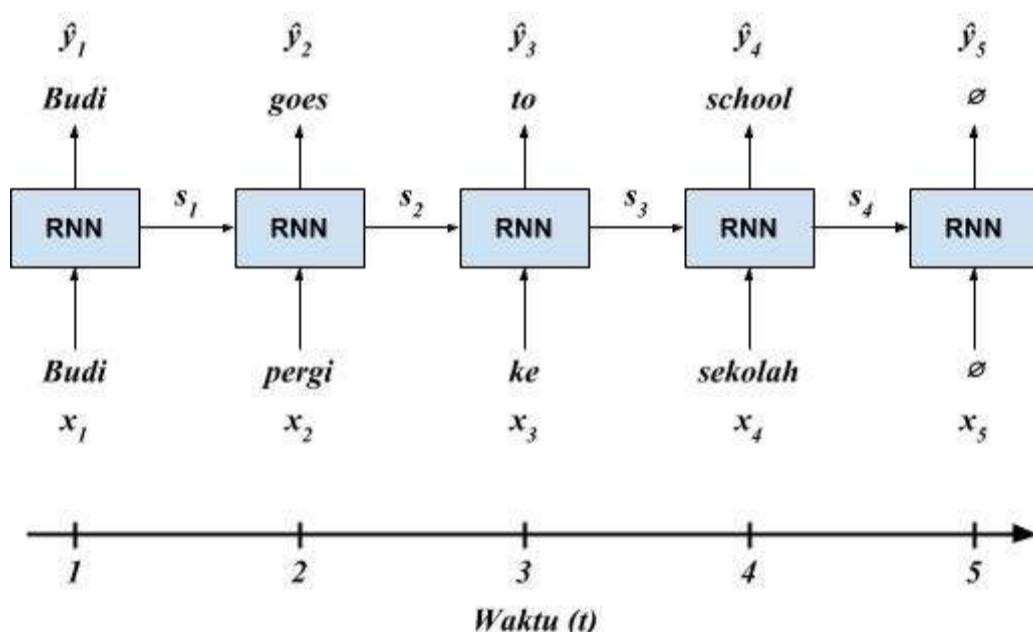
Pertemuan 14 (empatbelas) pada Microcredential Associate Data Scientist 2021 menyampaikan materi mengenai Membangun Model (RNN dan LSTM). silakan Anda kerjakan Latihan 1 s/d 5. Output yang anda lihat merupakan panduan yang dapat Anda ikuti dalam penulisan code :)

RNN

Jaringan saraf berulang atau recurrent neural network (RNN) adalah jenis arsitektur jaringan saraf tiruan yang pemrosesannya dipanggil berulang-ulang untuk memproses 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 memproses 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).

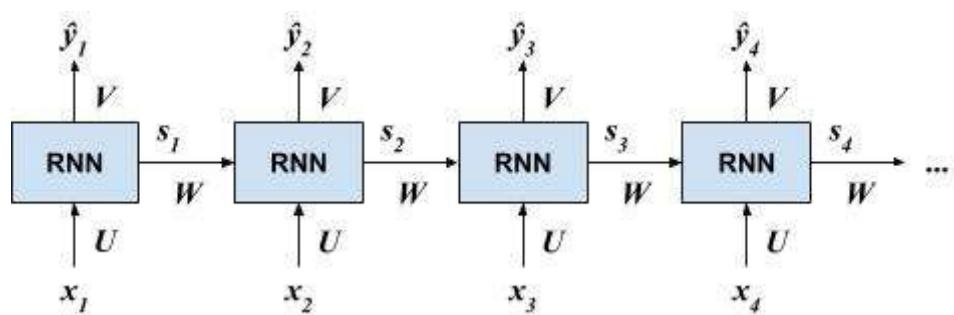
Berikut adalah ilustrasi bagaimana RNN bekerja. Misalkan kita membuat RNN untuk menerjemahkan bahasa Indonesia ke bahasa Inggris



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

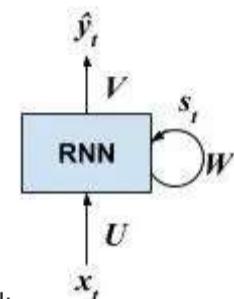
- sumbu horizontal adalah waktu, direpresentasikan dengan simbol t. Dapat kita bayangkan pemrosesan berjalan dari kiri ke kanan. Selanjutnya kita sebut t adalah langkah waktu (time step).
- Keseluruhan input adalah kalimat, dalam hal ini:
Budi pergi ke sekolah.
- Pemrosesan input oleh RNN adalah kata demi kata. Input kata-kata ini disimbolkan dengan x_1, x_2, \dots, x_5 , atau secara umum x_t .
- Output adalah kalimat, dalam hal ini:
Budi goes to school.
- RNN memberikan output kata demi kata, dan ini kita simbolkan dengan $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_5$, atau secara umum \hat{y}_t .
- Dalam tiap pemrosesan, RNN akan menyimpan state internal yaitu s_t , 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.

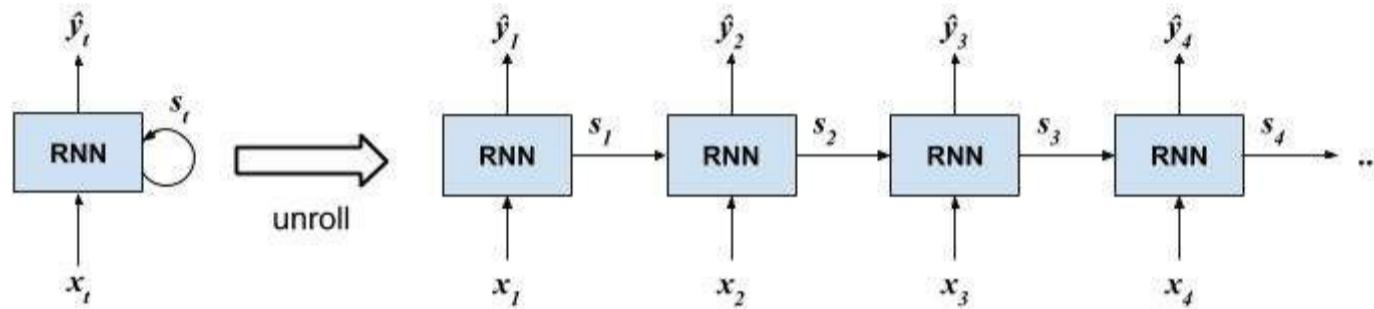
Penting untuk dipahami bahwa walaupun ada empat kotak RNN di gambar di atas, empat kotak itu mencerminkan satu modul RNN yang sama (satu instans model dengan parameter-parameter U , V , dan W yang sama). Penggambaran di atas hanya agar aspek sekvensialnya lebih tergambar.



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 [47]:

```
# 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 tensorflow.keras.layers import Dense, Dropout, SimpleRNN, LSTM
from tensorflow.keras.models import Sequential

# import Library untuk data preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

Load Dataset

In [2]:

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

In [3]:

```
# 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

	Date	Open	High	Low	Close	Volume	Name
4	2006-01-09	56.37	56.90	56.16	56.80	2467200	UTX

Review Data

In [4]:

```
# 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]:

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

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

In [6]:

```
# cek output Low_data
Low_data
```

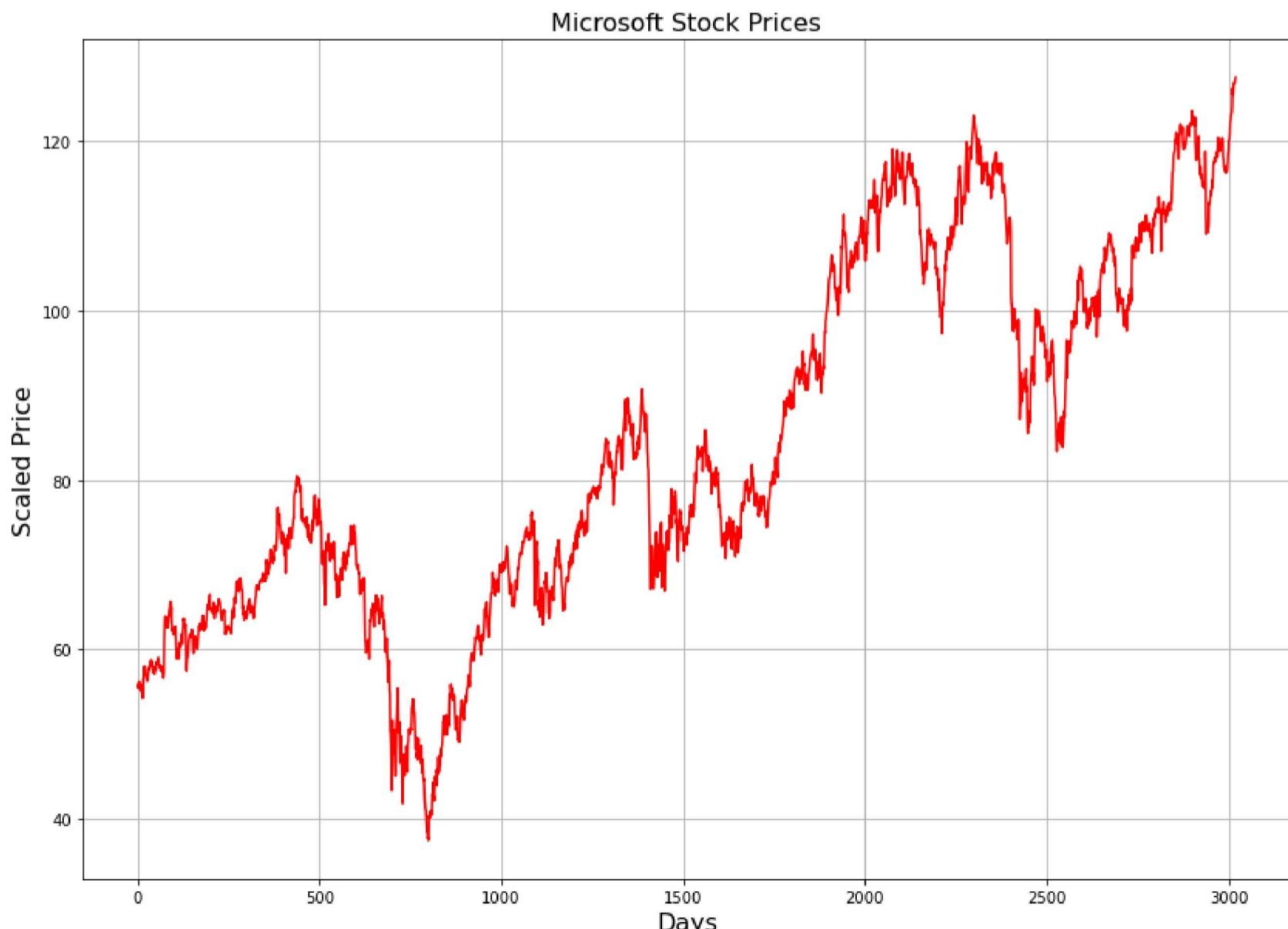
Out[6]:

```
array([[ 55.46],
       [ 55.84],
       [ 55.63],
       ...,
       [126.92],
       [127.29],
       [127.57]])
```

In [7]:

```
# 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]: # Menskalakan data antara 1 dan 0 (scaling) pada Low data
scaler = MinMaxScaler(feature_range=(0,1))
Low_data_scale = scaler.fit_transform(Low_data)

In [9]: Low_data_scale

Out[9]: array([[0.20028834],
   [0.20450261],
   [0.20217367],
   ...,
   [0.99279139],
   [0.99689475],
   [1.        ]])

In [36]: # definisikan variabel step dan train
step = 21
x_train = []
y_train = []

In [37]: # membuat fitur dan Lists Label
for i in range(step,3019):
    x_train.append(Low_data_scale[i-step:i,0])
    y_train.append(Low_data_scale[i,0])

In [38]: # mengonversi list yang telah dibuat sebelumnya ke array
x_train = np.array(x_train)
y_train = np.array(y_train)

In [39]: # cek dimensi data dengan function .shape
x_train.shape

Out[39]: (2998, 21)

In [40]: # 498 hari terakhir akan digunakan dalam pengujian
# 2500 hari pertama akan digunakan dalam pelatihan
x_test = x_train[2500:]
y_test = y_train[2500:]
x_train = x_train[:2500]
y_train = y_train[:2500]

In [42]: # reshape data untuk dimasukkan kedalam Keras model
x_train = np.reshape(x_train, (2500, step, 1))
x_test = np.reshape(x_test, (498, step, 1))

In [43]: # cek kembali dimensi data yang telah di reshape dengan function .shape
print(x_train.shape)
print(x_test.shape)

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

Sekarang kita bisa mulai membuat model kita, dimulai dengan RNN

Latihan (3)

Build Model - RNN

```
In [55]: # buat variabel penampung model RNN
rnn_model = Sequential()

In [56]: # 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=(x_train.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 [57]:

```
# menambahkan Loss function kedalam model RNN dengan tipe MSE

rnn_model.compile(optimizer="adam",loss="MSE")
```

In [58]:

```
# fit the model RNN, dengan epoch 20 dan batch size 25

rnn_model.fit(x_train,y_train,epochs=20,batch_size=25)
```

Train on 2500 samples
Epoch 1/20
2500/2500 [=====] - 3s 1ms/sample - loss: 0.1861
Epoch 2/20
2500/2500 [=====] - 1s 399us/sample - loss: 0.0595
Epoch 3/20
2500/2500 [=====] - 1s 404us/sample - loss: 0.0349
Epoch 4/20
2500/2500 [=====] - 1s 400us/sample - loss: 0.0231
Epoch 5/20
2500/2500 [=====] - 1s 402us/sample - loss: 0.0176
Epoch 6/20
2500/2500 [=====] - 1s 403us/sample - loss: 0.0139
Epoch 7/20
2500/2500 [=====] - 1s 405us/sample - loss: 0.0113
Epoch 8/20
2500/2500 [=====] - 1s 410us/sample - loss: 0.0104
Epoch 9/20
2500/2500 [=====] - 1s 406us/sample - loss: 0.0084
Epoch 10/20
2500/2500 [=====] - 1s 407us/sample - loss: 0.0081
Epoch 11/20
2500/2500 [=====] - 1s 408us/sample - loss: 0.0071
Epoch 12/20
2500/2500 [=====] - 1s 405us/sample - loss: 0.0067
Epoch 13/20
2500/2500 [=====] - 1s 406us/sample - loss: 0.0062
Epoch 14/20
2500/2500 [=====] - 1s 408us/sample - loss: 0.0058
Epoch 15/20
2500/2500 [=====] - 1s 408us/sample - loss: 0.0051
Epoch 16/20
2500/2500 [=====] - 1s 408us/sample - loss: 0.0049
Epoch 17/20
2500/2500 [=====] - 1s 410us/sample - loss: 0.0041
Epoch 18/20
2500/2500 [=====] - 1s 409us/sample - loss: 0.0042
Epoch 19/20
2500/2500 [=====] - 1s 405us/sample - loss: 0.0043
Epoch 20/20
2500/2500 [=====] - 1s 406us/sample - loss: 0.0043

Out[58]:

```
<tensorflow.python.keras.callbacks.History at 0x203e7421208>
```

In [69]:

```
# Prediksi Model RNN
rnn_pred = rnn_model.predict(x_test)

rnn_score = r2_score(y_test, rnn_pred)
```

In [70]:

```
rnn_score
```

Out[70]:

```
0.9480051863203837
```

Latihan (4)

Build Model - LSTM

In [61]:

```
# buat variabel penampung model LSTM
lstm_model = Sequential()
```

In [62]:

```
# 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=(x_train.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 [63]: # menambahkan loss function kedalam model Lstm dengan tipe MSE

lstm_model.compile(optimizer="adam", loss="MSE")
```

```
In [64]: # fit Lstm model, dengan epoch 20 dan batch size 25

lstm_model.fit(x_train, y_train, epochs=20, batch_size=25)
```

```
Train on 2500 samples
Epoch 1/20
2500/2500 [=====] - 6s 2ms/sample - loss: 0.0152
Epoch 2/20
2500/2500 [=====] - 2s 633us/sample - loss: 0.0032
Epoch 3/20
2500/2500 [=====] - 2s 632us/sample - loss: 0.0029
Epoch 4/20
2500/2500 [=====] - 2s 635us/sample - loss: 0.0025
Epoch 5/20
2500/2500 [=====] - 2s 638us/sample - loss: 0.0025
Epoch 6/20
2500/2500 [=====] - 2s 626us/sample - loss: 0.0025
Epoch 7/20
2500/2500 [=====] - 2s 640us/sample - loss: 0.0027
Epoch 8/20
2500/2500 [=====] - 2s 632us/sample - loss: 0.0025
Epoch 9/20
2500/2500 [=====] - 2s 631us/sample - loss: 0.0021
Epoch 10/20
2500/2500 [=====] - 2s 638us/sample - loss: 0.0021
Epoch 11/20
2500/2500 [=====] - 2s 677us/sample - loss: 0.0020
Epoch 12/20
2500/2500 [=====] - 2s 638us/sample - loss: 0.0019
Epoch 13/20
2500/2500 [=====] - 2s 659us/sample - loss: 0.0020
Epoch 14/20
2500/2500 [=====] - 2s 639us/sample - loss: 0.0019
Epoch 15/20
2500/2500 [=====] - 2s 633us/sample - loss: 0.0018
Epoch 16/20
2500/2500 [=====] - 2s 668us/sample - loss: 0.0016
Epoch 17/20
2500/2500 [=====] - 2s 672us/sample - loss: 0.0017
Epoch 18/20
2500/2500 [=====] - 2s 658us/sample - loss: 0.0016
Epoch 19/20
2500/2500 [=====] - 2s 648us/sample - loss: 0.0015
Epoch 20/20
2500/2500 [=====] - 2s 651us/sample - loss: 0.0016
<tensorflow.python.keras.callbacks.History at 0x203edb76390>
```

```
Out[64]:
```

```
In [71]: # Prediksi Model LSTM
lstm_pred = lstm_model.predict(x_test)

lstm_score = r2_score(y_test, lstm_pred)
```

```
In [72]: lstm_score
```

```
Out[72]: 0.9509356907431344
```

Latihan (5)

Evaluation

```
In [73]: # Cetak nilai prediksi masing-masing model dengan menggunakan r^2 square

print('R^2 Score dari model RNN', rnn_score)
print('R^2 Score dari model LSTM', lstm_score)
```

```
R^2 Score dari model RNN 0.9480051863203837
R^2 Score dari model LSTM 0.9509356907431344
```

Visualisasi Perbandingan Hasil Model prediksi dengan data original

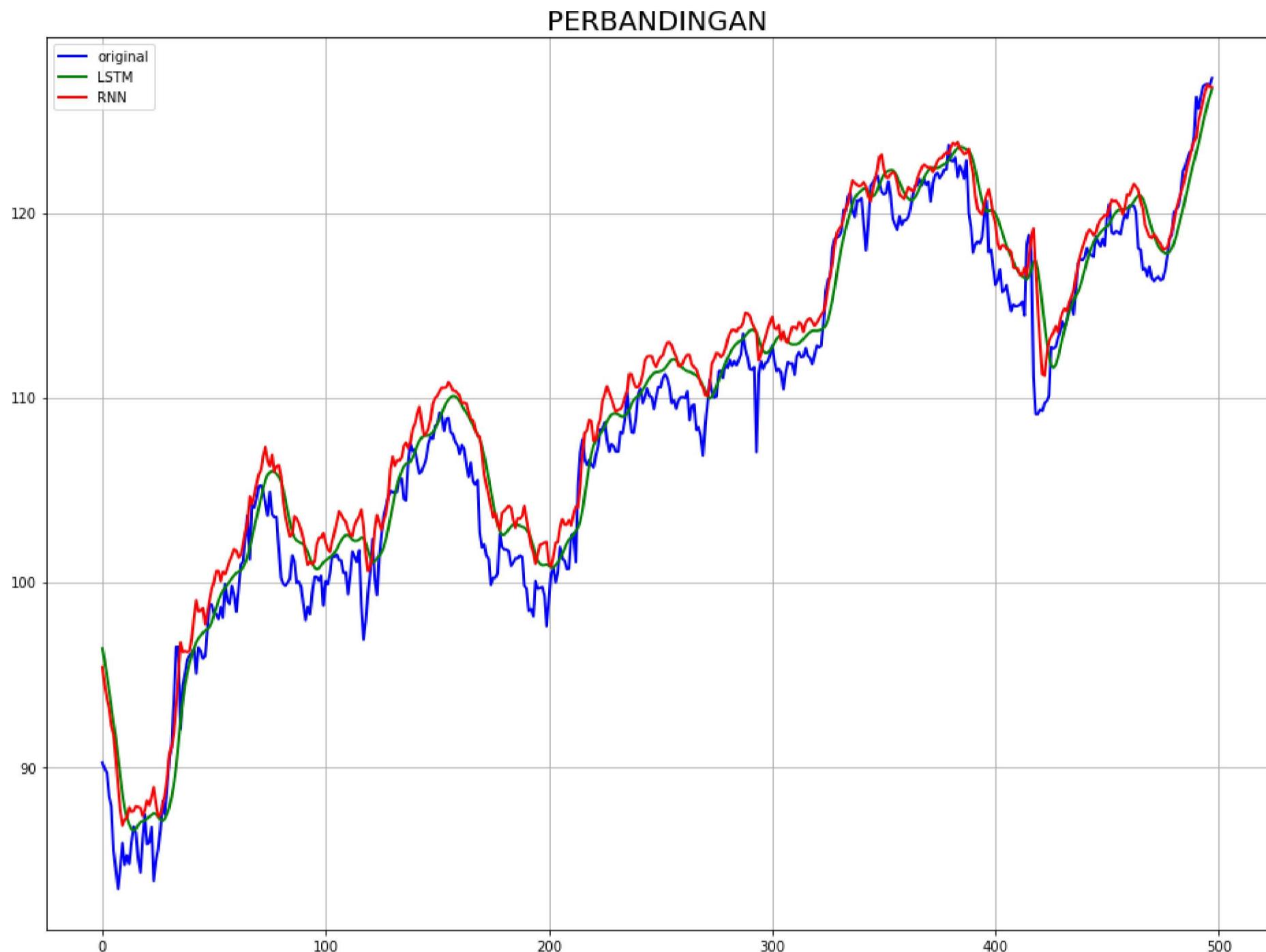
```
In [74]: lstm_pred = scaler.inverse_transform(lstm_pred)
```

```
rnn_pred = scaler.inverse_transform(rnn_pred)
y_test = scaler.inverse_transform(y_test.reshape(-1,1))
```

In [75]:

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

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



Berikan Kesimpulan Anda!

model RNN sederhana ini dapat banyak membantu untuk membuat prediksi yang lebih baik daripada model LSTM