Laporan Tugas Recurrent Neural Network dan Long-Short Term Memory

Amany Akhyar

Magister Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung
amanyakhyar@gmail.com

Abstrak. Dalam pengenalan pola, terdapat kasus dimana data yang ingin dicari polanya memiliki urutan yang perlu diperhatikan (sequential data). Beberapa jenis sequential data diantaranya adalah time series, urutan karakter di suatu kata atau kalimat, untaian DNA, dan lain-lain. Dalam mencari pola pada data tersebut, maka diperlukan algoritma khusus. Beberapa algoritma yang dapat digunakan untuk mencari pola pada sequential data adalah Recurrent Neural Network (RNN) dan Long-Short Term Memory (LSTM). Pada laporan ini akan dijelaskan tugas-tugas yang berhubungan dengan penerapan RNN dan LSTM terhadap sequential data.

1 Tujuan

Tujuan pembuatan tugas ini adalah :

1. Menerapkan tutorial RNN
2. Menerapkan RNN untuk menghasilkan nama dinosaurus baru
3. Menerapkan tutorial LSTM untuk prediksi pasar modal
4. Menerapkan LSTM untuk prediksi pasar modal dengan dataset baru

2 Pendahuluan

Recurrent Neural Network (RNN) adalah algoritma yang sangat efektif untuk Natural Language Processing (NLP) dan tugas lain yang mementingkan urutan (sequence) karena mempunyai "memory". RNN dapat membaca suatu input pada suatu waktu (x(t)) dan "mengingat" informasi mengenai input tersebut ketika melalui fungsi aktivasi hidden layer dari satu waktu ke waktu berikutnya. Pada tugas ini, RNN akan dibuat langkah demi langkah dan digunakan untuk menghasilkan nama dinosaurus baru. Long-Short Term Memory (LSTM) adalah algoritma yang sangat powerful untuk masalah prediksi sequence. Hal ini karena LSTM dapat menyimpan past information. Past information (informasi sebelumnya) sangat penting untuk kasus pasar modal, karena harga modal sebelumnya sangat penting dalam memprediksi harga modal di masa depan.

3 Proses

Proses pelaksanaan tugas ini terbagi menjadi 4 tahap sebagai berikut.

3.1 Tutorial RNN

Pada tugas ini, Recurrent Neural Network (RNN) akan diimplementasikan dengan numpy. Struktur RNN yang akan diimplementasikan adalah sebagai berikut.

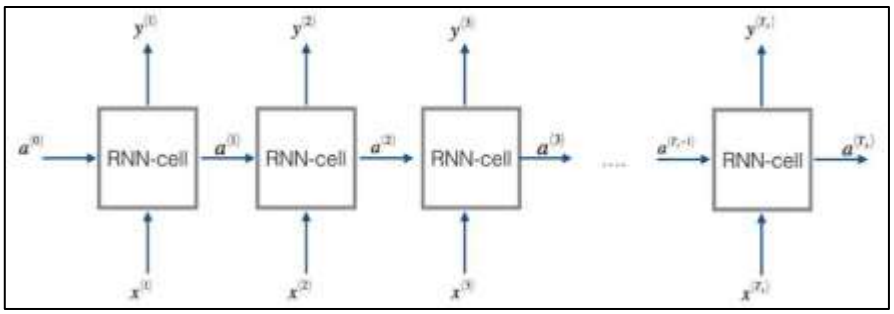

Fig. 1. Struktur Recurrent Neural Network

Pada **Fig. 1** diatas, proses yang terjadi di bagian dalam "RNN-cell" pada suatu waktu tertentu adalah sebagai berikut.

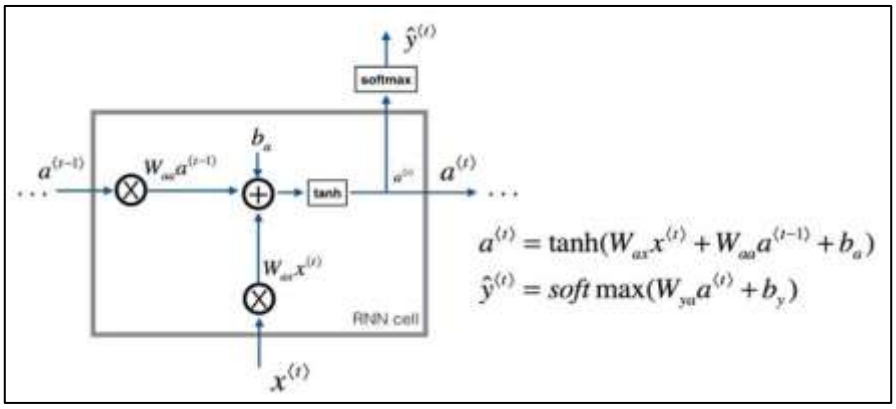

Fig. 2. Proses di dalam RNN-cell

Langkah-langkah yang dilakukan pada tutorial adalah sebagai berikut :

1. Import library numpy dan rnn_utils

```
import numpy as np
from rnn_utils import *
```

Fig. 3. Import library

2. Membuat fungsi **rnn_cell_forward**

   Fungsi rnn_cell_forward dibuat untuk mengimplementasikan tahap single forward RNN-cell. Berikut adalah fungsi rnn_cell_forward yang dibangun.

```
def rnn_cell_forward(xt, a_prev, parameters):

    Wax = parameters["Wax"]
    Waa = parameters["Waa"]
    Wya = parameters["Wya"]
    ba = parameters["ba"]
    by = parameters["by"]

    a_next = np.tanh(np.dot(Wax,xt)+np.dot(Waa,a_prev)+ba)
    yt_pred = softmax(np.dot(Wya,a_next)+by)

    cache = (a_next,a_prev,xt,parameters)

    return a_next, yt_pred, cache
```

Fig. 4. Fungsi rnn_cell_forward

Jika dibandingkan dengan isi RNN-cell (**Fig. 2**), maka dapat dilihat bahwa variable a_next merupakan implementasi dari rumus a(t) dan yt_pred merupakan implementasi dari rumus y(t).

Untuk menjalankan fungsi rnn_cell_forward maka jalankan script berikut.

```
np.random.seed(1)
xt = np.random.randn(3,10)
a_prev = np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Waa" : Waa, "Wax" : Wax, "Wya" : Wya, "ba" : ba, "by" : by}

a_next, yt_pred, cache = rnn_cell_forward(xt,a_prev,parameters)
print("a_next[4] = ", a_next[4])
print("a_next.shape = ", a_next.shape)
print("yt_pred[1] = ", yt_pred[1])
print("yt_pred.shape = ", yt_pred.shape)
```

Fig. 5. Menjalankan fungsi rnn_cell_forward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
a_next[4] =  [ 0.59584544  0.18141802  0.61311866  0.99808218  0.85016201  0.99980978
 -0.18887155  0.99815551  0.6531151   0.82872037]
a_next.shape =  (5, 10)
yt_pred[1] =  [0.9888161  0.01682021 0.21140899 0.36817467 0.98988387 0.88945212
 0.36920224 0.9966312  0.9982559  0.17746526]
yt_pred.shape =  (2, 10)
```

Fig. 6. Hasil fungsi rnn_cell_forward

3. Membuat fungsi **rnn_forward**

Fungsi rnn_forward dibuat untuk mengimplementasikan forward propagation algoritma RNN. Berikut adalah fungsi rnn_forward yang dibangun.

```
def rnn_forward(x, a0, parameters):
    caches=[]
    n_x, m, T_x = x.shape
    n_y, n_a = parameters["Wya"].shape

    a = np.zeros((n_a, m, T_x))
    y_pred = np.zeros((n_y, m, T_x))

    a_next = a0

    for t in range(0,T_x):
        a_next, yt_pred, cache = rnn_cell_forward(x[:,:,t],a_next,parameters)
        a[:,:,t] = a_next
        y_pred[:,:,t] = yt_pred
        caches.append(cache)

    caches = (caches,x)
    return a,y_pred,caches
```

Fig. 7. Fungsi rnn_forward

Pada tahap forward pass RNN, setiap cell mengambil input berupa hidden state dari cell sebelumnya (a(t-1)) dan input data pada current time-step (x(t)). Output dari setiap cell ialah berupa hidden state (a(t)) dan hasil prediksi pada current time-step (y(t)). Untuk ilustrasi proses ini dapat dilihat pada **Fig.1**.

Untuk menjalankan fungsi rnn_forward maka jalankan script berikut.

```
np.random.seed(1)
x =  np.random.randn(3,10,4)
a0 =  np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Waa" : Waa,"Wax" : Wax, "Wya" : Wya, "ba" : ba, "by" : by}

a, y_pred, caches = rnn_forward(x,a0,parameters)
print("a[4][1] = ",a[4][1])
print("a.shape = ",a.shape)
print("y_pred[1][3] = ",y_pred[1][3])
print("y_pred.shape = ",y_pred.shape)
print("caches[1][1][3] = ", caches[1][1][3])
print("len(caches) = ",len(caches))
```

Fig. 8. Menjalankan fungsi rnn_forward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
a[4][1] =  [-0.99999375  0.77911235 -0.99861469 -0.99833267]
a.shape =  (5, 10, 4)
y_pred[1][3] =  [0.79560373 0.86224861 0.11118257 0.81515947]
y_pred.shape =  (2, 10, 4)
caches[1][1][3] =  [-1.1425182  -0.34934272 -0.20889423  0.58662319]
len(caches) =  2
```

Fig. 9. Hasil fungsi rnn_forward

4. Membuat fungsi **lstm_cell_forward**

Fungsi lstm_cell_forward dibuat untuk mengimplementasikan tahap single forward LSTM-cell. Berikut adalah fungsi lstm_cell_forward yang dibangun.

```
def lstm_cell_forward(xt,a_prev,c_prev,parameters):
    Wf = parameters["Wf"]
    bf = parameters["bf"]
    Wi = parameters["Wi"]
    bi = parameters["bi"]
    Wc = parameters["Wc"]
    bc = parameters["bc"]
    Wo = parameters["Wo"]
    bo = parameters["bo"]
    Wy = parameters["Wy"]
    by = parameters["by"]

    n_x, m = xt.shape
    n_y, n_a = Wy.shape

    concat = np.zeros((n_x+n_a,m))
    concat[: n_a, :] = a_prev
    concat[n_a :, :] = xt

    ft = sigmoid(np.dot(Wf,concat)+bf)
    it = sigmoid(np.dot(Wi,concat)+bi)
    cct = np.tanh(np.dot(Wc,concat)+bc)
    c_next = c_prev*ft + it*cct
    ot = sigmoid(np.dot(Wo,concat)+bo)
    a_next = ot*np.tanh(c_next)
```

```
    yt_pred = softmax(np.dot(Wy,a_next)+by)

    cache = (a_next, c_next, a_prev, c_prev, ft, it, cct, ot, xt, parameters)

    return a_next,c_next,yt_pred,cache
```

Fig. 10. Fungsi lstm_cell_forward

Adapun ilustrasi LSTM-cell adalah sebagai berikut.

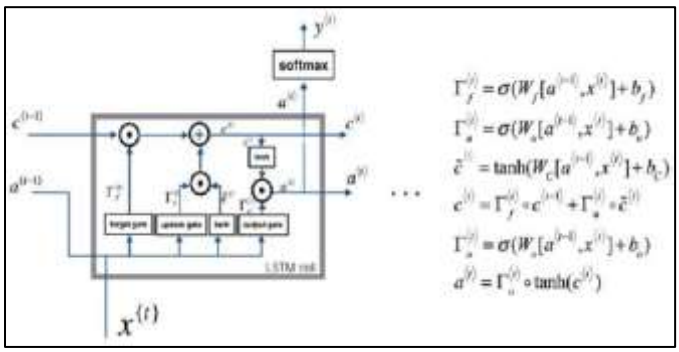

Fig. 11. LSTM-cell

Jıka fungsi lstm_cell_forward dibandingkan dengan **Fig.11**, maka terlihat bahwa variabel ft merupakan implementasi dari rumus $\Gamma_{f,}$ variabel it merupakan implementasi dari rumus $\Gamma_{u,}$ variabel cct merupakan implementasi dari rumus c̃, variabel c_next merupakan implementasi dari rumus c, variabel ot merupakan implementasi dari rumus Γ_{o}, dan variabel a_next merupakan implementasi dari rumus a.

Untuk menjalankan fungsi lstm_cell_forward maka jalankan script berikut.

```
np.random.seed(1)
xt = np.random.randn(3,10)
a_prev = np.random.randn(5,10)
c_prev = np.random.randn(5,10)
Wf = np.random.randn(5, 5+3)
bf = np.random.randn(5,1)
Wi = np.random.randn(5, 5+3)
bi = np.random.randn(5,1)
Wo = np.random.randn(5, 5+3)
bo = np.random.randn(5,1)
Wc = np.random.randn(5, 5+3)
bc = np.random.randn(5,1)
Wy = np.random.randn(2,5)
by = np.random.randn(2,1)

parameters = {"Wf": Wf, "Wi": Wi, "Wo": Wo, "Wc": Wc, "Wy": Wy, "bf": bf, "bi": bi, "bo": bo, "bc": bc, "by": by}

a_next, c_next, yt, cache = lstm_cell_forward(xt, a_prev, c_prev, parameters)

print("a_next[4] = ",a_next[4])
print("a_next.shape = ",a_next.shape)
print("c_next[2] = ",c_next[2])
print("c_next.shape = ",c_next.shape)
print("yt[1] = ",yt[1])
print("yt.shape = ",yt.shape)
print("cache[1][3] = ", cache[1][3])
print("len(cache) = ",len(cache))
```

Fig. 12. Menjalankan fungsi lstm_cell_forward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
a_next[4] =  [-0.66408471  0.0036921   0.02088357  0.22834167 -0.85575339  0.00138482
  0.76566531  0.34631421 -0.00215674  0.43827275]
a_next.shape =  (5, 10)
c_next[2] =  [ 0.63267805  1.00570849  0.35504474  0.20690913 -1.64566718  0.11832942
  0.76449811 -0.0981561  -0.74348425 -0.26810932]
c_next.shape =  (5, 10)
yt[1] =  [0.79913913 0.15986619 0.22412122 0.15606108 0.97057211 0.31146381
 0.00943007 0.12666353 0.39380172 0.07828381]
yt.shape =  (2, 10)
cache[1][3] =  [-0.16263996  1.03729328  0.72938082 -0.54101719  0.02752074 -0.30821874
  0.07651101 -1.03752894  1.41219977 -0.37647422]
len(cache) =  10
```

Fig. 13. Hasil fungsi lstm_cell_forward

5. Membuat fungsi **lstm_forward**

Fungsi lstm_forward dibuat untuk mengimplementasikan forward propagation algoritma RNN menggunakan LSTM-cell. Berikut adalah

fungsi lstm_forward yang dibangun.

```
def lstm_forward(x,a0,parameters):

    caches=[]

    n_x, m, T_x = x.shape
    n_y, n_a = parameters["Wy"].shape

    a = np.zeros((n_a, m, T_x))
    c = np.zeros((n_a, m, T_x))
    y = np.zeros((n_y, m, T_x))

    a_next = a0
    c_next = np.zeros((n_a,m))

    for t in range(0, T_x):
        a_next,c_next,yt,cache = lstm_cell_forward(x[:,:,t],a_next,c_next,parameters)

        a[:,:,t] = a_next

        y[:,:,t] = yt

        c[:,:,t] = c_next

        caches.append(cache)

    caches = (caches,x)

    return a,y,c,caches
```

Fig. 14. Fungsi lstm_forward

Fungsi lstm_forward menjalankan fungsi lstm_cell_forward berkali-kali dengan menggunakan iterasi for loop untuk mengimplementasikan struktur berikut ini.

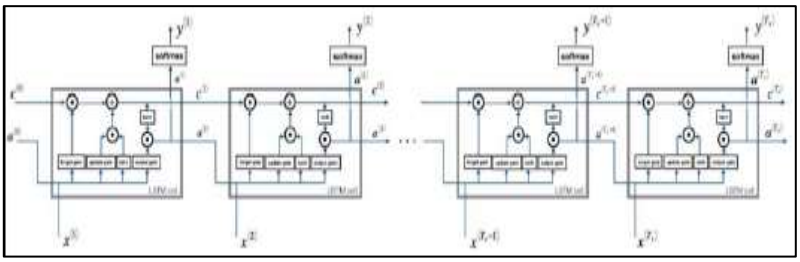

Fig. 15. Struktur LSTM forward pass untuk input sequence

Untuk menjalankan fungsi lstm_forward maka jalankan script berikut.

```
np.random.seed(1)
x = np.random.randn(3,10,7)
a0 =  np.random.randn(5,10)
Wf = np.random.randn(5, 5+3)
bf = np.random.randn(5,1)
Wi = np.random.randn(5, 5+3)
bi = np.random.randn(5,1)
Wo = np.random.randn(5, 5+3)
bo = np.random.randn(5,1)
Wc = np.random.randn(5, 5+3)
bc = np.random.randn(5,1)
Wy = np.random.randn(2,5)
by = np.random.randn(2,1)

parameters = {"Wf":Wf,"Wi":Wi,"Wo":Wo,"Wc":Wc,"Wy":Wy,"bf":bf,"bi":bi,"bo":bo,"bc":bc,"by":by}

a,y,c,caches = lstm_forward(x,a0,parameters)
print("a[4][3][6] = ",a[4][3][6])
print("a.shape = ",a.shape)
print("y[1][4][3] = ",y[1][4][3])
print("y.shape = ",y.shape)
print("caches[1][1][1] = ",caches[1][1][1])
print("c[1][2][1] = ",c[1][2][1])
print("len(caches) = ",len(caches))
```

Fig. 16. Menjalankan fungsi lstm_forward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
a[4][3][6] =  0.17211776753291672
a.shape =  (5, 10, 7)
y[1][4][3] =  0.9508734618501101
y.shape =  (2, 10, 7)
caches[1][1][1] =  [ 0.82797464  0.23009474  0.76201118 -0.22232814 -0.20075807  0.18656139
  0.41005165]
c[1][2][1] =  -0.8555449167181981
len(caches) =  2
```

Fig. 17. Hasil fungsi lstm_forward

6. Membuat fungsi **rnn_cell_backward**

Fungsi rnn_cell_backward dibuat untuk mengimplementasikan backward pass pada LSTM-cell (single time-step). Berikut adalah fungsi rnn_cell_backward yang dibangun.

```python
def rnn_cell_backward(da_next,cache):
    (a_next,a_prev,xt,parameters) = cache

    Wax = parameters["Wax"]
    Waa = parameters["Waa"]
    Wya = parameters["Wya"]
    ba = parameters["ba"]
    by = parameters["by"]

    dtanh = 1-np.power(a_next,2)

    dxt = np.dot(Wax.T, da_next*dtanh)
    dWax = np.dot(da_next*dtanh, xt.T)

    da_prev = np.dot(Waa.T, da_next*dtanh)
    dWaa = np.dot(da_next*dtanh, a_prev.T)

    dba = np.sum(da_next*dtanh, axis=1, keepdims=True)

    gradients = {"dxt":dxt, "da_prev":da_prev, "dWax":dWax, "dWaa":dWaa, "dba":dba}

    return gradients
```

Fig. 18. Fungsi rnn_cell_backward

Fungsi rnn_cell_backward menerapkan rumus-rumus yang diperlukan untuk backward pass RNN-cell sebagai berikut.

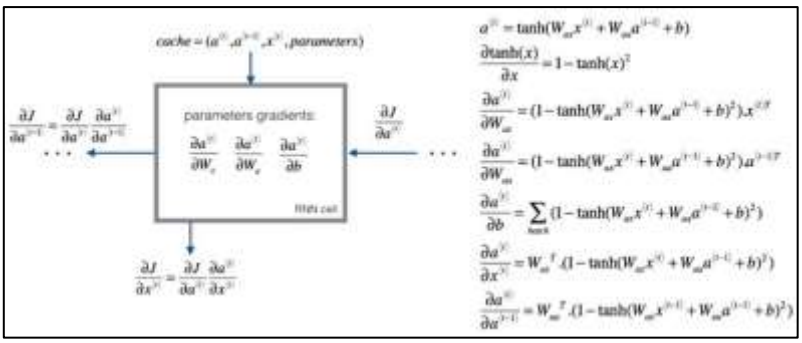

Fig. 19. Backward Pass RNN-cell

Untuk menjalankan fungsi rnn_cell_backward maka jalankan script berikut.

```
np.random.seed(1)
xt = np.random.randn(3,10)
a_prev = np.random.randn(5,10)
Wax = np.random.randn(5,3)
Waa = np.random.randn(5,5)
Wya = np.random.randn(2,5)
b = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Wax":Wax, "Waa":Waa, "Wya":Wya, "ba" : ba, "by" : by}

a_next, yt, cache = rnn_cell_forward(xt, a_prev, parameters)

da_next = np.random.randn(5,10)
gradients = rnn_cell_backward(da_next, cache)
print("gradients[\"dxt\"][1][2] =",gradients["dxt"][1][2])
print("gradients[\"dxt\"].shape =",gradients["dxt"].shape)
print("gradients[\"da_prev\"][2][3] =",gradients["da_prev"][2][3])
print("gradients[\"da_prev\"].shape =",gradients["da_prev"].shape)
print("gradients[\"dWax\"][3][1] =",gradients["dWax"][3][1])
print("gradients[\"dWax\"].shape =",gradients["dWax"].shape)
print("gradients[\"dWaa\"][1][2] =",gradients["dWaa"][1][2])
print("gradients[\"dWaa\"].shape =",gradients["dWaa"].shape)
print("gradients[\"dba\"][4] =",gradients["dba"][4])
print("gradients[\"dba\"].shape =",gradients["dba"].shape)
```

Fig. 20. Menjalankan fungsi rnn_cell_backward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
gradients["dxt"][1][2] = -0.4605641030588797
gradients["dxt"].shape = (3, 10)
gradients["da_prev"][2][3] = 0.08429686538067716
gradients["da_prev"].shape = (5, 10)
gradients["dWax"][3][1] = 0.3930818739219303
gradients["dWax"].shape = (5, 3)
gradients["dWaa"][1][2] = -0.28483955786960674
gradients["dWaa"].shape = (5, 5)
gradients["dba"][4] = [0.80517166]
gradients["dba"].shape = (5, 1)
```

Fig. 21. Hasil fungsi rnn_cell_backward

7. Membuat fungsi **rnn_backward**

Fungsi rnn_backward dibuat untuk mengimplementasikan backward pass RNN pada seluruh sequence data yang diinputkan. Berikut adalah fungsi

rnn_backward yang dibangun.

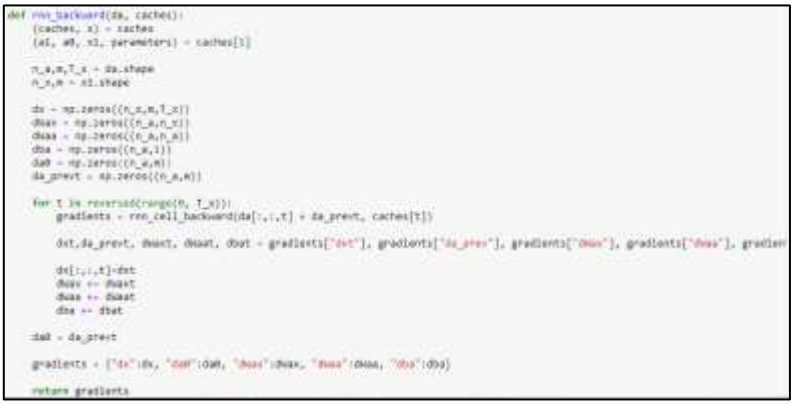

Fig. 22. Fungsi rnn_backward

Fungsi rnn_backward dijalankan untuk melakukan fungsi rnn_cell_backward sebanyak time-step yang telah dilakukan sebelumnya sehingga diperoleh gradients dari masing-masing time-step.

Untuk menjalankan fungsi rnn_backward maka jalankan script berikut.

```
np.random.seed(1)
x = np.random.randn(3,10,4)
a0 = np.random.randn(5,10)
Wax = np.random.randn(5,3)
Waa = np.random.randn(5,5)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Wax":Wax, "Waa":Waa, "Wya":Wya, "ba" : ba, "by" : by}
a, y, caches = rnn_forward(x, a0, parameters)
da = np.random.randn(5,10,4)
gradients = rnn_backward(da, caches)

print("gradients[\"dx\"][1][2] =",gradients["dx"][1][2])
print("gradients[\"dx\"].shape =",gradients["dx"].shape)
print("gradients[\"da0\"][2][3] =",gradients["da0"][2][3])
print("gradients[\"da0\"].shape =",gradients["da0"].shape)
print("gradients[\"dWax\"][3][1] =",gradients["dWax"][3][1])
print("gradients[\"dWax\"].shape =",gradients["dWax"].shape)
print("gradients[\"dWaa\"][1][2] =",gradients["dWaa"][1][2])
print("gradients[\"dWaa\"].shape =",gradients["dWaa"].shape)
print("gradients[\"dba\"][4] =",gradients["dba"][4])
print("gradients[\"dba\"].shape =",gradients["dba"].shape)
```

Fig. 23. Menjalankan fungsi rnn_backward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
gradients["dx"][1][2] = [-2.07101689 -0.59255627  0.02466855  0.01483317]
gradients["dx"].shape = (3, 10, 4)
gradients["da0"][2][3] = -0.3149423751266499
gradients["da0"].shape = (5, 10)
gradients["dWax"][3][1] = 11.264104496527777
gradients["dWax"].shape = (5, 3)
gradients["dWaa"][1][2] = 2.303333126579892
gradients["dWaa"].shape = (5, 5)
gradients["dba"][4] = [-0.74747722]
gradients["dba"].shape = (5, 1)
```

Fig. 24. Hasil fungsi rnn_backward

8. Membuat fungsi **lstm_cell_backward**

Fungsi lstm_cell_backward dibuat untuk mengimplementasikan backward pass pada LSTM-cell (single time-step). Berikut adalah fungsi lstm_cell_backward yang dibangun.

```
def lstm_cell_backward(da_next, dc_next, cache):
    (a_next,c_next,a_prev,c_prev,ft,it,cct,ot,xt,parameters) = cache

    n_x,m = xt.shape
    n_a,m = a_next.shape

    dot = da_next*np.tanh(c_next)
    dcct = (da_next*ot*(1-np.power(np.tanh(c_next), 2))+dc_next)*it
    dit = (da_next*ot*(1-np.power(np.tanh(c_next), 2))+dc_next)*cct
    dft = (da_next*ot*(1-np.power(np.tanh(c_next), 2))+dc_next)*c_prev

    dit = dit*it*(1-it)
    dft = dft*ft*(1-ft)
    dot = dot*ot*(1-ot)
    dcct = dcct*(1-np.power(cct,2))

    concat = np.zeros((n_x + n_a,m))
    concat[: n_a,:] = a_prev
    concat[n_a :,:] = xt
    dWf = np.dot(dft,concat.T)
    dWi = np.dot(dit,concat.T)
    dWc = np.dot(dcct,concat.T)
    dWo = np.dot(dot,concat.T)
    dbf = np.sum(dft, axis=1, keepdims=True)
    dbi = np.sum(dit, axis=1, keepdims=True)
    dbc = np.sum(dcct, axis=1, keepdims=True)
    dbo = np.sum(dot, axis=1, keepdims=True)
    da_prevx = np.dot(parameters['Wf'].T, dft) + np.dot(parameters['Wo'].T,dot)+np.dot(parameters['Wi'].T, dit) + np.dot(paramet
    da_prev = da_prevx[: n_a,:]
    dc_prev = (da_next*ot*(1-np.power(np.tanh(c_next),2))+dc_next)*ft
    dxt = da_prevx[n_a :,:]

    gradients = {"dxt":dxt, "da_prev":da_prev, "dc_prev":dc_prev, "dWf":dWf, "dbf":dbf, "dWi":dWi, "dbi":dbi, "dWc":dWc, "dbc":d

    return gradients
```

Fig. 25. Fungsi lstm_cell_backward

Fungsi lstm_cell_backward mengimplementasikan rumus-rumus untuk backward pass LSTM sebagai berikut.

$$d\Gamma_o^{\langle t \rangle} = da_{next} * \tanh(c_{next}) * \Gamma_o^{\langle t \rangle} * (1 - \Gamma_o^{\langle t \rangle})$$

$$d\tilde{c}^{\langle t \rangle} = dc_{next} * \Gamma_u^{\langle t \rangle} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * i_t * da_{next} * \tilde{c}^{\langle t \rangle} * (1 - \tanh(\tilde{c})^2)$$

$$d\Gamma_u^{\langle t \rangle} = dc_{next} * \tilde{c}^{\langle t \rangle} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * \tilde{c}^{\langle t \rangle} * da_{next} * \Gamma_u^{\langle t \rangle} * (1 - \Gamma_u^{\langle t \rangle})$$

$$d\Gamma_f^{\langle t \rangle} = dc_{next} * \tilde{c}_{prev} + \Gamma_o^{\langle t \rangle}(1 - \tanh(c_{next})^2) * c_{prev} * da_{next} * \Gamma_f^{\langle t \rangle} * (1 - \Gamma_f^{\langle t \rangle})$$

$$dW_f = d\Gamma_f^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_u = d\Gamma_u^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_c = d\tilde{c}^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_o = d\Gamma_o^{\langle t \rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

To calculate db_f, db_u, db_c, db_o you just need to sum across the horizontal (axis= 1) axis on $d\Gamma_f^{\langle t \rangle}, d\Gamma_u^{\langle t \rangle}, d\tilde{c}^{\langle t \rangle}, d\Gamma_o^{\langle t \rangle}$ respectively. Note that you should have the `keep_dims = True` option.

Finally, you will compute the derivative with respect to the previous hidden state, previous memory state, and input.

$$da_{prev} = W_f^T * d\Gamma_f^{\langle t \rangle} + W_u^T * d\Gamma_u^{\langle t \rangle} + W_c^T * d\tilde{c}^{\langle t \rangle} + W_o^T * d\Gamma_o^{\langle t \rangle}$$

Here, the weights for equations 13 are the first n_a, (i.e. $W_f = W_f[:n_a, :]$ etc...)

$$dc_{prev} = dc_{next}\Gamma_f^{\langle t \rangle} + \Gamma_o^{\langle t \rangle} * (1 - \tanh(c_{next})^2) * \Gamma_f^{\langle t \rangle} * da_{next}$$

$$dx^{\langle t \rangle} = W_f^T * d\Gamma_f^{\langle t \rangle} + W_u^T * d\Gamma_u^{\langle t \rangle} + W_c^T * d\tilde{c}_t + W_o^T * d\Gamma_o^{\langle t \rangle}$$

where the weights for equation 15 are from n_a to the end, (i.e. $W_f = W_f[n_a :, :]$ etc...)

Fig. 26. Rumus-rumus backward pass LSTM

Jıka fungsi lstm_cell_backward dibandingkan dengan **Fig. 26**, maka terlihat bahwa variabel dft merupakan implementasi dari rumus $d\Gamma_f$, variabel dit merupakan implementasi dari rumus $d\Gamma_u$, variabel dcct merupakan implementasi dari rumus $d\tilde{c}$, dan variabel dot merupakan implementasi dari rumus $d\Gamma_o$. Untuk nama variabel lainnya yang implementasi rumus di **Fig. 26** adalah sama.

Untuk menjalankan fungsi lstm_cell_backward maka jalankan script berikut.

```
np.random.seed(1)
xt = np.random.randn(3,10)
a_prev = np.random.randn(5,10)
c_prev = np.random.randn(5,10)
Wf = np.random.randn(5,5+3)
bf = np.random.randn(5,1)
Wi = np.random.randn(5,5+3)
bi = np.random.randn(5,1)
Wo = np.random.randn(5,5+3)
bo = np.random.randn(5,1)
Wc = np.random.randn(5,5+3)
bc = np.random.randn(5,1)
Wy = np.random.randn(2,5)
by = np.random.randn(2,1)

parameters = {"Wf":Wf,"Wi":Wi,"Wo":Wo,"Wc":Wc,"Wy":Wy,"bf":bf,"bi":bi,"bo":bo,"bc":bc,"by":by}

a_next,c_next,yt,cache = lstm_cell_forward(xt,a_prev,c_prev,parameters)

da_next = np.random.randn(5,10)
dc_next = np.random.randn(5,10)
gradients = lstm_cell_backward(da_next, dc_next, cache)
```

```
print("gradients[\"dxt\"][1][2] =",gradients["dxt"][1][2])
print("gradients[\"dxt\"].shape =",gradients["dxt"].shape)
print("gradients[\"da_prev\"][2][3] =",gradients["da_prev"][2][3])
print("gradients[\"da_prev\"].shape =",gradients["da_prev"].shape)
print("gradients[\"dc_prev\"][2][3] =",gradients["dc_prev"][2][3])
print("gradients[\"dc_prev\"].shape =",gradients["dc_prev"].shape)
print("gradients[\"dWf\"][3][1] =",gradients["dWf"][3][1])
print("gradients[\"dWf\"].shape =",gradients["dWf"].shape)
print("gradients[\"dWi\"][1][2] =",gradients["dWi"][1][2])
print("gradients[\"dWi\"].shape =",gradients["dWi"].shape)
print("gradients[\"dWc\"][3][1] =",gradients["dWc"][3][1])
print("gradients[\"dWc\"].shape =",gradients["dWc"].shape)
print("gradients[\"dWo\"][1][2] =",gradients["dWo"][1][2])
print("gradients[\"dWo\"].shape =",gradients["dWo"].shape)
print("gradients[\"dbf\"][4] =",gradients["dbf"][4])
print("gradients[\"dbf\"].shape =",gradients["dbf"].shape)
print("gradients[\"dbi\"][4] =",gradients["dbi"][4])
print("gradients[\"dbi\"].shape =",gradients["dbi"].shape)
print("gradients[\"dbc\"][4] =",gradients["dbc"][4])
print("gradients[\"dbc\"].shape =",gradients["dbc"].shape)
print("gradients[\"dbo\"][4] =",gradients["dbo"][4])
print("gradients[\"dbo\"].shape =",gradients["dbo"].shape)
```

Fig. 27. Menjalankan fungsi lstm_cell_backward

Jika berhasil dijalankan, maka akan tampil hasil sebagai berikut.

```
gradients["dxt"][1][2] = 3.2305591151091884
gradients["dxt"].shape = (3, 10)
gradients["da_prev"][2][3] = -0.0639621419710924
gradients["da_prev"].shape = (5, 10)
gradients["dc_prev"][2][3] = 0.7975220387970015
gradients["dc_prev"].shape = (5, 10)
gradients["dWf"][3][1] = -0.14795483816449723
gradients["dWf"].shape = (5, 8)
gradients["dWi"][1][2] = 1.0574980552259903
gradients["dWi"].shape = (5, 8)
gradients["dWc"][3][1] = 2.3045621636876668
gradients["dWc"].shape = (5, 8)
gradients["dWo"][1][2] = 0.33131159528921084
gradients["dWo"].shape = (5, 8)
gradients["dbf"][4] = [0.18864637]
gradients["dbf"].shape = (5, 1)
gradients["dbi"][4] = [-0.40142491]
gradients["dbi"].shape = (5, 1)
gradients["dbc"][4] = [0.25587763]
gradients["dbc"].shape = (5, 1)
gradients["dbo"][4] = [0.13893342]
gradients["dbo"].shape = (5, 1)
```

Fig. 28. Hasil fungsi lstm_cell_backward

3.2 RNN untuk Menghasilkan Nama Dinosaurus Baru

Pada tugas ini, kasus yang dihadapi adalah memberi nama spesies dinosaurus baru berdasarkan pola nama-nama dinosaurus yang pernah ada (https://raw.githubusercontent.com/carlb15/Dinosaur_Island_Character_Lvl_Language_Model/master/dinos.txt). Character level language model akan dibangun untuk menghasilkan (generate) nama-nama dinosaurus baru tersebut secara random.

Dalam membangun model ini, library rnn_utils akan digunakan agar dapat mengimplementasikan fungsi rnn_forward dan rnn_backward seperti yang sudah dibangun sebelumnya. Dataset kemudian dicari karakter uniknya dan disimpan untuk kemudian diberi index.

```
data = open('dinos.txt','r').read()
data = data.lower()
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
```

```
char_to_ix = { ch:i for i,ch in enumerate(sorted(chars)) }
ix_to_char = { i:ch for i,ch in enumerate(sorted(chars)) }
print(ix_to_char)
```

Fig. 29. Dataset dicari karakter uniknya dan diberi index

Model terdiri dari beberapa bagian. Bagian pertama adalah forward propagation untuk menghitung loss. Berikutnya adalah backward propagation untuk menghitung gradient loss. Untuk mencegah terjadinya exploding gradient (nilai yang sangat besar), maka akan dilakukan gradient clipping (simple element-wise). Dari gradient yang diperoleh, maka parameter akan diperbarui dengan gradient descent (stochastic).

Berikut adalah fungsi-fungsi yang dibuat untuk membangun model yang menghasilkan nama dinosaurus baru.

1. Membuat fungsi **clip** untuk menerapkan gradient clipping.

```
def clip(gradients,maxValue):
    dWaa,dWax,dWya,db,dby = gradients['dWaa'], gradients['dWax'], gradients['dWya'], gradients['db'], gradients['dby']

    for gradient in [dWax, dWaa, dWya, db, dby]:
        np.clip(gradient, -maxValue, maxValue, gradient)

    gradients = {"dWaa":dWaa,"dWax":dWax,"dWya":dWya,"db":db,"dby":dby}
    return gradients
```

Fig. 30. Fungsi untuk gradient clipping

2. Membuat fungsi **sample** untuk membentuk nama baru dinosaurus.

   Berikut adalah struktur untuk membentuk nama baru dinosaurus per karakter.

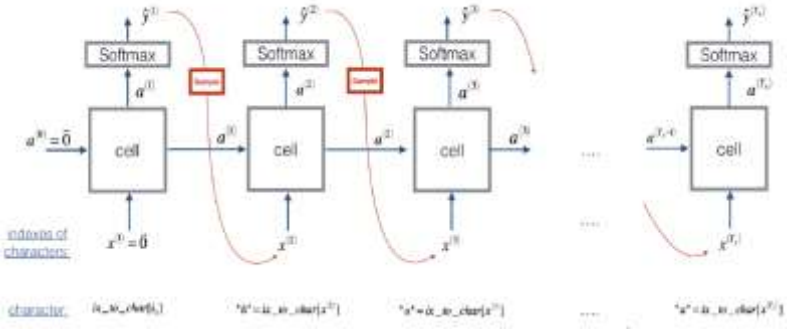

Fig. 31. Struktur untuk membentuk nama baru dinosaurus

   Struktur diatas kemudian diterapkan menjadi sebuah fungsi (fungsi sample) sebagai berikut.

```
def sample(parameters,char_to_ix,seed):
    Waa,Wax,Wya,by,b = parameters['Waa'], parameters['Wax'], parameters['Wya'], parameters['by'], parameters['b']
    vocab_size = by.shape[0]
    n_a = Waa.shape[1]
    n_x = Wax.shape[1]

    x = np.zeros((n_x,1))

    a_prev = np.zeros((n_a,1))

    indices = []

    idx = -1

    counter = 0
    newline_character = char_to_ix['\n']

    while (idx != newline_character and counter != 50):
        a = np.tanh(np.dot(Wax,x)+np.dot(Waa,a_prev)+b)
        z = np.dot(Wya,a)+by
        y = softmax(z)

        np.random.seed(counter+seed)

        idx = np.random.choice(range(0,vocab_size),p=y.ravel())

        indices.append(idx)

        indices.append(idx)

        x=np.zeros((vocab_size,1))
        x[idx] = 1

        a_prev = a

        seed += 1
        counter += 1

    if (counter == 50):
        indices.append(char_to_ix['\n'])

    return indices
```

Fig. 32. Fungsi untuk membentuk nama baru dinosaurus

3. Membuat fungsi **optimize** untuk menerapkan one step gradient descent dengan cara memanggil fungsi untuk forward RNN, fungsi untuk backward RNN, fungsi untuk gradient clipping (fungsi clip) agar nilai gradient tidak lebih dari 5, dan fungsi untuk update parameter.

```
def optimize(X,Y,a_prev,parameters,learning_rate = 0.01):
    loss,cache = rnn_forward(X,Y,a_prev,parameters)

    gradients,a = rnn_backward(X,Y,parameters, cache)

    gradients = clip(gradients,5)

    parameters = update_parameters(parameters,gradients,learning_rate)

    return loss,gradients, a[len(X)-1]
```

Fig. 33. Fungsi optimize

4. Membuat fungsi **model** untuk membentuk model dengan cara menjalankan fungsi optimize dan fungsi sample.

```
def model(data,ix_to_char,char_to_ix,num_iterations=35000,n_a=50,dino_names=7,vocab_size=27):
    n_x,n_y = vocab_size,vocab_size

    parameters = initialize_parameters(n_a,n_x,n_y)

    loss = get_initial_loss(vocab_size, dino_names)

    with open("dinos.txt") as f:
        examples = f.readlines()
    examples = [x.lower().strip() for x in examples]

    np.random.seed(0)
    np.random.shuffle(examples)

    a_prev = np.zeros((n_a,1))

    for j in range (num_iterations):
        index = j % len(examples)
        X = [None] + [char_to_ix[ch] for ch in examples[index]]
        Y = X[1:] + [char_to_ix["\n"]]

        curr_loss,gradients,a_prev = optimize(X,Y,a_prev,parameters,learning_rate = 0.01)

        loss = smooth(loss,curr_loss)

        if j % 2000 == 0:
            print('Iteration: %d, Loss: %f' % (j,loss) + '\n')

            seed = 0
            for name in range(dino_names):

                sampled_indices = sample(parameters,char_to_ix,seed)
                print_sample(sampled_indices,ix_to_char)

                seed += 1

            print('\n')

    return parameters
```

Fig. 34. Pembentukan model

Fungsi optimize yang telah dibuat sebelumnya akan dijalankan sebanyak iterasi (35.000 kali). Setiap kelipatan 2000 iterasi, akan dipanggil fungsi sample sebanyak 7 kali untuk menghasilkan 7 nama dinosaurus secara random. Pada iterasi awal, akan terbentuk nama yang terdiri dari karakter-karakter sangat acak seperti berikut.

```
Iteration: 0, Loss: 23.087336

Nkzxwtdmfqoeyhsqwasjkjvu
Kneb
Kzxwtdmfqoeyhsqwasjkjvu
Neb
Zxwtdmfqoeyhsqwasjkjvu
Eb
Xwtdmfqoeyhsqwasjkjvu


Iteration: 2000, Loss: 27.884160

Liusskeomnolxeros
Hmdaairus
Hytroligoraurus
Lecalosapaus
Xusicikoraurus
Abalpsamantisaurus
Tpraneronxeros
```

Fig. 35. Hasil iterasi awal

Namun, semakin mendekati iterasi akhir (35.000), semakin jelas nama dinosaurus yang dihasilkan.

```
Iteration: 32000, Loss: 22.408924

Meutosaurus
Jrabainopeirus
Kusrochenorontosaurus
Macaisina
Wurodonthauroroclimisaurus
Eg
Trodonti


Iteration: 34000, Loss: 22.523032

Mawrosaurus
Inee
Iustodon
Mabaeosaurus
Wusiaggnhaurosaurus
Eg
Stichaniangosaurus
```

Fig. 36. Hasil iterasi akhir

3.3 Tutorial LSTM untuk Prediksi Pasar Modal

Pada tugas ini, LSTM akan diimplementasikan untuk data time series. Langkah pertama dimulai dengan mengimpor library yang akan digunakan sebagai berikut.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Fig. 37. Import Library

Kemudian dataset yang akan digunakan pun dimasukkan. Data yang digunakan merupakan data time series berupa dataset pasar modal TATA GLOBAL. Dataset ini berisi 2035 data. Pada tugas ini, hanya kolom "Open" saja yang digunakan sebagai training set.

```
dataset_train = pd.read_csv('NSE-TATAGLOBAL.csv')
training_set = dataset_train.iloc[:,1:2].values
```

Fig. 38. Dataset untuk traning set

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

Fig. 39. Dataset pasar saham TATA GLOBAL

Training set tersebut kemudian diterapkan scaling dengan MinMaxScaler agar berada pada range 0 hingga 1.

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))
training_set_scaled = sc.fit_transform(training_set)
```

Fig. 40. Scaling training set

Dataset kemudian dibagi menjadi beberapa bagian, dimana disetiap bagian berisi 60 data time series.

```
X_train = []
y_train = []
for i in range(60,2035):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train,y_train = np.array(X_train),np.array(y_train)

X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
```

Fig. 41. Training set dibentuk menjadi 60 data per bagian

Selanjutnya dibangun model menggunakan keras. Library keras diimport terlebih dahulu.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

Fig. 42. Import library keras

Sequential digunakan untuk inisialisasi neural network. Dense digunakan untuk menambah layer neural network yang terhubung secara dense. LSTM digunakan untuk menambahkan layer Long-Short Term Memory. Dropout digunakan untuk me-dropout layer-layer sehingga mencegah terjadinya overfitting. Berikut adalah kode yang membangun neural network yang dense dengan tambahan LSTM dan dropout.

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train,y_train,epochs = 100,batch_size = 32)
```

Fig. 43. Membuat model

Pada model ini, digunakan Adam optimizer dan mean squared error sebagai loss-nya. Model dijalankan sebanyak 100 epoch dan ukuran batchnya adalah 32.

Untuk melakukan prediksi, maka digunakan test set TATA GLOBAL sebagai berikut. Test set terdiri dari 16 data. Hanya kolom "Open" juga yang digunakan.

```
dataset_test = pd.read_csv('tatatest.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

Fig. 44. Test set

Test set tersebut kemudian dilakukan scaling dan pembagian menjadi 60 data per bagian seperti training set sebelumnya.

```
dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60,76):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0], X_test.shape[1],1))
```

Fig. 45. Scaling test set dan pembagian menjadi 60 data per bagian

Kemudian dilakukan prediksi dari test set dan dilakukan inverse scaling.

```
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Fig. 46. Prediksi test set dan inverse scaling

Hasil prediksi harga modal dan harga modal sebenarnya kemudian ditampilkan ke dalam plot dengan kode sebagai berikut.

```
plt.plot(real_stock_price,color = 'black',label = 'TATA Stock Price')
plt.plot(predicted_stock_price, color = 'green', label = 'Predicted TATA Stock Price')
plt.title('TATA Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('TATA Stock Price')
plt.legend()
plt.show()
```

Fig. 47. Penyajian hasil ke dalam bentuk plot

Hasil yang diperoleh adalah sebagai berikut.

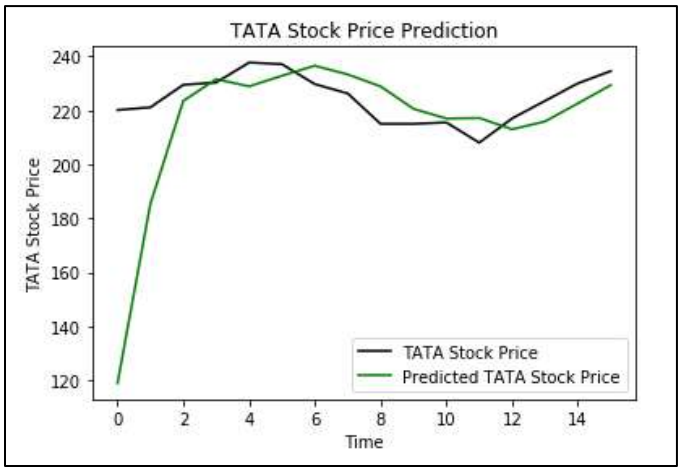

Fig. 48. Hasil prediksi pasar modal TATA GLOBAL

3.4 LSTM untuk Predıksı Pasar Modal dengan Dataset Baru

Model LSTM dibangun kembali untuk memprediksi dataset pasar modal yang baru. Dataset yang digunakan merupakan data time series pasar modal Hang Seng Index (https://finance.yahoo.com/quote/%5EHSI/history?p=%5EHSI) yang berjumlah 250 data. Untuk training set, hanya kolom "Open" dari data set saja yang digunakan.

	Date	Open	High	Low	Close	Adj Close	Volume
0	10-10-19	25625.57031	25809.58008	25521.94922	25707.92969	25707.92969	1.434232e+09
1	09-10-19	25736.88086	25866.83984	25656.66016	25682.81055	25682.81055	1.588334e+09
2	08-10-19	25848.73047	26180.02930	25761.50000	25893.40039	25893.40039	1.779110e+09
3	04-10-19	26169.50977	26169.50977	25612.49023	25821.02930	25821.02930	1.137244e+09
4	03-10-19	25831.43945	26192.86914	25809.47070	26110.31055	26110.31055	1.142351e+09

Fig. 49. Dataset pasar saham Hang Seng Index (HSI)

Data kemudian dilakukan scaling sehingga berada pada range 0 hingga 1 dan dibagi menjadi 7 data per bagian.

```
X_train = []
y_train = []
for i in range(7,250):
    X_train.append(training_set_scaled[i-7:i,0])
    y_train.append(training_set_scaled[i,0])
X_train,y_train = np.array(X_train),np.array(y_train)

X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
```

Fig. 50. Training set dibentuk menjadi 7 data per bagian

Untuk pembangunan model, digunakan struktur yang sama. Hanya saja, ukuran batch diperkecil menjadi 3 saja.

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train,y_train,epochs = 100,batch_size = 3)
```

Fig. 51. Membuat model

Untuk melakukan prediksi, test set Hang Seng Index digunakan, dimana test set ini berjumlah 25 data.

Test set kemudian di-scaling dan dibagi menjadi 7 data per bagian. Kemudian dilakukan prediksi terhadap test set dari model yang telah dibuat sebelumnya. Setelah prediksi selesai, maka hasilnya di-inverse scaling kembali.

```
dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 7:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(7,32): #7(Timestamp) + 25 (jumlah data test)
    X_test.append(inputs[i-7:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0], X_test.shape[1],1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Fig. 52. Scaling, pembagian menjadi 7 data per bagian, prediksi, dan inverse-scaling

Hasil prediksi harga modal dan harga modal sebenarnya lalu disajikan dalam bentuk plot. Hasil yang diperoleh dari prediksi dataset pasar modal baru (HSI) dengan LSTM adalah sebagai berikut.

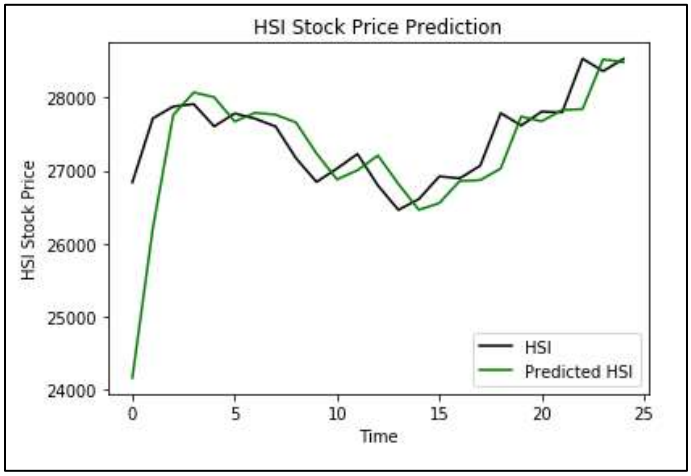

Fig. 53. Hasil prediksi pasar modal HSI

4 Kesimpulan

Dalam melaksanakan serangkaian tugas diatas, maka tujuan telah berhasil dicapai dalam menerapkan tutorial RNN, menerapkan RNN untuk menghasilkan nama dinosaurus baru, menerapkan tutorial LSTM untuk prediksi pasar modal, dan menerapkan LSTM untuk prediksi pasar modal dengan dataset baru.

Referensi

1. Bishop, C.M. : Pattern Recognition and Machine Learning. Springer, USA (2006)
2. Dinosaur_Island_Character_Lvl_Language_Model/dinos.txt at master · carlb15/Dinosaur_Island_Character_Lvl_Language_Model, https://raw.githubusercontent.com/carlb15/Dinosaur_Island_Character_Lvl_Language_Model/master/dinos.txt
3. ^HSI 26,498.37 281.33 1.07% : HANG SENG INDEX - Yahoo Finance, https://finance.yahoo.com/quote/%5EHSI/history?p=%5EHSI