

Implementasi Recurrent Neural Network (RNN) dan Long Short-Term Memory (LSTM)

Zalid Qomalita Hijriana (23519013)

Magister Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung, Indonesia
23519013@std.stei.itb.ac.id

Abstraksi. Keefektifan RNN dan LSTM telah diakui dalam berbagai jurnal ilmiah. RNN dan LSTM banyak diimplementasikan terutama untuk menangani persoalan-persoalan yang bersifat sekvensial karena keunggulannya yang seolah memiliki “memori”. Pada laporan ini, akan dipaparkan secara detail teori tentang RNN dan LSTM, tahap demi tahap pembangunan modelnya dalam bahasa pemrograman python, serta beberapa implementasinya, seperti pada studi kasus *Dinosaurs Land* dan *Stock Market Prediction*.

Kata kunci: RNN, LSTM, python.

1 Pendahuluan

Pada neural network standar, data dari suatu persoalan yang akan diselesaikan perlu didefinisikan panjang input dan outputnya. Namun pada kenyataannya, tidak semua data dapat ditentukan panjang input dan outputnya secara pasti. Misalnya pada mesin penerjemah, panjang input dan outputnya akan sangat variatif. Oleh karena itu, diperlukan solusi untuk mengatasi permasalahan tersebut. Saat ini, metode yang terkenal untuk menangani permasalahan seperti ini adalah *Recurrent Neural Network* (RNN). RNN sangat efektif untuk *Natural Language Processing* dan *sequence task* lainnya, karena RNN seolah memiliki “memory”.

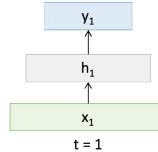
Long Short Term Memory (LSTM) merupakan salah satu arsitektur RNN yang paling handal. LSTM memperkenalkan istilah “*memory cell*” dalam pembahasannya. Dengan *memory cell* ini, banyak informasi penting yang dapat diingat. Sehingga LSTM ini sangat baik untuk memprediksi.

Laporan ini menjabarkan tentang teori dasar mengenai RNN dan LSTM, penjelasan mengenai implementasi RNN dan LSTM menggunakan python, serta implementasinya pada beberapa kasus seperti untuk menangani data sekvensial pada studi kasus *Dinosaurs Land* dan data *time-series* yang fluktuatif seperti prediksi *stock* (saham) pada pasar modal.

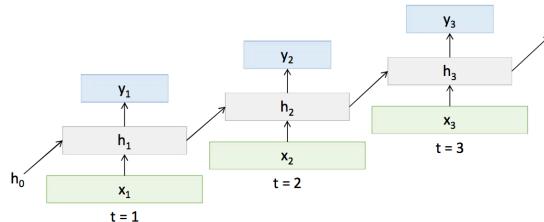
2 Dasar Teori

2.1 Recurrent Neural Network (RNN)

Berbeda dengan *feedforward network* atau yang lebih dikenal dengan MLP (Multilayer Perceptron), RNN mengambil input dari dua sumber, yakni input dari sumber “masa kini” dan dari sumber “masa lampau”. Informasi dari kedua sumber ini digunakan untuk memutuskan keluaran dari suatu data baru. Caranya adalah dengan menerapkan *feedback loop* dimana output dari masing-masing *instance* akan digunakan sebagai input untuk momen berikutnya. Darisini, kita dapat mengatakan bahwa *recurrent neural network* memiliki memori, dimana masing-masing urutan kejadian memiliki sekumpulan informasi, dan informasi-informasi ini akan disimpan dalam *hidden state*.

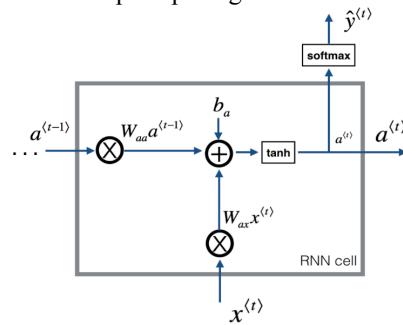


Gbr. 1. Contoh arsitektur *feedforward network*.



Gbr. 2. Contoh arsitektur RNN.

Jaringan RNN tersusun dari cell RNN yang berulang selama T *timestep*. Untuk setiap cell RNN, dilustrasikan seperti pada gambar berikut:



Gbr. 3. Arsitektur cell RNN.

Dari arsitektur tersebut maka dapat dirumuskan formula untuk setiap cell RNN.

$$\begin{aligned} a^{(t)} &= \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b_a) \\ \hat{y}^{(t)} &= \text{softmax}(W_{ya}a^{(t)} + b_y) \end{aligned}$$

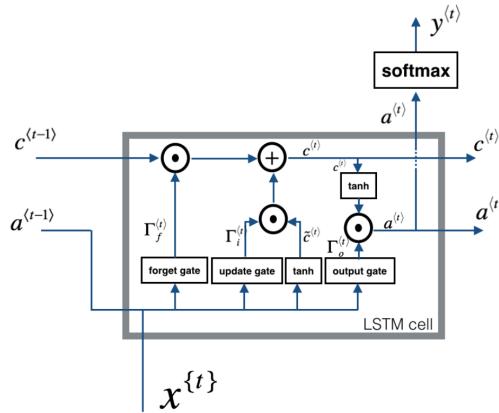
dimana,

- $x^{(t)}$: input data pada timestep ke- t
- $a^{(t)}$: hidden state pada timestep ke- t
- $a^{(t-1)}$: hidden state pada timestep ke $(t - 1)$
- W_{aa} : weight pada hidden state
- W_{ax} : weight pada input
- W_{ya} : weight diantara hidden state dan output
- b_a : bias
- b_y : bias diantara hidden state dan output
- $\hat{y}^{(t)}$: prediksi pada timestep ke- t

2.2 Long Short Term Memory (LSTM)

LSTM diperkenalkan oleh Hochreiter dan Schmidhuber pada tahun 1997. LSTM adalah tipe spesial dari RNN. Jaringan LSTM ahli dalam menangani kasus yang memiliki *long-term dependencies* karena LSTM mampu mengingat informasi dalam jangka panjang. LSTM menggunakan ide dari “Constant Error Flow” pada RNN untuk membentuk “Constant Error Carousel” yang dapat memastikan gradient tetap dalam keadaan baik. Komponen utama LSTM adalah memori yang bertindak sebagai *accumulator* yang berisi informasi dari waktu ke waktu.

Perbedaan tersebut berpengaruh pada struktur jaringan LSTM yang berbeda dengan RNN. RNN memiliki struktur yang sederhana dengan *feedback loop*. Pada LSTM terdapat tambahan yaitu blok memori.



Gbr. 4. Arsitektur cell LSTM.

Pada gambar diatas, garis horizontal cell state $c^{(t-1)}$ dan $c^{(t)}$ bertindak seperti *conveyor belt* yang melintasi seluruh cell pada jaringan. Garis ini membawa informasi dari cell-cell sebelumnya yang dibawa menuju cell-cell berikutnya. Tidak semua informasi disimpan. Ada juga informasi yang dihapuskan. Hanya informasi yang dianggap penting saja yang disimpan. Penghapusan informasi ini berdasarkan threshold, dimana perhitungan dilakukan pada *forget gate layer* (Γ) atau yang juga dikenal dengan sigmoid layer. Output pada *forget layer* berupa vektor 0 dan 1. Jika menghasilkan nilai 0 berarti informasi tersebut dihapus dan 1 berarti informasi tersebut tetap disimpan. Informasi yang diperoleh kemudian dilakukan operasi *element-wise multiplication* dengan cell state sebelumnya. Sehingga, jika ditulis dalam persamaan, persamaan untuk forget gate adalah:

$$\Gamma_f^{(t)} = \sigma(W_f[a^{(t-1)}, x^{(t)}] + b_f)$$

Selanjutnya, setelah melakukan tahap forget gate, dilakukan juga tahap update gate untuk menyimpan informasi yang terupdate. Persamaan untuk update gate adalah:

$$\Gamma_u^{(t)} = \sigma(W_u[a^{(t-1)}, x^{(t)}] + b_u)$$

Setelah gate terupdate, berikutnya adalah mengupdate cell dengan menggunakan persamaan berikut:

$$\begin{aligned} c^{(t)} &= \tanh(W_c[a^{(t-1)}, x^{(t)}] + b_c) \\ c^{(t)} &= \Gamma_f^{(t)} * c^{(t-1)} + \Gamma_u^{(t)} * c^{(t)} \end{aligned}$$

Sehingga diperoleh formula untuk output adalah:

$$\begin{aligned} \Gamma_o^{(t)} &= \sigma(W_o[a^{(t-1)}, x^{(t)}] + b_o) \\ a^{(t)} &= \Gamma_o^{(t)} * \tanh(c^{(t)}) \end{aligned}$$

3 Implementasi

Pada tugas ini, implementasi RNN dan LSTM dilakukan secara bertahap. Terdapat 4 tahap implementasi, yaitu:

- RNN dan LSTM Step-by-step
- Implementasi RNN untuk studi kasus Dinosaurs Land
- Implementasi untuk studi kasus Pasar Modal
- Implementasi untuk studi kasus Pasar Modal dengan data baru

Pengimplementasian RNN dan LSTM ini, digunakan bahasa pemrograman python 3.7 dan tools jupyter notebook. Library yang perlu dipersiapkan/diinstall antara lain numpy, scikit-learn, matplotlib, serta Keras.

3.1 RNN dan LSTM Step by Step

Implementasi RNN dan LSTM dimulai dari mengimplementasikan arsitektur cell

kedalam program dilanjutkan arsitektur utuh dari RNN dan LSTM. Terdapat proses forward dan backward.

Pertama, perlu dilakukan import library-library yang dibutuhkan seperti numpy untuk melakukan fungsi-fungsi matematis, serta rnn_utils untuk fungsi-fungsi pada RNN.

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

Gbr. 5. Import library.

3.1.1 RNN Cell Forward

Jaringan RNN dapat dilihat sebagai pengulangan dari single cell. Sehingga perlu dibuat terlebih dahulu fungsi untuk model single cell (fungsi rnn_cell_forward).

```
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
```

Gbr. 5. Import library.

Program tersebut merupakan pengimplementasian dari formula RNN cell yang telah dijabarkan pada landasan teori. Disini terdapat cache yang nantinya akan digunakan untuk proses backward pass. Cache ini berisi informasi: xt, a_next, a_prev, dan parameters. Untuk mengetes fungsi ini berjalan dengan baik, maka digunakan nilai random.

```
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)
```

Gbr. 6. RNN cell forward tes.

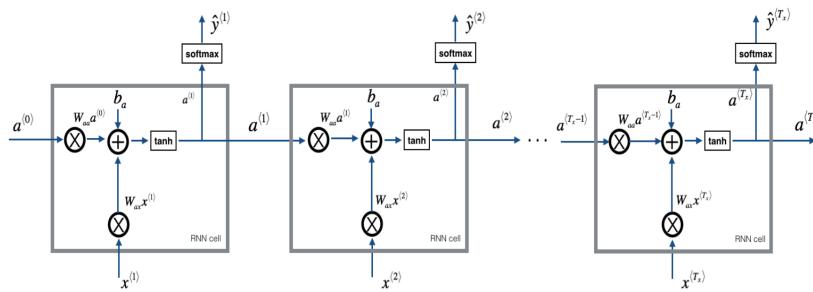
Maka, jika dijalankan akan diperoleh output:

```
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)
```

Gbr. 7. Hasil RNN cell forward

3.1.2 RNN Forward

Selanjutnya, dilakukan pengulangan terhadap RNN cell, seperti ditunjukkan pada gambar berikut:



Gbr. 8. RNN forward

Diberikan input sekuens x selama $timestep T$, menghasilkan output y untuk setiap timestep. Langkah-langkah implementasinya adalah sebagai berikut :

```
#Keterangan :
# a -- hidden state untuk setiap time step
# a0 -- initial hidden state
# x -- input untuk setiap time step
# y_pred -- prediksi untuk setiap time step
# caches -- berisi list of cache dan x
```

```
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(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
```

Gbr. 9. RNN forward

Jika fungsi berjalan, maka akan dihasilkan output sebagai 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))

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

```

Gbr. 10. Hasil RNN forward

3.1.3 LSTM Cell

```

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

```

Gbr. 11. LSTM cell forward

Fungsi ini mengimplementasikan arsitektur dari LSTM cell pada Gbr 4. LSTM cell dibentuk melalui formula-formula yang telah didefinisikan pada poin 2.2 diatas. Input dari LSTM cell ini adalah x_t yaitu input pada timestep ke-t, a_{prev} merupakan hidden state pada timestep ke t-1, c_{prev} adalah memory state pada timestep t-1. Setelah dilakukan perhitungan ke dalam formula LSTM, diperoleh output a_{next} , c_{next} , y_t _pred, dan cache. a_{next} berarti next hidden state, c_{next} berarti next memory state, y_t _pred berarti prediksi pada timestep ke-t.

Jika dilakukan tes untuk random number terhadap fungsi tersebut, maka menghasilkan output sebagai 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))
```

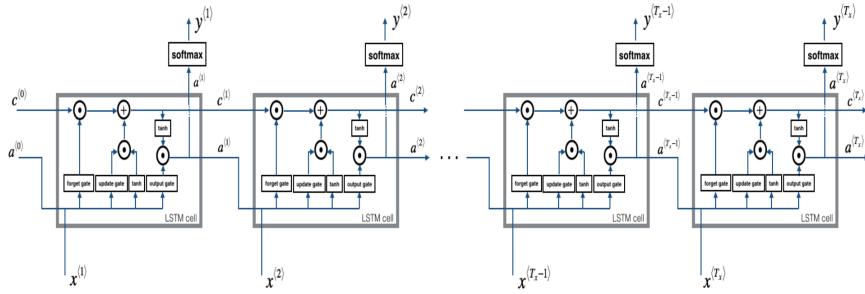
Gbr. 12. LSTM cell forward tes

```
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
```

Gbr. 13. Hasil LSTM cell forward

3.1.4 LSTM Forward

Arsitektur jaringan LSTM berupa pengulangan dari LSTM Cell. Membentuknya dengan menggunakan *for-loop instruction*.



Gbr. 14. Arsitektur LSTM forward

Sehingga jika diimplementasikan dengan code akan menjadi:

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

    a_next = a0
    c_next = np.zeros(a_next.shape)

    for t in range(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
```

Gbr. 15. LSTM forward

```

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,cache = 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("cache[1][1][1] = ", cache[1][1][1])
print("len(cache) = ", len(cache))

print("c[1][2][1] = ", c[1][2][1])

```

Gbr. 16. LSTM forward tes

```

a[4][3][6] =  0.7316245102699646
a.shape = (5, 10, 7)
y[1][4][3] =  0.9508734618501101
y.shape = (2, 10, 7)
cache[1][1][1] = [ 0.82797464  0.23009474  0.76201118 -0.22232814 -0.20075807  0.18656139
                  0.41005165]
len(cache) = 2
c[1][2][1] = -0.8555449167181982

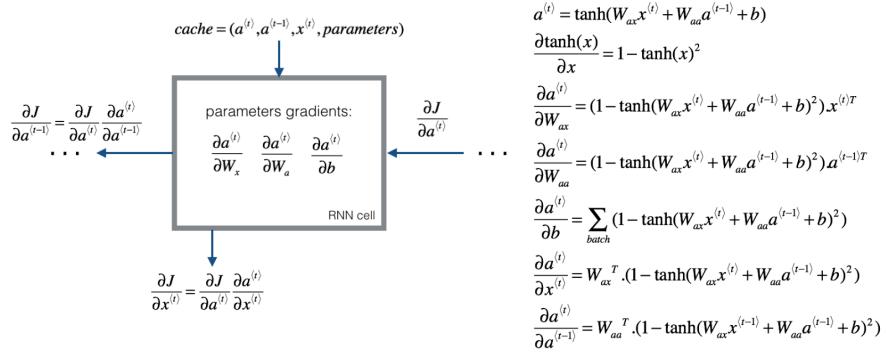
```

Gbr. 17. Hasil LSTM forward tes

3.1.5 RNN Cell Backward

Proses backward bersifat opsional. Pada Deep learning yang modern, hanya perlu dilakukan proses forward dan framework yang akan menangani proses backward. Namun pada tugas ini, perlu mengetahui detail dari proses didalam RNN dan LSTM, sehingga dilakukan pengimplementasian juga untuk proses backward.

Pada backward proses di RNN cell, diilustrasikan seperti pada gambar berikut:



Gbr. 18. Formula untuk RNN cell backward

Sehingga pengimplementasian formula tersebut kedalam code akan menjadi sebagai berikut:

```
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
```

Gbr. 19. RNN cell backward

Untuk mengetahui isi setiap variable dalam fungsi tersebut, dilakukan dengan memberikan input *random number*.

```
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}

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)
```

Gbr. 20. RNN cell backward tes.

Sehingga menghasilkan output sebagai berikut:

```
gradients["dxt\[1\]\[2\] -0.4605641030588796
gradients["dxt"].shape (3, 10)
gradients["da_prev\[2\]\[3\] 0.08429686538067718
gradients["da_prev"].shape (5, 10)
gradients["dWax\[3\]\[1\] 0.3930818739219303
gradients["dWax"].shape (5, 3)
gradients["dWaa\[1\]\[2\] -0.2848395578696067
gradients["dWaa"].shape (5, 5)
gradients["dba\[4\] [0.80517166]
gradients["dba"].shape (5, 1)
```

Gbr. 21. Hasil RNN cell backward tes

3.1.6 RNN Backward

Seperti pada proses forward, pembentukan jaringan untuk RNN dilakukan dengan penerapan *looping* dari RNN cell backward.

```

def rnn_backward(da, caches):
    (caches, x) = caches
    (a1, a0, x1, parameters) = caches[0]

    n_a, m, T_x = da.shape
    n_x, m = x1.shape

    dx = np.zeros((n_x, m, T_x))
    dWax = np.zeros((n_a, n_x))
    dWaa = np.zeros((n_a, n_a))
    dba = np.zeros((n_a, 1))
    da0 = np.zeros((n_a, m))
    da_prevt = np.zeros((n_a, m))

    for t in reversed(range(T_x)):
        gradients = rnn_cell_backward(da[:, :, t] + da_prevt, caches[t])
        dxt, da_prevt, dWaxt, dbat = gradients["dxt"], gradients["da_prevt"],
        gradients["dWax"], gradients["dWaa"], gradients["dba"]
        dx[:, :, t] = dxt
        dWax += dWaxt
        dWaa += dWaa
        dba += dbat

        da0 = da_prevt
        gradients = {"dx": dx, "da0": da0, "dWax": dWax, "dWaa": dWaa, "dba": dba}

    return gradients

```

Gbr. 22. RNN backward

Kemudian, dilakukan tes untuk menlihat apakah fungsi berjalan dengan baik.

```

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)

gradients["dx"][1][2] = [-2.07101689 -0.59255627  0.02466855  0.01483317]
gradients["dx"].shape = (3, 10, 4)
gradients["da0"][2][3] = -0.31494237512664996
gradients["da0"].shape = (5, 10)
gradients["dWax"][3][1] = 11.264104496527777
gradients["dWax"].shape = (5, 3)
gradients["dWaa"][1][2] = 2.303333126579893
gradients["dWaa"].shape = (5, 5)
gradients["dba"][4] = [-0.74747722]
gradients["dba"].shape = (5, 1)

```

Gbr. 23. Hasil RNN backward

3.1.7 LSTM Cell Backward

Pada LSTM Backward dilakukan pengimplementasian terhadap formula berikut:

$$\begin{aligned}
 d\Gamma_o^{(t)} &= da_{next} * \tanh(c_{next}) * \Gamma_o^{(t)} * (1 - \Gamma_o^{(t)}) \\
 dc^{(t)} &= dc_{next} * \Gamma_i^{(t)} + \Gamma_o^{(t)}(1 - \tanh(c_{next})^2) * i_t * da_{next} * c^{(t)} * (1 - \tanh(c)^2) \\
 d\Gamma_u^{(t)} &= dc_{next} * c^{(t)} + \Gamma_o^{(t)}(1 - \tanh(c_{next})^2) * c^{(t)} * da_{next} * \Gamma_u^{(t)} * (1 - \Gamma_u^{(t)}) \\
 d\Gamma_f^{(t)} &= dc_{next} * \tilde{c}_{prev} + \Gamma_o^{(t)}(1 - \tanh(c_{next})^2) * c_{prev} * da_{next} * \Gamma_f^{(t)} * (1 - \Gamma_f^{(t)}) \\
 dW_f &= d\Gamma_f^{(t)} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \\
 dW_u &= d\Gamma_u^{(t)} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \\
 dW_c &= dc^{(t)} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \\
 dW_o &= d\Gamma_o^{(t)} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \\
 da_{prev} &= W_f^T * d\Gamma_f^{(t)} + W_u^T * d\Gamma_u^{(t)} + W_c^T * dc^{(t)} + W_o^T * d\Gamma_o^{(t)} \\
 dc_{prev} &= dc_{next} \Gamma_f^{(t)} + \Gamma_o^{(t)} * (1 - \tanh(c_{next})^2) * \Gamma_f^{(t)} * da_{next} \\
 dx^{(t)} &= W_f^T * d\Gamma_f^{(t)} + W_u^T * d\Gamma_u^{(t)} + W_c^T * dc_t + W_o^T * d\Gamma_o^{(t)}
 \end{aligned}$$

Gbr. 24. Formula LSTM backward

Sehingga, jika diimplementasikan kedalam code akan ada parameter-parameter seperti dxt yang merupakan gradient dari input data pada timestep t, da_prev yang merupakan gradient dari previous hidden state, dc_prev yang merupakan gradient dari previous memory state, dWf merupakan weight dari forget gate, dWi merupakan weight dari update gate, dWc merupakan weight dari memory gate, dWo merupakan weight dari output gate.

```

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(parameters['Wc'].T,dcct)
    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":dbc, "dWo":dWo, "dbo":dbo}
    return gradients

```

Gbr. 25. LSTM backward

Untuk mengetahui masing-masing hasil dari formula LSTM Backward, maka fungsi tersebut dijalankan dan diperoleh hasil sebagai 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)

```

Gbr. 26. LSTM backward tes

Output dari proses tersebut adalah:

```
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] -0.1479548381644968
gradients["dWf"].shape (5, 8)
gradients["dWi"][[1][2] 1.0574980552259903
gradients["dWi"].shape (5, 8)
gradients["dWc"][[3][1] 2.304562163687667
gradients["dWc"].shape (5, 8)
gradients["dWo"][[1][2] 0.3313115952892109
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)
```

Gbr. 23. Hasil LSTM backward tes

3.2 Implementasi RNN untuk Studi Kasus Dinosaurs Land

Inti utama dari studi kasus ini adalah mengenerate nama baru untuk dinosaurus, belajar dari karakteristik nama-nama dinosaurus yang telah ada sebelumnya. Untuk membuat nama baru tersebut, maka akan dibangun character level languange model, dimana algoritma akan belajar pola-pola nama kemudian secara random membuat nama baru. Dataset yang digunakan adalah dino.txt.

Program ini akan tetap menggunakan fungsi-fungsi pada *rnn_util* terutama *rnn_forward* dan *rnn_backward* seperti yang telah diimplementasikan sebelumnya. Dataset di dino.txt tersebut dalam proses ini akan disimpan per karakter.

Struktur model yang dibangun dalam pengimplementasian RNN dalam studi kasus Dinosaurs Land ini antara lain:

- i. Inisialisasi parameter
- ii. Menjalankan *optimization loop*
 - Forward propagation untuk menghitung loss function.
 - Backward propagation untuk menghitung gradient dengan memperhatikan loss function
 - Gradient clipping untuk menghindari exploding gradient (nilai yang sangat besar).

- Menggunakan gradient dan mengupdate parameter dengan gradient descent

iii. Kembali pada parameter yang dipelajari.

3.2.1 Import Library dan Load Data

```
import numpy as np
from utils import *
import random

data = open('dinos.txt', 'r').read()
data= data.lower()
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print('There are %d total characters and %d unique characters in your data.' % (data_size, vocab_size))

There are 19909 total characters and 27 unique characters in your data.
```

Gbr. 24. Impor library dan load data

Dataset dino.txt berisi nama-nama spesies dinosaurus yang pernah ada. Berikut contoh isi dari dataset dino.txt yang telah di lowecase.

```
'aachenosaurus\naardonyx\nabdallahsaurus\nabelisaurus\nabriktosaurus\nabrosaurus\nabydosaurus\nacanthopholis\nachelousaurus\nacheroraptor\nachillesaurus\nachillobator\nacristsavus\nacrocanthosaurus\nacrotholus\nactiosaurus\nadamantisaurus\nadasaurus\nadelolophus\nadeopapposaurus\naegyptosaurus\naeolosaurus\naepisaurus\naepyornithomimus\naerosteon\naetonyxafromimus\nafrovocator\nagathumas\naggiosaurus\nagilisaurus\nagnosphitys\nagrosaurus\nagujaceratops\nagustinia\nnahshilepelta\nnairakoraptor\nnajancingenia\nnjkaceratops\nnalamosaurus\nnalaskacephale\\nalbaphosaurus\\nalbertaceratops\\nalbertadromeus\\nalbertavenator\\nalbertonykus\\nalbertosaurus\\nalbinykus\\nalbisaurus\\nalcovasaurus\\nalectrosaurus\\naletopelta\\nalgasaurus\\naliaramus\\naliwali\\nalloaurus\\nalmas\\nlnashetri\\nalcodon\\naltirhinus\\naltispinax\\navezsaurus\\nawalkeria'
```

Gbr. 25. Data dino.txt

Data ini kemudian dilakukan pemecahan per karakter. Total karakter yang diperoleh dari dataset ini sebesar 19.909 karakter, dengan 27 karakter unik (26 karakter a-z, dan 1 karakter *newline*). Setelah itu karakter tersebut diberi indeks untuk membantu mendefinisikan karakter yang muncul pada output nantinya (probability distribution output pada softmax layer).

```
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)

{0: '\n', 1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o', 16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't', 21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y', 26: 'z'}
```

Gbr. 26. Indeks karakter

3.2.2 Pembangunan Model

Model yang dibangun adalah model RNN forward, seperti yang digambarkan pada Gbr.8. ditambah dengan *gradient clipping* (untuk menghindari exploding gradient) dan *sampling* (teknik untuk mengenerate karakter).

Gradient clipping dipanggil dalam *optimization loop*. Pada optimization loop biasanya terdiri dari proses forward pass, cost computation, backward pass, dan parameter update. Disini sebelum parameter update ditambahkan gradient clipping, sehingga prosesnya menjadi: forward pass → cost computation → backward pass → gradient clipping → parameter update.

Ada berbagai cara clipping gradient, pada tugas ini digunakan simple element-wise, dimana setiap elemen gradient di clip pada range [-5,5]. *Clipping* yang dimaksud adalah jika ada komponen dari vektor gradient yang nilainya lebih dari 5, maka akan di-set dengan 5, begitupun sebaliknya jika ada komponen dari vektor gradient yang nilainya kurang dari -5, maka akan di-set dengan -5.

```
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
```

Gbr. 27. Fungsi Gradient Clipping

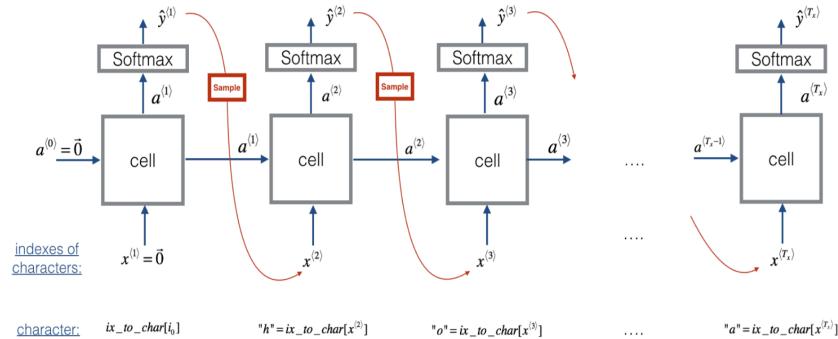
Fungsi clip tersebut dipanggil di fungsi optimize yaitu fungsi untuk menghitung gradient.

```
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]
```

Gbr. 28. Fungsi Optimize

Proses sampling dilakukan sebelum input, terdiri dari 3 step sebagai berikut:



Gbr. 29. Sampling

- Inisialisasi semua input dengan vektor zero.
- Setelah menjalani 1 step forward propagation, maka akan diperoleh $a^{(1)}$ dan $\hat{y}^{(1)}$, lalu dilakukan proses sebagai berikut:

$$a^{(t+1)} = \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t)} + b)$$

$$z^{(t+1)} = W_{ya}a^{(t+1)} + b_y$$

$$\hat{y}^{(t+1)} = softmax(z^{(t+1)})$$

- Ambil indeks karakter selanjutnya berdasarkan probablity distribution yang didefinisikan $\hat{y}^{(t+1)}$ yaitu dengan menggunakan fungsi `np.random.choices`
- Overwrite variabel x (input berikutnya) dengan hasil dari proses *sampling* ini.

```

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)
        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

```

Gbr. 30. Fungsi sample

Sehingga untuk pemodelan RNN pada studi kasus Dinosaurs Land ini dapat dituliskan sebagai berikut.

```

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

```

Gbr. 31. Fungsi model

Untuk menjalankan program, hanya perlu untuk memanggil fungsi model dengan parameter-parameter inputnya.

```
parameters = model(data, ix_to_char, char_to_ix)
```

Gbr. 32. Memanggil fungsi model

Pada model ini, iterasi dilakukan sebanyak 34.000 kali. Awal iterasi, huruf-huruf masih tergenerate secara acak dan belum dapat terbaca. Pada pertengahan iterasi, nama dinosaurus sudah mulai memiliki arti. Hingga pada iterasi terakhir, nama-nama dinosaurus yang baru sudah tergenerate dengan baik.

```
iteration : 32000, Loss :22.374718
```

```
Matruraptor
Incaaeratidbshuchesaurus
Ivusanon
Madalisaurus
Yusidon
Dabarnaiantatasaurus
Usilonis
```

```
iteration : 34000, Loss :22.488935
```

```
Mawprodon
Inga
Iustrangosaurus
Macaersaurus
Yusiamamiangosaurus
Edahosaurus
```

Gbr. 33. Output

3.3 Implementasi untuk Studi Kasus Pasar Modal

Pasar modal adalah tempat dimana saham dari sebuah perusahaan diperjualbelikan. Pasar modal memiliki karakteristik yakni fluktuasinya yang tinggi dan data *time-series* yang non linear. Data *time-series* adalah sekumpulan data yang yang diperoleh dari waktu ke waktu dari suatu aktivitas. Berdasarkan karakteristiknya ini, peramalan pasar modal memiliki resiko yang jauh lebih tinggi dibandingkan dengan sektor lainnya. Namun, pada kenyataannya banyak investor yang tertarik pada penelitian tentang prediksi harga saham ini. Untuk bisa berinvestasi dengan baik, tentunya investor perlu untuk mengetahui situasi pasar saham dimasa yang akan datang. Semakin baik dan efektif sistem untuk memprediksi harga saham ini, akan sangat membantu para investor trader, serta para analis pasar.

Banyak metode yang telah digunakan untuk membangun model untuk memprediksi pasar modal ini. Pada tugas ini, dilakukan pemodelan dengan

menggunakan LSTM yang merupakan metode yang saat ini sangat diandalkan untuk permasalahan data-data sekuens atau time-series.

Untuk pengimplementasiannya, digunakan bahasa pemrograman python dengan library Keras yang didalamnya sudah termasuk library tentang LSTM yang dapat langsung digunakan.

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

Gbr. 34. Import library Keras

Data yang digunakan adalah data Stock Market yaitu NSE-TATAGLOBAL yang merupakan data harian periode 21-07-2010 sampai 28-09-2018 dengan total jumlah data yaitu 2035 data. Dataset ini terdiri dari informasi-informasi seperti yang ditampilkan pada gambar berikut.

dataset_train.head()									
	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	

Gbr. 35. Contoh isi data

Kolom “Open” menunjukkan harga awal, kolom “Close” menunjukkan harga akhir dari saham pada hari itu, “High” dan “Low” menunjukkan harga tertinggi dan terendah pada hari tertentu. Namun, tidak semua informasi dalam data tersebut dimasukkan ke dalam training model LSTM. Untuk melakukan prediksi, hanya dibutuhkan data pada kolom “Open” saja yang menginterpretasikan *Stock Price*.

Tahap berikutnya adalah melakukan preprocessing terhadap data terlebih dahulu sebelum masuk kedalam model LSTM. Preprocessing yang dilakukan adalah Scalling, menggunakan *MinMaxScaler* yang ada dalam library *sklearn.preprocessing*. Nilai Stock Price diskalakan ke dalam nilai 0 sampai 1.

Pendefinisian data training menjadi data input (X) dan data prediksi (Y) juga

dilakukan. Pada tutorial tersebut data input saling overlap per 60 data (60 timestep) dan data prediksinya adalah 1 data setelah data input. Misalnya: X1 adalah data ke-0 sampai 59 , maka Y1 adalah data ke 60. Jika X2 adalah data ke-2 sampai 60, maka Y1 adalah data ke 61, dan seterusnya. Selanjutnya dilakukan reshape.

```
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))
```

Gbr. 35. Data training

Proses berikutnya adalah proses inti, yakni membangun model untuk LSTM dengan menggunakan library-library pada Keras yang telah dijelaskan sebelumnya.

```
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)
```

Gbr. 35. Training LSTM dengan Keras

Terdapat beberapa tahap modul dalam pembangunan model dengan library Keras, antara lain:

- *Sequential* digunakan untuk inisialisasi neural network.
- *Dense* digunakan untuk menambah kepadatan Neural Network layer.
- *LSTM* digunakan untuk menambahkan LSTM layer.
- *Dropout* digunakan untuk menambahkan dropout layer yang dapat mencegah overfitting.

Untuk parameteranya, pada LSTM digunakan 50 unit neuron. `return_sequence = True` dan `dropout layer = 0.2` yang berarti 20% dari layer akan di drop, serta `dense = 1` unit output. Sedangkan optimizer yang digunakan pada model ini adalah `adam`, dimana `adam` telah terkenal sebagai optimizer yang dapat diandalkan. Untuk perhitungan loss digunakan `mean_squared_error`.

Training dilakukan dalam 100 epoch. Hasil loss yang diperoleh saat training adalah `2.3012e-04`. Hasil ini sangat bagus, sehingga model siap untuk digunakan untuk memprediksi data baru.

Data test digunakan untuk *predicting future stock*. Dataset yang digunakan adalah `tatatest.csv` yang dengan jumlah data 15.

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

Gbr. 36. Load data tes.

Sebelum memprediksi future stock, ada hal-hal yang perlu dilakukan terlebih dahulu diantaranya:

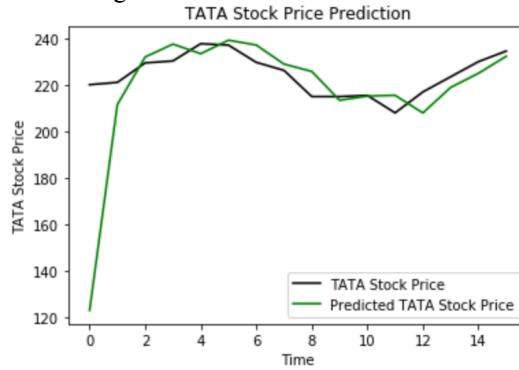
- Merge training set dan tes set pada 0 axis
- Mengatur `timestep = 60`
- Menggunakan `minmaxscaler` untuk data baru
- Reshape dataset seperti yang dilakukan pada tahap sebelumnya.

Setelah membuat prediksi, dilakukan *inverse_transform* untuk mengembalikan format data stock seperti sedia kala.

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

Gbr. 37. Tahap prediksi

Hasil prediksi dapat dilihat dari grafik berikut.



Gbr. 38. Grafik hasil prediksi

Berdasarkan Gbr.38, dapat diketahui bahwa grafik hasil prediksi sejalan dengan real data. Ketika grafik real data menunjukkan penurunan, grafik prediksi juga menunjukkan penurunan. Sehingga dapat disimpulkan bahwa prediksi terhadap stock price sudah tepat.

3.4 Implementasi untuk Studi Kasus Pasar Modal dengan Data Baru

Program yang telah dibuat pada poin 3.3 digunakan kembali untuk mencoba dataset baru. Disini saya menggunakan dataset *stock price* Hang Seng Index dari yahoo finance (<https://finance.yahoo.com/quote/%5EHSI/history?p=%5EHSI>) dari periode 8 Desember 2017 hingga 8 Juni 2019 untuk data training dan periode 8 Juli 2019 hingga 8 Oktober 2019 untuk data testing. Data training berjumlah 458 sedangkan data testing berjumlah 34.

```
dataset_train = pd.read_csv('^HSItrain.csv')
training_set = dataset_train.iloc[:,1:2].values

#load data test
dataset_test = pd.read_csv('^HSItest.csv')
real_stock_price = dataset_test.iloc[:,1:2].values
```

Gbr. 39. Load data baru (training dan testing)

Perlu dilakukan penyesuaian untuk proses looping pada beberapa tempat, seperti pada bagian berikut, perlu disesuaikan dengan jumlah data. Timestep yang digunakan tetap 60 timestep seperti pada poin 3.2.

```

x_train = []
y_train = []
for i in range(60,458):
    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))

```

Gbr. 40. Range training loop

Pada bagian training, loop timestep diset pada range 60 sampai 458 sesuai dengan jumlah data training saat ini. Sedangkan pada data testing diubah menjadi range 60 sampai 94.

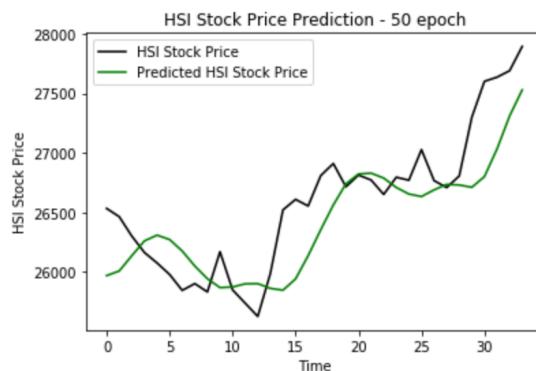
```

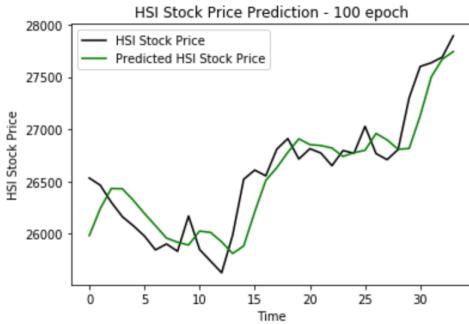
dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']),axis = 0)
inputs = dataset_total[len(dataset_total)-len(dataset_test)-60: ].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
x_test = []
for i in range(60,94):
    x_test.append(inputs[i-60:i,0])
x_test = np.array(x_test)
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))
predicted_stock_price = regressor.predict(x_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)

```

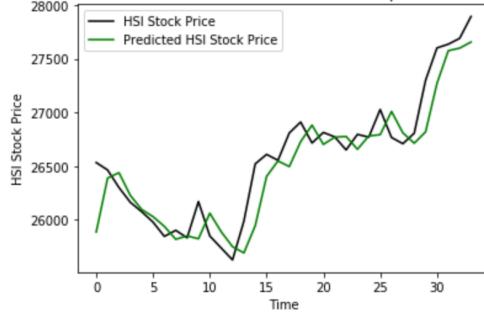
Gbr. 41. Range test loop

Untuk tahap selanjutnya sama seperti pada poin 3.2. Untuk parameter model, saya mencoba untuk mengubah nilai epoch menjadi 50, 100, 150, dan 200. Dari percobaan tersebut, diperoleh hasil sebagai berikut:

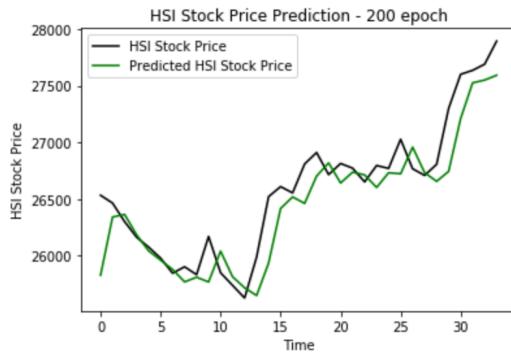
**Gbr. 42.** Prediksi Stock Price dengan 50 epoch



Gbr. 43. Prediksi Stock Price dengan 100 epoch
HSI Stock Price Prediction - 100 epoch



Gbr. 44. Prediksi Stock Price dengan 150 epoch
HSI Stock Price Prediction - 150 epoch



Gbr. 45. Prediksi Stock Price dengan 200 epoch
HSI Stock Price Prediction - 200 epoch

Pada eksperimen dengan hanya menggunakan jumlah epoch 50, hasil prediksi kurang bisa mengikuti grafik pada real data. Perbedaannya masih sangat jauh. Hasil prediksi kurang bagus. Ketika jumlah epoch ditambah menjadi 100 dan 150, bentuk grafik semakin bagus dan sudah mulai mengikuti bentuk grafik dari real data. Pada hasil menggunakan epoch 200, grafik sudah semakin baik, bahkan sudah ada yang tepat memprediksi seperti pada real data. Ketika grafik pada real data naik, grafik prediksi juga naik, dan sebaliknya. Hal ini menandakan bahwa

hasil prediksi sudah sesuai dengan yang diharapkan.

4 Kesimpulan

Berdasarkan empat tahap yang telah dikerjakan yakni RNN step-by-step, implementasi untuk permasalahan penamaan spesies baru pada studi kasus *Dinosaurs Land*, serta dua *task* tentang prediksi *stock market* pada Pasar Modal, secara keseluruhan tujuan dari tugas ini sudah dapat dicapai. RNN dan LSTM step-by-step dapat diimplementasikan dan memperoleh hasil sesuai dengan target output yang diharapkan. RNN juga berhasil menggenerate nama spesies baru untuk dinosaurus pada *Dinosaurs Land*. LSTM handal untuk memprediksi *stock price* pada pasar modal, serta ditambah kesimpulan dari hasil eksperiment saat penerapan LSTM kedalam dataset baru yakni mengenai perubahan parameter jumlah epoch akan mempergaruh hasil prediksi. Semakin besar epoch yang digunakan maka akan semakin presisi prediksi yang dihasilkan.

References

1. Murtaza Ronndiwala, Harshal Patel, Shraddha Varma: Predicting Stock Price Using LSTM. International Journal of Science and Research (IJSR) (2015)
2. Hiransha M, Gopalakrishnan E.A, Vijai Khrishna Menon, Soman K.P: NSE Stock Market Prediction Using Deep Learning Models. International Conference on Computational Intelligence and Data Science. Elsevier (2018)
3. Alex Sherstinsky: Fundamentals of Recurrent Neural (RNN) and Long Short Term Memory (LSTM) Network. Arxiv (2018)
4. RNN-basic [slide kuliah]
5. Pasar Modal Basic [slide kuliah]