



Tugas 2

# DEEP LEARNING

Recurrent Neural Networks (RNN)

DAN

Long Short-Term Memory (LSTM)

[unsia.ac.id](http://unsia.ac.id)



YAYASAN MEMAJUKAN ILMU DAN KEBUDAYAAN

**UNIVERSITAS SIBER ASIA**

Kampus Menara, Jl. RM. Harsono, Ragunan - Jakarta Selatan. Daerah Khusus Ibukota Jakarta 12550.

Telp. (+6221) 27806189. [asiacyberuni@acu.ac.id](mailto:asiacyberuni@acu.ac.id). [www.unsia.ac.id](http://www.unsia.ac.id)

Mata Kuliah	: Deep Learning
Kelas	: IT501
Prodi	: PJJ Informatika
Nama Mahasiswa	: Hendro Gunawan
NIM	: 200401072103
Dosen	: Catur Nogroho, S.Kom., M.Kom.

[unsia.ac.id](http://unsia.ac.id)



# Kata Pengantar

Dalam pertemuan kali ini saya akan membahas tentang mata kuliah Deep Learning terutama mengenai *Long Short-Term Memory* (LSTM) yang telah dijelaskan oleh dosen saya yaitu Bapak Catur Nugroho S.Kom., M.Kom. Dimana pertemuan tersebut merupakan salah satu bab dalam pertemuan kesatu sampai dengan pertemuan ke tigabelas. Tepatnya pada pertemuan ketujuh yang membahas tentang *Recurrent Neural Networks* (RNNs). Pada pertemuan ketujuh tersebut terdapat sub bab yang membahas tentang LSTM, sebelumnya kita telah banyak melakukan diskusi dengan teman-teman untuk menyelesaikan masalah-masalah yang dihadapi dalam menyelesaikan kasus-kasus pada *Deep Learning* (DL) dan juga kita telah melakukan praktikum prediksi LSTM pada pertemuan ketujuh dan praktikum RNN pada pertemuan kesepuluh serta praktikum CNN pada pertemuan ketiga belas. Disini kita akan melakukan prediksi menggunakan LSTM dan optimasi ADAM untuk memperkirakan harga emiten saham Bank Rakyat Indonesia dengan range waktu antara 20 januari 2019 – 15 juni 2023.

Saya berharap, hal ini dapat meningkatkan pengetahuan kita lebih dalam lagi mengenai apa itu *Long-Short Term Memory* dan implementasinya dalam kehidupan sehari-hari. Semoga dengan adanya makalah ini kita bisa menambah wawasan tentang mata kuliah *Deep Learning*. Koreksi, komentar, kritik, dan saran dapat disampaikan melalui surel (*email*) ke alamat: [hendro.gnwn@gmail.com](mailto:hendro.gnwn@gmail.com), [hendro.gnwn@ymail.com](mailto:hendro.gnwn@ymail.com), atau [hendro.gnwn@outlook.com](mailto:hendro.gnwn@outlook.com). Semoga mendapat manfaat dari makalah ini. Kami mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Catur Nugroho S.Kom., M.Kom yang telah memberikan bimbingannya dalam mempelajari Deep Learning pada semester lima ini.

Gresik, 14 Maret 2023

Penulis

(Hendro Gunawan)



# Daftar Isi

Kata Pengantar.....	i
Daftar Isi.....	ii
BAB I PENDAHULUAN.....	1
BAB II TINJAUAN PUSTAKA.....	3
BAB III PEMBAHASAN.....	8
BAB IV PENUTUP.....	44
UCAPAN TERIMA KASIH.....	44
DAFTAR PUSTAKA.....	45
Biodata Penulis.....	45
Link File.....	46
Tabel Nilai.....	47



## BAB I PENDAHULUAN

*Recurrent Neural Networks* (RNN) memiliki arsitektur dengan sedikit *layer* (lapisan), namun susunanya rumit dengan adanya aliran koneksi mundur. Dengan sejumlah koneksi mundur ini, RNN sesuai untuk data sekuens (deretan atau barisan), seperti data time series (deretan data-data historis), teks(sekuens kalimat, kata, subkata, atau huruf), suara(barisan amplitudo, frekuensi, spectrum, cepstrum), atau video(deretan gambar) yang disebut sebagai *sequence of image frames*. Walaupun hanya berisi sedikit lapisan, RNN bisa dianggap DL karena ada koneksi mundur (*looping*). Dahulu, para ahli kesulitan melatih RNN karena sering terjebak pada *vanishing gradient*, tetapi teknik pembelajaran DL mampu mengatasi masalah *vanishing gradient* tersebut.

### A. Latar Belakang Masalah

Pernahkah Anda memperhatikan bagaimana proses seorang manusia dalam berpikir? Misalkan Anda membaca suatu cerita dan memproses informasi di dalam cerita tersebut. Informasi yang Anda dapatkan dari cerita tersebut tidak hanya berasal dari satu kata atau bahkan satu kalimat. Tapi informasi tersebut berasal dari serangkaian kalimat yang membentuk satu cerita utuh. Pada kenyataannya, kita tidak dapat memproses informasi yang utuh hanya dari satu bagian teks saja. Namun, kita harus membaca setiap rangkaian teks yang ada di dalam cerita tersebut untuk memperoleh informasi.

Contoh lain pada saat kita menonton sebuah film. Informasi dari film tersebut kita dapatkan utuh jika kita menyaksikan setiap detik dari film tersebut. Atau dalam sebuah percakapan, informasi kita dapatkan dari serangkaian suara dari pembicara dari detik demi detik. Ketiga contoh tersebut merupakan contoh data yang bersifat sekuensial. Antara data sebelumnya dan data setelahnya terdapat keterhubungan yang rangkaiannya membentuk sebuah informasi.

### B. Rumusan Masalah

Dengan melihat latar belakang masalah yang telah dikemukakan maka, beberapa masalah yang dapat penulis rumuskan dan akan dibahas dalam laporan ini adalah:

1. Apakah ide dasar dan motivasi Recurrent Neural Network itu?
2. Apakah arsitektur Recurrent Neural Network itu?
3. Bagaimana formulasi perhitungan Recurrent Neural Network ?
4. Apakah algoritma pembelajaran yang digunakan pada Recurrent Neural Network?
5. Apakah Recurrent Neural Network data sekuens itu?
6. Apakah Short-Term Memory itu?



### **C. Tujuan Dan Manfaat**

Tujuan dan manfaat penelitian yang ingin dicapai adalah:

1. Mengetahui bagaimana ide dasar dan motivasi Recurrent Neural Network?
2. Mengetahui bagaimana arsitektur Recurrent Neural Network?
3. Mengetahui bagaimana formulasi perhitungan menggunakan Recurrent Neural Network?
4. Mengetahui bagaimana algoritma pembelajaran yang digunakan pada Recurrent Neural Network?
5. Mengetahui bagaimana Recurrent Neural Network untuk data sekuens?
6. Mengetahui bagaimana arsitektur Long Short-Term Memory?

### **D. Metode penelitian**

Metode yang digunakan oleh penulis dalam menyusun makalah ini yaitu dengan mengumpulkan informasi dari berbagai sumber buku dan browsing di internet menggunakan aplikasi *Edge*, *Microsoft Bing*, *Google Search*, dan *Youtube*. Tools dan kebutuhan perangkat lunak dari metode yang digunakan dalam penelitian ini yaitu menggunakan Anaconda3 (64-bit), Jupyter Notebook, Python 3.5 dengan tensorflow versi 2.12.0. Portable Computer (PC) yang digunakan yaitu CPU Intel® Core™ i9-12900KF LGA 1700 yang berjalan pada 3.19 GHz dan GPU NVIDIA G-Force GTX 1650 OC Edition 4GB DUAL, dengan RAM terinstal 24GB, sistem operasi 64-bit, dengan spesifikasi windows 11 Pro Insider Preview. Versi 22H2, Build OS 23493.1000.

### **E. Estetika Penulisan**

Dalam penyusunan makalah ini terdiri dari hal-hal yang saling berkaitan antara bab I sampai dengan bab IV yang memuat beberapa isi sebagai berikut:

#### **BAB I Pendahuluan**

Membahas tentang latar belakang masalah, rumusan masalah, tujuan penulisan dan sistematika penulisan.

#### **BAB II Tinjauan Pustaka**

Membahas tinjauan tentang ide dasar dan motivasi RNN, tinjauan tentang arsitektur *Recurrent Neural Network*, tinjauan tentang formulasi *Recurrent Neural Network*, tinjauan tentang algoritma pembelajaran *Recurrent Neural Network*, tinjauan tentang *Recurrent Neural Network* untuk data sekuens, dan tinjauan tentang *Long Short-Term Memory*.

#### **BAB III Pembahasan**

Membahas tentang apakah pengertian *Artificial Neural Network*?, apakah pengertian *Convolution Neural Network*?, apakah pengertian *Deep Unsupervised Learning*?

#### **BAB IV Penutup**



Membahas tentang kesimpulan, saran, ucapan terima kasih dan daftar pustaka.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 Tinjauan Tentang Ide Dasar dan Motivasi Recurrent Neural Network

Ide dasar dari *Recurrent Neural Network* (RNN) adalah kemampuan untuk memproses data berurutan atau data yang memiliki ketergantungan temporal. RNN dirancang khusus untuk mengatasi masalah di mana input dan output memiliki hubungan sekuensial, seperti dalam teks, suara, atau data waktu.

Dalam jaringan saraf tradisional, setiap input dianggap independen satu sama lain. Namun, dalam banyak kasus, informasi kontekstual dari input sebelumnya diperlukan untuk memahami input saat ini. RNN dirancang untuk mengatasi masalah ini dengan mengenali dan memanfaatkan hubungan sekuensial dalam data.

Motivasi utama di balik pengembangan RNN adalah untuk memiliki arsitektur jaringan saraf yang dapat mengenali pola sekuensial dan mempertahankan informasi kontekstual sepanjang waktu. Dalam RNN, informasi dari input sebelumnya dipertahankan dalam unit memori internal yang disebut "*state*" atau "*hidden state*". Informasi ini kemudian digunakan sebagai konteks untuk memproses input saat ini dan mempengaruhi output yang dihasilkan.

Dengan demikian, RNN memungkinkan pemodelan data yang memiliki hubungan sekuensial dan memperhatikan konteks sebelumnya. Ini berguna dalam berbagai tugas seperti pemrosesan bahasa alami, pengenalan ucapan, terjemahan mesin, prediksi deret waktu, dan banyak lagi.

RNN memiliki kemampuan untuk "mengingat" informasi kontekstual dari sekuensi input sebelumnya dan "memberikan" informasi ini ke sekuensi input berikutnya. Ini memungkinkan RNN untuk mengambil keputusan yang lebih baik berdasarkan sejarah input yang telah diproses.

Meskipun RNN memiliki kelebihan dalam memodelkan data sekuensial, namun ada tantangan dalam melatihnya, seperti masalah *vanish gradient* dan *explode gradient* yang dapat mempengaruhi kemampuan jaringan untuk menjaga dan memanfaatkan informasi kontekstual sepanjang waktu. Ini telah mendorong pengembangan variasi RNN yang lebih maju, seperti *Long Short-Term Memory* (LSTM) dan *Gated Recurrent Unit* (GRU), yang dirancang untuk mengatasi tantangan ini.

Dengan demikian, ide dasar RNN adalah memanfaatkan hubungan sekuensial dalam data dan mempertahankan informasi kontekstual untuk memproses data yang memiliki



ketergantungan temporal. Ini memberikan jaringan saraf kemampuan untuk memodelkan pola sekuensial dan melakukan tugas-tugas yang melibatkan data yang diurutkan.

## 2.2 Tinjauan Tentang Arsitektur Recurrent Neural Network

Arsitektur *Recurrent Neural Network* (RNN) adalah bentuk arsitektur jaringan saraf yang dirancang khusus untuk memproses data sekuensial atau data yang memiliki ketergantungan temporal. RNN memiliki struktur yang rekursif, di mana output pada waktu sebelumnya menjadi masukan pada waktu berikutnya.

Dalam arsitektur RNN, ada beberapa jenis layer yang membentuk struktur dasarnya:

*Layer Input*: Layer input pada RNN menerima input sekuensial, seperti kata-kata dalam kalimat atau deret waktu. Setiap elemen dalam sekuensi dianggap sebagai input pada waktu yang berbeda.

*Layer Rekurensi*: Layer rekurensi adalah komponen inti dalam arsitektur RNN. Pada setiap waktu, layer ini menerima input saat ini dan juga output dari waktu sebelumnya sebagai masukan. Informasi dari waktu sebelumnya disimpan dalam unit memori internal yang disebut "*state*" atau "*hidden state*". Hal ini memungkinkan layer rekurensi untuk mempertahankan informasi kontekstual dan memperhitungkan hubungan sekuensial dalam data.

*Layer Output*: Layer output pada RNN menghasilkan output dari layer rekurensi pada setiap waktu. Output ini dapat digunakan untuk tugas-tugas seperti klasifikasi, regresi, atau prediksi sekuensial berikutnya. Fungsi aktivasi yang sesuai digunakan pada layer output, tergantung pada jenis tugas yang dihadapi.

Selain komponen dasar tersebut, ada juga variasi arsitektur RNN yang lebih kompleks, seperti *Long Short-Term Memory* (LSTM) dan *Gated Recurrent Unit* (GRU). Arsitektur LSTM dan GRU telah dikembangkan untuk mengatasi masalah *vanish gradient* dan *explode gradient* yang dapat terjadi saat melatih RNN tradisional. Kedua arsitektur tersebut menggunakan mekanisme *gate* untuk mengatur aliran informasi dalam jaringan dan mempertahankan informasi jangka panjang.

Secara umum, arsitektur RNN memungkinkan jaringan saraf untuk mengatasi data sekuensial dan memodelkan hubungan temporal dalam data tersebut. Ini membuat RNN menjadi pilihan yang kuat untuk tugas-tugas seperti pemrosesan bahasa alami, pengenalan ucapan, prediksi deret waktu, dan lainnya.

## 2.3 Tinjauan Tentang Formulasi Recurrent Neural Network

Formulasi matematis dari *Recurrent Neural Network* (RNN) adalah sebagai berikut:



Pada setiap waktu  $t$ , RNN menerima input  $x_t$  dan *state* dari waktu sebelumnya  $h_{t-1}$ , kemudian menghasilkan output  $o_t$  dan *state* baru  $h_t$ . Secara formal, formulasi RNN dapat dinyatakan sebagai berikut:

$$h_t = f(W_{xh} * x_t + W_{hh} * h_{t-1} + b_h) \quad (2-1)$$

$$o_t = g(W_{hy} * h_t + b_y) \quad (2-2)$$

Di sini:

$x_t$  adalah input pada waktu  $t$ .

$h_t$  adalah *state* pada waktu  $t$ , yang menyimpan informasi kontekstual dari waktu sebelumnya.

$W_{xh}$  adalah matriks bobot yang menghubungkan input  $x_t$  dengan *state*  $h_t$ .

$W_{hh}$  adalah matriks bobot yang menghubungkan *state*  $h_{t-1}$  dengan *state*  $h_t$ .

$b_h$  adalah bias untuk *state*  $h_t$ .

$f$  adalah fungsi aktivasi yang diterapkan pada input linier.

$o_t$  adalah output pada waktu  $t$ .

$W_{hy}$  adalah matriks bobot yang menghubungkan *state*  $h_t$  dengan output  $o_t$ .

$b_y$  adalah bias untuk output  $o_t$ .

$g$  adalah fungsi aktivasi yang diterapkan pada output linier.

Fungsi aktivasi  $f$  dan  $g$  biasanya adalah fungsi non-linear seperti *sigmoid*, tangen hiperbolik (*tanh*), atau ReLU (*Rectified Linear Unit*).

Secara umum, formulasi ini memungkinkan RNN untuk memproses data sekuensial dengan mempertahankan informasi kontekstual dalam *state* dan menghasilkan output pada setiap waktu. Dengan iterasi melalui waktu, RNN dapat memodelkan hubungan sekuensial dalam data dan mempelajari pola temporal yang kompleks.

## 2.4 Tinjauan Tentang Algoritma Pembelajaran Recurrent Neural Network

Algoritma pembelajaran yang umum digunakan untuk melatih *Recurrent Neural Network* (RNN) adalah algoritma *Backpropagation Through Time* (BPTT). BPTT adalah modifikasi dari algoritma backpropagation yang digunakan untuk melatih jaringan syaraf rekursif seperti RNN.

Berikut adalah langkah-langkah umum dari algoritma BPTT:

**Inisialisasi Bobot:** Langkah pertama adalah menginisialisasi bobot dan bias dalam RNN dengan nilai acak kecil atau menggunakan metode inisialisasi yang lebih canggih seperti Xavier atau He initialization.

**Langkah Maju (*Forward Pass*):** Mulailah dengan memberikan input sekuensial ke RNN dan lakukan langkah maju untuk menghasilkan output pada setiap waktu. Selama langkah maju,





*state* dari waktu sebelumnya diumpankan ke waktu berikutnya untuk mempertahankan informasi kontekstual.

**Perhitungan Loss:** Setelah langkah maju, hitung loss antara output yang dihasilkan oleh RNN dan target yang diinginkan. Loss biasanya dihitung menggunakan metode seperti *Mean Squared Error* (MSE) atau *Cross-Entropy Loss*, tergantung pada jenis masalah yang dihadapi.

**Langkah Mundur (*Backward Pass*):** Mulai dari waktu terakhir, lakukan langkah mundur untuk menghitung gradien loss terhadap bobot dan bias dalam RNN. Gradien ini dihitung menggunakan aturan rantai (*chain rule*) dan diteruskan ke waktu sebelumnya untuk memperbarui bobot dan bias dalam RNN.

**Pembaruan Bobot:** Setelah menghitung gradien untuk semua waktu, lakukan pembaruan bobot dan bias dengan menggunakan algoritma optimasi seperti Gradient Descent atau varian lainnya. Pembaruan ini menggerakkan bobot dan bias dalam arah yang mengurangi *loss*.

**Ulangi Langkah 2-5:** Ulangi langkah-langkah 2 hingga 5 untuk setiap *batch* atau *epoche* dalam pelatihan. Dalam setiap iterasi, bobot dan bias diperbarui berdasarkan gradien loss yang dihitung dari sekuensial waktu.

**Evaluasi dan Penyesuaian:** Setelah pelatihan, evaluasi kinerja RNN menggunakan data validasi atau data uji yang tidak digunakan selama pelatihan. Jika performanya tidak memuaskan, Anda dapat menyesuaikan *hyperparameter* atau melakukan langkah-langkah pengoptimalan tambahan.

Proses ini diulang hingga RNN mencapai tingkat kinerja yang diinginkan atau sampai kriteria penghentian yang ditetapkan terpenuhi.

Algoritma BPTT dapat diterapkan dengan menggunakan metode komputasi grafik atau dengan menerapkan langkah-langkah secara manual menggunakan pustaka deep learning seperti *TensorFlow* atau *PyTorch*.

Perlu dicatat bahwa ada juga variasi dan perbaikan dari algoritma BPTT, seperti *Truncated Backpropagation Through Time* (TBPTT) yang membatasi jumlah langkah mundur yang dihitung untuk mengurangi kompleksitas perhitungan

## **2.5 Recurrent Neural Network Untuk Data Sekuens**

*Recurrent Neural Network* (RNN) sangat cocok untuk memproses dan menganalisis data sekuensial, seperti teks, deret waktu, atau suara. Dengan kemampuannya untuk mempertahankan informasi kontekstual dari waktu sebelumnya, RNN dapat mengidentifikasi pola dan hubungan temporal dalam data sekuensial.

Berikut adalah beberapa contoh penerapan RNN untuk data sekuensial:



Pemrosesan Bahasa Alami (*Natural Language Processing* - NLP): RNN dapat digunakan untuk tugas-tugas seperti pemodelan bahasa, pengenalan entitas berbasis teks, atau penerjemahan mesin. Dalam NLP, RNN dapat mempelajari hubungan antara kata-kata dalam kalimat atau urutan kata dalam dokumen untuk menghasilkan prediksi atau analisis yang relevan.

Prediksi Deret Waktu: RNN dapat digunakan untuk menganalisa dan memprediksi deret waktu, seperti harga saham, suhu, atau data sensor. Dengan memanfaatkan informasi kontekstual dari waktu sebelumnya, RNN dapat mengenali pola temporal dan melakukan prediksi yang lebih akurat.

Pengenalan Suara: RNN dapat digunakan dalam pengenalan suara untuk mengenali dan memahami ucapan manusia. Dengan memproses data suara sekuensial, RNN dapat memodelkan hubungan antara fonem atau suku kata untuk melakukan pengenalan dan transkripsi suara.

Musik: RNN dapat digunakan untuk memodelkan pola dan struktur musik. Dalam generasi musik, RNN dapat mempelajari urutan nada atau ritme dalam komposisi musik dan menghasilkan musik yang baru berdasarkan pembelajaran dari pola-pola tersebut.

Pengenalan Tulisan Tangan: RNN dapat digunakan untuk pengenalan tulisan tangan, di mana urutan gerakan pena dalam menulis dianggap sebagai data sekuensial. RNN dapat mempelajari hubungan antara gerakan pena dan karakter tulisan untuk melakukan pengenalan karakter atau pengenalan tulisan tangan secara keseluruhan.

Kelebihan utama RNN dalam memproses data sekuensial adalah kemampuannya untuk memperhatikan konteks sebelumnya dan memodelkan pola temporal dalam data. Namun, RNN juga memiliki tantangan, seperti masalah *vanish gradient* dan *explode gradient* yang dapat mempengaruhi pelatihan dan kinerja RNN. Oleh karena itu, variasi RNN seperti *Long Short-Term Memory* (LSTM) dan *Gated Recurrent Unit* (GRU) sering digunakan untuk mengatasi kendala ini dan meningkatkan kinerja RNN dalam memproses data sekuensial.

## 2.6 Tinjauan Tentang Long Short Term Memory

*Long Short-Term Memory* (LSTM) adalah salah satu varian arsitektur *Recurrent Neural Network* (RNN) yang dirancang khusus untuk mengatasi masalah memori jangka panjang dan mengatasi tantangan *vanish gradient* dalam pelatihan RNN.

LSTM menggunakan unit memori internal yang kompleks untuk mengontrol aliran informasi dalam jaringan. Unit memori ini memungkinkan LSTM untuk "mengingat" dan "melupakan" informasi kontekstual dalam sekuensial waktu yang lebih lama. Dengan demikian, LSTM



mampu mempertahankan informasi jangka panjang dan lebih efektif dalam memodelkan hubungan temporal dalam data.

Dalam arsitektur LSTM, ada beberapa komponen penting, termasuk:

*Cell State*: *Cell state* atau *state* memori adalah komponen inti dalam LSTM. Ini adalah jalur informasi kontekstual yang melewati seluruh rangkaian waktu. *Cell state* berfungsi untuk menyimpan informasi jangka panjang dan mempertahankan hubungan sekuensial dalam data.

*Forget Gate*: *Forget gate* adalah mekanisme yang mengatur bagian mana dari informasi yang harus "dilupakan" atau diabaikan dalam *cell state*. *Gate* ini memutuskan apakah informasi lama di *cell state* harus dipertahankan atau dihapus, berdasarkan input saat ini dan *state* sebelumnya.

*Input Gate*: *Input gate* mengontrol sejauh mana informasi baru harus dimasukkan ke dalam *cell state*. Ini memungkinkan LSTM untuk menyerap informasi yang relevan dari input saat ini dan mengintegrasikannya ke dalam *state* yang ada.

*Output Gate*: *Output gate* mengatur sejauh mana informasi dalam *cell state* harus diekstrak dan digunakan dalam *output*. Ini memungkinkan LSTM untuk mengambil informasi yang relevan dari *cell state* dan menghasilkan output yang sesuai pada setiap waktu.

Dengan menggunakan komponen-komponen ini, LSTM dapat mengatasi masalah *vanish gradient* yang sering terjadi pada RNN tradisional. Ini memungkinkan LSTM untuk melatih jaringan dengan sekuensi waktu yang panjang dan mempertahankan informasi kontekstual yang penting.

LSTM telah terbukti berhasil dalam berbagai tugas seperti pemrosesan bahasa alami, prediksi deret waktu, dan pengenalan suara. Kemampuannya untuk memodelkan hubungan sekuensial dan memperhitungkan informasi kontekstual jangka panjang menjadikannya pilihan yang kuat dalam memproses data sekuensial.

### **BAB III**

#### **PEMBAHASAN**

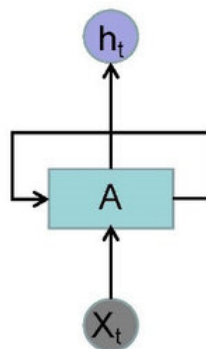
Jenis metode yang digunakan dalam makalah ini yaitu menggunakan klasifikasi *Deep Supervised Learning* (DSL), *Deep Supervised Learning* (DSL), dan *Deep Semi Supervised Learning* (DSSL) di mana *Recurrent Neural Network* (RNN) dan *Long Short-Term Memory* (LSTM) merupakan teknik pembelajaran *Deep Learning* (DL) yang termasuk ke dalam ketiga teknik model pembelajaran tersebut.

#### **3.1 Ide Dasar dan Motivasi RNN**

Algoritma-algoritma *deep learning* yang telah kita bahas pada bab sebelumnya menggunakan asumsi bahwa setiap data berdiri sendiri. Artinya, tidak ada keterkaitan antara data pada suatu

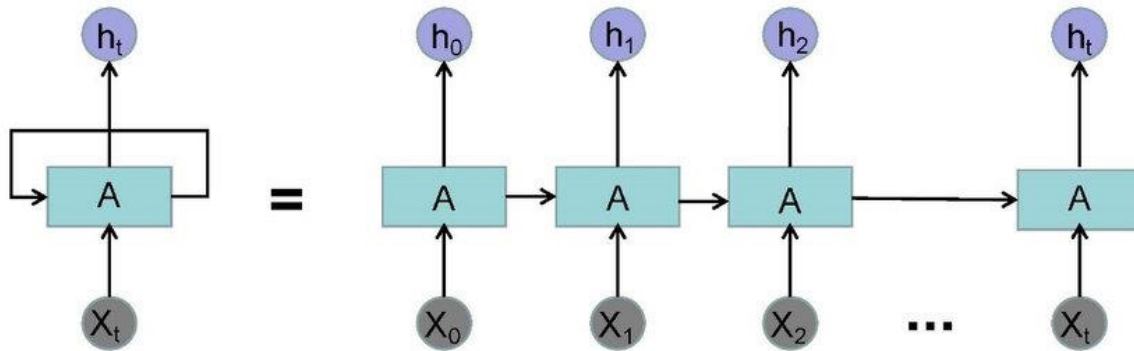
baris dengan data pada baris setelahnya. Namun bagaimana jika kita harus memproses data yang bersifat sekuensial seperti data teks, data percakapan atau data video? Hal ini akan sulit dilakukan jika menggunakan CNN atau DBN. Pada algoritma-algoritma tersebut, tidak ada mekanisme untuk memberikan informasi tambahan dari data sebelumnya pada saat pemrosesan data berikutnya dilakukan ( *Networks and Solberg*, 2018).

Untuk dapat memproses data sekuensial, algoritma *deep learning* harus menyimpan informasi dari satu waktu untuk digunakan pada pemrosesan waktu setelahnya. Hal ini dimungkinkan dengan menambahkan suatu koneksi loop dari satu neuron yang kembali ke neuron tersebut. Koneksi *loop* tersebut menyimpan informasi yang dihasilkan oleh suatu neuron pada satu saat. Informasi yang disimpan pada *loop* tersebut digunakan pada saat pemrosesan saat selanjutnya. Sehingga, pada saat pemrosesan data berikutnya, informasi yang digunakan bukan hanya dari data sebelumnya. Jika dimisalkan pada data percakapan, informasi yang diproses pada suatu saat tidak hanya data suara percakapan dari segmen tersebut, namun juga ditambahkan dengan informasi dari data suara percakapan segmen sebelumnya. Arsitektur ini disebut *Recurrent Neural Network* (RNN), seperti diilustrasikan pada gambar 1 (Caterini and Chang, 2018).



**Gambar 1. Arsitektur Recurrent Neural Network (RNN)**

Misalkan kita ingin memproses suatu data  $x$  pada segmen waktu tertentu  $t$ , kita sebut sebagai  $x_t$ . Data  $x_t$  akan diproses oleh neuron A untuk menjadi nilai output  $h_t$ . Hasil dari pemrosesan neuron A akan disimpan pada koneksi loop tersebut untuk digunakan pada pemrosesan data berikutnya  $x_{t+1}$ . Proses loop ini dapat dilakukan beberapa kali. Semakin banyak *loop* yang dilakukan, semakin informasi yang perlu disimpan. Dan semakin kaya informasi yang diproses pada tahap berikutnya. Sebuah RNN dapat dianggap sebagai *multiple copy* dari sebuah *neural network* suksesornya. Jika koneksi *loop*-nya dijabarkan, arsitektur RNN dapat diilustrasikan seperti pada gambar 2 berikut ini (Caterinei and Chang, 2018).

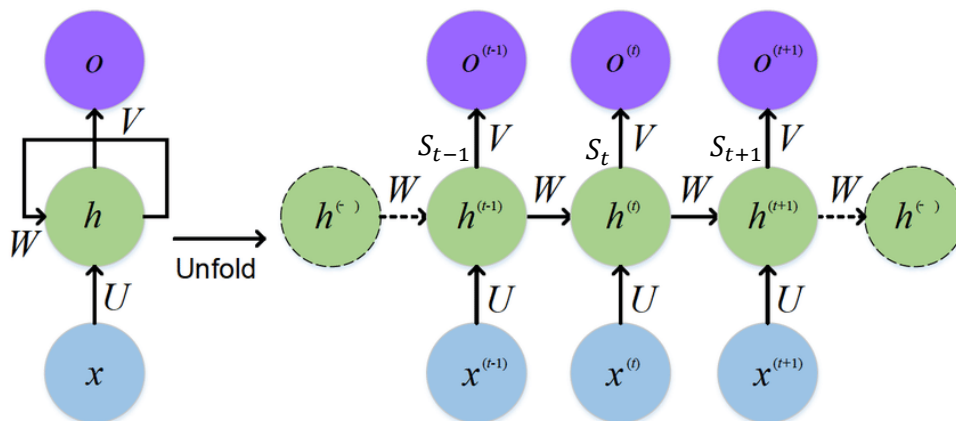


**Gambar 2. Arsitektur RNN yang dijabarkan**

Arsitektur RNN tersebut seolah-olah berbentuk rantai dengan setiap elemen dari rantai merupakan *neural network* yang memproses data pada suatu saat ( $x_t$ ) dengan tambahan informasi dari hasil pemrosesan pada saat sebelumnya ( $x_{t-1}$ ) dan informasi yang dihasilkan digunakan pada saat pemrosesan data berikutnya ( $x_{t+1}$ ). Hal ini yang menyebabkan arsitektur RNN cocok untuk digunakan pada data yang bersifat sekuensial seperti data teks, percakapan atau video.

### 3.2 Arsitektur RNN

Dengan menggunakan formulasi matematis, arsitektur *Recurrent Neural Network* diilustrasikan pada gambar 3.

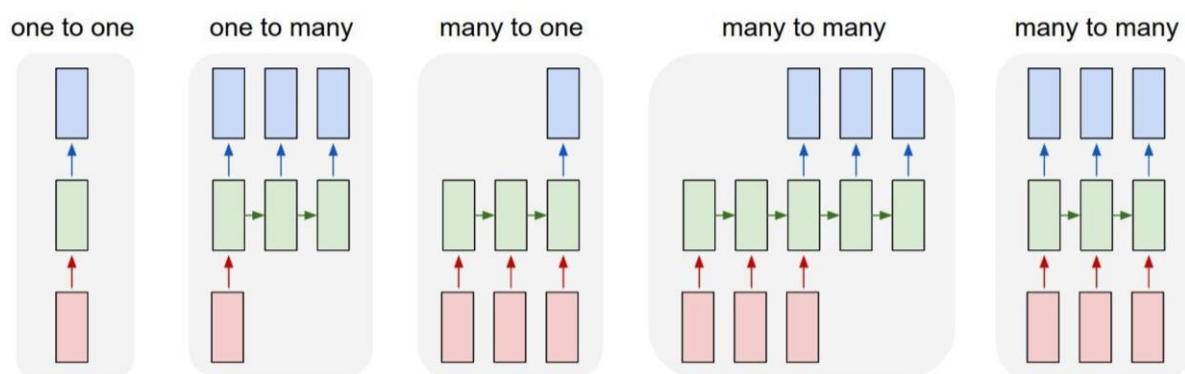


**Gambar 3. Arsitektur RNN dengan formulasi matematis**

$x_{t-1}$ ,  $x_t$  dan  $x_{t+1}$  merupakan data input masing-masing pada *time step*  $t-1$ ,  $t$  dan  $t+1$ .  $S_t$  adalah *hidden state* pada *time step* ke- $t$ . *Hidden state* di sini bisa dianggap sebagai memori yang digunakan untuk menyimpan hasil pemrosesan dari neuron. Nilai dari *hidden state* sebelumnya (disimbolkan dengan  $W$ ) dan juga nilai dari sinyal *input current state*  $x_t$ .  $S_t = f(Ux_t + W_{S_{t-1}})$ . Fungsi  $f$  adalah fungsi aktivasi yang biasanya berupa fungsi *tangen hiperbolik* ( $\tanh$ ) atau *rectified linear unit* (ReLU). Pada kondisi awal,  $S_{t-1}$  secara default bernilai 0.  $o_t$  adalah output pada *stste* ke- $t$ . Nilai dari  $o_t$  berupa *output* dari fungsi aktivasi dari hasil perhitungan pada  $S_t$ , bisa berupa *softmax* atau *sigmoid* biasa.

Sekilas terlihat bahwa RNN tidak memiliki banyak lapisan seperti pada arsitektur *deep learning* lainnya. Namun, proses RNN yang secara berulang-ulang menyimpan hasil perhitungan pada *time step* sebelumnya untuk digunakan pada *time step* berikutnya dapat dianggap sebagai kumpulan beberapa lapis *neural network*. Perbedaannya adalah pada algoritma *deep learning* lainnya, nilai bobot pada setiap layer pada umumnya berbeda-beda. Namun, pada RNN karena satu *layer* digunakan berulang kali, maka dapat dianalogikan RNN menggunakan banyak layer dengan nilai bobot yang sama. Hal ini membuat RNN lebih efisien dari pada algoritma *deep learning* yang lainnya karena parameter yang perlu disimpan hanya bobot pada satu *layer*, bukan banyak layer seperti pada arsitektur CNN dan DBN.

RNN dapat memiliki beberapa variasi bergantung pada problem yang ingin diselesaikan. Setiap arsitektur RNN memiliki karakteristik yang berbeda-beda baik karakteristik *input* maupun *output*-nya. Beberapa variasi arsitektur RNN diilustrasikan pada gambar 4. Setiap kotak mempresentasikan vektor bobot dan panah mempresentasikan fungsi perkalian matriks. Arah panah menunjukkan arah aliran data dari input menuju *output*.



**Gambar 4. Empat variasi arsitektur RNN (Li, Johnson and Yeung, 2017)**

Dari kiri ke kanan adalah variasi arsitektur RNN sebagai berikut.

- Arsitektur RNN *one to many*, digunakan pada problem yang memiliki *output* berupa *sequence*. Contohnya pada problem *image captioning*, yang menerima input berupa citra dan mengeluarkan output berupa rangkaian kata-kata menjadi kalimat.
- Arsitektur RNN *many to one*, digunakan pada problem yang memiliki input berupa *sequence* dan output berukuran *fixed*. Contohnya pada problem analisis sentimen, dengan *input* berupa rangkaian kata-kata dalam kalimat dan *output* berupa label sentimen positif atau negatif.
- Arsitektur RNN *many to many*, digunakan pada problem yang memiliki input dan output berupa *sequence*. Contohnya pada problem mesin penerjemah. *Input* dari sistem berupa rangkaian kata-kata dari bahasa asal dan output juga berupa rangkaian kata-kata dari bahasa tujuan.

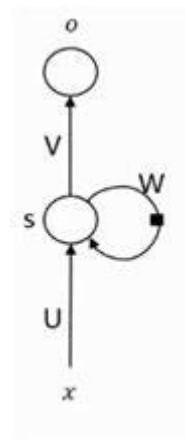
d. Arsitektur *many to many* dengan input dan output sinkron (tanpa *delay*). Contoh masalahnya adalah klasifikasi video yang melebihi setiap *frame* pada video.

### 3.3 Formulasi Recurrent Neural Network

Selanjutnya, kita membahas bagaimana formulasi perhitungan RNN. Secara sederhana, formulasi perhitungan RNN dapat diilustrasikan seperti pada gambar 5.

$o_t$  adalah output dari sebuah neuron pada satu time step ke- $t$ .  $h_t$  adalah nilai dari hasil perhitungan pada sebuah neural network pada satu time step ke- $t$ . Nilai dari  $h_t$  diformulasikan sebagai berikut.

$$h_t = f_w (h_{t-1}, x_t) \quad (3-1)$$



Gambar 5. Arsitektur RNN dengan formulasi matematis

Dengan  $f_w$  adalah sebuah fungsi aktivasi pada neuron  $h$  yang biasanya merupakan fungsi aktivasi *tanh* atau fungsi ReLU dan nilai  $h_{t-1}$  didapatkan dari perhitungan  $h$  pada *time step* sebelumnya. Bobot yang ada pada *hidden layer*  $h$  terdiri dari dua, yaitu bobot untuk vektor  $h_{t-1}$  dan juga bobot untuk vektor input  $x_t$ . Sehingga untuk layer  $h$ , didefinisikan tiga parameter  $w_{hh}$  yang merupakan bobot dari output *hidden layer* pada time step ke  $t-1$  ke *hidden layer* pada time step ke- $t$ ,  $w_{xh}$  yang merupakan bobot dari input layer ke *hidden layer*, dan  $b_h$  yang merupakan nilai bias pada *hidden layer*. Maka formulasi perhitungan dari *output hidden layer* pada time step ke- $t$  adalah sebagai berikut.

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h) \quad (3-2)$$

Dan nilai dari *output layer* pada time step ke- $t$  adalah sebagai berikut.

$$o_t = f_0(W_{h0} h_t + b_0) \quad (3-3)$$

Dengan  $f_0$  adalah fungsi aktivasi dari output layer yang biasanya berupa sigmoid atau softmax,  $W_{h0}$  adalah bobot dari *hidden layer* ke *output layer* dan  $b_0$  adalah nilai bias pada *output layer*.



### 3.4 Algoritma Pembelajaran Recurrent Neural Network

RNN memiliki arsitektur yang seolah-olah berlapis namun pada dasarnya hanya terdiri dari neural network yang sederhana namun digunakan secara berulang. Setiap satu layer neural network akan digunakan beberapa kali pada beberapa time step yang berurutan. Sehingga, proses pembelajaran pada RNN juga akan menelusuri setiap lapis neural network pada satu rangkaian waktu. Oleh karena itu, proses pembelajaran pada RNN disebut sebagai algoritma pembelajaran *Back Propagation Through Time* (BPTT) (Han, 2015).

Kita telah mengetahui bahwa di dalam arsitektur RNN kita menghitung output dari *hidden layer* pada satu time step ke- $t$  dengan formulasi berikut.

$$h_t = f_w(h_{t-1}, x_t) \quad (3-4)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \quad (3-5)$$

Dan output pada satu time step ke- $t$  dihitung menggunakan formula berikut.

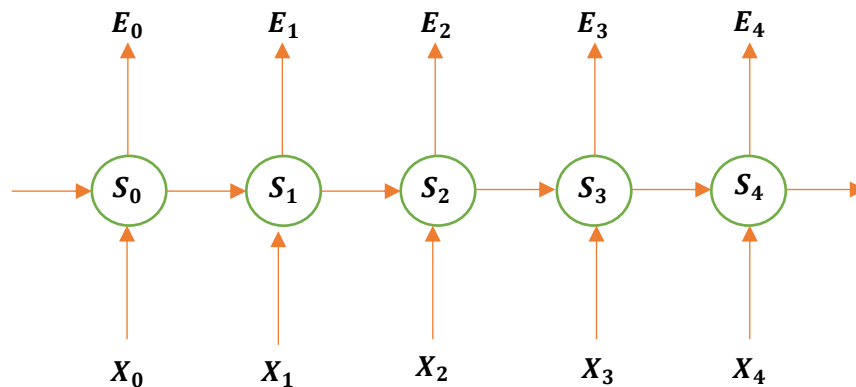
$$o_t = f_o(W_{ho}h_t + b_o) \quad (3-6)$$

Kita definisikan *loss function* menggunakan formula *cross entropy* sebagai berikut.

$$E_t = (o_t, o_t) = -O_t \log(o_t) \quad (3-7)$$

$$E = \sum_t E_t(o_t, o_t) = \sum_t -O_t \log(o_t) \quad (3-8)$$

Dengan  $E$  adalah total loss function pada satu sequence input,  $E_t = (o_t, o_t)$  adalah *loss function* untuk satu *time step*  $t$ , dan  $o_t$  adalah nilai sebenarnya dari output pada *time step* ke- $t$ .



Gambar 6. Arsitektur RNN dengan formulasi matematis

Seperti pada algoritma pembelajaran *Backpropagation* pada umumnya, BPTT juga menggunakan pendekatan *Gradient Descent* dengan mencari nilai dari  $U$ ,  $V$  dan  $W$  yang meminimumkan nilai loss function. Sama seperti pada proses perhitungan error pada BPTT juga menghitung total dari *gradient* yang didapatkan pada setiap *time step*.

$$\frac{\partial E_3}{\partial w} = \sum_t \frac{\partial E_t}{\partial w} \quad (3-9)$$



Untuk menghitung gradien kita menggunakan aturan rantai dari diferensiasi tersebut. Kita misalkan untuk  $t=3$ , maka proses penurunan perhitungan gradien menggunakan aturan rantai adalah sebagai berikut.

$$\frac{\partial E_3}{\partial w} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial v} = (\hat{y}_3 - y_3) \otimes s_3 \quad (3-10)$$

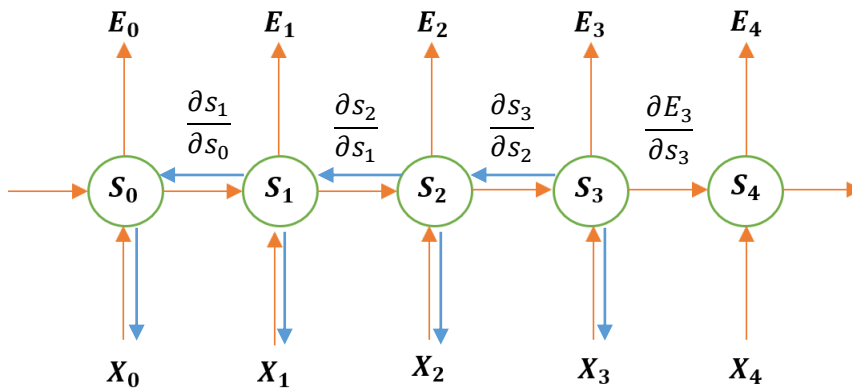
Dengan  $z_3 = V \cdot s_3$  dan  $\otimes$  adalah hasil perkalian dari dua vektor. Perhatikan bahwa hasil dari perhitungan tersebut hanya menggunakan informasi dari time step  $t = 3$ . Namun, perhitungan untuk gradient terhadap  $U$  akan berbeda. Perhitungan nilai gradient terhadap  $W$  dapat dijabarkan sebagai berikut.

$$\frac{\partial E_3}{\partial w} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial w} \quad (3-11)$$

Perhatikan bahwa  $s_3 = \tanh(U_{x_t} + W_{s_2})$  bergantung dari nilai  $W$  dan  $s_2$  yang merupakan nilai yang didapatkan pada time step sebelumnya lagi. Sehingga jika kita ingin menghitung nilai gradien pada time step ke 3 terhadap  $W$ , kita tidak dapat menganggap bahwa nilai  $s_2$  tersebut konstan. Kita harus memperhitungkan nilai tersebut hingga ke  $s_1$ . Maka dengan aturan rantai, akan kita dapatkan perhitungan sebagai berikut.

$$\frac{\partial E_3}{\partial w} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_k} \frac{\partial s_k}{\partial w} \quad (3-12)$$

Sehingga nilai dari gradient didapatkan dari total perhitungan nilai dari setiap *time step*. Semakin banyak *loop* dari arsitektur RNN yang kita bangun, akan semakin kompleks proses perhitungan dari gradient tersebut.



Gambar 7. Arsitektur RNN dengan formulasi matematis

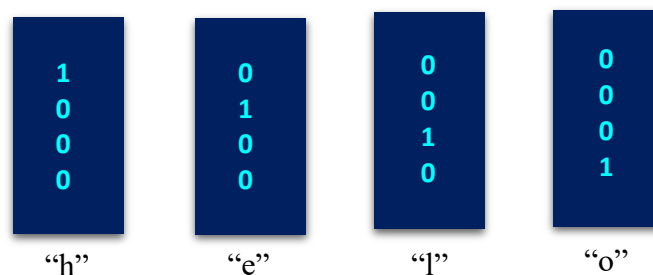
Perhatikan, bahwa ini pada dasarnya sama seperti algoritma *Backpropagation* yang ada pada arsitektur *deep neural network* lainnya. Perbedaannya adalah pada nilai  $W$  yang digunakan pada setiap *layer*. Berbeda dengan arsitektur *deep learning* lainnya seperti CNN dan DBN yang setiap *layer*-nya memiliki nilai unik, sehingga kita tidak perlu menghitung nilai total dari setiap layer. Proses perhitungan gradient ini menjadi sangat kompleks jika jumlah loop



pada RNN sangat banyak. Untuk mengimplementasikan algoritma BPTT agar lebih cepat, digunakan skema *truncated* yang membagi proses *forward pass* dan *backward pass* ke dalam *subsequences* yang lebih kecil dengan jumlah *time step* yang seragam.

### 3.5 RNN untuk Data Sekuens

Sekarang, marilah kita mendiskusikan penggunaan RNN untuk data sekuens. Kita akan lihat beberapa contoh dari penggunaan RNN dalam beberapa kasus yang data sekuens yang umum. Kasus pertama adalah *Character Level Language* model yaitu bagaimana membangun model bahasa dari rangkaian karakter (Bullinaria, 2015). Misalnya kita ingin melatih model RNN kita untuk mengenali sekuens *h-e-l-o*. Artinya jika model kita diberikan pola *h-e-l*, maka model RNN harus mengembalikan sekuens sisanya yaitu karakter *l* dan *o*. *Vocabulary* yaitu kita susun untuk model kita terdiri dari 4 huruf masing-masing adalah *h*, *e*, *l*, dan *o* dengan mekanisme pengkodean sebagai berikut.



Gambar 8. Representasi Encoding karakter

Parameter yang kita butuhkan pada *hidden layer* adalah  $W_{xh}$ ,  $W_{hh}$ , dan  $b_h$  dan pada *output layer* adalah  $W_{hy}$  dan  $b_y$ . Fungsi pada *hidden layer* dan *output layer* masing-masing adalah:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \quad (3-13)$$

$$y_t = W_{hy}h_t \quad (3-14)$$

Pada time step pertama, input adalah karakter *h* dan outputnya memprediksi karakter berikutnya yaitu *e*. Karakter *h* dikodekan menjadi vektor  $[1, 0, 0, 0]^T$ . Lalu dilakukan perhitungan perkalian matriks dengan bobot  $W_{xh}$  didapatkan vektor  $[0.3, -0.1, 0.9]^T$ .

### 3.6 Long Short-Term Memory

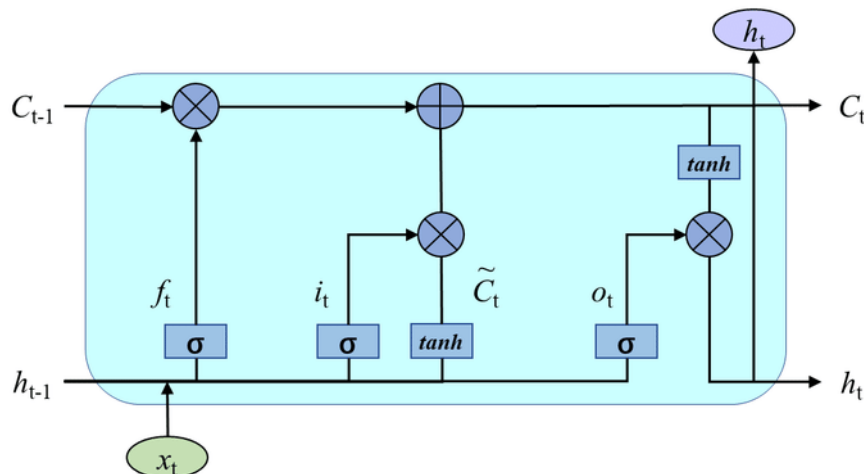
Kita perhatikan lagi algoritma pembelajaran BPTT. Kita perhatikan bahwa semakin panjang suatu input sequences, maka semakin kompleks juga proses perhitungan gradient. Hal ini secara tidak langsung akan membatasi kemampuan dari RNN. Padahal dalam beberapa kasus, kadang kita memerlukan sequence yang agak panjang. Misalkan pada kasus pengenalan makna semantik kalimat. Misalkan terdapat kalimat Laki-laki yang memakai rambut palsu pada kepalanya masuk ke dalam. Inti dari kalimat tersebut adalah laki-laki yang masuk ke

dalam, bukan laki-laki memakai rambut palsu. Sehingga, sequence yang harus diproses adalah sepanjang kalimat tersebut. Namun arsitektur RNN biasa tidak akan dapat memproses sequence yang panjang. Kita perhatikan pada persamaan gradien untuk mengupdate nilai  $W$  dengan asumsi *sequence* sepanjang 3.

$$\frac{\partial E_3}{\partial w} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_k} \frac{\partial s_k}{\partial w} \quad (3-15)$$

Perhatikan bahwa persamaan tersebut didapatkan dari urutan rantai sehingga nilai dari  $\frac{\partial s_3}{\partial s_k}$  sendiri dapat berupa rantai perkalian yang mungkin sangat panjang, misalkan untuk panjang  $sequence=3$  akan menjadi  $\frac{\partial s_3}{\partial s_2} \times \frac{\partial s_2}{\partial s_1}$ . Nilai gradient untuk satu *times step* didapatkan dari hasil aktivasi fungsi *tanh* dengan *range*  $[-1, 1]$ . Nilai range ini memungkinkan pecahan sehingga hasil perkalian dari beberapa gradient sangat mungkin menghasilkan nilai nol. Kondisi ini kita kenal dengan istilah *vanishing gradient*. Atau jika nilai bobot pada  $W > 1$ , maka nilai gradient akan terus membesar yang disebut sebagai kondisi *exploding gradient*.

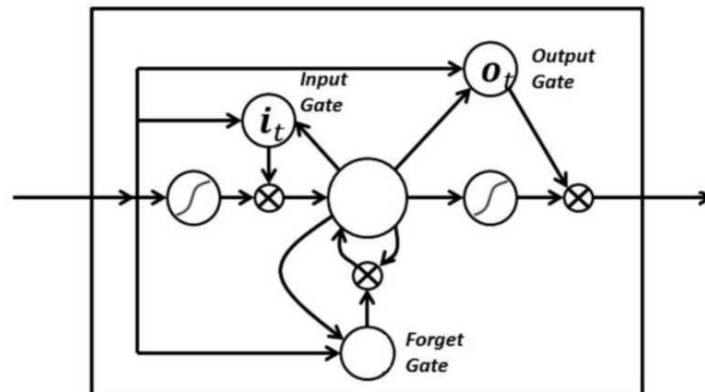
Untuk mengatasi problem *vanishing gradient* tersebut, diusulkan sebuah solusi berupa arsitektur RNN yang bernama *Long Short-Term Memory* (LSTM). LSTM pertama kali diusulkan tahun 1997 oleh Sepp Hochreiter dan Jürgen Schmidhuber dan saat ini menjadi arsitektur RNN yang paling populer digunakan. LSTM menggunakan mekanisme 4 gerbang untuk mengatasi problem *vanishing gradient* yang muncul pada arsitektur RNN biasa. Berikut adalah arsitektur dari LSTM (Hochreiter and Urgen Schmidhuber, 1997).



**Gambar 9. Arsitektur LSTM (Hocreiter and Urgent Schmidhuber, 1997)**

*Long Short-Term Memory* adalah ekstensi untuk *Recurrent Neural Network* (RNN) yang pada dasarnya memperluas memori. Oleh karena itu, sangat cocok untuk belajar dari pengalaman penting yang memiliki jeda waktu sangat lama. Unit *Long Short-Term Memory* (LSTM) digunakan sebagai unit bangunan untuk lapisan *Recurrent Neural Network* (RNNs), sering disebut jaringan *Long Short-Term Memory* (LSTM), *Long Short-Term Memory* (LSTM)

memungkinkan *Recurrent Neural Network* (RNNs) untuk mengingat input dalam jangka waktu yang lama. Ini karena LSTM berisi informasi dalam memori, seperti memori komputer. LSTM dapat membaca, menulis, dan menghapus informasi dari memorinya.



Gambar 9. Gerbang LSTM

Gerbang dalam LSTM adalah analog dalam bentuk *sigmoid*, artinya berkisar dari nol hingga satu. Fakta bahwa mereka analog memungkinkan mereka untuk melakukan *backpropagation*.

### 3.6.1 Gradient

Gradient adalah turunan parsial terhadap inputnya. Jika Anda tidak tahu apa artinya, pikirkan saja seperti ini: gradien mengukur seberapa banyak output dari suatu fungsi berubah jika Anda mengubah inputnya sedikit, gradien sebagai kemiringan suatu fungsi. Semakin tinggi gradien, semakin curam kemiringannya dan semakin cepat model dapat belajar. Tetapi jika kemiringannya nol, model akan mengalami *vanishing gradient*.

### 3.6.2 Exploding Gradients

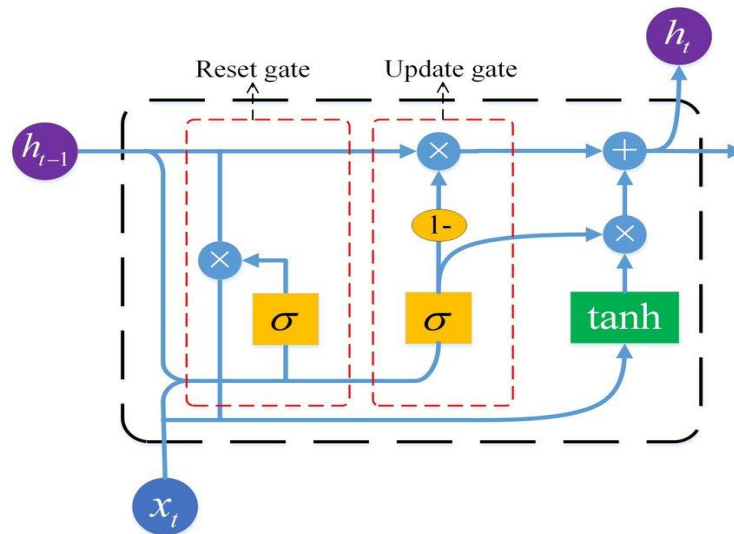
Gradien yang meledak adalah ketika algoritma, tanpa banyak alasan, memberikan bobot yang sangat penting. Untungnya, masalah ini dapat dengan mudah diselesaikan dengan memotong atau menekan gradien.

### 3.6.3 Vanishing Gradients

Gradien menghilang terjadi ketika nilai gradien terlalu kecil dan model berhenti belajar atau membutuhkan waktu terlalu lama. Ini adalah masalah besar pada 1990-an dan jauh lebih sulit untuk dipecahkan daripada gradien yang meledak. Untungnya, itu diselesaikan melalui konsep LSTM oleh Sepp Hochreiter dan Juergen Schmidhuber.

LSTM sendiri memiliki beberapa variasi, seperti misalnya LSTM yang memiliki *peephole connection*, LSTM yang menggabungkan *input gate* dan *forget gate*, dan variasi yang cukup terkenal adalah *Gated Recurrent Unit* (GRU) yang dipopulerkan oleh Cho dan Chung pada tahun 2014. GRU menyederhanakan proses komputasi dalam LSTM namun dengan kinerja yang masih cukup setara dengan LSTM. Pada GRU, *forget gate* digabungkan dengan *input*

gate menjadi satu gate yang disebut *update gate*. GRU juga menggabungkan *cell state* ke dalam *hidden state*. Berikut adalah arsitektur GRU (Wang, Liao and Chang, 2018).



Gambar 10. Arsitektur GRU (Wang, Liao and Chang, 2018)

### 3.6.4 Kelebihan Dan Kekurangan Algoritma LSTM

Kelebihan algoritma LSTM bila dibandingkan dengan RNN konvensional adalah:

- LSTM mampu memodelkan urutan kronologis dan dependensi jarak jauh.
- Cenderung lebih baik untuk masalah memori pendek karena adanya modifikasi formula pada memori internal.

Sedangkan kekurangan dari LSTM, yaitu:

- Terjadi peningkatan kompleksitas komputasi dibandingkan dengan RNN karena lebih banyak parameter untuk dipelajari.
- Memori yang dibutuhkan lebih tinggi dari pada RNN konvensional karena adanya beberapa memori *cell*.
- Cenderung terjadi masalah *overfitting*.

### 3.7 Implementasi LSTM dengan Python

Untuk lebih memahami cara Long Short-Term Memory (LSTM), kita akan menggunakan kasus dengan Python. Dataset yang akan digunakan adalah dataset BBRI.JK dari Yahoo Finance. Data tersebut dapat didownload di situs <https://finance.yahoo.com/quote/BBRI.JK/history?p=BBRI.JK>. Langkah-langkah yang akan kita gunakan untuk memproses dataset tersebut adalah sebagai berikut.

#### 3.7.1 Import Tensorflow

Sebelum kita memulai memproses dataset, terlebih dahulu kita akan melihat berapa versi tensorflow yang akan kita gunakan saat ini dengan cara mengetikkan program sebagai berikut.

```
import tensorflow as tf
```



```
tf.__version__
```

```
'2.12.0'
```

Keterangan kode program:

- `import tensorflow as tf` digunakan untuk mengimpor library TensorFlow ke dalam program Python. TensorFlow adalah salah satu library yang populer digunakan untuk keperluan deep learning dan machine learning.  
TensorFlow menyediakan kerangka kerja untuk membangun, melatih, dan menggunakan model machine learning dan deep learning, terutama neural networks. Dengan menggunakan TensorFlow, Anda dapat membuat berbagai model machine learning, seperti model regresi, model klasifikasi, dan model neural networks seperti Convolutional Neural Networks (CNN) dan Recurrent Neural Networks (RNN).
- `tf.__version__` digunakan untuk mengetahui versi TensorFlow yang saat ini digunakan dalam lingkungan Python Anda. Dengan menjalankan perintah `tf.__version__`, Anda dapat dengan mudah mendapatkan informasi tentang versi TensorFlow yang terpasang pada sistem Anda.

### 3.7.2 Import library

Berikut ini adalah beberapa library yang kita butuhkan untuk memproses dataset BBRIJK.

```
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import yfinance as yf
```

Keterangan kode program:

- `import math`, digunakan untuk mengimpor modul `math` dalam bahasa pemrograman Python. Modul `math` adalah modul bawaan (built-in module) yang menyediakan berbagai fungsi matematika dan konstanta matematika yang siap digunakan dalam program Anda.
- `import numpy as np`, digunakan untuk mengimpor modul NumPy dalam bahasa pemrograman Python. NumPy (Numerical Python) adalah sebuah pustaka (library) yang sangat populer untuk komputasi numerik dan manipulasi array multidimensi. Dengan mengimpor modul NumPy dan memberikan alias `np`, Anda dapat menggunakan berbagai fungsi dan fitur yang disediakan oleh NumPy dalam kode Python Anda.



- `import pandas as pd`, digunakan untuk mengimpor modul Pandas dalam bahasa pemrograman Python. Pandas adalah salah satu library yang paling populer digunakan untuk manipulasi dan analisis data dalam Python. Dengan mengimpor module Pandas dan memberikan alias `pd`, Anda dapat menggunakan berbagai fungsi dan fitur yang disediakan oleh Pandas dalam kode Python Anda.
- `from sklearn.preprocessing import MinMaxScaler`, digunakan untuk mengimpor kelas `MinMaxScaler` dari modul `preprocessing` dalam library `scikit-learn` (`sklearn`) dalam bahasa pemrograman Python. `Scikit-learn` adalah salah satu library yang paling populer untuk machine learning dan data science di Python.
- `from keras.models import Sequential`, digunakan untuk mengimpor kelas `Sequential` dari modul `models` dalam library Keras (sekarang telah diintegrasikan ke dalam TensorFlow) dalam bahasa pemrograman Python. Keras adalah library yang populer untuk pembuatan dan pelatihan model neural network (jaringan saraf) dalam lingkungan Python.
- `from keras.layers import Dense, LSTM` digunakan untuk mengimpor kelas `Dense` dan `LSTM` dari modul `layers` dalam library Keras (sekarang telah diintegrasikan ke dalam TensorFlow) dalam bahasa pemrograman Python. Keras adalah library yang populer untuk pembuatan dan pelatihan model neural network (jaringan saraf) dalam lingkungan Python.
- `import matplotlib.pyplot as plt` digunakan untuk mengimpor modul `pyplot` dari library `matplotlib` dalam bahasa pemrograman Python. `Matplotlib` adalah library yang sangat populer untuk membuat visualisasi dan plot grafik. Modul `pyplot` dalam `matplotlib` menyediakan fungsi-fungsi yang mirip dengan MATLAB untuk membuat berbagai jenis plot seperti scatter plot, line plot, bar plot, histogram, dan banyak lagi.
- `plt.style.use('fivethirtyeight')` digunakan untuk menerapkan gaya (style) visual "fivethirtyeight" pada gambar (plot) yang akan dibuat menggunakan library `matplotlib` dalam bahasa pemrograman Python. `Matplotlib` menyediakan beberapa gaya visual bawaan yang dapat digunakan untuk mengubah tampilan default plot menjadi lebih menarik dan sesuai dengan preferensi atau tema tertentu.
- `import yfinance as yf` digunakan untuk mengimpor library `yfinance` dalam bahasa pemrograman Python. `yfinance` adalah library yang memungkinkan Anda untuk mengakses dan mendapatkan data historis dan real-time dari Yahoo Finance, sebuah platform penyedia informasi keuangan dan data pasar saham.

### 3.7.3 Install Yfinance

Untuk menginstall *library Yahoo Finance* menggunakan package manager `pip`, berikut ini adalah programnya.





```
!pip install yfinance
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: yfinance in c:\users\hendr\appdata\roaming\python\python310\site-packages (0.2.20)
Requirement already satisfied: pandas>=1.3.0 in c:\anaconda3\lib\site-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in c:\anaconda3\lib\site-packages (from yfinance) (1.23.5)
Requirement already satisfied: requests>=2.26 in c:\anaconda3\lib\site-packages (from yfinance) (2.28.1)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\hendr\appdata\roaming\python\python310\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\anaconda3\lib\site-packages (from yfinance) (4.9.1)
Requirement already satisfied: appdirs>=1.4.4 in c:\anaconda3\lib\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\anaconda3\lib\site-packages (from yfinance) (2022.7)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\hendr\appdata\roaming\python\python310\site-packages (from yfinance) (2.3.8)
Requirement already satisfied: cryptography>=3.3.2 in c:\anaconda3\lib\site-packages (from yfinance) (39.0.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\anaconda3\lib\site-packages (from yfinance) (4.11.1)
Requirement already satisfied: html5lib>=1.1 in c:\users\hendr\appdata\roaming\python\python310\site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in c:\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.3.2.post1)
Requirement already satisfied: cffi>=1.12 in c:\anaconda3\lib\site-packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: six>=1.9 in c:\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in c:\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2022.12.7)
Requirement already satisfied: pycparser in c:\anaconda3\lib\site-packages (from cffi>=1.12->cryptography>=3.3.2->yfinance) (2.21)
```

Keterangan kode program:

`!pip install yfinance`, digunakan untuk menginstal library `yfinance` menggunakan package manager `pip` dalam lingkungan Python. Perintah `pip` adalah alat yang digunakan untuk mengelola paket atau library pihak ketiga (third-party) dalam Python.

Dengan menjalankan perintah `!pip install yfinance`, Anda memerintahkan Python untuk mengunduh dan menginstal library `yfinance` dari Python Package Index (PyPI) ke lingkungan





Python yang aktif saat ini. Setelah proses instalasi selesai, Anda dapat mengimpor dan menggunakan library yfinance dalam kode Python Anda.

Pastikan Anda telah terhubung dengan internet saat menjalankan perintah `!pip install yfinance` agar Python dapat mengunduh paket yfinance dari PyPI. Jika instalasi berhasil, Anda akan dapat menggunakan library tersebut untuk mengakses data keuangan dan saham dari Yahoo Finance seperti yang telah dijelaskan sebelumnya.

Sebagai tambahan, perlu diingat bahwa instalasi paket menggunakan perintah `!pip` biasanya dilakukan di lingkungan seperti Jupyter Notebook, Google Colab, atau lingkungan interaktif lainnya. Jika Anda menggunakan lingkungan lokal, Anda dapat menggunakan perintah `pip install yfinance` langsung dari terminal atau command prompt untuk menginstal yfinance pada lingkungan Python lokal.

### 3.7.4 Download BBRI.JK File

```
A_df = yf.download('BBRI.JK')
```

```
[*****100%*****] 1 of 1 completed
```

Keterangan kode program:

- `A_df = yf.download('BBRI.JK')`, digunakan untuk mengunduh data historis harga saham dari Yahoo Finance untuk saham dengan kode "BBRI.JK". Di sini, "BBRI.JK" adalah simbol pasar saham dari PT Bank Rakyat Indonesia Tbk (Bank BRI) yang tercatat di Bursa Efek Indonesia.
- Setelah menjalankan perintah ini, data historis harga saham Bank BRI akan diunduh dan disimpan dalam bentuk DataFrame yang ditugaskan ke variabel `A_df`. DataFrame adalah struktur data tabular dua dimensi yang disediakan oleh library pandas, yang memungkinkan Anda untuk menyimpan dan memanipulasi data dengan mudah.

### 3.7.5 Show dataset

Berikut adalah kode program untuk melihat isi dari dataset BBRI.JK yang ada di library Yahoo Finance dan kemudian memasukkannya ke dalam variabel `A_df`.

```
A_df
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2003-11-10	105.0	110.0	95.0	97.5	53.342930	5144140000
2003-11-11	97.5	100.0	95.0	100.0	54.710709	2938555000



	Open	High	Low	Close	Adj Close	Volume
Date						
2003-11-12	97.5	105.0	97.5	105.0	57.446243	2002915000
2003-11-13	105.0	105.0	102.5	105.0	57.446243	1190050000
2003-11-14	105.0	105.0	100.0	105.0	57.446243	1725265000
...	...	...	...	...	...	...
2023-07-03	5500.0	5550.0	5450.0	5475.0	5475.000000	168559800
2023-07-04	5500.0	5500.0	5425.0	5450.0	5450.000000	94462600
2023-07-05	5400.0	5450.0	5400.0	5450.0	5450.000000	109632800
2023-07-06	5450.0	5475.0	5400.0	5425.0	5425.000000	109446200
2023-07-07	5400.0	5425.0	5375.0	5375.0	5375.000000	114043800

4879 rows × 6 columns

Keterangan kode program:

- Setelah menjalankan perintah `A_df = yf.download('BBRI.JK')` dan mengunduh data historis harga saham dari Yahoo Finance untuk saham Bank BRI, variabel `A_df` akan berisi DataFrame yang menyimpan data tersebut.
- DataFrame `A_df` berisi data historis harga saham Bank BRI yang diunduh dari Yahoo Finance, dan dapat digunakan untuk melakukan berbagai analisis, pemrosesan data, dan visualisasi terkait harga saham. Anda dapat menggunakan berbagai fungsi dan fitur dari library pandas untuk memanipulasi dan menganalisis data yang ada dalam DataFrame `A_df`.

### 3.7.6 Import Datetime

Berikut adalah kode python untuk mengimport datetime.

```
import datetime
```

Keterangan kode program:



- `import datetime`, digunakan untuk mengimpor modul `datetime` dalam bahasa pemrograman Python. Modul `datetime` adalah bagian dari library standar Python dan menyediakan berbagai fungsi dan kelas untuk bekerja dengan tanggal (`date`) dan waktu (`time`).
- Dengan mengimpor modul `datetime`, Anda dapat mengakses berbagai fungsi dan kelas yang memungkinkan Anda untuk melakukan hal-hal seperti:
  1. Mendapatkan waktu saat ini (`current time`).
  2. Membuat objek tanggal (`date`) atau waktu (`time`).
  3. Menghitung selisih antara dua tanggal atau waktu.
  4. Memformat tanggal atau waktu menjadi bentuk yang diinginkan.
  5. Dan banyak lagi operasi terkait tanggal dan waktu.

### 3.7.7 Install Datetime

Untuk menginstall *library datetime* menggunakan package manager `pip`, berikut ini adalah programnya.

```
!pip install datetime
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: datetime in c:\users\hendr\appdata\roaming\python\python310\site-packages (5.1)
Requirement already satisfied: zope.interface in c:\anaconda3\lib\site-packages (from datetime) (5.4.0)
Requirement already satisfied: pytz in c:\anaconda3\lib\site-packages (from datetime) (2022.7)
Requirement already satisfied: setuptools in c:\anaconda3\lib\site-packages (from zope.interface->datetime) (65.6.3)
```

Keterangan kode program:

- Dengan menggunakan modul `datetime`, Anda dapat dengan mudah bekerja dengan tanggal dan waktu dalam kode Python, melakukan operasi terkait tanggal dan waktu, serta memanipulasi format tanggal dan waktu sesuai dengan kebutuhan aplikasi Anda.

### 3.7.8 Time Range

Untuk menentukan rentang waktu (*time range*) yang akan kita gunakan dalam memprediksi saham BBRI.JK, kita dapat mengetikkan program sebagai berikut.

```
start = datetime.datetime(2019,1,20)
end = datetime.datetime(2023,6,15)
```

Keterangan kode program:

- `start = datetime.datetime(2019, 1, 20)` digunakan untuk membuat objek `start` yang merupakan representasi dari tanggal dan waktu tertentu dalam bahasa pemrograman



Python. Objek start diinisialisasi dengan tanggal 20 Januari 2019. Disini kita menggunakan time range antara tanggal 20 Januari 2019 sampai dengan 15 Juni 2023.

### 3.7.9 Download BBRI.JK File

Untuk mendownload file BBRI.JK kita dapat menggunakan perintah sebagai berikut, dengan menyertakan variabel A\_df beserta tanggal awal dan akhir.

```
A_df = yf.download('BBRI.JK', start=start, end=end)
```

```
[*****100%*****] 1 of 1 completed
```

Keterangan kode program:

- `A_df = yf.download('BBRI.JK', start=start, end=end)` digunakan untuk mengunduh data historis harga saham dari Yahoo Finance untuk saham dengan kode "BBRI.JK" dalam rentang tanggal tertentu. Di sini, start dan end adalah dua objek datetime yang menentukan rentang tanggal untuk data yang akan diunduh.
- Ketika Anda menjalankan perintah ini, data historis harga saham Bank BRI (kode "BBRI.JK") akan diunduh dari Yahoo Finance mulai dari tanggal yang ditentukan oleh objek start hingga tanggal yang ditentukan oleh objek end. Data tersebut akan disimpan dalam bentuk DataFrame yang ditugaskan ke variabel A\_df.

### 3.7.10 Show Dataset

Berikut adalah kode program Python untuk melihat seluruh isi dari dataset BBRI.JK.

```
A_df
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2019-01-21	3800.0	3830.0	3800.0	3800.0	3149.417236	80926000
2019-01-22	3770.0	3790.0	3740.0	3770.0	3124.553223	119687600
2019-01-23	3760.0	3810.0	3730.0	3770.0	3124.553223	163458700
2019-01-24	3750.0	3810.0	3750.0	3790.0	3141.129150	176166500
2019-01-25	3820.0	3820.0	3780.0	3780.0	3132.841064	127141300
...	...	...	...	...	...	...
2023-06-08	5375.0	5475.0	5350.0	5475.0	5475.000000	92794100



	Open	High	Low	Close	Adj Close	Volume
Date						
2023-06-09	5425.0	5450.0	5375.0	5425.0	5425.000000	77251000
2023-06-12	5400.0	5400.0	5375.0	5400.0	5400.000000	92314200
2023-06-13	5400.0	5550.0	5400.0	5550.0	5550.000000	108831400
2023-06-14	5575.0	5575.0	5475.0	5550.0	5550.000000	79137600

1083 rows × 6 columns

Keterangan kode program:

- Setelah Anda menjalankan perintah `A_df = yf.download('BBRI.JK', start=start, end=end)` dan mengunduh data historis harga saham Bank BRI dari Yahoo Finance dalam rentang tanggal tertentu, variabel `A_df` akan berisi `DataFrame` yang menyimpan data tersebut.

`DataFrame A_df` berisi data historis harga saham Bank BRI dalam rentang tanggal yang telah ditentukan oleh objek `start` dan `end`. `DataFrame` ini memuat informasi seperti harga pembukaan (`Open`), harga tertinggi (`High`), harga terendah (`Low`), harga penutupan (`Close`), volume perdagangan (`Volume`), dan mungkin kolom lainnya tergantung pada data yang tersedia dari Yahoo Finance.

Anda dapat menggunakan `DataFrame A_df` untuk melakukan berbagai analisis, pemrosesan data, dan visualisasi terkait harga saham Bank BRI selama rentang tanggal tersebut.

### 3.7.11 Show Shape

Untuk melihat shape kita dapat menggunakan perintah program sebagai berikut.

```
A_df.shape
```

```
(1083, 6)
```

Keterangan kode program:

- `A_df.shape` digunakan untuk mengetahui dimensi (shape) dari `DataFrame A_df`, yaitu jumlah baris dan kolom yang ada dalam `DataFrame`. Hasil dari `A_df.shape` adalah sebuah tuple dengan dua elemen, di mana elemen pertama merupakan jumlah baris dan elemen



kedua merupakan jumlah kolom dalam DataFrame. Disini terlihat bahwa DataFrame memiliki sebanyak 1083 baris dan terdiri dari 6 kolom.

### 3.7.12 Plot The Prediction Results On The Training Data

```
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(A_df['Close'])
plt.xlabel('Date',fontsize=18)
plt.ylabel('Close Price USD($)',fontsize=18)
plt.show()
```



Keterangan kode program:

- `plt.figure(figsize=(16,8))` digunakan untuk mengatur ukuran (lebar dan tinggi) dari sebuah gambar (plot) yang akan dihasilkan menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini memungkinkan Anda untuk membuat gambar dengan ukuran yang lebih besar atau lebih kecil dari ukuran default.

Berikut adalah penjelasan lebih lanjut:

- `plt`: Ini merujuk ke objek pyplot dari library matplotlib. pyplot menyediakan antarmuka yang mirip dengan MATLAB dan banyak digunakan untuk membuat visualisasi dalam Python.
- `figure()`: Fungsi `figure()` digunakan untuk membuat sebuah gambar (plot) baru atau mengatur atribut dari gambar yang sedang aktif. Sebagai argumen opsional, Anda dapat menyertakan parameter `figsize` untuk mengatur ukuran gambar.
- `figsize=(16, 8)`: Ini adalah argumen dari fungsi `figure()`. Dalam contoh ini, kita mengatur ukuran gambar dengan lebar 16 inci dan tinggi 8 inci. Nilai (16, 8) adalah dalam satuan inci.



- Dengan menggunakan `plt.figure(figsize=(16, 8))`, Anda dapat membuat gambar yang lebih besar atau lebih lebar untuk menampilkan grafik atau visualisasi dengan lebih jelas dan terlihat lebih baik.
- `plt.title('Close Price History')` digunakan untuk memberikan judul pada gambar (plot) yang dibuat menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini digunakan untuk menambahkan judul yang deskriptif di atas gambar, sehingga memberikan informasi tentang konten atau isi dari plot tersebut.
- `plt.plot(A_df['Close'])` digunakan untuk membuat plot garis dari data kolom 'Close' dari DataFrame `A_df` menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini akan menghasilkan visualisasi data harga penutupan (close prices) dalam bentuk garis.
- `plt.xlabel('Date', fontsize=18)` digunakan untuk menambahkan label pada sumbu x (horizontal) dari gambar (plot) yang dibuat menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini memberikan deskripsi yang jelas tentang nilai atau data yang ditampilkan pada sumbu x.
- `plt.ylabel('Close Price USD($)', fontsize=18)` digunakan untuk menambahkan label pada sumbu y (vertikal) dari gambar (plot) yang dibuat menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini memberikan deskripsi yang jelas tentang nilai atau data yang ditampilkan pada sumbu y, serta memungkinkan untuk mengatur ukuran font dari label tersebut.
- `plt.show()` digunakan untuk menampilkan gambar (plot) yang telah dibuat menggunakan library matplotlib dalam bahasa pemrograman Python. Fungsi ini menampilkan visualisasi data atau grafik ke dalam jendela pop-up atau output konsol (tergantung pada lingkungan pengembangan atau IDE yang digunakan).

### 3.7.13 Data Conversion Into Numpy Array Form

```
data=A_df.filter(['Close'])
dataset=data.values
training_data_len=math.ceil(len(dataset)*.8)
```

Keterangan kode program:

- `data = A_df.filter(['Close'])` digunakan untuk membuat dataframe baru dengan nama data, yang berisi kolom 'Close' dari dataframe `A_df`.  
Pada kode ini, kita menggunakan metode `filter()` pada dataframe `A_df` untuk memilih kolom 'Close'. Hasilnya adalah dataframe baru dengan nama data, yang hanya berisi kolom 'Close' dari dataframe awal.



- `dataset = data.values` digunakan untuk mengubah dataframe data menjadi bentuk array NumPy. Dalam konteks ini, variabel `dataset` akan berisi array NumPy yang berisi nilai dari kolom 'Close' dari dataframe data.
- `training_data_len=math.ceil(len(dataset)*.8)` digunakan untuk menghitung panjang data training (jumlah baris data) yang akan digunakan saat melatih model. Dalam konteks ini, variabel '`training_data_len`' akan berisi nilai yang merupakan 80% dari total jumlah baris data pada array '`dataset`', yang telah diambil dari kolom 'Close' dataframe '`data`'.

#### 3.7.14 Scale The Data

```
scaler=MinMaxScaler(feature_range=(0,1))
scaled_data=scaler.fit_transform(dataset)
scaled_data
```

```
array([[0.47521866],
       [0.4664723 ],
       [0.4664723 ],
       ...,
       [0.94169096],
       [0.98542274],
       [0.98542274]])
```

#### Keterangan kode program:

- `scaler=MinMaxScaler(feature_range=(0,1))` digunakan untuk membuat objek `scaler` dari kelas `MinMaxScaler` pada library `scikit-learn`. Objek `scaler` ini akan digunakan untuk melakukan normalisasi data dengan metode Min-Max Scaling pada data historis harga saham sebelum dilatih menggunakan model machine learning. Normalisasi dengan metode Min-Max Scaling akan mengubah nilai-nilai data ke dalam rentang yang ditentukan, dalam hal ini (0,1). Proses normalisasi ini berguna untuk menyamakan skala data sehingga data memiliki nilai terkecil 0 dan nilai terbesar 1. Normalisasi ini memudahkan model machine learning untuk belajar dari data yang memiliki rentang nilai yang lebih kecil dan seragam, serta mencegah masalah divergensi saat melatih model.
- `scaled_data = scaler.fit_transform(dataset)` berguna untuk melakukan normalisasi data historis harga saham dalam array '`dataset`' menggunakan metode Min-Max Scaling dengan rentang normalisasi (0,1). Metode '`fit_transform()`' dari objek '`scaler`' akan menghitung nilai minimum dan maksimum dari data, lalu melakukan normalisasi data sesuai dengan rentang yang telah ditentukan.

#### 3.7.15 Show Shape

Untuk melihat shape kembali berikut adalah kode programnya.





```
A_df.shape
```

```
(1083, 6)
```

Keterangan kode program:

- `A_df.shape` digunakan untuk mengetahui dimensi (shape) dari DataFrame `A_df`, yaitu jumlah baris dan kolom yang ada dalam DataFrame. Hasil dari `A_df.shape` adalah sebuah tuple dengan dua elemen, di mana elemen pertama merupakan jumlah baris dan elemen kedua merupakan jumlah kolom dalam DataFrame.

### 3.7.16 Create Training Data Set

```
train_data=scaled_data[0:training_data_len,:]
```

Keterangan kode program:

- `train_data = scaled_data[0:training_data_len, :]` digunakan untuk membagi data yang telah diubah skala (scaled data) menjadi data pelatihan (training data). Di sini, `training_data_len` adalah panjang (jumlah baris) data pelatihan yang ingin Anda gunakan.
- Dalam analisis dan pemodelan data, seringkali kita membagi data menjadi dua bagian: data pelatihan (training data) dan data pengujian (test data). Data pelatihan digunakan untuk melatih model atau algoritma machine learning, sedangkan data pengujian digunakan untuk menguji kinerja model yang telah dilatih.
- Dalam contoh di atas, `scaled_data` adalah data yang telah diubah skala, mungkin sebagai persiapan untuk model machine learning tertentu. Anda ingin menggunakan sebagian data ini sebagai data pelatihan. Dengan menggunakan `scaled_data[0:training_data_len, :]`, Anda mengambil bagian pertama dari `scaled_data` sebanyak `training_data_len` baris dan semua kolom.

### 3.7.17 Split The Data Into x\_train And y\_train Data Sets

```
x_train=[]  
y_train=[]
```

Keterangan kode program:

- `x_train=[]` dan `y_train=[]`, digunakan untuk mendefinisikan sebuah list kosong yang akan digunakan untuk menyimpan data pelatihan (input) pada kasus machine learning atau data yang akan digunakan sebagai fitur (features) dalam model.
- Dalam banyak kasus machine learning, data pelatihan seringkali memiliki format matriks atau array. Jika Anda ingin menggunakan pendekatan supervised learning, Anda perlu membagi data pelatihan menjadi dua bagian: fitur (features) yang akan disimpan dalam `x_train` dan label (target) yang akan disimpan dalam variabel lain (misalnya `y_train`).



### 3.7.18 Menjelaskan Y dengan Data 60 Hari Terakhir

##### 過去 60 日分のデータでYを説明する、という形#####

```
for i in range(60,len(train_data)):
    x_train.append(train_data[i-60:i,0])
    y_train.append(train_data[i,0])
    if i<=60:
        print(x_train)
        print(y_train)
        print()

[array([0.47521866, 0.4664723 , 0.4664723 , 0.47230321, 0.46938776,
        0.46938776, 0.44314869, 0.4606414 , 0.48979592, 0.51020408,
        0.50437318, 0.50437318, 0.51020408, 0.51311953, 0.50145773,
        0.49562682, 0.48979592, 0.47230321, 0.47521866, 0.4664723 ,
        0.48688047, 0.50728863, 0.50145773, 0.50728863, 0.50437318,
        0.52186589, 0.51603499, 0.49562682, 0.48979592, 0.49562682,
        0.49854227, 0.49271137, 0.50437318, 0.50437318, 0.48979592,
        0.49271137, 0.48688047, 0.47230321, 0.48688047, 0.51895044,
        0.5335277 , 0.52769679, 0.53061224, 0.53061224, 0.54810496,
        0.53644315, 0.55102041, 0.54810496, 0.55393586, 0.56559767,
        0.57725948, 0.57725948, 0.57725948, 0.59766764, 0.6122449 ,
        0.60641399, 0.62099125, 0.62390671, 0.62099125, 0.62390671])] ]
[0.6355685131195334]
```

Keterangan kode program:

- `for i in range(60, len(train_data)):` digunakan untuk melakukan iterasi melalui data pelatihan (`train_data`) dengan memulai dari indeks ke-60 hingga indeks terakhir. Biasanya, ini digunakan saat Anda ingin membuat sekuens data berurutan dengan ukuran (panjang) tertentu yang akan digunakan sebagai input untuk model machine learning.

### 3.7.19 Show Shape

```
A_df.shape
```

```
(1083, 6)
```

Keterangan kode program:

- `A_df.shape` digunakan untuk mendapatkan dimensi (shape) dari DataFrame `A_df`, yaitu jumlah baris dan kolom yang ada dalam DataFrame. Hasil dari `A_df.shape` adalah sebuah tuple dengan dua elemen, di mana elemen pertama merupakan jumlah baris dan elemen kedua merupakan jumlah kolom dalam DataFrame.

### 3.7.20 Convert Train Data To Numpy Arrays

```
x_train,y_train=np.array(x_train),np.array(y_train)
```

Keterangan kode program:



- `x_train, y_train = np.array(x_train), np.array(y_train)`, digunakan untuk mengubah data pelatihan (fitur dan label) dari bentuk list menjadi bentuk array numpy. Dalam banyak kasus machine learning, library numpy sangat umum digunakan untuk manipulasi data dan operasi matriks.
- Setelah melakukan proses pembentukan `x_train` dan `y_train`, kita dapat mengubahnya menjadi array numpy untuk mempermudah proses analisis dan pemodelan.

### 3.7.21 Reshape

```
x_train.shape
```

```
(807, 60)
```

Keterangan kode program:

- `x_train.shape` digunakan untuk mendapatkan dimensi (shape) dari array `x_train`. Hasil dari `x_train.shape` adalah sebuah tuple dengan dua elemen, di mana elemen pertama merupakan jumlah baris (sampel) dan elemen kedua merupakan jumlah kolom (fitur) dalam array.
- Dalam konteks machine learning, `x_train` biasanya berisi data fitur yang akan digunakan sebagai input untuk melatih model atau algoritma machine learning. Bentuk (shape) dari `x_train` adalah penting untuk memastikan bahwa data fitur telah diubah dengan benar dan sesuai dengan ekspektasi model yang akan digunakan.

### 3.7.22 Train Shape

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

```
(807, 60, 1)
```

Keterangan kode program:

- `X_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))` berfungsi untuk mengubah bentuk (shape) dari array `x_train` dalam konteks pemrosesan data untuk model sequential dalam deep learning, khususnya untuk model recurrent neural network (RNN) atau convolutional neural network (CNN).
- Dalam pemrosesan data untuk model RNN atau CNN, kita sering membutuhkan data dalam bentuk tiga dimensi dengan format (jumlah\_sampel, panjang\_sequence, jumlah\_fitur). `X_train` adalah array yang menyimpan data fitur dalam bentuk ini.
- `x_train.shape` berfungsi untuk mendapatkan dimensi (shape) dari array `x_train`. Hasil dari `x_train.shape` adalah sebuah tuple dengan dua elemen, di mana elemen pertama merupakan jumlah baris (sampel) dan elemen kedua merupakan jumlah kolom (fitur) dalam array.



### 3.7.23 Build LSTM Model

```
model=Sequential()  
model.add(LSTM(50,return_sequences=True,input_shape=(x_train.shape[1],1)))  
model.add(LSTM(50,return_sequences=False))  
model.add(Dense(25))  
model.add(Dense(1))
```

Keterangan kode program:

- `model = Sequential()` digunakan untuk membuat objek model sequential dalam library Keras, yang akan digunakan untuk membangun model neural network dalam machine learning atau deep learning.
- Dalam Keras, model sequential adalah tipe model neural network yang paling umum dan paling sederhana. Model ini terdiri dari sejumlah lapisan (layers) yang tersusun secara berurutan, di mana keluaran dari satu lapisan menjadi masukan untuk lapisan berikutnya. Arsitektur sequential sangat cocok untuk model-model yang memiliki urutan sederhana dalam penyusunan lapisan-lapisannya.
- `model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)))` digunakan untuk menambahkan lapisan LSTM (Long Short-Term Memory) ke dalam model sequential dalam library Keras. LSTM adalah tipe lapisan rekuren dalam neural network yang digunakan untuk menangani data sekuensial, seperti data time series atau data berurutan lainnya.

Pada baris kode tersebut:

- `LSTM(50)` menunjukkan bahwa lapisan LSTM memiliki 50 unit neuron. Ini adalah jumlah neuron atau unit yang akan digunakan dalam lapisan LSTM. Jumlah unit ini bisa disesuaikan dengan kompleksitas masalah dan ukuran data yang Anda miliki.
- `return_sequences=True` menandakan bahwa lapisan LSTM akan menghasilkan keluaran untuk setiap waktu atau timestep dalam data sekuensial. Jika `return_sequences=False`, maka lapisan LSTM hanya akan menghasilkan keluaran untuk timestep terakhir dalam sekuens data.
- `input_shape=(x_train.shape[1], 1)` mendefinisikan bentuk input data untuk lapisan LSTM. Di sini, `x_train.shape[1]` adalah panjang dari setiap sekuens atau jumlah fitur dalam setiap timestep, dan `1` mengindikasikan bahwa data input memiliki satu dimensi. Dalam kasus ini, kita menggunakan sekuens data berdimensi satu, namun dalam beberapa kasus lain, sekuens data bisa memiliki lebih dari satu dimensi, seperti sekuens data gambar (RGB images) dengan bentuk (panjang, lebar, channel).

### 3.7.24 Compile The Model



```
model.compile(optimizer='adam',loss='mean_squared_error')
```

Keterangan kode program:

- `model.compile(optimizer='adam', loss='mean_squared_error')` digunakan untuk mengkompilasi model sequential dalam library Keras sebelum dilakukan proses pelatihan. Proses kompilasi ini menentukan bagaimana model akan melakukan pembelajaran dengan menggunakan fungsi optimisasi dan fungsi kerugian tertentu.

Pada baris kode tersebut:

- `optimizer='adam'` menunjukkan bahwa kita akan menggunakan algoritma optimisasi Adam untuk mengoptimalkan model saat dilakukan pelatihan. Adam adalah salah satu algoritma optimisasi yang populer dalam deep learning karena memiliki adaptivitas yang baik dan efisien untuk menemukan minimum global fungsi loss.
- `loss='mean_squared_error'` menunjukkan bahwa kita akan menggunakan mean squared error (MSE) sebagai fungsi kerugian (loss function) untuk melatih model. Fungsi MSE sangat cocok untuk tugas regresi, di mana kita ingin meminimalkan selisih kuadrat antara nilai prediksi dan nilai sebenarnya.

### 3.7.25 Train The Model

```
model.fit(x_train,y_train,batch_size=1,epochs=1)
```

```
807/807 [=====] - 15s 15ms/step - loss: 0.0057  
<keras.callbacks.History at 0x2a4a32841f0>
```

Keterangan kode program:

- `model.fit(x_train, y_train, batch_size=1, epochs=1)` digunakan untuk melatih model sequential yang telah dikompilasi menggunakan data pelatihan `x_train` dan `y_train`. Proses pelatihan ini dilakukan selama satu epoch (iterasi) dengan ukuran batch sebesar 1.

Pada baris kode tersebut:

- `x_train` adalah data fitur yang digunakan sebagai input untuk melatih model.
- `y_train` adalah data target yang merupakan nilai yang ingin diprediksi oleh model berdasarkan data fitur.
- `batch_size=1` menunjukkan bahwa pelatihan model akan dilakukan dengan satu sampel (data) pada setiap batch. Dalam hal ini, pelatihan dilakukan dengan metode stochastic gradient descent (SGD), di mana model akan diperbarui setiap kali melihat satu data.
- `epochs=1` menunjukkan bahwa proses pelatihan akan berlangsung selama satu epoch (iterasi). Satu epoch berarti seluruh data pelatihan akan dilihat oleh model satu kali dalam proses pelatihan.



### 3.7.26 Create The Testing Data Set

```
test_data=scaled_data[training_data_len-60:,:]
```

Keterangan kode program:

- `test_data = scaled_data[training_data_len-60:, :]` digunakan untuk mengambil data uji (test data) dari data skala (scaled\_data) setelah data pelatihan (training data) sebelumnya dipisahkan. Dalam konteks pemodelan time series atau data sekuensial, biasanya data uji digunakan untuk menguji performa model yang telah dilatih pada data pelatihan dan melihat sejauh mana model dapat memprediksi data masa depan yang belum pernah dilihat sebelumnya.

### 3.7.27 Create The Data Sets x\_test And y\_test

```
x_test=[]  
y_test=dataset[training_data_len:,:]  
for i in range(60,len(test_data)):  
    x_test.append(test_data[i-60:i,0])
```

Keterangan kode program:

- `x_test=[]` digunakan untuk mendefinisikan sebuah list kosong yang akan digunakan untuk menyimpan data uji (test data) pada kasus machine learning atau data yang akan digunakan sebagai fitur (features) dalam model ketika melakukan evaluasi atau prediksi.
- `y_test = dataset[training_data_len:, :]` digunakan untuk memisahkan data uji (test data) dari dataset utuh (biasanya berupa DataFrame) setelah data pelatihan sebelumnya dipisahkan. Dalam konteks machine learning, dataset biasanya dibagi menjadi dua bagian utama: data pelatihan dan data uji.
- `for i in range(60, len(test_data)):` digunakan untuk mengiterasi melalui data uji (test data) dengan jangkauan indeks dari 60 hingga panjang data uji (`len(test_data)`).
- `x_test.append(test_data[i-60:i, 0])` berguna untuk menambahkan data fitur dari data uji (test data) ke dalam list `x_test`. Dalam banyak kasus machine learning, data fitur harus diatur dengan format yang sesuai sebelum digunakan untuk evaluasi atau prediksi menggunakan model machine learning.

### 3.7.28 Convert The Data To A Numpy Array

```
x_test=np.array(x_test)
```

Keterangan kode program:



- `x_test=np.array(x_test)`, berguna untuk mengubah list `x_test` yang berisi sekuens data fitur dari data uji menjadi array numpy. Transformasi ini diperlukan untuk memastikan data fitur memiliki bentuk (shape) yang sesuai dengan model machine learning yang telah dilatih menggunakan data pelatihan (`x_train`).

### 3.7.29 Reshape

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

Keterangan kode program:

- `x_test=np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))`, digunakan untuk mengubah bentuk (shape) dari array `x_test` menjadi bentuk yang sesuai dengan input yang diperlukan oleh model sequential, terutama ketika menggunakan lapisan LSTM atau lapisan rekuren lainnya dalam Keras.

### 3.7.30 Get The Models Predicted Price Values

```
predictions=model.predict(x_test)
predictions=scaler.inverse_transform(predictions)
```

7/7 [=====] - 1s 12ms/step

Keterangan kode program:

- `predictions=model.predict(x_test)`, digunakan untuk melakukan prediksi pada data fitur `x_test` menggunakan model sequential yang telah dilatih sebelumnya.
- `predictions=scaler.inverse_transform(predictions)`, berguna untuk mengubah kembali prediksi yang telah dilakukan pada data fitur yang telah dinormalisasi (scaled) ke dalam skala aslinya sebelum dinormalisasi. Langkah ini diperlukan karena sebelum dilakukan pelatihan model, data fitur dan data target biasanya akan dinormalisasi untuk memastikan bahwa semua fitur memiliki rentang yang seragam, yang dapat membantu proses pelatihan model.

### 3.7.31 Get The Root Mean Squared Error (RMSE)

```
rmse=np.sqrt( np.mean((predictions - y_test)**2))
```

Keterangan kode program:

- `predictions=scaler.inverse_transform(predictions)` berguna untuk mengubah kembali prediksi yang telah dilakukan pada data fitur yang telah dinormalisasi (scaled) ke dalam skala aslinya sebelum dinormalisasi. Langkah ini diperlukan karena sebelum dilakukan pelatihan model, data fitur dan data target biasanya akan dinormalisasi untuk



memastikan bahwa semua fitur memiliki rentang yang seragam, yang dapat membantu proses pelatihan model.

### 3.7.32 Print RMSE

```
rmse
```

```
233.14250210648245
```

Keterangan kode program:

- RMSE (Root Mean Squared Error) berguna untuk mengukur seberapa baik model prediksi dalam memprediksi nilai target atau label yang sebenarnya pada data uji. Metrik evaluasi ini sangat umum digunakan dalam tugas regresi di bidang machine learning. RMSE memberikan gambaran tentang tingkat kesalahan prediksi model dalam satuan yang sama dengan variabel target.

Manfaat RMSE adalah sebagai berikut:

1. Evaluasi Performa Model: RMSE membantu kita dalam mengevaluasi performa model prediksi. Semakin kecil nilai RMSE, semakin baik model dalam memprediksi nilai target yang sebenarnya pada data uji.
2. Pembandingan Model: RMSE memungkinkan kita untuk membandingkan performa beberapa model yang berbeda dalam tugas regresi. Model dengan RMSE yang lebih rendah akan dianggap lebih baik dalam memprediksi nilai target.
3. Mengidentifikasi Kesalahan: RMSE memberikan informasi tentang kesalahan prediksi yang dibuat oleh model. Dengan mengevaluasi nilai RMSE, kita dapat mengidentifikasi data di mana model cenderung memberikan prediksi yang lebih buruk, sehingga dapat membantu dalam melakukan perbaikan atau penyesuaian.
4. Pengoptimalan Model: Dalam beberapa kasus, RMSE digunakan sebagai fungsi objektif (objective function) yang ingin dioptimalkan saat melatih model. Tujuan pelatihan adalah untuk mencari parameter model yang dapat menghasilkan nilai RMSE sekecil mungkin.
5. Interpretasi Hasil: Nilai RMSE yang lebih kecil membuat hasil prediksi lebih mudah diinterpretasikan. Misalnya, jika kita menggunakan RMSE dalam tugas prediksi harga rumah, maka RMSE akan memberikan gambaran tentang sejauh mana model memprediksi harga rumah dengan baik atau tidak.
6. Secara keseluruhan, RMSE adalah metrik penting yang digunakan untuk memahami seberapa baik model regresi dalam memprediksi nilai target pada data uji dan membantu kita membuat keputusan tentang performa model dan peningkatan yang mungkin diperlukan.

### 3.7.33 Train Prediction Dataset





```
train=data[:training_data_len]
valid=data[training_data_len:]
valid['Predictions']=predictions
```

C:\Users\hendr\AppData\Local\Temp\ipykernel\_13352\3270863671.py:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

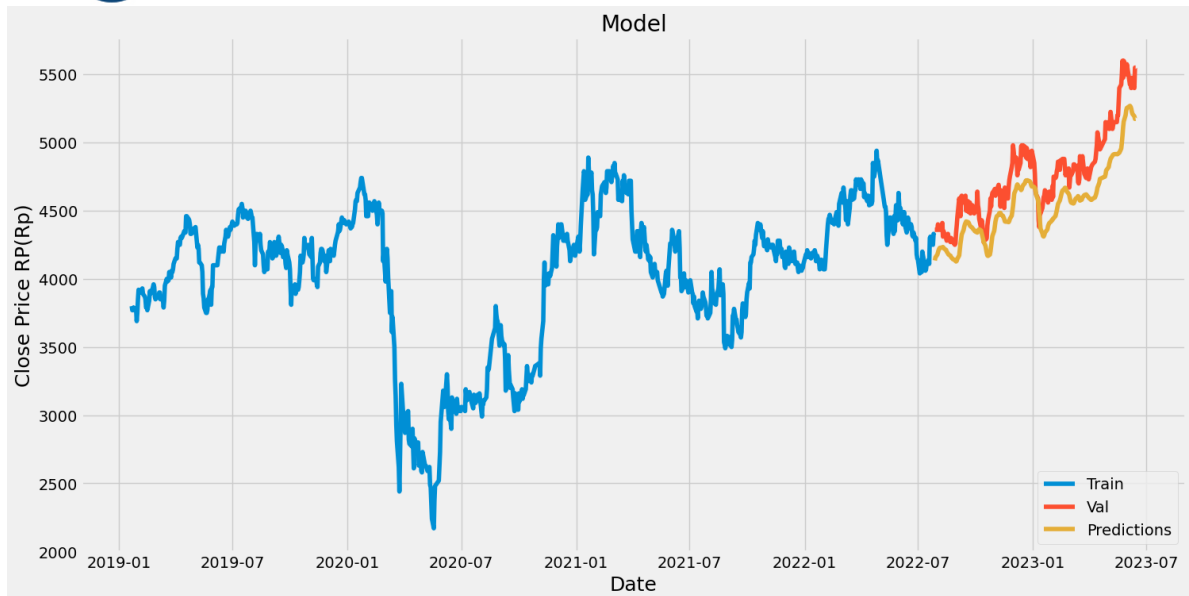
See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
valid['Predictions']=predictions

Keterangan kode program:

- `train=data[:training_data_len]` berguna untuk memisahkan data pelatihan dari dataset utuh (data). Lebih tepatnya, ini akan mengambil sejumlah baris data dari awal dataset hingga indeks `training_data_len-1`, yang akan digunakan sebagai data pelatihan.
- `valid=data[training_data_len:]` berguna untuk memisahkan data validasi dari dataset utuh (data). Lebih tepatnya, ini akan mengambil sejumlah baris data mulai dari indeks `training_data_len` hingga akhir dataset, yang akan digunakan sebagai data validasi.
- `valid['Predictions']=predictions` digunakan untuk menambahkan kolom baru dengan nama 'Predictions' ke dalam dataframe 'valid', dan mengisi kolom tersebut dengan nilai prediksi ('predictions') yang telah dihasilkan oleh model.
- Dalam banyak kasus, setelah model dilatih dan menghasilkan prediksi pada data validasi, kita ingin menyimpan hasil prediksi tersebut sebagai kolom baru di dataframe validasi. Hal ini berguna untuk membandingkan nilai prediksi dengan nilai sebenarnya dan melakukan analisis lebih lanjut terhadap performa model.

### 3.7.34 Visualize The Data

```
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date',fontsize=18)
plt.ylabel('Close Price RP(Rp)',fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Train','Val','Predictions'],loc='lower right')
plt.show()
```



Keterangan kode program:

- `plt.figure(figsize=(16,8))` digunakan untuk mengatur ukuran (dimensi) dari gambar (plot) yang akan digambarkan dengan menggunakan library matplotlib. Fungsi ini memungkinkan kita untuk mengontrol ukuran dari plot yang dihasilkan, sehingga kita dapat dengan mudah menyesuaikannya sesuai dengan kebutuhan tampilan grafik yang diinginkan.
- `plt.title('Model')` digunakan untuk menambahkan judul (title) pada plot yang dihasilkan menggunakan library matplotlib. Judul berfungsi untuk memberikan informasi singkat tentang isi atau maksud dari plot yang ditampilkan.
- `plt.xlabel('Date', fontsize=18)` digunakan untuk menambahkan label pada sumbu x (horizontal) dari plot yang dihasilkan menggunakan library matplotlib. Label sumbu x memberikan informasi tentang variabel atau data apa yang ditampilkan dalam sumbu x.
- `plt.ylabel('Close Price RP(Rp)', fontsize=18)` digunakan untuk menambahkan label pada sumbu y (vertikal) dari plot yang dihasilkan menggunakan library matplotlib. Label sumbu y memberikan informasi tentang variabel atau data apa yang ditampilkan dalam sumbu y.
- `plt.plot(train['Close'])` digunakan untuk menggambar garis plot dari data harga penutupan (Close) pada data pelatihan (train). Fungsi ini akan menampilkan perubahan harga penutupan dari waktu ke waktu, membentuk garis grafik yang terhubung antara titik-titik data harga penutupan.
- `plt.plot(valid[['Close','Predictions']])` digunakan untuk menggambar garis plot dari data harga penutupan (Close) pada data validasi (valid) dan data prediksi



(Predictions) yang dihasilkan oleh model. Fungsi ini akan menampilkan dua garis grafik yang terhubung antara titik-titik data harga penutupan dan prediksi.

- `plt.legend(['Train','Val','Predictions'], loc='lower right')` digunakan untuk menambahkan legenda (legend) pada plot yang dihasilkan menggunakan library matplotlib. Legenda berfungsi untuk memberikan keterangan atau label yang mengidentifikasi berbagai elemen atau garis plot yang ada dalam grafik.
- `plt.show()` digunakan untuk menampilkan plot yang telah digambar menggunakan library matplotlib. Fungsi ini memberikan perintah untuk menampilkan plot di layar setelah semua elemen plot selesai diatur dan siap untuk ditampilkan.

### 3.7.35 Show the valid and predicted prices

valid

	Close	Predictions
Date		
2022-07-28	4360.0	4135.095703
2022-07-29	4360.0	4157.719727
2022-08-01	4360.0	4177.507324
2022-08-02	4400.0	4193.425293
2022-08-03	4380.0	4209.616211
...	...	...
2023-06-08	5475.0	5226.413574
2023-06-09	5425.0	5210.356934
2023-06-12	5400.0	5192.645020
2023-06-13	5550.0	5173.699219
2023-06-14	5550.0	5174.261719

216 rows x 2 columns



Keterangan kode program:

- Dalam konteks kode sebelumnya, 'valid' merupakan sebuah dataframe yang digunakan untuk menyimpan data validasi. DataFrame adalah salah satu struktur data utama dalam library pandas yang digunakan untuk menyimpan dan mengorganisir data dalam bentuk tabel dua dimensi dengan baris dan kolom.

### 3.7.36 Get The Quote

```
apple_quote=A_df = yf.download('BBRI.JK')
```

```
[*****100%*****] 1 of 1 completed
```

Keterangan kode program:

- Kode `apple_quote = yf.download('BBRI.JK')`, berfungsi untuk mengunduh data historis harga saham dari pasar keuangan menggunakan library `yfinance` (`yf`) untuk saham BBRI (Bank Rakyat Indonesia) dengan simbol 'BBRI.JK' dari Yahoo Finance.

### 3.7.37 Create A New Dataframe

```
new_df=apple_quote.filter(['Close'])
```

Keterangan kode program:

- `new_df=apple_quote.filter(['Close'])` digunakan untuk membuat dataframe baru yang hanya berisi kolom 'Close' dari dataframe `apple_quote`. Pada dasarnya, fungsi filter pada dataframe digunakan untuk mengambil subset kolom tertentu dari dataframe yang ada.

### 3.7.38 Get The Last 60 Day Closing Price Values And Convert The Dataframe To An Array

```
last_60_days=new_df[-60:].values
```

Keterangan kode program:

- `last_60_days = new_df[-60:].values` berfungsi untuk mengambil 60 baris terakhir (60 hari terakhir) dari dataframe `new_df` yang berisi data historis harga penutupan saham BBRI, dan mengonversi data tersebut menjadi bentuk array NumPy

### 3.7.39 Scale The Data To be Values Between 0 And 1

```
last_60_days_scaled=scaler.transform(last_60_days)
```

Keterangan kode program:

- `last_60_days_scaled = scaler.transform(last_60_days)` berguna untuk melakukan penskalaan (scaling) data dari array `last_60_days` menggunakan objek `scaler`



yang telah dibuat sebelumnya. Skalasi data ini berguna untuk mengubah rentang nilai data menjadi skala yang diinginkan, biasanya dalam rentang tertentu seperti 0 hingga 1.

#### 3.7.40 Create An Empty List

```
X_test=[]
```

Keterangan kode program:

- `X_test=[]` berguna untuk mendefinisikan sebuah list kosong yang akan digunakan untuk menyimpan data tes (testing data). Dalam konteks pemodelan atau analisis data, list ini biasanya akan diisi dengan data yang akan diuji atau diprediksi oleh model yang telah dilatih.

#### 3.7.41 Append The Past 60 Days

```
X_test.append(last_60_days_scaled)
```

Keterangan kode program:

- `X_test.append(last_60_days_scaled)`, berguna untuk menambahkan data yang telah di-scaling (`last_60_days_scaled`) ke dalam list `X_test`. Dalam konteks pemodelan atau analisis data, hal ini berarti kita menambahkan data tes (testing data) yang telah diubah skala ke dalam list `X_test`, sehingga data tersebut dapat digunakan untuk melakukan prediksi menggunakan model yang telah dilatih sebelumnya.

#### 3.7.42 Convert The X\_test Data To A Numpy Array

```
X_test=np.array(X_test)
```

Keterangan kode program:

- `X_test=np.array(X_test)` berguna untuk mengubah list `X_test` menjadi array NumPy. Dalam analisis data atau pemodelan, seringkali lebih nyaman dan mudah bekerja dengan data dalam bentuk array NumPy daripada dalam bentuk list.

#### 3.7.43 Reshape

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

Keterangan kode program:

- `X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))` berguna untuk mengubah bentuk array `X_test` menjadi bentuk yang sesuai untuk digunakan sebagai input pada model recurrent neural network (RNN) atau Long Short-Term Memory (LSTM) dalam deep learning.

#### 3.7.44 Get The Predicted Scaled Price

```
pred_price=model.predict(X_test)
```



1/1 [=====] - 0s 22ms/step

Keterangan kode program:

- `pred_price = model.predict(X_test)` berguna untuk melakukan prediksi harga saham berdasarkan data testing `X_test` menggunakan model yang telah dilatih sebelumnya. Model yang digunakan adalah model neural network, seperti LSTM atau RNN, yang telah didefinisikan dan dilatih sebelumnya untuk memprediksi harga saham berdasarkan data historis harga saham.

### 3.7.45 Undo The Scaling

```
pred_price=scaler.inverse_transform(pred_price)
print(pred_price)
```

```
[[5149.7866]]
```

Keterangan kode program:

- `pred_price=scaler.inverse_transform(pred_price)` digunakan untuk mengubah kembali skala hasil prediksi ('`pred_price`') dari skala yang telah di-scaling sebelumnya ke skala aslinya. Fungsi `inverse_transform` dari objek `scaler` bertujuan untuk melakukan transformasi balik (revert) pada data yang telah diubah skala sebelumnya, sehingga hasil prediksi akan kembali ke skala harga saham aslinya.
- `print(pred_price)` digunakan untuk menampilkan hasil prediksi harga saham yang telah diubah kembali ke skala harga saham aslinya setelah dilakukan `inverse transform`. Dengan melakukan `print` pada variabel '`pred_price`', kita dapat melihat nilai hasil prediksi yang telah diubah kembali ke skala aslinya, sehingga dapat memahami prediksi harga saham yang dihasilkan oleh model secara lebih terperinci.

### 3.7.46 Get The Quote

```
apple_quote2=A_df = yf.download('BBRI.JK')
print(apple_quote2['Close'])
```

```
Date
2003-11-10    97.5
2003-11-11   100.0
2003-11-12   105.0
2003-11-13   105.0
2003-11-14   105.0
...
2023-07-03  5475.0
2023-07-04  5450.0
2023-07-05  5450.0
2023-07-06  5425.0
2023-07-07  5375.0
Name: Close, Length: 4879, dtype: float64
```



Keterangan kode program:

- Kode `apple_quote2 = yf.download('BBRI.JK')` digunakan untuk mengunduh data historis harga saham dari BBRI (Bank Rakyat Indonesia) dengan simbol 'BBRI.JK' dari pasar keuangan menggunakan library `yfinance` (`yf`).

Variabel `apple_quote2` akan berisi data historis harga saham BBRI dalam bentuk dataframe setelah proses unduhan selesai. DataFrame ini akan mencakup informasi seperti tanggal, harga pembukaan (Open), harga tertinggi (High), harga terendah (Low), harga penutupan (Close), volume perdagangan, dll.

- `print(apple_quote2['Close'])` berguna untuk menampilkan data historis harga penutupan saham BBRI dari dataframe `apple_quote2`.

DataFrame `apple_quote2` berisi data historis harga saham BBRI yang telah diunduh dari pasar keuangan menggunakan library `yfinance` (`yf`). Pada kolom 'Close' dari dataframe ini, terdapat harga penutupan saham BBRI pada setiap tanggal yang tersedia.

## BAB IV

### PENUTUP

#### 4.1 Kesimpulan

Dari penelitian dan eksperimen yang telah dilakukan di atas dapat disimpulkan bahwa, penutupan saham PT Bank Rakyat Indonesia (Persero) Tbk pada tanggal 21 Januari 2019 sampai dengan 14 Juni 2023 mengalami kenaikan yaitu dari Rp 3.800 menjadi sebesar Rp 5.475 dengan nilai *loss* sebesar 0.0057 atau 0.57% dengan *Root Mean Square Error* (RMSE) sebesar 233.14250210648245. Dapat dilihat pula bahwa saham penutupan PT Bank Rakyat Indonesia (Persero) Tbk dari awal berdiri tanggal 10 November 2003 sampai sekarang tepatnya tanggal 07 Juli 2023 juga mengalami kenaikan yang signifikan yaitu dari Rp 97.5 menjadi sebesar Rp 5.375.

#### 4.2 Saran

Dalam mempelajari *Deep Learning* yang meliputi *Recurrent Neural Network* (RNN), dan *Long Short-Term Memory* (LSTM) kita harus dapat memahami secara mendalam dan mengkaji sejauh mana capaian yang telah didapat serta apa saja permasalahan yang dihadapi dalam mempelajari *Deep Learning* (DL). Kita juga harus dapat memprediksi bagaimana masa depan *Deep Learning* serta tantangan yang dihadapinya di masa yang akan datang bagi kehidupan manusia.

### UCAPAN TERIMA KASIH

Terima kasih saya ucapkan kepada Bapak Catur Nugroho S.Kom., M.Kom yang telah memberikan ilmunya kepada saya dan teman-teman mahasiswa Universitas Siber Asia



(UNSIA) dalam mempelajari *Deep Learning* (DL) sehingga kami menjadi paham dan mengerti istilah-istilah serta seluk beluk mengenai pembelajaran *Deep Learning* selama semester lima ini. Kami juga telah mengikuti beberapa pertemuan yang telah diadakan oleh dosen kita mengenai pembelajaran tersebut. Pada pertemuan kesatu kita membahas tentang Pengenalan, pada pertemuan kedua membahas tentang Jaringan Saraf Tiruan, pada pertemuan ketiga membahas tentang *Convolution Neural Network* (CNN), kemudian pada pertemuan keempat kita membahas *Deep Unsupervised Learning* (DUL), pada pertemuan kelima membahas mengenai *Back Propagation Algorithm*, terus pada pertemuan keenam kita belajar tentang *Capsule Network*, pada pertemuan ketujuh membahas tentang *Recurrent Neural Network* (RNN) dan *Long Short-Term Memory* beserta praktikum nya, pada pertemuan ke sembilan kita membahas *Reinforcement Learning* (RL), pada pertemuan kesepuluh membahas *Self Organising MAP* (SOM) beserta praktikum RNN, kemudian pada pertemuan ke sebelas kita membahas tentang *Evolving Deep Neural Networks*, kemudian pada pertemuan kedua belas membahas tentang Seleksi dan Estimasi Model *Deep Learning*. Tak lupa juga saya ucapkan terima kasih kepada Bapak Muhammad Ikhwani Saputra S.Kom., M.Kom selaku dosen pembimbing, terima kasih semuanya.

#### DAFTAR PUSTAKA

1. Deep Learning . (2023). Dalam M. Catur Nugroho S.Kom., *Sesi-7 Recurrent Neural Network dan Long Short-Term Memory* (hal. 12-15). Jakarta: UNSIA.
2. Dr. Suyanto, S. M. (2019). *Deep Learning Modernisasi Machine Learning untuk Big Data*. Bandung: INFORMATIKA.
3. *Data Historis*. (2023, Juli 07). Diambil kembali dari Yahoo Finance: <https://finance.yahoo.com/quote/BBRI.JK/history?p=BBRI.JK>. Diakses tanggal 07 Juli 2023.
4. Trivusi. (2022, September 17). *Mengenal Algoritma Long Short Term Memory (LSTM)*. Diambil kembali dari Trivusi: <https://www.trivusi.web.id/2022/07/algoritma-lstm.html>. Diakses tanggal 09 Juli 2023.

#### Biodata Penulis



Hendro Gunawan, lahir di Jakarta, pada tanggal 1 Januari 1981. Menyelesaikan pendidikan Taman Kanak-Kanak (TK) dan Sekolah Dasar (SD) di Desa Sumber Sari Kecamatan Sine





Kabupaten Ngawi Jawa Timur, kemudian Sekolah Menengah Pertama (SMP), dan Sekolah Menengah Atas (SMA) di kabupaten Muara Enim Provinsi Sumatra Selatan. Saat ini sedang menempuh pendidikan S1 jurusan PJJ Informatika di Universitas Siber Asia (UNSA) Jakarta. Selain sebagai mahasiswa, penulis juga aktif sebagai karyawan di PT Indospring Tbk. Penulis juga menyukai bahasa pemrograman PHP, Java, Lua, dan Python. Dengan membaca makalah ini penulis berharap dapat menambah pengetahuan dan *skill* dalam bidang pembelajaran *Machine Learning*, terutama mengenai *Recurrent Neural Network* (RNN) dan *Long Short-Term Memory* (LSTM).

Penulis dapat dihubungi melalui:

Telp : 081259640815

Email : [hendro.gnwn@gmail.com](mailto:hendro.gnwn@gmail.com)  
[hendro.gnwn@outlook.com](mailto:hendro.gnwn@outlook.com)  
[hendro.gnwn@ymail.com](mailto:hendro.gnwn@ymail.com)

#### Link File



#### Link Video Youtube

<https://youtu.be/F91us2kAgjE>

#### Link File Github

<https://github.com/Hendro10/Prediksi-Dengan-LSTM-dan-Optimisasi-ADAM-Untuk-Memperkiraan-Harga-Emiten-Saham-Bank-Rakyat-Indonesia>



#### Link File Google Drive

[https://drive.google.com/file/d/1IN8PeO7qAJpDia4\\_uAB4c2rrjIzjSV41/view?usp=sharing](https://drive.google.com/file/d/1IN8PeO7qAJpDia4_uAB4c2rrjIzjSV41/view?usp=sharing)

#### Link Google Collab

<https://colab.research.google.com/drive/1f1HB9aINPEvjFDQ561AAsmJlk3phiImG#scrollTo=wOxUzcisQp6H>

**Tabel Nilai**

<b>Nilai</b>	<b>Tanda Tangan Dosen Pengampu</b>	<b>Tanda Tangan Mahasiswa</b>
	 <b>(Catur Nugroho, S.Kom., M.Kom)</b>	 <b>(Hendro Gunawan)</b>
	Diserahkan pada Tanggal :	Tanggal Mengumpulkan :
		09/07/2023