BAB II

TUNJAUAN PUSTAKA

2.1 Investasi Emas

Pada umumnya orang memilih berinvestasi dalam bentuk emas untuk memperoleh keuntungan (Maya Apriyanti, 2012: 37). Investasi emas dibedakan menjadi 2 yaitu, emas batangan dan saham emas (sertifikat). Kedua investasi tersebut mempunyai peluang dan resiko. Berinvestasi dalam emas batangan membutuhkan biaya untuk menyewa safe deposit box dan memungkinkan resiko lebih besar daripada berinvestasi emas dalam bentuk saham. Dalam berinvestasi emas berbentuk saham, investor hanya perlu keahlian membaca bursa saham.

Emas memiliki *supply* yang terbatas dan tidak mudah didapat, sementara permintaan terhadap emas tidak pernah berkurang, akibatnya harga emas cenderung mengalami kenaikan dari tahun ke tahun.

Pada kenyataan sehari-hari, harga emas tidak hanya bergantung kepada situasi permintaan dan penawaran. Harga emas juga dipengaruhi oleh situasi perekonomian secara keseluruhan. Faktor-faktor yang mempengaruhi harga emas sebagai berikut (Maya Apriyanti, 2012: 58):

1. Inflasi

   Inflasi terjadi jika nilai mata uang mengalami penurunan sehingga harga barang di pasaran mengalami kenaikan. Hal ini didorong oleh meningkatnya permintaan barang dan jasa yang kemudian diperparah

dengan tersendatnya distribusi barang. Kondisi inflasi seperti inilah yang mendorong naiknya harga emas.

2. Krisis Finansial

   Saat terjadi krisis finansial, orang lebih jeli dalam berinvestasi. Orang akan mencari keamanan dalam berinvestasi, sehingga mereka akan memilih investasi yang memberikan keuntungan. Dalam hal ini adalah berinvestasi dengan emas.

3. Naiknya Permintaan Emas di Pasaran

   Harga emas dapat naik karena permintaan akan emas dipasaran yang mengalami peningkatan.

4. Kurs Dollar

   Karena harga emas dihitung berdasarkan kurs dollar, maka jika dollar mengalami kenaikan, harga emas akan ikut terdorong naik.

5. Harga Minyak

   Pada korelasi antara emas dan minyak, mereka memiliki hubungan berbanding lurus. Jika harga minyak melonjak, hal ini menyebabkan produksi emas akan menurun dikarenakan biaya produksi semakin mahal. Penambangan emas sangatlah bergantung pada minyak untuk operasionalnya. Pada situasi lain, ketika harga minyak melonjak naik, kinerja perusahaan akan menurun. Produksi mereka pun menurun dan bisa jadi tidak mencapai target. Hal ini menyebabkan banyak para investor dari perusahaan tersebut yang melepas kepemilikan sahamnya. Hal ini menyebabkan harga saham perusahaan itu pun melorot. Para investor pun

mencari laternatif lain untuk mengembangkan financial mereka, salah satunya emas. Permintaan yang tinggi terhadap emas di saat para investor melepas kepemilikan sahamnya menyebabkan harga emas merangkak naik.

6. Situasi Politik

   Kenaikan harga emas pada akhir tahun 2002 dan awal tahun 2003 terjadi karena sekutu yang dikomando AS akan melakukan serangan ke Iraq. Pelaku pasar beralih investasi dari pasar uang dan pasar saham ke investasi emas sehingga permintaan emas melonjak tajam.

2.2 Data *Time Series*

Time Series (Deret Waktu) adalah serangakaian nilai-nilai variabel yang disusun berdasarkan waktu, Analisis *time series* mempelajari pola gerakan nilainilai variabel pada satu interval waktu seperti mingguan, bulanan, tahunan yang teratur. Metode *time series* didasarkan pada asumsi bahwa pola lama akan terulang. Manfaat dari analisis *time series* dapat diperoleh ukuran-ukuran yang dapat digunakan untuk membuat keputusan pada saat ini, untuk peramalan dan untuk merencanakan masa depan (Jayanti, 2013).

2.3 Statistika Deskriptif

Statistika Deskriptif merupakan suatu metode yang berhubungan dengan mengumpulkan dan menyajikan data sehingga data tersebut mudah dipahami dan dapat memberikan informasi yang berguna. Statistika deskriptif mampu memberikan infiormasi yang ringkas dari banyaknya data awal sehingga mudah

dipahami dan di cerna oleh orang umum. Hasan (2004:185) menjelaskan bahwa analisis deskriptif merupakan bentuk analisis data penelitian untuk menguji generalisasi hasil penelitian berdasarkan satu sampel. Analisa deskriptif ini dilakukan dengan pengujian hipotesis deskriptif. Hasil analisisnya adalah apakah hipotesis penelitian dapat digeneralisasikan atau tidak. Jika hipotesis nol (H_0) diterima, berarti hasil penelitian dapat digeneralisasikan. Analisis deskriptif ini menggunakan satu variabel atau lebih tapi bersifat mandiri, oleh karena itu analisis ini tidak berbentuk perbandingan atau hubungan.

Selanjutnya Hasan (2001:7) menjelaskan bahwa statistik deskriptif atau statistik deduktif adalah bagian dari statistik mempelajari cara pengumpulan data dan penyajian data sehingga muda dipahami. Statistik deskriptif hanya berhubungan dengan hal menguraikan atau memberikan keterangan-keterangan mengenai suatu data atau keadaan atau fenomena. Dengan kata statistik deskriptif berfungsi menerangkan keadaan, gejala, atau persoalan. Beberapa penyajian statistika deskriptif adalah penyajian statistika deskriptif secara visualisasi dan secara pemusatan data. Beberapa contoh penyajian statistika deskriptif adalah histogram, *polygon*, *pie chart*, grafik garis, dan lain sebagainya.

2.3.1 Penyajian Statistika Deskriptif Secara Ukuran Pemusatan Data

Statistika deskriptif yang biasa digunakan dalam menggamarkan kondisi suat data adalah ukuran pemusatan data. Terdapat tiga jenis ukuran pemusatandata (tendensi *central)* yaitu (Tiyas, 2019):

1. *Mean*

   Mean atau Rata-rata hitung adalah suatu metode yang paling banyak dipakai dalam menunjukan ukuran tendensi sentral. *Mean* ini dihitung dengan menjumlahkan seluruh nilai data pengamatan lalu dibagi dengan banyaknya data.

2. Median

   Median adalah nilai yang membagi himpunan pengamatan menjadi dua bagian yang sama besar atau 50% dari pengamatan yang berada di bawah median serta 50% lagi berada di atas median. Median dari n pengukuran/ pengamatan x1, x2, …, xn merupakan suatu nilai pengamatan yang berada di tengah gugus data sesudah data tersebut diurutkan. Jika banyaknya pengamatan (n) ganjil, median berada tepat ditengah gugus data, sementara jika n genap, median didapatkan dengan cara interpolasi. Yakni cara di mana rata-rata dari dua data yang berada di tengah gugus data.

3. Modus

   Modus adalah suatu data yang paling sering muncul atau terjadi. Untuk menentukan adanya modus, pertama kali dengan menyusun data dalam urutan meningkat atau sebaliknya. Lalu diikuti dengan menghitung frekuensinya. Nilai yang frekuensinya paling besar atau sering muncul itulah yang dinamakan sebagai modus. Modus dipakai baik untuk tipe data numerik maupun data kategoris.

2.4 Prediksi

Prediksi adalah memperkirakan keadaan dimasa yang akan datang melalui pengujian keadaan dimasa lalu. Dalam kehidupan sosial segala sesuatu itu serba tidak pasti dan sukar diperkirakan secara tepat, sehingga diperlukan prediksi. Prediksi yang dibuat selalu diupayakan agar dapat meminimumkan pengaruh ketiadakpastian ini terhadap sebuah masalah. Dengan kata lain prediksi bertujuan mendapatkan peramalan yang bisa meminimumkan kesalahan meramal (*forecast error*) yang biasanya diukur dengan *mean square error*, *mean absoulute error*, dan sebagainya. (Makridakis, 1999).

Baik tidaknya suatu prediksi yang disusun selain ditentukan oleh metode yang digunakan, juga ditentukan oleh baik tidaknya informasi yang digunakan. Selama informasi yang digunakan tidak dapat menyakinkan untukmendapat hasil yang bagus, hasil peramalan yang disusun juga akan sukar dipercaya ketepatannya. Keberhasilan dari suatu prediksi sangat ditentukan oleh:

a. Pengetahuan teknik tentang pengumpulan informasi (data) masa lalu, dapat ataupun informasi tersebut bersifat kuantitatif.
b. Teknik dan metode yang tepat dan sesuai dengan pola data yang dikumpulkan.

Gambaran perkembangan pada masa lalu yang akan datang diperoleh dari hasil analisa data yang didapat dari penelitian yang dilakukan. Perkembangan pada masa depan merupakan perkiraan apa yang akan terjadi, sehingga dapat dikatan bahwa peramalan selalu diperlukan dalam penelitian. Ketepatan penelitian merupakan hal yang penting, walaupun demikian perlu diketahui bahwa sesuatu ramalan selalu ada

unsur kesalahannya, sehingga yang perlu diperhatikan adalah usaha untuk memperkecil kesalahan dari ramalan tersebut.

2.5 *Artificial Neural Networks*

Artificial Neural Networks (ANN) atau juga disebut Jaringan Saraf Tiruan (JST) adalah teknik pembelajaran mesin yang terinspirasi oleh jaringan saraf biologis di otak manusia. Algoritma ANN itu sendiri ditemukan oleh Warren McCulloch dan Walter Pitts pada tahun 1943 yang memperkenalkan jaringan saraf, Warren McCulloch dan Walter Pitts menciptakan perangkat jaringan elektronik berdasarkan *neuron* dan menujukan bahwa jaringan sederhana semacam ini bahkan dapat menghitung hampir semua logika atau aritmatika fungsi. Tujuan dari ANN ini yaitu untuk mengenali pola dari data yang ada (Ananta,2017).

ANN terdiri dari sejumlah prosesor sangat sederhana dan saling berhubungan yang disebut *neuron* (Kurniawansyah, 2018). Penentuan jumlah lapisan dan jumlah *neuron* di lapisan tersembunyi dan koneksi di antara *neuron* tersebut adalah hal yang penting (Saritas & Yasar, 2019). *Neuron* mempunyai karakteristik yang sama dalam ANN, terdiri dalam kelompok-kelompok yang disebut *layer*. *Neuron-neuron* dalam satu *layer* terhubung dalam *layer-layer* lainnya yang berdekatan. Kekuatan hubungan antar *neuron* yang berdekatan direpresentasikan dalam kekuatan hubungan atau bobot (Dharma, Putera, & Ardana, 2011).

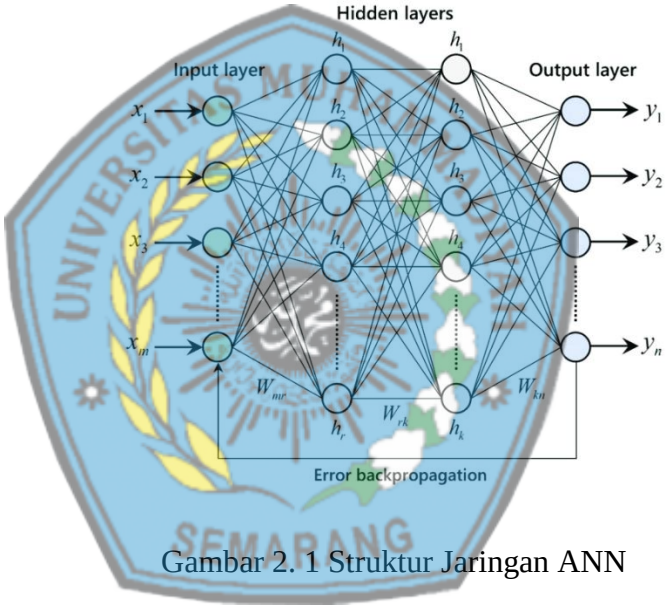
Struktur ANN terdiri dari sejumlah lapisan *input layer* dan *output layer*, yang terkoneksi dan pada setiap koneksinya terdapat *weight* yang bisa diubah-ubah dengan tujuan untuk mendapatkan hasil prediksi sesuai dengan yang diinginkan,

serta (dalam kebanyakan kasus) *hidden layer* yang terdiri dari unit yang mengubah input menjadi sesuatu yang dapat digunakan oleh lapisan output. Berikut merupakan lapisan yang ada dalam ANN:

- *Input layer* (Lapisan masukan): Lapisan terluar yang menghubungkan sumber data ke dalam jaringan pemrosesan. Setiap masukan akan merepresentasikan variabel-variabel bebas yang nantinya akan berpenagruh terhadap keluaran.
- *Hidden layer* (Lapisan tersembunyi): Lapisan perambat variable-variable input untuk mendapatkan hasil ou tput yang sesuai dengan keinginan. Pada ANN multi layer dapat memiliki satu atau lebih *hidden layer*.
- *Output layer* (Lapisan keluaran): Lapisan yang merupakan hasil dari pemrosesan ANN. Output yang dihasilkan dipengaruhi oleh bobot, jumlah lapisan tersembunyi (*hidden layer*), dan fungsi aktifasi yang diterapkan.

Input, *output*, dan *hidden layer* adalah alat yang sangat baik untuk menemukan pola yang terlalu rumit atau banyak bagi *programmer* untuk mengekstraksi dan mengajar mesin untuk mengenali. Sementara jaringan saraf (juga disebut "*perceptrons*") telah ada sejak 1940-an, hanya dalam beberapa dekade terakhir di mana mereka telah menjadi bagian utama dari kecerdasan buatan. Hal ini disebabkan oleh temuan teknik yang disebut "*backpropagation*" yang memungkinkan jaringan untuk menyesuaikan lapisan neuron tersembunyi mereka dalam situasi di mana hasilnya tidak sesuai dengan harapan peneliti misalnya seperti jaringan yang dirancang untuk mengenali kucing, tapi mengidentifikasi kelinci. (Larasati, 2020).

Suatu informasi (α) diterima input *layer* menggunakan bobot kedatangan (w) tertentu. Kemudia akan dilakukan penjumlahan bobot pada *hidden layer*. Selanjutnya hasil dari penjumlahan tersebut dibandingkan dengan nilai ambang (*threshold*). Jika nilai melewati ambang batas, maka akan diteruskan ke *output layer*, sedangkan jika nilai tidak melewati ambang batas, maka tidak akan diteruskan ke *output layer* (Habibi & Riksakomara, 2017). Berikut Gambar 2.1 yang menunjukkan struktur jaringan dari ANN.



Gambar 2. 1 Struktur Jaringan ANN

Sumber: (Ghiffary, 2020)

Layer input terdiri dari *neuron-neuron* yang menerima sebuah input dari lingkungan luar, *Input* disini dapat berupa penggambaran dari suatu masalah. Layer tersembunyi (*hidden layer*) terdiri dari *neuron-neuron* yang menerima masukan dari input layer dan kemudian membawa output ke layer berikutnya. Lapisan output disebut unit-unit output, terdiri dari neuron-neuron yang menerima output dari *hidden layer* dan mengirimkannya kepada pemakai (Dharma, Putera, & Ardana, 2011).

Langkah penting dalam pengembangan sebuah model ANN adalah penentuan bobot matrik melalui pelatihan (*training*). Ada dua tipe mekanisme *training* yaitu *supervised training* dan *unsupervised training*. *Supervised training* memerlukan supervisi dari luar untuk memandu proses *training*. Algoritma ini menggunakan sejumlah pasangan data input-output yang dipergunakan sebagai contoh, dimana data yang dipergunakan sebagai contoh sebaiknya menggunakan data yang sudah diketahui kebenarannya. Output dari jaringan lalu dibandingkan dengan data output yang diharapkan untuk mendapatkan selisih antara output perkiraan dengan output sebenarnya. Selisih inilah yang dipergunakan untuk mengubah bobot jaringan sehingga diperoleh output yang sama atau mendekati target. Mekanisme sebuah *supervised training* yang disebut algoritma *backpropagation training* umumnya banyak digunakan dalam aplikasi-aplikasi *enginering*. Karena ANN tidak mempertimbangkan fisik dari permasalahan, ANN merupakan sebuah model *blackbox* namun dapat mendeteksi proses fisik dalam model ANN yang sudah di*training* (Dharma, Putera, & Ardana, 2011).

Pemodelan jaringan pada ANN terdapat 3 macam, (Ghiffary, 2020) yaitu sabagai berikut:

1. *Single Layer* dimana *neuron* disusun dalam bentuk lapisan (*layer*). Pembentukan ANN yang paling sederhana yaitu *single layer*. Cara kerja dari *single layer*, input layer yang berasal dari sumber node di proyeksikan langsung ke output layer dari neuron (node komputasi), tetapi tidak berlaku sebaliknya. Permodelan ini merupakan jenis jaringan feedforward yang dapat dilihat pada gambar 2. 15. Pada gambar tersebut input dan output

memiliki 4 node, namun yang dimaksud dengan single layer yaitu output dari jaringan, sedangkan inputnya tidak memiliki pengaruh karena pada saat melakukan input tidak terjadi proses komputasi.

2. *Multi Layer*, pada *single layer* apabila terdapat tambahan satu atau dua *hidden layer* maka jaringan akan terganggu karena *input* dan *output* dari jaringan tidak dapat melihat *hidden layer* yang di masukkan. Sehingga memerlukan jaringan yang bisa menampung nya yaitu bernama *multi layer*. Cara kerja *multi layer* adalah *input layer* menyuplai *input* vektor pada jaringan, kemudian input yang dimasukkan melakukan komputasi pada *layer* yang kedua, lalu *output* dari *layer* yang kedua digunakan sebagai *input* dari *layer* yang ketiga dan seterusnya.
3. *Recurrent Network*, terbentuk karena pada jaringan *single layer* dan *multi layer* harus memiliki *feedback* untuk dirinya sendiri pada setiap *loop* jaringan nya, pada *reccurent network* jaringan tidak memerlukan *feedback* untuk dirinya sendiri melainkan *feedback* dari *input* yang digunakan.

Namun pada *Artificial Neural Network* tidak dapat mengingat informasi yang diberikan sebelumnya. Hal ini diatasi oleh *Recurrent Neural Network* (RNN). RNN adalah jaringan dengan *loop* di dalamnya, memungkinkan informasi untuk bertahan (Larasati, 2020).

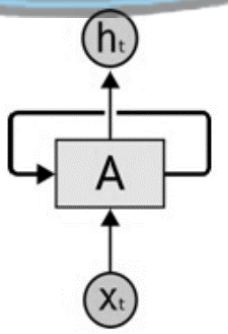
2.6 *Recurrent Neural Network*

RNN adalah sejenis jaringan saraf tiruan yang memiliki kemampuan yang untuk melihat korelasi tersembunyi yang terjadi pada data dalam aplikasi untuk

pengenalan suara, pemrosesan bahasa alami, dan prediksi deret waktu. RNN sangat baik untuk masalah pemodelan urutan dengan beroperasi pada informasi input serta jejak informasi yang diperoleh sebelumnya karena koneksi berulang (Tian & dkk, 2018).

Pada umumnya, dalam kehidupan sehari hari manusia tidak membuat keputusan secara tunggal setiap saat. Manusia akan memperhitungkan masa lalu dalam membuat sebuah keputusan. Cara berpikir seperti ini yang menjadi dasar dari pengembangan Recurrent neural network. Sama seperti analogi tersebut, RNN tidak membuang begitu saja informasi dari masa lalu dalam proses pembelajarannya. Hal inilah yang membedakan RNN dari *Artificial Neural Network* biasa (Lapian, Osmond, & Saputra, 2018).

Recurrent Neural Network (RNN) pertama kali dikembangkan oleh Jeff Elman pada tahun 1990. RNN merupakan variasi dari *Artificial Neural Network* (ANN) yang di rancang khusus untuk memproses data yang bersambung atau berurutan. RNN biasa digunakan untuk menyelesaikan masalah dengan data *time series*.

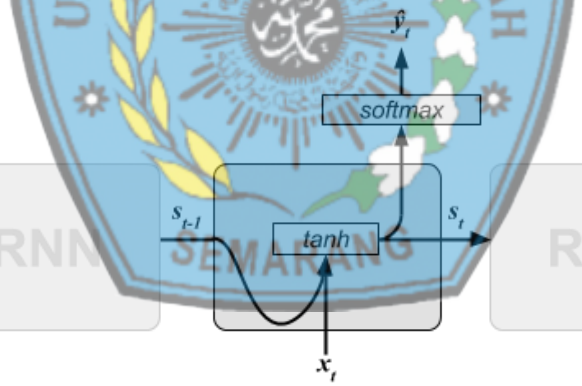


Gambar 2. 2 Proses perulangan informasi pada RNN

Sumber: (Colah, 2015)

Pada gambar 2.2 di atas dapat dijelaskan, bahwa struktur dari RNN akan lebar tengahnya selebar panjang pola data yang ingin di pelajari oleh RNN karena itu dikatakan RNN di desain khusus untuk meng*handle* data berurutan. *Input* x_t menghasilkan *output* h_t.

Recurrent Neural Network (RNN) terdiri dari unit *input*, unit *output*, dan unit tersembunyi (*hidden*). Model RNN pada dasarnya memiliki aliran informasi satu arah dari unit input ke unit tersembunyi, dan sintesis aliran informasi satu arah dari unit tersembunyi sementara sebelum ke unit tersembunyi waktu saat ini. Unit tersembunyi dapt dilihat sebagai penyimpanan seluruh jaringan, yang mengingat informasi dari ujung ke ujung (Yin, 2017). Seperti yang ditunjukkan pada Gambar 2. 3.



Gambar 2. 3 Layer tanh pada RNN

Sumber: (IndoML, 2018)

Persamaan tanh yang dapat diperoleh dari gambar 2.3 diuraikan pada persamaan sebagai berikut.

$$S_t = tanh(U\,.x_t + W.S_{t-1}) \quad (2.1)$$

$$\hat{y}_t = softmax(V.S_t) \quad (2.2)$$

Dengan S_t adalah memori jaringan pada waktu t; U, W, dan V adalah matriks bobot berbagi di setiap lapisan; X_t dan O_t mewakili input dan output pada waktu t; dan f (.) dan g (.) mewakili fungsi nonlinear.

Pada model RNN sinyal dapat mengalir secara *forward* dan *backward* secara berulang. Untuk bisa melakukan hal tersebut, maka ditambahkan sebuah *layer* baru yang disebut dengan *context layer*. Selain melewati *input* antar *layer*, *output* dari setiap *layer* juga menuju ke *context layer* untuk digunakan sebagai inputan pada timestep berikutnya. RNN menyimpan informasi di *context layer*, yang membuatnya dapat mempelajari urutan data dan menghasilkan output atau urutan lain. Jika ditarik kesimpulan maka dapat dikatakan bahwa RNN memiliki memori yang berisikan hasil rekaman informasi yang dihasilkan sebelumnya (Juanda, Jondri, & Rohmawati, 2018).

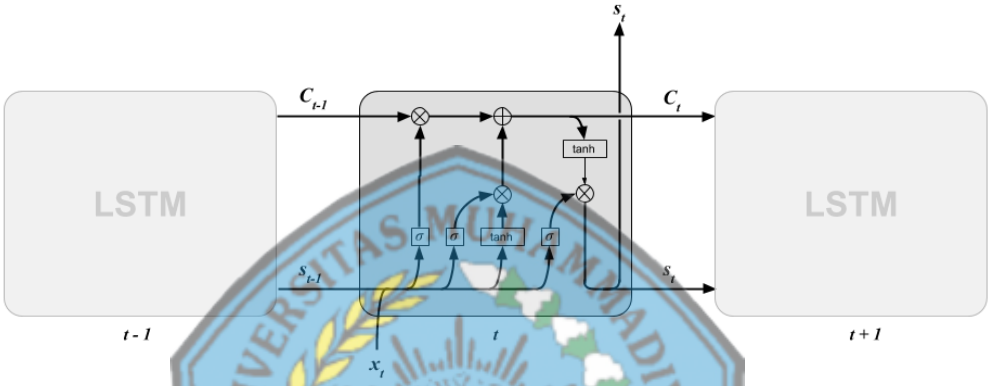
Berbeda dengan koneksi bobot yang dibangun antara lapisan dalam jaringan saraf dasar, RNN dapat menggunakan status *internal* (memori) untuk memproses urutan *input*. Status tersembunyi menangkap informasi pada titik waktu sebelumnya, dan *output* berasal dari waktu saat ini dan memori sebelumnya. RNN berkinerja baik ketika output dekat dengan input terkait informasi dari node sebelumnya diteruskan ke node berikutnya. Secara teori, RNN juga mampu menangani ketergantungan jangka panjang. Namun, dalam aplikasi praktis, RNN tidak dapat menyimpan informasi sebelumnya dengan baik ketika interval waktu lama karena masalah gradien menghilang. Untuk mengatasi kelemahan ini dan meningkatkan kinerja RNN, sebuah tipe khusus arsitektur RNN yang disebut LSTM diusulkan oleh Hochreiter & Schmidhuber (1997) (Tian & dkk, 2018).

2.7 *Long Short Term Memory*

Long Short Term Memory (LSTM) merupakan pengembangan dari *Recurrent Neural Network* (RNN) dengan mengatasi salah satu kekurangan RNN yaitu kemampuan pengelolaan informasi dalam periode yang lama. Diusulkan oleh Sepp Hochreiter dan Jurgen Schmidhuber pada tahun 1997, LSTM banyak dipilih untuk prediksi berbasis waktu atau time-series karena dikenal lebih unggul dan handal dalam melakukan prediksi dalam waktu lama dibanding algoritma lain (Zahara, Sugianto, & Ilmiddafiq, 2019). Satu keuntungan dari model LSTM dibandingkan dengan model RNN adalah kemampuan untuk menangkap struktur *autoregresif* dengan panjang yang sewenang-wenang. RNN memerlukan spesifikasi sebelumnya tentang berapa banyak '*feedback loops*' yang akan abda, padahal sebenarnya tidak diperlukan saat berhadapan dengan jaringan LSTM (Hansson, 2017).

Salah satu variasi dari *Recurrent Neural Network* yang diciptakan untuk menghindari masalah ketergantungan dalam waktu jangka panjang pada *Recurrent Nueral Network* (RNN) adalah LSTM. LSTM baik dalam mengingat informasi untuk waktu yang lama. Pada RNN perulangan jaringan hanya menggunakan satu layer sederhana, yaitu layer tanh. (Habibie, 2018). Karena lebih banyak informasi sebelumnya dapat mempengaruhi akurasi model, LSTM menjadi pilihan penggunaan yang wajar. Modul LSTM khas yang disebut modul berulang memiliki empat modul lapisan jaringan saraf berinteraksi dengan cara yang unik seperti yang ditunjukkan pada Gambar 2.4. Komponen dasar LSTM adalah status sel, sebuah baris yang berjalan dari *memory cell* sebelumnya (S_{t-1}) ke *memory cell* saat ini (S_t). *Memory cell* adalah garis horizontal yang menghubungkan semua *output layer* pada

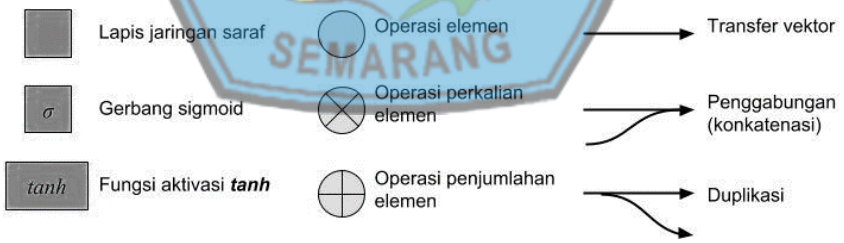
LSTM. Dengan adanya jalur tersebut, suatu nilai *memory cell* yang lama akan dengan mudah diteruskan ke *memory cell* yang baru dengan sedikit sekali modifikasi. Ini memungkinkan informasi mengalir lurus ke bawah. Jaringan dapat menentukan jumlah informasi sebelumnya mengalir. Itu dikendalikan melalui lapisan pertama (σ_1) (Kumar & dkk, 2018).



Gambar 2. 4 Struktur LSTM

Sumber: (IndoML,2018)

Notasi yang digunakan pada Gambar 2.4 adalah sebagai berikut: (IndoML,2018)



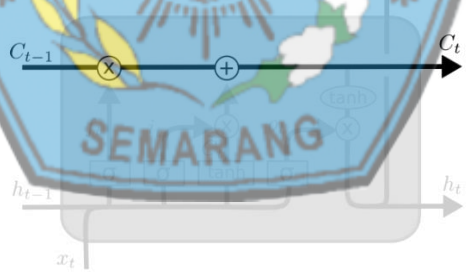
Gambar 2. 5 Notasi dalam diagram LSTM

Sumber: (IndoML, 2018)

Pada diagram di atas, setiap garis membawa seluruh vektor, dari output satu simpul (*node*) ke input yang lain. Lingkaran merah muda mewakili operasi elemen, seperti penambahan atau perkalian elemen vektor, sedangkan kotak kuning adalah lapis jaringan saraf (mengandung parameter dan bias) yang bisa belajar. Dua garis

yang bergabung menandakan penggabungan dua matriks/vektor, sementara garis berpisah menandakan kontennya disalin dan salinannya pergi ke simpul yang berbeda.

Model LSTM menyaring informasi melalui struktur gerbang untuk mempertahankan dan memperbarui keadaan sel memori. Struktur pintunya mencakup *input gate*, *forget gate*, dan *output gate*. Setiap sel memori memiliki tiga lapisan *sigmoid* dan satu lapisan *tanh* (Qiu, Wang, & Zhou, 2020). Informasi baru yang akan disimpan dalam status sel dihitung menggunakan dua lapisan jaringan. Sebuah lapisan *sigmoid* (σ_2) itu memutuskan nilai untuk memperbarui (I_t) dan *tanh layer* ϕ_1 yang mengembangkan vektor nilai kandidat baru ($\tilde{S}_t$). Kunci utama dari LSTM adalah *cell state*. *Cell state* adalah garis horizontal yang menghubungkan semua output layer pada LSTM seperti terlihat pada Gambar 2.6.

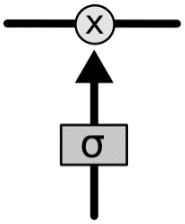


Gambar 2. 6 *Cell state* pada LSTM

Sumber: (Colah, 2015)

LSTM itu sendiri memiliki kemampuan untuk menambah dan menghapus informasi dari *cell state*. Kemampuan ini disebut dengan *gates*. *Gates* sebagai pengatur apakah informasi akan diteruskan atau diberhentikan. *Gates* terdiri dari

sigmoid layer dan *pointwise multiplication operation* seperti yang terlihat pada gambar 2.7.



Gambar 2. 7 Sigmoid *layer* pada LSTM

Sumber: (Colah, 2015)

Output dari sigmoid *layer* adalah angka 1 atau 0 yang menunjukkan apakah informasi tersebut akan diteruskan atau diberhentikan. Angka 0 menunjukkan bahwa tidak ada informasi yang akan diteruskan, sedangkan angka 1 menunjukkan bahwa semua informasi akan diteruskan. Persamaan sigmoid dan tanh diuraikan pada persamaan 2.3 dan 2.4

$$(x) = \frac{1}{1+e^{-x}} \quad (2.3)$$

$$\tanh(x) = 2\sigma(2x) - 1 \quad (2.4)$$

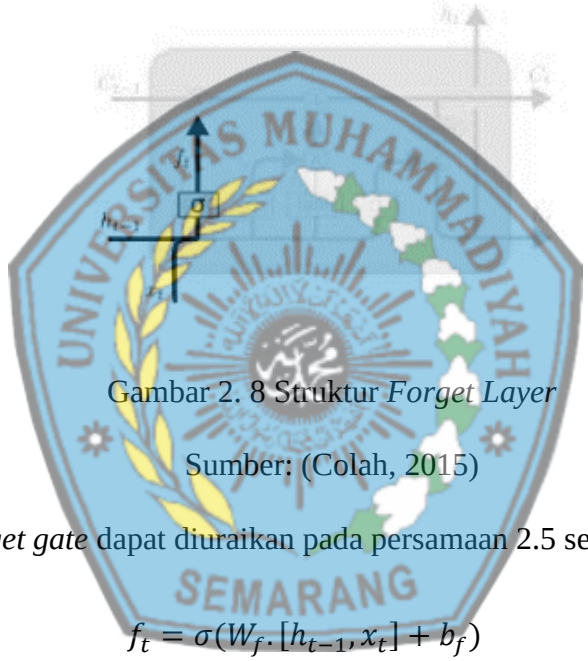
Dimana:

x = data input

e = konstanta matematika

LSTM memiliki 3 jenis gates diantaranya adalah *forget gate*, *input gate*, dan *output gate*. *Forget gate* adalah *gate* yang memutuskan informasi mana yang akan dihapus dari *cell*. *Input gate* adalah *gate* yang memutuskan nilai dari *input* untuk di diperbarui pada *state* memori. *Output gate* adalah *gate* yang memutuskan apa yang

akan dihasilkan output sesuai dengan *input* dan memori pada *cell*. Langkah – langkah panduan jalannya metode LSTM, langkah pertama adalah LSTM memutuskan informasi apa yang akan dihapus dari *cell state*. Keputusan ini dibuat oleh sigmoid *layer* yang bernama "*forget gate layer*". *Forget gate layer* akan memproses h_{t-1} dan x_t sebagai *input*, dan menghasilkan output berupa angka 0 atau 1 pada cell state C_{t-1} seperti yang terlihat pada gambar 2.8.



Gambar 2. 8 Struktur *Forget Layer*

Sumber: (Colah, 2015)

Persamaan *forget gate* dapat diuraikan pada persamaan 2.5 sebagai berikut:

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f) \quad (2.5)$$

Dimana:

f_t = *forget gate*

σ = fungsi sigmoid

W_f = nilai *weight* untuk *forget gate*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_f = nilai bias pada *forget gate*

Nilai *weight* dapat diuraikan pada persamaan 2.6 sebagai berikut:

$$W = \left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right) \quad (2.6)$$

Dimana:

W = *weight*

d = jumlah data

Langkah selanjutnya adalah menentukan apakah informasi akan disimpan pada sel. Pertama, sebuah layer sigmoid bernama "*input gate layer*" menentukan nilai mana yang akan diperbaharui. Selanjutnya, sebuah layer tanh membuat vector dari nilai kandidat baru C_t, yang dapat ditambah ke state. Langkah selanjutnya, kedua layer ini akan dikombinasikan untuk memperharui state.

Gambar 2. 9 Struktur *Input gate layer* & *tanh layer* (*Remember gate*)

Sumber: (Colah, 2018)

Persamaan *input gate* dapat diuraikan pada persamaan 2. 7 sebagai berikut:

$$I_t = \sigma(W_I.[h_{t-1}, x_t] + b_I) \quad (2.7)$$

Dimana:

I_t = *input gate*

σ = fungsi sigmoid

W_I = nilai *weight* untuk *input gate*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_I = nilai bias pada *input gate*

Persamaan kandidat baru dapat diuraikan pada persamaan 2.8 sebagai berikut:

$$\tilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \quad (2.8)$$

Dimana:

$\tilde{C}_t$ = nilai baru yang dapat ditambahkan ke *cell state*

$tanh$ = fungsi tanh

W_C = nilai *weight* unutk *cell state*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_C = nilai bias pada *cell gate*

Selanjutnya, state lama akan diperbaharui, C_{t-1} ke state sel baru C_t. Kemudian, f_t akan dikalikan dengan state lama dengan mengabaikan informasi yang sudah dilupakan sebelumnya. Lalu, I_t ditambahkan dengan C_t.

$$C_t = f_t \times C_{t-1} + I_t \times \tilde{C}_t \quad (2.9)$$

Dimana:

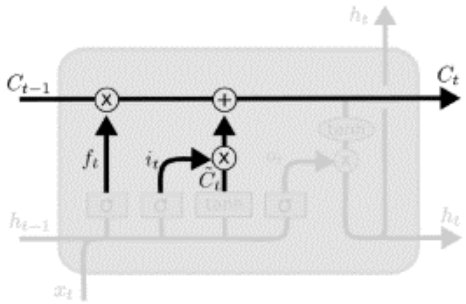
C_t = *cell state*

f_t = *forget gate*

C_{t-1} = *cell state* sebelum orde ke t

I_t = *input gate*

$\tilde{C}_t$ = nilai baru yang dapat ditambahkan ke *cell state*



Gambar 2. 10 Struktur *Update Layer*

Sumber: (Colah, 2018)

Langkah terakhir adalah menentukan apa keluarannya. Pertama, Layer sigmoid akan menentukan bagian dari sel yang akan dikeluarkan. Kemudian, sel tersebut akan dilewatkan pada Layer tanh dan mengalikan dengan keluaran dari gerbang sigmoid.

Gambar 2. 11 Struktur *Output Layer*

Sumber: (Colah, 2018)

Persamaan *output gate* dapat diuraikan pada persamaan (2.10) sebagai berikut:

$$O_t = \sigma(W_O[h_{t-1}, x_t] + b_O) \tag{2.10}$$

Dimana:

O_t = *output gate*

σ = fungsi sigmoid

W_O = nilai *weight* untuk *output gate*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_O = nilai bias pada *output gate*

Persamaan nilai output orde t dapat diuraikan pada persamaan 2.11 sebagai berikut:

$$h_t = O_t * \tanh(C_t) \quad (2.11)$$

Dimana:

h_t = nilai *output* orde t

O_t = *output gate*

$tanh$ = fungsi tanh

C_t = *cell state*

Algoritma yang digunakan dalam LSTM adalah sebagai berikut (Brownlee, 2017):

1. Menghitung nilai sigmoid dan tanh.
2. Mengubah data yang akan digunakan menjadi *supervised learning* problem. *Supervised learning* problem adalah algoritma yang digunakan untuk mempelajari fungsi pemetaan dari input ke output. Karena data yang digunakan adalah data time series maka inputnya adalah data hari kemarin (t-1) dan outputnya adalah data hari ini (t)
3. Melakukan normalisasi menggunakan *min-max scalling*.
4. Menghapus variabel yang tidak dibutuhkan.
5. Membagi data menjadi data *training* dan *testing*.

6. Membuat model RNN LSTM yang akan digunakan dengan menentukan banyaknya *hidden layer*, *neuron* dan *epoch* yang akan digunakan. Dalam model RNN LSTM akan dilakukan proses berupa:
   - Menghitung nilai *forget gate*.
   - Menghitung nilai *input gate*.
   - Memperbarui memori yang berada pada *cell*.
   - Menghitung *output gate* dan nilai *output* akhir.
7. Melakukan evaluasi terhadap model RNN LSTM yang telah dibuat.

2.8 Parameter Evaluasi

Menurut Carlo Vercilis (Varcellis, 2009) terdapat dua alasan utama untuk dapat melihat tingkat akurasi pada suatu prediksi dengan model *time series*. Pertama, pada tahap pengembangan dan identifikasi model, ukuran akurasi digunakan dalam membandingkan antar model alternatif yang satu dengan lainnya dan untuk menentukan nilai parameter yang muncul dalan fungsi prediksi. Dalam identifikasi model prediksi yang paling akurat, masing masing model di anggap diterapkan pada data masa lalu, dan nilai model dengan tingkat *error* terendah atau minimum yang dipilih. Kedua, setelah dilakukan pengembangan model prediksi dan digunakan untuk menghasilkan prediksi untuk masa mendatang, perlu secara berkala untuk menilai keakuratan, untuk mendeteksi kelainan dan kekurangan dalam model yang mungkin timbul di lain waktu. Evaluasi keakuratan prediksi pada tahap ini membuat mungkin dalam menentukan apakah model masih akurat atau membutuhkan suatu revisi. Dalam mengevaluasi akurasi dan peramalan kinerja

model berbeda, beberapa indeks evaluasi yang sering digunakan salah satu diantaranya yaitu *Mean Absolute Persentage Error* (MAPE). Formula untuk menghitung indeks ini adalah sebagai berikut (Budiman, 2016):

2.8.1 *Mean Absolute Persentage Error* (MAPE)

Mean Absolute Percentage Error (MAPE) adalah nilai *absolute* dari persentase *error* data teradap *mean*. Sehingga didapatkan persamaan rumus sebagai berikut:

$$MAPE = \frac{\sum_{i=1}^{n} \frac{|\widehat{Yi} - Yi|}{Yi}}{n} * 100\% \quad (2.14)$$

Dimana:

n = jumlah data

Yi = nilai data sebenarnya

$\widehat{Yi}$ = nilai data prediksi

Berikut merupakan nilai parameter evaluasi MAPE:

Tabel 2. 1 Nilai Akurasi MAPE

Nilai MAPE	Akurasi Peramalan MAPE
MAPE ≤ 10%	Tinggi
10% < MAPE ≤ 20%	Baik
20% < MAPE ≤ 50%	*Reasonable*
MAPE ≥ 50%	Rendah

Sumber: Julio Warmansyah & Dida Hilpiah

2.9 Algoritma Optimasi *Adaptive Moment Estimation* (ADAM)

Optimasi berbasis gradien stokastik adalah kepentingan praktis inti dalam banyak bidang sains dan teknik. Banyak masalah dalam bidang ini dapat dilemparkan sebagai optimalisasi beberapa fungsi tujuan berstandarisasi scalar yang membutuhkan maksimalisasi atau minimalisasi berkenaan dengan parameternya. *Adaptive Moment Estimation* (ADAM) adalah metode optimasi yang menghitung tingkat pembelajaran secara adaptif untuk setiap parameter. Adam merupakan algoritma optimasi yang dikembangkan dengan memanfaatkan kelebihan dari algoritma *Adaptive Gradient* (AdaGrad) yang bekerja dengan baik dengan gradien jarang dan *Root Mean Square Propagation* (RMSProp) yang bekerja dengan baik secara *online* dan non Pengaturan Stasioner. Adam tidak hanya mengadaptasi tingkat pembelajaran parameter berdasarkan rata-rata pertama (*mean*) seperti dalam RMSProp, namun ADAM juga menggunakan rata-rata kedua dari gradien (*varians uncentered*). Algoritma menghitung rata-rata pergerakan eksponensial dari gradien dan gradient kuadratnya, dan parameter $\beta 1$ dan $\beta 2$ mengontrol tingkat peluruhan rata-rata pergerakan (Aldi, Jondri, & Aditsania, 2018).

Beberapa keuntungan ADAM adalah bahwa besarnya pembaruan parameter tidak sama dengan penskalaan gradien, ukurannya kira-kira dibatasi oleh *hyperparameter stepsize*, tidak memerlukan alat tulis stasioner. *Adaptive Moment Estimation* (ADAM) adalah metode yang menghitung *adaptive learning rate* untuk setiap parameter. Nilai parameter konfigurasi dalam ADAM adalah Alpha atau juga disebut sebagai *learning rate* atau *step size*, proporsi yang digunakan weight saat

pembaruan α = 0,001, $\beta 1$= 0.9, $\beta 2$ = 0.999, dan ϵ = 10^{-8} dengan $\beta 1$ = $\beta 2$ = tingkat penurunan eksponensial dan ϵ = nilai epsilon untuk update parameter (Kingma & Ba, 2015).

2.10 Sistem LSTM

Dalam membangun sistem pada metode pendekatan RNN LSTM maka dapat dilakukan dengan beberapa langkah, diantaranya adalah *preprocessing* data, inisialisasi parameter, *training* LSTM *network*, dan juga melakukan uji terhadap data *testing*. Dalam sistem yang dibangun, dataset yang didapatkan terlebih dahulu dioalah dengan menggunakan teknik normalisasi *min max scaling*. Dilakukan inisialisasi pada setiap parameter, setelah itu dilakukan *training* pada jaringan yang dibuat sesuai dengan parameter yang telah ditentukan. Selanjutnya dilakukan uji pada model yang telah didapatkan dari proses *training* terhadap data *testing*. Proses tersebut terus diulang hingga mendapatkan model dengan akurasi yang dapat diterima. Langkah lebih jelasnya adalah sebagai berikut: (Aldi, Jondri, & Aditsania, 2018):

2.10.1 Preprocessing data

Untuk meminimalisir terjadinya *error*, dilakukan normalisasi pada dataset dengan mengubah data aktual menjadi nilai dengan *range interval* [0,1]. Teknik normalisasi yang digunakan menggunakan *min-max scaling*. Adapun untuk rumus normalisasi *min-max scaling* adalah:

$$x' = \frac{x - min_x}{max_x - min_x} \quad (2.15)$$

Dimana x' adalah hasil normalisasi, x adalah data yang akan dinormalisasikan, min_x adalah nilai minimum dari keseluruhan data, dan max_x adalah nilai maksimum dari keseluruhan data.

2.10.2 Inisialisasi Parameter

Adapun bobot-bobot dalam RNN yaitu bobot dari input *layer* ke *hidden layer*, bobot dari *hidden layer* ke *output layer* dan bobot dari *context layer* ke *hidden layer*. Selain bobot, proses RNN LSTM juga termasuk inisialisasi parameter pembelajaran. Parameter pembelajaran dalam RNN LSTM pada penelitian ini adalah nilai *learning rate*, *epoch* dan fungsi aktivasi (Rizal & Soraya, 2018).

Setelah dilakukannya *preprocessing* data, langkah selanjutnya adalah menentukan inisialisasi parameter-parametar dasar yang dibutuhkan, diantaranya sebagai berikut:

1. Jumlah *hidden layer*
2. Jumlah *neuron* pada *hidden layer*
3. Target nilai *error* yang berupa *Mean Absolute Error* (MAE)
4. *Epoch* maksimum

2.10.3 *Training* LSTM

Langkah selanjutnya adalah *training* LSTM, penjelasan dari proses ini adalah sebagai berikut:

1. Hitung semua fungsi gates unit pada setiap *neuron*. Dengan berurut fungsi *gates* yang akan dihitung adalah *forget gates* dengan persamaan (2.5), fungsi *input gates* dengan persamaan (2.6) dan (2.7), fungsi *cell gates*

dengan persamaan (2.9), dan yang terakhir fungsi *output gates* dengan persamaan (2.10) dan (2.11).

2. Jika telah melakukan perulangan sebanyak *epoch* yang telah ditentukan, maka berhenti. Jika belum, akan dilakukan optimasi dan memperbarui bobot dan bias pada sistem, kemudiankembali ke langkah dua.

2.10.4 *Testing* LSTM

Model yang telah didapatkan pada proses *training* akan diuji dengan menggunakan data *testing* yang telah didapat dari *preprocessing* data, dengan metode akurasi yang paling akurat yaitu menggunakan nilai MAPE.

2.10.5 Denormalisasi Data

Sebelum menghitung akurasi hasil prediksi, terlebih dahulu dilakukan proses denormaliasi hasil output dari RNN LSTM. Data yang pada awalnya sudah dilakukan normalisasi, maka akan di kembalikan ke data sebenarnya untuk mengetahui hasil prediksi sebenarnya. Denormalisasi dilakukan untuk mendapatkan nilai real dari hasil prediksi yang diberikan. Sementara itu akurasi dihitung dengan melihat persen akurasi dari hasil prediksi. Proses denormalisasi dilakukan dengan persamaan 2.16 sebagai berikut (Rizal & Soraya, 2018):

$$dn_i = ((n_i)(x_{max} - x_{min})) + (x_{min}) \quad (2.16)$$

Dimana:

dn_i = data hasil denormalisasi

n_i = data ke-i

x_{max} = data aktual dengan nilai maksimum

x_{min} = data aktual dengan nilai minimum