BAB II

TINJAUAN PUSTAKA

2.1 Tinjauan Statistik

2.1.1 Peramalan

Peramalan merupakan suatu kegiatan yang dilakukan untuk memprediksi atau memperkirakan suatu hal yang akan terjadi pada waktu mendatang berdasarkan data masa lalu (Wardana, 2020). Peramalan biasa digunakan sebagai dasar pengambilan keputusan atau perencanaan kebijakan suatu perusahaan, baik itu rencana jangka pendek, panjang, maupun menengah. Metode peramalan dibagi menjadi 2, diantaranya (Syarif, 2012) :

1. Metode Peramalan Kualitatif

   Yaitu peramalan yang didasarkan atas pendapat seseorang menurut insting, pengetahuan, atau pengalamannya. Contohnya estimasi pelanggan yang dapat diperoleh dengan realisasi teknologi 3G (Yulius, 2014) atau peramalan tentang permintaan produk baru dimana data masa lalu belum tersedia (Hutahaean, 2019) sehingga perusahaan akan meminta pendapat dari staf ahli penjualan atau pemasaran.

2. Metode Peramalan Kuantitatif

   Yaitu peramalan yang didasarkan atas data kuantitatif masa lalu dengan menggunakan pendekatan *time series* maupun korelasi yang dapat diwujudkan dalam angka serta terbukti secara ilmiah. Bentuk pola datanya bermacam-macam antara lain *trend*, siklus, musiman, dan horizontal. Contohnya prediksi

harga minyak dunia (Hussein, 2021) dan prediksi IHK berdasarkan harga bahan makanan pokok (Zahara, 2021).

2.1.2 *Time Series*

Time series atau deret waktu merupakan serangkaian pengamatan terhadap suatu peristiwa, kejadian, gejala atau perubahan yang terjadi secara berurutan dari waktu ke waktu (Wardana, 2020). *Time Series* pertama kali diperkenalkan oleh George Box dan Gwilyn Jenkins pada tahun 1976. Analisis *time series* adalah teknik analisis data yang mengamati serangkaian variabel yang dicatat secara urut dalam beberapa waktu berdasarkan kejadiaannya dengan interval waktu tetap. Periode waktu dari data *time series* dapat berupa mingguan, bulanan, semester, tahunan, kuartal dan lain-lain.

Peramalan menggunakan *time series* bertujuan untuk mengetahui lebih awal suatu kondisi atau kejadian di masa depan dengan harapan dapat membuat suatu keputusan dengan tepat. Setiap pengamatan dinyatakan sebagai variabel Y_t yang diperoleh berdasarkan indeks waktu tertentu (t_i) dengan i =1, 2, ..., n. Jadi, bentuk penulisan data *time series* adalah $Y_{t_1}, Y_{t_2}, \dots, Y_{t_n}$ (Nabilah, 2017).

2.1.3 *Preprocessing*

Preprocessing merupakan tahap awal yang perlu dilakukan untuk mempersiapkan data agar menjadi informasi yang siap untuk diolah pada tahapan berikutnya (pemodelan, prediksi, peramalan, dan sebagainya). Berikut beberapa proses yang termasuk dalam *preprocessing* (Wardana, 2020) :

1. *Cleaning* Data

Tahap ini bertujuan untuk menghilangkan *missing value* dan *noise* pada data penelitian. Untuk mengatasi *missing value*, dapat dilakukan dengan menghilangkan data yang tidak ada atau dapat menggantinya dengan nilai rata-rata dan modus. Sedangkan untuk mengatasi *noise*, biasanya dapat dilakukan dengan teknik binning, regresi, dan *clustering*.

2. Normalisasi Data

Normalisasi merupakan proses memberikan skala nilai pada data dalam rentang 0 hingga 1. Hal ini bertujuan untuk menghindari fitur yang memiliki nilai besar lebih mendominasi daripada fitur yang bernilai kecil sehingga dapat meminimalkan *error* yang dihasilkan. Normalisasi dilakukan memakai rumus *min-max scaler*, berikut rumusnya (Aulia, 2020) :

$$X_{sn} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Keterangan:

X_{sn} = Data setelah dinormalisasi

X = Data *input* yang akan dinormalisasi

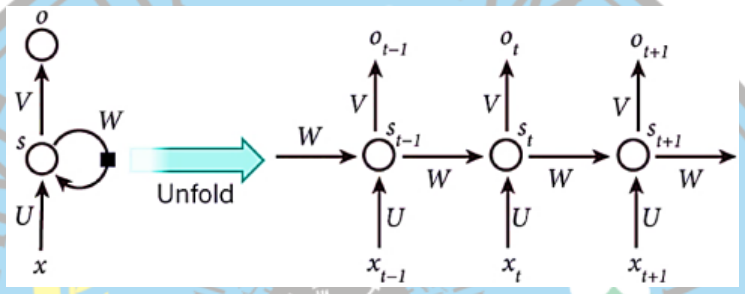
X_{min} = Nilai terkecil dari keseluruhan data

X_{max} = Nilai terbesar dari keseluruhan data

2.1.4 *Recurrent Neural Network* (RNN)

Recurrent Neural Network adalah variasi lain dari jaringan saraf tiruan, yang diciptakan oleh seorang professor bernama Jeff Elman Tahun 1990. RNN dirancang secara khusus untuk memproses data menjadi bentuk sekuensial atau berurutan. Selain biasa digunakan dalam peramalan berbasis data *time series*, RNN

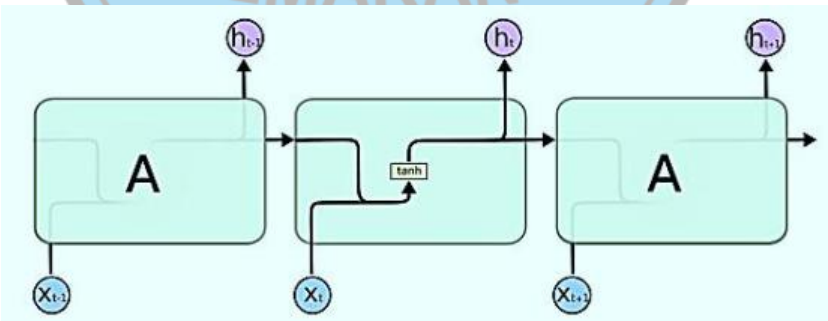
juga dapat menyelesaikan permasalahan yang kaitannya dengan domain *Natural Language Processing* (NLP) seperti mesin pengenalan suara, aplikasi terjemahan, hingga klasifikasi gambar. Dalam RNN, alur pemrosesan data terjadi secara *forward* dan *backward* dengan sistem yang berulang, yang berfokus kepada waktu sebelumnya dan saat ini (t) akan mempengaruhi *output* pada waktu berikutnya (t + 1) (Aulia, 2020). Proses kerja secara berulang dalam RNN dapat dilihat dalam gambar berikut :



Gambar 2.1 Proses Kerja RNN

Sumber : Faadilah, 2020.

Dari gambar diatas, dapat terlihat jaringan RNN yang bekerja secara berurutan atau *sequence*, mulai dari *layer* pertama hingga terakhir. Berdasarkan gambar proses kerja RNN, secara lebih jelasnya arsitektur RNN digambarkan sebagai berikut (Wardani, 2021) :



Gambar 2.2 Arsitektur RNN

Dan berikut penjabaran rumus RNN :

$$S_t = f(U \times X_t) + (W \times S_{t-1})$$

$$h_t = g(V \times S_t)$$

Keterangan :

X_t = *input* dalam tiap waktu

S_t = *hidden state* pada tiap waktu atau memori dalam suatu jaringan pada waktu ke-t yang berguna menyimpan hasil perhitungan dan merekam informasi sebelumnya. S_t dihitung menurut *hidden state* sebelumnya dan *input* saat ini. Untuk *hidden state* pertama menggunakan S_{t-1}.

h_t = *output* setiap langkah pada waktu ke-t atau nilai prediksi.

t = nilai sebenarnya

U, V, W = matriks pembobot

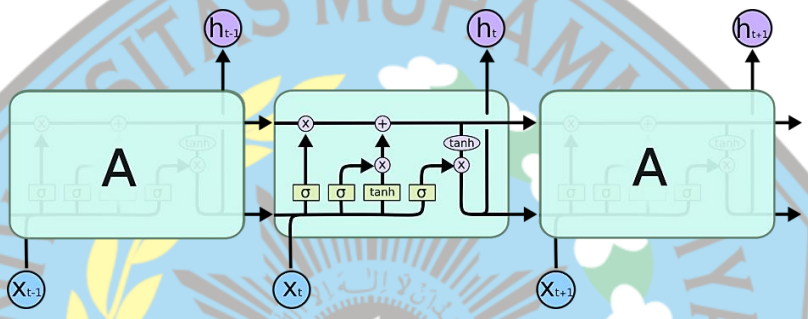
f dan g = fungsi non-linier tanh.

RNN akan menyimpan memori sebelumnya sehingga dapat mengenali pola data dengan baik, karena itu bisa menghasilkan prediksi yang akurat. Hal ini dilakukan dengan cara perulangan pada arsitekturnya, maka secara otomatis informasi masa lalu akan tersimpan. Namun, RNN juga memiliki kekurangan tidak bisa mengatasi masalah gradien hilang. Oleh karena itu, ada algoritma pengembangannya yaitu LSTM dan GRU yang dapat mengatasi masalah tersebut.

2.1.5 *Long Short Term Memory* (LSTM)

Long Short Term Memory adalah algoritma pengembangan dari jaringan saraf berulang atau RNN, yang memiliki keunggulan mampu mengatasi masalah

vanishing gradient yang tidak dapat diatasi oleh RNN. Pada tahun 1997, LSTM diciptakan oleh 2 orang professor bernama Hochreiter dan Jurgen Schmidhuber. Selain dapat diterapkan dalam hal klasifikasi, LSTM juga banyak digunakan untuk peramalan berbasis data runtun waktu, karena terbukti mahir dalam membuat prediksi dengan jangka waktu yang lama dibandingkan dengan algoritma yang lain (Zahara, 2019). Berikut gambar struktur jaringan LSTM :



Gambar 2.3 Struktur Jaringan LSTM

Sumber : Larasati, 2020.

LSTM memiliki gerbang atau lapisan sigmoid dengan struktur yang terdiri dari *input gate*, *forget gate*, dan *output gate*. Gerbang ini sebagai jalan untuk menyaring informasi agar tetap memperbarui dan mempertahankan keadaan *memory cell*. Tiap *memory cell* terdapat 1 lapisan tanh dan 3 lapisan sigmoid (Aulia, 2020). Berikut rumus dari fungsi tanh dan sigmoid :

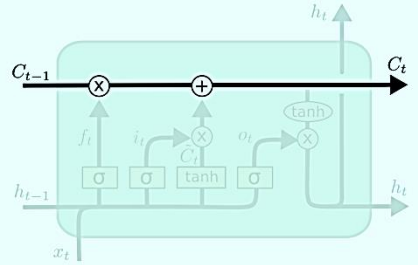
$$\tanh(x) = 2\sigma(2x) - 1$$

$$\sigma(x) = 1/1+e^{-x}$$

Keterangan :

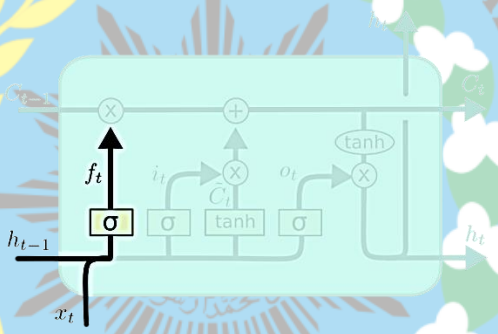
σ = Fungsi aktivasi sigmoid

x = Data *input*



Gambar 2.4 *Memory Cell* LSTM

Gambar diatas, menampilkan jalur yang menyambungkan antara sel memori lama (C_{t-1}) dengan sel memori baru (C_t). Sel memori ini berupa garis horizontal yang menghubungkan semua *layer* dalam LSTM. Jalur tersebut berfungsi memudahkan nilai dari sel memori lama untuk diteruskan menuju sel memori baru.



Gambar 2.5 Arsitektur *Forget Gate*

Proses kerja pada LSTM diawali dengan memutuskan data atau informasi yang akan dibuang dari sel. Hal ini dilakukan oleh salah satu lapisan sigmod yaitu *forget gate*, yang digambarkan dengan simbol h_{t-1} dan x_t dengan menghasilkan angka antara 0 dan 1. 0 artinya informasi tersebut akan dibuang atau tidak diteruskan pada *step* berikutnya, sedangkan 1 artinya informasi tetap dipertahankan dan lanjut ke langkah berikutnya. Dalam hal ini, sel memori menerima *output* (h_{t-1}) pada momen sebelumnya dan *input* berupa informasi eksternal (x_t) pada momen saat ini, keduanya digabungkan dalam suatu vektor panjang melalui transformasi yang dijabarkan pada rumus berikut :

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

Keterangan :

f_t = *Forget gate*

σ = Fungsi sigmoid

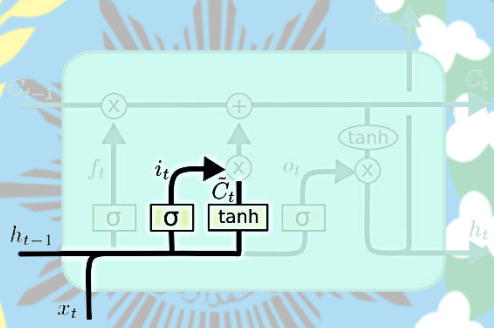
W_f = Nilai bobot *forget gate*

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

x_t = Nilai *input* orde ke-t

b_f = Nilai bias *forget gate*

Notasi $[h_{t-1}, x_t]$ artinya menambahkan baris dari x_t dengan baris dari h_{t-1}.



Gambar 2.6 Arsitektur *Input Gate*

Proses berikutnya yaitu menyeleksi informasi baru manakah yang akan disimpan dalam sel menggunakan *input gate*. Untuk lebih jelasnya, *input gate* memiliki 2 fungsi, yaitu mencari keadaan sel yang perlu diperbarui nilainya dan mengontrol seberapa banyak informasi yang ditambahkan dengan memakai vektor C_t pada lapisan Tanh. Berikut persamaannya :

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i)$$

$$\hat{C}_t = \tanh(W_i . [h_{t-1}, x_t] + b_c)$$

Keterangan :

i_t = *Input gate*

σ = Fungsi sigmoid

W_i = Nilai bobot *input gate*

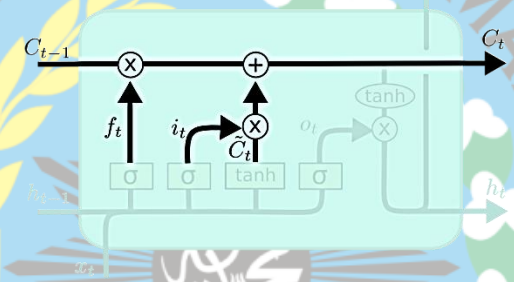
h_{t-1} = Nilai *output* sebelum orde ke-t

x_t = Nilai *input* pada orde ke-t

b_i = Nilai bias pada *input*

$\hat{C}_t$ = Nilai baru yang ditambahkan ke *cell state*

b_c = Nilai bias pada *cell state*



Gambar 2.7 Proses Memperbarui Status Sel

Gambar di atas, merupakan proses pembaruan dari sel lama (C_{t-1}) ke dalam sel baru (C_t). Untuk lebih jelasnya, digambarkan dalam persamaan berikut :

$$C_t = f_t \times C_{t-1} + i_t \times \hat{C}_t$$

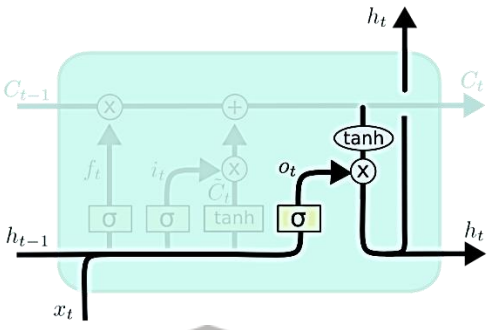
Keterangan :

C_t = *Cell state* orde ke-t

f_t = *Forget gate*

C_{t-1} = *Cell state* sebelum orde ke-t (sel lama)

$i_t \times \hat{C}_t$ = *Input gate* (nilai kandidat baru)



Gambar 2.8 Arsitektur *Output Gate*

Langkah terakhir dalam LSTM berlangsung pada *Output Gate*, lapisan ini berfungsi untuk mengontrol seberapa banyak informasi yang dibuang dalam keadaan sel saat ini dan menghasilkan nilai akhir atau luaran dari informasi yang masih tersimpan. Berikut persamaannya :

$$O_t = \sigma(W_o . [h_{t-1}, x_t] + b_o)$$

Keterangan :

O_t = *Output gate*

σ = Fungsi sigmoid

W_o = Nilai bobot 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*

Nilai *output* akhir sel dijabarkan sebagai berikut :

$$h_t = O_t \times tanh(C_t)$$

Keterangan :

h_t = Nilai *output* orde ke-t

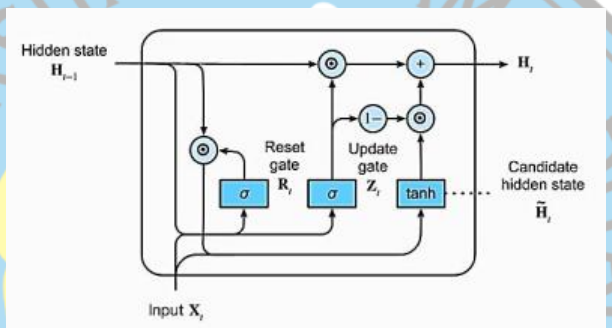
tanh = Fungsi tanh

O_t = *Output gate*

C_t = *Cell state*

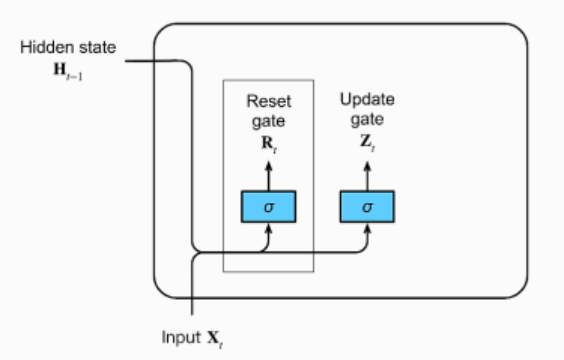
2.1.6 *Gated Recurrent Unit* (GRU)

Gated Recurrent Unit adalah algoritma variasi atau bentuk lain dari LSTM, GRU pertama kali diperkenalkan oleh 3 orang professor bernama Chung, Kyung, dan Yoshua pada tahun 2014. Dibandingkan dengan LSTM, perbedaannya terletak pada arsitektur atau komputasi yang lebih sederhana dan gerbang yang dimiliki, dimana GRU hanya terdapat 2 gerbang yaitu *reset gate* dan *update gate*. Berikut gambar arsitektur dari GRU (Siringoringo, 2021) :



Gambar 2.9 Arsitektur GRU

Langkah pertama yang dilakukan oleh GRU yaitu menggabungkan informasi dari waktu sebelumnya dengan masukan dari *reset gate*. *Reset Gate* berfungsi untuk menentukan seberapa banyak informasi terdahulu yang tidak relevan sehingga akan dihilangkan menggunakan fungsi aktivasi sigmoid. *Output* yang dihasilkan dari *reset gate* hanya berkisar antara nilai 0 hingga 1. Jika mendekati 0 artinya informasi pada waktu sebelumnya tidak begitu berpengaruh maka akan dihapus. Namun, jika nilainya mendekati 1 artinya informasi sebelumnya memiliki pengaruh dan tetap akan disimpan. Berikut gambar proses dari *reset gate* dan *update gate* :



Gambar 2.10 Arsitektur *Reset Gate* dan *Update Gate*

Untuk penghitungannya, dijabarkan dalam persaman berikut :

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$$

Keterangan :

R_t = *reset gate*

σ = fungsi aktivasi sigmoid

X_t = data *input*

W_{xr} = nilai bobot *input* pada waktu ke-t

H_{t-1} = *hidden state* pada waktu sebelumnya

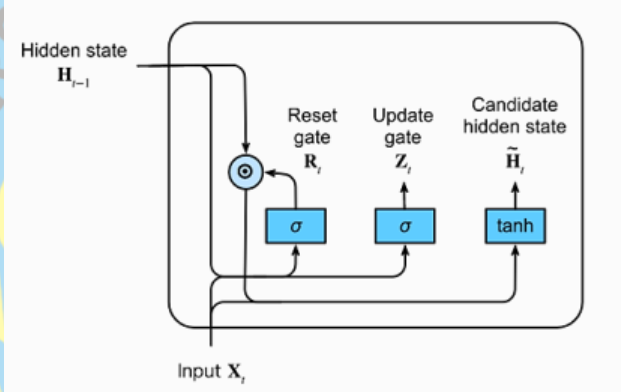
W_{hr} = nilai bobot *output* pada waktu t-1

b_r = nilai bias pada *reset gate*

Langkah selanjutnya yaitu menentukan berapa banyak informasi dari waktu sebelumnya yang dapat disimpan sebagai *input* di waktu berikutnya untuk dilakukan perhitungan pada *hidden layer* dan menentukan pengaruh informasi di waktu sebelumnya pada *output* saat ini. Tahapan ini terjadi dalam lapisan *update gate* dengan menggunakan fungsi aktivasi sigmoid. *Output* yang dihasilkan juga berkisar antara 0 hingga 1. Untuk perhitungannya, dijabarkan dalam persamaan berikut :

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

Update gate berfungsi menentukan seberapa banyak status sel yang akan diperbarui, dengan rentang nilai antara 0 hingga 1. Ketika nilai pembaruannya mendekati 0, maka keadaan sebelumnya tetap akan disimpan. Sebaliknya, jika nilai unit pembaruannya mendekati 1, maka keadaan sebelumnya akan dihapus dan digantikan dengan nilai baru. Proses kerja *update gate* mirip dengan *input gate* dan *forget gate* pada LSTM.



Gambar 2.11 Alur Kerja Hidden State

Pada gambar diatas, menjelaskan langkah berikutnya, yaitu menentukan nilai kandidat *hidden state* atau *output* pada waktu saat ini (t) dari data yang relevan di waktu sebelumnya (t-1) menggunakan fungsi aktivasi tanh. Ini bukanlah *output* terakhir, tetapi merupakan memori pada waktu saat ini. Nilai ini dipengaruhi oleh *output* dari *reset gate*. Berikut jika digambarkan dalam suatu persamaan :

$$\widehat{H}_t = \tanh(X_t W_{xh} + (R_t . H_{t-1}) W_{hh} + b_h)$$

Keterangan :

$\widehat{H}_t$ = nilai kandidat *hidden state*

$\mathrm{R_t}$ = *reset gate*

X_t = data *input*

W_{xh} = nilai bobot *input* pada waktu ke-t

H_{t-1} = *hidden state* pada waktu sebelumnya

W_{hh} = nilai bobot *output* pada waktu t-1

b_h = nilai bias

Langkah terakhir yaitu menghitung informasi akhir dari unit saat ini, penghitungannya dipengaruhi oleh kandidat *hidden state*, nilai *hidden state* pada waktu sebelumnya, dan *output* dari *update gate*. Informasi ini akan diteruskan pada waktu berikutnya sebagai keadaan tersembunyi atau *hidden state*, kemudian digunakan kembali untuk menghitung *output* pada unit waktu tersebut. Proses ini berjalan sama dan terus berulang kedepannya hanya dengan nilai *input* yang berbeda. Berikut bentuk persamaannya :

$$H_t = Z_t . H_{t-1} + (1 - Z_t) . \widehat{H}_t$$

Keterangan :

H_t = *output*

$\widehat{H}_t$ = kandidat *hidden state*

Z_t = *output update gate*

H_{t-1} = *hidden state* pada waktu sebelumnya

2.1.7 Nesterov Adam

Nadam atau Nesterov Adam merupakan algoritma optimasi pengembangan dari algoritma *Adaptive Moment Estimation* (Adam) dengan adanya penambahan momentum *Nesterov Accelerated Gradient* (NAG) atau yang biasa disebut Nesterov. Nadam pertama kali diperkenalkan oleh seorang professor bernama

Timothy Dozat dalam makalahnya yang berjudul *"Incorporating Nesterov Momentum into Adam"* pada tahun 2016 (Dozat, 2016). Momentum bekerja dengan mempercepat proses perhitungan gradien yang menurun secara eksponensial (momen pertama) dari gradien menuju ke algoritma penurunan gradien. Ini memiliki dampak untuk menghaluskan fungsi tujuan dan meningkatkan konvergensi.

Selain itu, dari penelitian yang dilakukan oleh Zahara, Soffa dkk (2019), mencari model terbaik dengan melakukan percobaan pada 7 algoritma optimasi diantaranya meliputi SGD, RMSProp, AdaGrad, Adam, Adadelta, Nadam, serta Adamax. Dari ketujuh algoritma tersebut, Nadam terbukti memiliki performa yang paling baik dalam memprediksi dengan nilai RMSE terkecil. Dapat disimpulkan bahwa Nadam bekerja dengan memperbarui bobot untuk menghaluskan gradien yang berdampak dalam mempercepat proses *training* dan meningkatkan akurasi. Persamaannya dapat ditulis sebagai berikut (Michael, 2020):

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_t + \frac{1-\beta_1}{1-\beta_1^t}\right)$$

Keterangan:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Dengan nilai m_t dan v_t diinisialisasi ke-0.

Berikut beberapa penjelasan nilai parameter pada algoritma optimasi Nadam yang sudah disediakan oleh *library* Keras pada python (Wardana, 2020) :

1. *Learning rate* (α)

   Nilai ini bekerja menentukan seberapa cepat dan lambat jalannya proses *training*, default nilainya adalah 0.001, dan bisa di ubah ketika hasilnya belum optimal. Namun, semakin kecil nilainya maka akan semakin lambat jalannya proses *training*, sedangkan nilai yang besar akan mempercepat proses *training*, tetapi jika terlau besar juga akan menyebabkan overfitting.

2. Beta 1 (β_1)

   Nilai konstan 0,9 untuk laju penurunan eksponensial pada estimasi momen pertama.

3. Beta 2 (β_2)

   Nilai konstan 0,999 pada laju penurunan eksponensial pada estimasi momen kedua.

4. Epsilon (ϵ)

   Merupakan sebuah konstanta kecil (1e-07) untuk presisi atau menstabilkan angka yang dihasilkan dari proses *training*.

2.1.8 Inisialisasi Hyperparameter

Inisialisasi adalah pendeklarasian variabel atau pemberian suatu nilai dalam sebuah variabel maupun objek. Sedangkan hyperparameter merupakan pengaturan eksternal atau suatu nilai yang besar kecilnya bisa ditentukan sendiri bahkan tidak dapat ditebak dari data. Jadi inisialisasi hyperparameter dapat diartikan sebagai pendeklarasian variabel dengan pemberian nilai yang ditentukan sebelum model

dilatih dan memberikan pengaruh pada kecepatan dan ketepatan proses pembelajaran pada model. Berikut hyperparameter yang digunakan dalam penelitian ini :

1. *Epoch*

   Epoch merupakan masa yang menunjukkan seluruh *dataset* sudah melewati proses *training* dalam jaringan saraf. 1 *epoch* artinya proses *training* telah terjadi dalam satu putaran, *epoch* berlangsung secara berulang-ulang lebih dari 1, hal ini bertujuan agar mencapai konvergensi (ketidakbiasan) pada nilai bobot. Karena 1 e*poch* saja akan mengakibatkan underfitting pada model, dan jika terlalu banyak juga dapat berakibat overfitting. Tidak ada penentuan mutlak jumlah e*poch* dan cukup beragam, serta bisa juga disesuaikan dengan jumlah *dataset* yang dimiliki (Wardana, 2020).

2. *Neuron*

   Neuron merupakan sel saraf yang bertugas menerima *input* untuk diteruskan ke *output*. Dalam algoritma LSTM dan GRU, *neuron* biasa disebut sebagai unit, ia terletak pada lapisan sel di *hidden layer* dan *output layer*. Jumlah *neuron* memberikan efek yang cukup signifikan terhadap model jaringan saraf. Tidak ada ketentuan mutlak jumlah *neuron*, sehingga untuk mengetahui model yang terbaik, maka perlu dilakukan percobaan lebih dari 1 kali atau *trial and error* (Ryandhi, 2017).

2.1.9 *Mean Absolute Error* (MAE)

Mean Absolute Error merupakan salah satu parameter untuk mengukur model terbaik dalam peramalan. Nilai MAE menunjukkan *error* atau rata-rata kesalahan *absolute* antara hasil peramalan dengan nilai sebenarnya, semakin kecil nilai MAE maka semakin baik model yang terbentuk, secara umum rumus MAE dijabarkan sebagai berikut (Sautomo, 2021) :

$$\text{MAE} = \left| \frac{\sum_{i=1}^{n} (X_t - F_t)}{n} \right|$$

Keterangan :

X_t = Nilai riil saat periode ke-t

F_t = Nilai peramalan saat periode ke-t

n = Jumlah periode peramalan

2.1.10 Denormalisasi Data

Denormalisasi merupakan proses mengembalikan data yang sudah dinormalisasi ke data sebenarnya untuk kemudian dibandingkan dengan data hasil peramalan (Aulia, 2020). Berikut rumusnya :

$$X_t = y\ (X_{max} - X_{min}) + X_{min}$$

Keterangan :

X_t = Nilai data sebenarnya

y = Nilai *output* yang dinormalisasi

X_{max} = Nilai maksimum data aktual

X_{min} = Nilai minimum data aktual

2.1.11 *Mean Absolute Percentage Error*

MAPE (*Mean Absolute Percentage Error*) merupakan persentase rata-rata *absolute* dari kesalahan meramal dan digunakan untuk mengukur tingkat akurasi dari hasil peramalan (Elmunim, 2013). Semakin kecil rata-rata persentase errornya maka tingkat keakuratan peramalan akan semakin besar. Persamaan nilai MAPE dapat dirumuskan sebagai berikut :

$$\text{MAPE} = \frac{\sum \frac{|F_t - X_t|}{X_t}}{n} \text{ x } 100\%$$

Keterangan :

X_t = Data riil periode ke-t

F_t = Hasil peramalan periode ke-t

n = Jumlah observasi

Nilai MAPE dibagi menjadi beberapa kriteria, dapat dilihat dalam tabel berikut (Arfianti, 2021) :

Tabel 2.1 Kriteria Nilai MAPE

MAPE	Keterangan
< 10%	Hasil peramalan sangat baik
10 – 20%	Hasil peramalan baik
20 – 50%	Hasil peramalan cukup baik
> 50%	Hasil peramalan buruk

2.2 Tinjauan Non-Statistik

Ekspor Migas merupakan kegiatan jual beli komoditi minyak dan gas dari suatu negara ke negara lain menggunakan pembayaran valuta asing yang dinyatakan dalam satuan juta USD (Putra, 2017). Menurut Kertayuga (2021), minyak dan gas bumi merupakan bahan utama industri. Minyak bumi didefinisikan sebagai hidrokarbon berbentuk fasa cair atau padat seperti aspal, lilin mineral, dan bitumen yang berada dalam tekanan dan temperatur yang tinggi dan didapatkan melalui proses penambangan. Sedangkan gas bumi yaitu hidrokarbon berbentuk fasa gas yang berada dalam tekanan serta temperatur yang tinggi dan didapatkan dari proses penambangan (Sihombing, 2021).