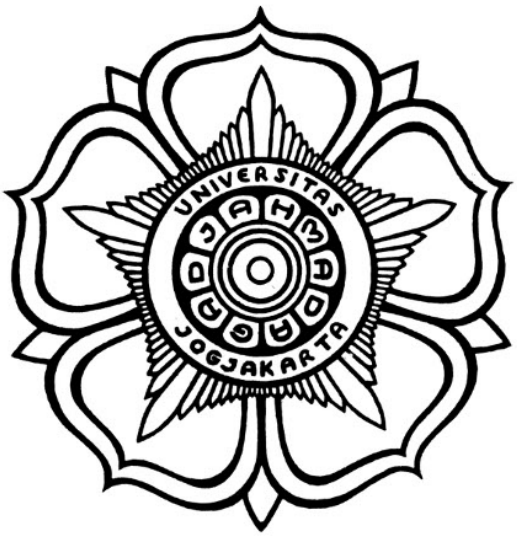
PENGARUH JUMLAH *HIDDEN LAYER* TERHADAP KINERJA AKURASI DAN WAKTU PEMBELAJARAN PADA JARINGAN SYARAF TIRUAN

SKRIPSI



THE SUSTAINABLE DEVELOPMENT GOALS
Industry, Innovation and Infrastructure
Affordable and Clean Energy
Climate Action

Disusun oleh:

Muhamad Alif Ridha
19/439613/TK/48343

PROGRAM SARJANA PROGRAM STUDI TEKNIK ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI
FAKULTAS TEKNIK UNIVERSITAS GADJAH MADA
YOGYAKARTA
2025

HALAMAN PENGESAHAN

PENGARUH JUMLAH *HIDDEN LAYER* TERHADAP KINERJA AKURASI DAN WAKTU PEMBELAJARAN PADA JARINGAN SYARAF TIRUAN

SKRIPSI

Diajukan Sebagai Salah Satu Syarat untuk Memperoleh
Gelar Sarjana Teknik
pada Departemen Teknik Elektro dan Teknologi Informasi
Fakultas Teknik
Universitas Gadjah Mada

Disusun oleh:

Muhamad Alif Ridha
19/439613/TK/48343

Telah disetujui dan disahkan

Pada tanggal . . . . . .

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Risanuri Hidayat, M.Sc.	**Prof. Dr. Ir. Sasongko Pramono Hadi, DEA.**
196708021993031002	**196708021993031002**

PERNYATAAN BEBAS PLAGIASI

Saya yang bertanda tangan di bawah ini :

Nama	: Muhamad Alif Ridha
NIM	: 19/439613/TK/48343
Tahun terdaftar	: 2019
Program Studi	: Teknik Elektro
Fakultas	: Teknik Universitas Gadjah Mada

Menyatakan bahwa dalam dokumen ilmiah Skripsi ini tidak terdapat bagian dari karya ilmiah lain yang telah diajukan untuk memperoleh gelar akademik di suatu lembaga Pendidikan Tinggi, dan juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang/lembaga lain, kecuali yang secara tertulis disitasi dalam dokumen ini dan disebutkan sumbernya secara lengkap dalam daftar pustaka.

Dengan demikian saya menyatakan bahwa dokumen ilmiah ini bebas dari unsur-unsur plagiasi dan apabila dokumen ilmiah Skripsi ini di kemudian hari terbukti merupakan plagiasi dari hasil karya penulis lain dan/atau dengan sengaja mengajukan karya atau pendapat yang merupakan hasil karya penulis lain, maka penulis bersedia menerima sanksi akademik dan/atau sanksi hukum yang berlaku.

Yogyakarta, tanggal-bulan-tahun

Materai Rp10.000

(Tanda tangan)

Muhamad Alif Ridha
19/439613/TK/48343

HALAMAN PERSEMBAHAN

Tugas akhir ini kupersembahkan kepada kedua orang tuaku. Kupersembahkan pula kepada keluarga dan teman-teman semua, serta untuk bangsa, negara, dan agamaku.

KATA PENGANTAR

Puji dan syukur telah dilayangkan kepada kehadirat Allah SWT yang telah melimpahkan kasih dan sayang-Nya, sehingga tugas akhir yang berjudul "Pengaruh Jumlah *Hidden Layer* Terhadap Kinerja Akurasi dan Waktu Pembelajaran pada Jaringan Syaraf Tiruan" dapat terselesaikan. Adapun tujuan dari penyusunan tugas akhir ini adalah untuk memenuhi salah satu persyaratan dalam menempuh ujian jenjang sarjana pada program studi Teknik Elektro yang berada di Fakultas Teknik Elektro dan Informatika Universitas Gadjah Mada.

Dengan tersusunnya laporan ini, pemahaman diperoleh bahwa dalam proses penyusunannya tak terhindar dari bantuan, doa, dan bimbingan dari berbagai pihak kepada. Oleh karena itu, banyak terima kasih diungkapkan kepada:

1. Bapak Ir. Hanung Adi Nugroho, S.T., M.Eng., Ph.D., IPM. selaku Ketua Departemen Teknik Elektro dan Teknologi Informasi dan Ir. Adha Imam Cahyadi, S.T., M.Eng., D.Eng., IPM. selaku Ketua Program Studi S1 Teknik Elektro Fakultas Teknik Universitas Gadjah Mada.
2. Ir. Lesnanto Multa Putranto, S.T., M.Eng, Ph.D., IPM. selaku Sekretaris Departemen Teknik Elektro dan Teknologi Informasi.
3. Bapak Dr. Ir. Risanuri Hidayat, M.Sc. selaku dosen pembimbing 1.
4. Bapak Prof. Dr. Ir. Sasongko Pramono Hadi, DEA. selaku dosen pembimbing 2.
5. Seluruh dosen, staf dan karyawan Program studi Teknik Elektro Universitas Gadjah Mada.
6. Terkhusus kepada yang tercinta dan saya banggakan bapak Adri Hartono S.Si dan Sugihartini S.Pd yang telah banyak berkorban dalam mengasuh, mendidik, mendukung dan mendoakan penulis dengan penuh kasih sayang yang tulus dan ikhlas. Serta Risda Putri Indriani dan Fauzan Dwiputra yang turut mendukung penulis secara moral.
7. Sahabat-sahabatku dan rekan-rekan seperjuangan mahasiswa DTETI angkatan 2019.

Akhir kata penulis berharap semoga skripsi ini dapat memberikan manfaat bagi kita semua, aamiin.

DAFTAR ISI

HALAMAN PENGESAHAN ii
PERNYATAAN BEBAS PLAGIASI iii
HALAMAN PERSEMBAHAN iv
KATA PENGANTAR v
DAFTAR ISI vi
DAFTAR TABEL ix
DAFTAR GAMBAR x
DAFTAR SINGKATAN xi
INTISARI xiii
ABSTRACT xiv
BAB I Pendahuluan 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan Penelitian 2
1.4 Batasan Penelitian 3
1.5 Manfaat Penelitian 3
1.6 Sistematika Penulisan 4
BAB II Tinjauan Pustaka dan Dasar Teori 5
2.1 Tinjauan Pustaka 5
2.2 Dasar Teori 8
2.2.1 Persamaan linear diskriminan 8
2.2.2 Jaringan Syaraf Tiruan 11
2.2.3 *Perceptron* 13
2.2.4 *Multilayer Perceptron* 15
2.2.5 Proses Pelatihan 16
2.2.5.1 *forward propagation* 16
2.2.5.2 *Loss Function* 17
2.2.5.3 *Backpropagation* 18
2.2.5.4 Gradient Descent 18
2.2.6 Normalisasi 20
2.2.7 *Relative error* 21
BAB III Metode Penelitian 22
3.1 Alat dan Bahan Tugas akhir 22
3.1.1 Alat Tugas akhir 22
3.1.2 Bahan Tugas akhir 22
3.2 Metode yang Digunakan 22

3.2.1 Implementasi Algoritma Pelatihan 23
3.2.2 *Dataset* yang digunakan 23
3.2.3 Desain Eksperimen 23
3.3 Alur Tugas Akhir 23
3.3.1 Studi Literatur 24
3.3.2 Pengembangan program jaringan syaraf tiruan dengan python 24
3.3.2.1 Import Library 25
3.3.2.2 *pre-processing* data 25
3.3.2.3 Pembuatan *Class Neural Network* 28
3.3.3 Pengujian program jaringan syaraf tiruan 34
3.3.4 Pengujian dengan satu hidden layer dengan variasi jumlah neuron 34
3.3.5 Pengujian dengan variasi jumlah hidden layer dengan jumlah neuron yang sama 35
3.3.6 Pengujian dengan variasi jumlah hidden layer dengan jumlah bobot yang sama 36
3.3.7 Pengambilan data 37
BAB IV Hasil dan Pembahasan 38
4.1 Hasil Optimal untuk Parameter 38
4.2 Pembahasan Tujuan 1 dengan Hasil Penelitian 1 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas) 38
4.3 Pembahasan Tujuan 1 dengan Hasil Penelitian 2 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas) 38
4.4 Pembahasan Tujuan 2 dengan Hasil Penelitian 3 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas) 38
4.5 Perbandingan Hasil Penelitian dengan Hasil Terdahulu 38
BAB V Tambahan (Opsional) 39
BAB VI Kesimpulan dan Saran 40
6.1 Kesimpulan 40
6.2 Saran 40
DAFTAR PUSTAKA 41
LAMPIRAN L-1
L.1 Isi Lampiran L-1
L.2 Panduan Latex L-2
L.2.1 Syntax Dasar L-2
L.2.1.1 Penggunaan Sitasi L-2
L.2.1.2 Penulisan Gambar L-2
L.2.1.3 Penulisan Tabel L-2
L.2.1.4 Penulisan formula L-2
L.2.1.5 Contoh list L-3

L.2.2 Blok Beda Halaman L-3
L.2.2.1 Membuat algoritma terpisah L-3
L.2.2.2 Membuat tabel terpisah L-3
L.2.2.3 Menulis formula terpisah halaman L-4
L.3 Format Penulisan Referensi L-6
L.3.1 Book L-6
L.3.2 Handbook L-8
L.4 Contoh Source Code L-10
L.4.1 Sample algorithm L-10
L.4.2 Sample Python code L-11
L.4.3 Sample Matlab code L-12

DAFTAR TABEL

Tabel 2.1 Ringkasan Penelitian Terdahulu Mengenai Jumlah Hidden Layer pada Jaringan Saraf Tiruan 7
Tabel 2.2 Masukan dan Keluaran fungsi OR 9
Tabel 2.3 Data *dummy* harga rumah di Jakarta 20

Tabel 3.1 Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah diberi label 26
Tabel 3.2 Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah normalisasi 27

Tabel L.1 Tabel ini L-2
Tabel L.2 Contoh tabel panjang L-4

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi bagaimana fungsi OR dapat dipisahkan menjadi 2 kelas. 10
Gambar 2.2 Diagram Venn ini menggambarkan hubungan antara bidang ilmu [1]. 11
Gambar 2.3 *Action Potential* pada jaringan syaraf 12
Gambar 2.4 Jaringan Syaraf Tiruan 12
Gambar 2.5 Perceptron yang menerima tiga input dan mengeluarkan satu output 13
Gambar 2.6 Output fungsi aktivasi *step function* 14
Gambar 3.7 *Flowchart* alur penelitian 24
Gambar 3.8 Diagram alir program 25
Gambar 3.9 Grafik dari dataset setelah melalui Minmax Scaling. *Cyan* adalah lemari, *magenta* adalah buffet, dan kuning adalah wardrobe . . 28
Gambar 3.10 Diagram Alir Class Neural Network 28
Gambar L.1 Contoh gambar. L-2

DAFTAR SINGKATAN

ANN	=	Artificial Neural Network
ML	=	Machine Learning
AI	=	Artificial Intelligence
JST	=	Jaringan Syaraf Tiruan
DL	=	Deep Learning
MSE	=	Mean Squared Error
KA	=	Kecerdasan Buatan
STRANAS	=	Strategi Nasional
DTETI	=	Departemen Teknik Elektro dan Teknologi Informasi
FT	=	Fakultas Teknik
UGM	=	Universitas Gadjah Mada
MLP	=	Multilayer Perceptron
CNN	=	Convolutional Neural Network
RNN	=	Reccurent Neural Network
GAN	=	Generative Adversarial Network
MSE	=	Mean Square Error
BCE	=	Binary Cross-Entropy
CCE	=	Categorical Cross-Entropy
HL	=	Huber Loss
KL	=	Kullback-Leibler
ReLU	=	Rectified Linear Unit
PReLU	=	Parametric Rectified Linear Unit
ELU	=	Exponential Linear Unit
SGD	=	Stochastic Gradient Descent
Adam	=	Adaptive Moment Estimation
BPPT	=	Badan Pengkajian dan Penerapan Teknologi
NLP	=	Natural Language Processing

INTISARI

Intisari ditulis menggunakan bahasa Indonesia dengan jarak antar baris 1 spasi dan maksimal 1 halaman. Intisari sekurang-kurangnya berisi tentang latar belakang dan tujuan penelitian, metodologi yang digunakan, hasil penelitian, kesimpulan dan implikasi, dan Kata kunci yang berhubungan dengan penelitian.

Kata Kunci ditulis maksimal 5 kata yang paling berhubungan dengan isi skripsi. Silakan mengacu pada ACM / IEEE *Computing classification* jika Anda adalah mahasiswa Sarjana TI http://www.acm.org/about/class/ atau mengacu kepada IEEE keywords http://www.ieee.org/documents/taxonomy_v101.pdf jika Anda berasal dari Prodi Sarjana TE.

Kata kunci : Kata kunci 1, Kata kunci 2, Kata kunci 3, Kata kunci 4, Kata kunci 5

Contoh Abstrak Teknik Elektro:

"Penelitian ini bertujuan untuk mengembangkan sistem pengendalian suhu ruangan dengan menggunakan microcontroller. Metodologi yang digunakan adalah desain sirkuit, implementasi sistem pengendalian, dan pengujian performa. Hasil penelitian menunjukkan bahwa sistem pengendalian suhu ruangan yang dikembangkan mampu mengendalikan suhu ruangan dengan akurasi sebesar ±0,5°C. Kesimpulan dari penelitian ini adalah sistem pengendalian suhu ruangan yang dikembangkan efektif dan efisien.

Kata kunci: microcontroller, sistem pengendalian suhu, akurasi."

Contoh Abstrak Teknik Biomedis:

"Penelitian ini bertujuan untuk mengevaluasi keefektifan prototipe alat pemantau denyut jantung berbasis elektrokardiogram (ECG) untuk pasien jantung. Metodologi yang digunakan meliputi desain dan pembuatan prototipe, pengujian dengan pasien, dan analisis data. Hasil penelitian menunjukkan bahwa prototipe alat pemantau denyut jantung berbasis ECG memiliki akurasi yang baik dan mampu memantau denyut jantung pasien secara efektif. Kesimpulan dari penelitian ini adalah prototipe alat pemantau denyut jantung berbasis ECG merupakan solusi yang efektif dan efisien untuk memantau pasien jantung.

Kata kunci: elektrokardiogram, alat pemantau denyut jantung, akurasi."

Contoh Abstrak Teknologi Informasi:

"Penelitian ini bertujuan untuk mengevaluasi keamanan dan privasi pengguna aplikasi media sosial terpopuler. Metodologi yang digunakan meliputi analisis kebijakan privasi dan pengaturan keamanan, pengujian penetrasi, dan survei pengguna. Hasil penelitian menunjukkan bahwa beberapa aplikasi media sosial memiliki kebijakan privasi yang kurang jelas dan rendahnya tingkat keamanan. Kesimpulan dari penelitian ini adalah pentingnya meningkatkan kebijakan privasi dan tingkat keamanan pada aplikasi media sosial untuk melindungi privasi dan data pengguna.

Kata kunci: media sosial, keamanan, privasi, pengguna."

ABSTRACT

Abstract italic (miring) menggunakan bahasa Inggris dengan jarak antar baris 1 spasi dan maksimal 1 halaman. Abstract adalah versi Bahasa Inggris dari intisari. Abstract dapat ditulis dalam beberapa paragraf. Baris pertama paragraph harus menjorok ke dalam sekitar 1 cm. Tidak dsarankan menggunakan mesin penerjemah melainkan tulis ulang.

Keywords : Keyword 1, Keyword 2, Keyword 3, Keyword 4, Keyword 5

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kecerdasan artifisial (KA) atau, *artificial intelligence* (AI) adalah bidang ilmu yang berfokus pada pengembangan sistem komputer untuk melakukan tugas-tugas yang membutuhkan kecerdasan manusia, seperti belajar, menalar, dan mengenali pola [2]. Bidang ini belakangan berkembang sangat pesat dan menjadi perhatian global. Data dari laporan terbaru menunjukkan bahwa investasi global dalam teknologi KA terus meningkat secara signifikan, dengan perkiraan mencapai lebih dari 300 miliar dolar Amerika pada tahun 2023 [3]. Teknologi KA telah diintegrasikan secara luas ke dalam berbagai sektor. Dalam bidang kesehatan, KA digunakan untuk menganalisis data medis dan membantu diagnosis penyakit [4]. Sementara itu, dalam sektor keuangan, KA dimanfaatkan untuk mengoptimalkan portofolio investasi dan memprediksi kondisi obligasi di masa depan. Sejumlah platform digital besar, seperti Facebook dan Google, telah mengadopsi teknologi kecerdasan artifisial guna meningkatkan efisiensi layanan dan meningkatkan pengalaman penggunanya [5].

Indonesia, melalui Badan Pengkajian dan Penerapan Teknologi (BPPT), menanggapi perkembangan ini dengan membentuk Strategi Nasional Kecerdasan Artifisial Indonesia 2020 - 2045 (STRANAS KA) [6]. Pembentukan STRANAS KA ini merupakan perwujudan Visi Indonesia Emas 2045 dan menekankan pentingnya penelitian dan penggunaan KA untuk sektor-sektor yang diprioritaskan. Dalam bidang ilmu ini, salah satu teknik yang fundamental dan banyak dikembangkan adalah Jaringan Syaraf Tiruan (JST). Oleh karena itu, penelitian yang mendukung pengembangan JST, seperti yang dilakukan dalam skripsi ini, selaras dengan tujuan yang ditetapkan oleh pemerintah.

Jaringan syaraf tiruan telah menjadi fondasi bagi berbagai aplikasi KA, seperti pengenalan wajah, penerjemah bahasa, dan *driver assistance system* [7]. Keberhasilan jaringan syaraf tiruan dalam menangani tugas-tugas kompleks didukung dengan meningkatnya daya komputasi dan ketersediaan data dalam jumlah besar melalui teknologi *cloud*.

Jaringan syaraf tiruan adalah model matematis yang meniru cara kerja otak manusia. Jaringan syaraf tiruan terdiri dari serangkaian *neuron* yang terhubung satu sama lain dan tersusun membentuk lapisan-lapisan. *Input layer* menerima data, *output layer* digunakan untuk mengeluarkan prediksi, dan di antara keduanya terdapat satu atau lebih *hidden layer*. *Hidden layer* (HL) inilah yang bertanggung jawab untuk mengekstraksi dan mempelajari fitur-fitur kompleks dari data. Jumlah dan konfigurasi *hidden layer* akan mempengaruhi kemampuan jaringan dalam memahami dan memodelkan pola dalam

data. Dengan konfigurasi *hidden layer* yang tepat, jaringan syaraf tiruan dapat meningkatkan kinerja dan efisiensi dalam berbagai tugas klasifikasi maupun regresi.

Salah satu permasalahan yang sering dihadapi dalam penggunaan jaringan syaraf tiruan adalah penentuan jumlah *hidden layer* yang akan digunakan. Penentuan jumlah dan ukuran *hidden layer* ini tidak selalu dapat terlihat langsung didalam data. Pemilihan jumlah dan ukuran *hidden layer* memerlukan pengaturan yang baik untuk mendapatkan hasil yang diinginkan. Secara umum, jika jumlah *hidden layer* terlalu sedikit, jaringan mungkin gagal mempelajari representasi yang cukup kompleks dari data. Sebaliknya, jika jumlahnya terlalu banyak, dapat terjadi *overfitting*, yaitu jaringan menjadi terlalu cocok dengan data latih dan kinerjanya menurun saat diuji dengan data baru. Oleh karena itu, menemukan jumlah *hidden layer* yang tepat merupakan tantangan dalam merancang jaringan syaraf tiruan yang efektif.

Beberapa pendekatan, seperti penggunaan *cross-validation* atau teknik optimasi seperti algoritma genetika, telah digunakan untuk membantu menentukan jumlah *hidden layer* yang optimal dalam suatu jaringan syaraf tiruan. Meskipun kecerdasan artifisial merupakan bidang penelitian yang masih berkembang, penelitian atas permasalahan ini diharapkan akan memperkuat kemampuan jaringan syaraf tiruan untuk menghadapi berbagai tugas pemodelan dan prediksi dengan lebih akurat dan efisien.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, permasalahan utama dalam penelitian ini adalah penentuan jumlah dan ukuran *hidden layer* pada suatu jaringan syaraf tiruan belum memiliki panduan yang jelas. pengurangan ataupun penambahan pada *hidden layer* memberikan efek yang kompleks dan tidak selalu linier terhadap kinerja model dan diperlukan penelitian lebih lanjut. Oleh karena itu, rumusan masalah disusun seperti berikut:

1. Ketidakpastian jumlah *hidden layer* menyebabkan kesulitan dalam merancang arsitektur JST yang efektif.
2. Penelitian mengenai efek apa yang terjadi pada pengurangan dan penambahan *hidden layer* pada jaringan syaraf tiruan masih menunjukan hasil yang beragam dan perlu dikaji lebih lanjut.
3. Penelitian mengenai pengaruh jumlah *hidden layer* terhadap waktu komputasi dan akurasi menghasilkan perbedaan yang beragam.

1.3 Tujuan Penelitian

Dalam penelitian ini terdapat beberapa tujuan yang ingin dicapai, yaitu:

1. Menganalisis pengaruh penambahan atau pengurangan *hidden layer* terhadap tingkat akurasi yang dihasilkan dalam tugas klasifikasi.
2. Menganalisis pengaruh penambahan atau pengurangan *hidden layer* terhadap lama waktu komputasi yang diperlukan selama proses pelatihan model.
3. Menemukan konfigurasi jumlah *hidden layer* yang optimal untuk dataset yang digunakan, dengan mempertimbangkan *trade-off* antara akurasi yang tinggi dan waktu komputasi yang efisien.

1.4 Batasan Penelitian

Untuk memfokuskan cakupan penelitian, maka ditetapkan batasan-batasan masalah sebagai berikut:

1. Objek Penelitian : Objek yang diteliti pada penelitian ini adalah pengaruh variasi jumlah *hidden layer* pada arsitektur *Multilayer Perceptron* (MLP) terhadap kinerja model dalam melakukan klasifikasi pada data *linearly separable*.
2. Waktu dan tempat penelitian : Penelitian ini dimulai pada Januari 2023 hingga Januari 2025
3. Variabel : Variabel Bebas dari penelitian ini adalah jumlah *hidden layer* pada arsitektur MLP. Variabel Terikat dari penelitian ini adalah akurasi klasifikasi dan waktu komputasi yang diperlukan selama proses pelatihan (*training*) model.
4. Populasi dan Sampel : Populasi dalam penelitian ini adalah seluruh kemungkinan dataset yang *linearly separable*. Sampel yang digunakan adalah satu dataset sintetis dengan spesifikasi 16 instance, 2 fitur, dan 3 kelas.
5. Data Penelitian : Data yang digunakan adalah data sintetis yang dibuat secara khusus agar bersifat *linearly separable*. Data ini memiliki 16 instance, 2 fitur, dan tiga kelas target.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah:

1. Akademisi, Hasil penelitian ini dapat menambah wawasan ilmu dalam bidang kecerdasan artifisial, khususnya mengenai pemahaman mendalam tentang pengaruh kedalaman JST pada MLP serta menjadi pijakan untuk penelitian lanjutan.
2. Praktisi, Memberikan panduan dalam melakukan tuning arsitektur jaringan saraf tiruan yang lebih terarah, sehingga dapat menghemat waktu dan sumber daya komputasi dalam pengembangan model.

1.6 Sistematika Penulisan

BAB I PENDAHULUAN

Pada ini dijelaskan latar belakang, rumusan masalah, batasan, tujuan, manfaat, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA DAN LANDASAN TEORI

Pada bab ini dijelaskan teori dan penelitian yang digunakan sebagai acuan dalam penelitian.

BAB III : METODOLOGI PENELETIAN

Pada bab ini berisi tentang metodologi penelitian yang terdiri atas alat dan bahan, langkah kerja, dan alur tahapan penelitian.

BAB IV : ANALISIS DAN PEMBAHASAN

Pada bab ini dijelaskan hasil dari penelitian dan pembahasan pengujianya.

BAB V : KESIMPULAN DAN SARAN

Pada bab ini ditulis kesimpulan penelitian dan saran untuk penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Pengujian mengenai pengaruh jumlah *hidden layer* terhadap kinerja jaringan saraf tiruan (JST) telah banyak dilakukan oleh berbagai peneliti. Perbedaan jumlah *hidden layer* terbukti memengaruhi kemampuan jaringan dalam melakukan klasifikasi, kecepatan pelatihan, serta kemampuan generalisasi terhadap data baru. Beberapa hasil penelitian yang relevan disajikan berikut ini.

Surkan dan Singleton (1995) [8] dalam penelitiannya membuat sebuah JST berdasarkan rating obligasi dari moody's or standard and poor's. Data yang digunakan berisi tujuh buah *feature* dan 126 *instance* pola obligasi. Moody's memisahkan rating obligasi menjadi tujuh kelas, yaitu Aaa, Aal, Aa2, Aa3, A1 , A2, dan A3. Aaa dikategorikan sebagai kualitas tinggi, Aa1-Aa3 sebagai kualitas menengah, dan A1-A3 sebagai kualitas rendah. Jaringan syaraf tiruan yang dibuat akan digunakan untuk membedakan kelas kualitas tinggi (Aaa) dan kelas kualitas rendah (A1-A3). JST dibuat dengan tiga model arsitektur. satu dari tiga model menggunakan satu *hidden layer*. Dua model lainnya menggunakan dua *hidden layer* dengan jumlah *neuron* yang berbeda, yaitu masing - masing memiliki arsitektur [7-14-2], [7-5-10-2], dan [7-10-5-2]. Hasilnya menunjukkan bahwa arsitektur dengan dua *hidden layer* memberikan akurasi yang lebih tinggi. Arsitektur [7-10-5-2] mencapai akurasi 88% pada data uji, sedangkan [7-14-2] yang hanya mencapai 65%. Penelitian ini menyimpulkan bahwa penggunaan dua *hidden layer* dapat meningkatkan kinerja klasifikasi tanpa secara signifikan memperlama waktu pelatihan.

De Villiers dan Barnard (1993) [9] mengevaluasi perbedaan kinerja antara jaringan saraf tiruan (JST) satu *hidden layer* dengan dua *hidden layer*. Studi ini menggunakan pendekatan "*distribution of distributions*". Metode tersebut menggunakan sejumlah dataset acak dari beragam distribusi *Gaussian* untuk mensimulasikan variasi alami pada data pelatihan. Dalam setiap eksperimen, kompleksitas jaringan dikendalikan agar tetap konstan dengan menyamakan jumlah total bobot antar arsitektur (± 20, 40, atau 60 bobot). Dengan cara ini, perbandingan kinerja berfokus pada pengaruh kedalaman topologi, bukan pada perbedaan kapasitas model. Hasil penelitian menunjukkan bahwa, secara rata-rata, jaringan satu *hidden layer* menghasilkan kinerja pelatihan (training) dan generalisasi yang lebih stabil dibandingkan jaringan dua *hidden layer*, yang cenderung lebih sensitif terhadap inisialisasi bobot dan lebih sering terjebak pada local minima. Namun, beberapa konfigurasi jaringan dua *hidden layer* khususnya ketika jumlah neuron pada dua *hidden layer* relatif seimbang menunjukkan konvergensi yang lebih cepat dan kadang mencapai performa setara dengan jaringan satu *hidden layer*. Penelitian ini

juga menegaskan bahwa peningkatan jumlah sampel pelatihan secara signifikan meningkatkan akurasi klasifikasi pada kedua arsitektur, serta memperkecil variabilitas hasil antar pelatihan. Secara keseluruhan, temuan tersebut menunjukkan bahwa penambahan *hidden layer* tidak otomatis meningkatkan performa bila kompleksitas jaringan dan jumlah data tetap terbatas, meskipun dapat memberikan keuntungan efisiensi pelatihan pada kondisi topologi tertentu.

Berdasarkan penelitian Panchal (2011) [10], metode *Hopfield Neural Networks* dan *Akaike's Information Criterion* (AIC) digunakan untuk menentukan jumlah *hidden layer* dan jumlah *neuron* yang optimal. Pendekatan ini digunakan untuk menghindari masalah *overfitting* dan *underfitting* dengan memilih arsitektur jaringan yang sesuai dengan kompleksitas data. Hasil eksperimen menunjukkan bahwa penambahan jumlah *hidden layer* secara signifikan meningkatkan *Correct Classification Rate* (CCR) dan menurunkan *error*, meskipun diiringi dengan peningkatan waktu pelatihan. Hasil penelitiannya menyimpulkan bahwa dalam aplikasi yang memprioritaskan akurasi tinggi, penambahan *hidden layer* dapat memberikan peningkatan performa yang signifikan. Sebaliknya, untuk aplikasi yang memerlukan waktu respons cepat atau sumber daya komputasi terbatas, penggunaan satu *hidden layer* sudah dianggap cukup memadai. Dengan demikian, pemilihan arsitektur dapat disesuaikan secara fleksibel berdasarkan *trade-off* antara akurasi dan efisiensi komputasi.

Thomas (2017) [11] melakukan studi untuk menguji perbandingan antara jaringan dengan satu *hidden layer* (*Single Hidden Layer Feedforward Network*, SLFN) dan dua *hidden layer* (*Two-Layer Feedforward Network*, TLFN). Penelitian ini menggunakan sepuluh dataset publik dari berbagai domain (seperti *Abalone, Airfoil Noise, Concrete*, dan *White Wine Quality*) serta algoritma pelatihan Levenberg–Marquardt. Hasil eksperimen menunjukkan bahwa sembilan dari sepuluh dataset menghasilkan *error* yang lebih rendah pada arsitektur dengan dua *hidden layer*. Secara rata-rata, jaringan dengan dua *hidden layer* menunjukkan peningkatan performa sebesar 5–6% dibandingkan jaringan dengan satu *hidden layer*. Penelitian ini menyimpulkan bahwa dua *hidden layer* umumnya memberikan kemampuan generalisasi yang lebih baik, khususnya pada dataset kompleks, meskipun peningkatan tersebut bergantung pada karakteristik data dan tingkat *noise* yang ada.

Artikel dari Uzair dan Jamil (2020) [12] mengkaji berbagai hasil penelitian mengenai pengaruh jumlah *hidden layer* terhadap kompleksitas waktu dan akurasi dalam jaringan saraf tiruan. Hasil kajian menunjukkan bahwa menentukan jumlah *hidden layer* yang optimal tidak dapat hanya didasarkan pada jumlah *input layer* dan *output* saja. Meskipun terdapat berbagai metode untuk memperkirakannya, akurasi aproksimasi sangat bergantung pada karakteristik data yang digunakan. Apabila tujuannya adalah mencapai akurasi tinggi, arsitektur jaringan yang lebih dalam dapat menjadi pilihan. Namun,

jika efisiensi waktu menjadi prioritas, penggunaan jaringan yang dalam kurang tepat digunakan. Selain itu, penambahan *neuron* atau *hidden layer* yang berlebihan berisiko menyebabkan *overfitting*. Oleh karena itu, sangat disarankan untuk melakukan analisis awal terhadap sampel data guna memperkirakan konfigurasi *hidden layer* yang sesuai.

Dari berbagai penelitian di atas, dapat disimpulkan bahwa penambahan jumlah *hidden layer* umumnya dapat meningkatkan kemampuan jaringan dalam mengenali pola yang kompleks, tetapi tidak selalu memberikan peningkatan signifikan jika jumlah data atau kompleksitas model terbatas. Semakin banyak *hidden layer* akan meningkatkan waktu pelatihan dan risiko *overfitting*. Oleh karena itu, pemilihan jumlah *hidden layer* yang optimal harus mempertimbangkan keseimbangan antara akurasi, generalisasi, dan efisiensi komputasi.

Pada penelitian ini, penulis akan membangun jaringan saraf tiruan dengan jumlah *hidden layer* dan *neuron* yang bervariasi, untuk kemudian dibandingkan berdasarkan akurasi dan waktu komputasi yang dihasilkan. Data yang digunakan merupakan data dimensi tinggi dan lebar suatu barang. Data akan diklasifikasikan menjadi tiga kategori, yaitu wardrobe, lemari, dan buffet. Melalui penelitian ini diharapkan dapat diperoleh pemahaman yang lebih jelas mengenai pengaruh jumlah *hidden layer* terhadap kinerja jaringan saraf tiruan dalam konteks klasifikasi berbasis dimensi objek.

Tabel 2.1. Ringkasan Penelitian Terdahulu Mengenai Jumlah Hidden Layer pada Jaringan Saraf Tiruan

No	**Peneliti (Tahun)**	**Judul / Konteks Penelitian**	**Jumlah Hidden Layer yang Diuji**	**Dataset / Aplikasi**	**Kesimpulan**
1	Surkan & Singleton (1995)	*Neural Networks for Bond Rating Improved by Multiple Hidden Layers*	1 HL: [7–14–2]; 2 HL: [7–5–10–2], [7–10–5–2]	126 data obligasi (Moody's & S&P)	Dua *hidden layer* meningkatkan akurasi klasifikasi tanpa memperlama waktu pelatihan secara signifikan.
2	De Villiers & Barnard (1993)	*Backpropagation Neural Nets with One and Two Hidden Layers*	1 HL dan 2 HL (20–60 bobot total)	Dataset sintetis Gaussian dan data nyata	Penambahan *hidden layer* tidak selalu meningkatkan kinerja; tergantung pada kompleksitas data dan topologi jaringan.

No	Peneliti (Tahun)	Judul / Konteks Penelitian	Jumlah Hidden Layer yang Diuji	Dataset / Aplikasi	Kesimpulan
3	Panchal et al. (2011)	*Behaviour Analysis of MLPs with Multiple Hidden Neurons and Hidden Layers*	1–4 HL, variasi jumlah *neuron*	Data retensi karyawan (9–17 *feature*)	1 HL cukup untuk masalah sederhana, 2 HL cocok untuk pola kompleks; AIC efektif untuk menentukan arsitektur optimal.
4	Thomas et al. (2017)	*Two Hidden Layers Are Usually Better Than One*	1 HL (SLFN) vs 2 HL (TLFN)	10 dataset publik (UCI, MATLAB, Bilkent, dll.)	Dua *hidden layer* umumnya memberikan generalisasi lebih baik, terutama untuk dataset kompleks.
5	Uzair & Jamil (2020)	*Effects of Hidden Layers on the Efficiency of Neural Networks*	1–5 HL (studi literatur dan analisis empiris)	Kajian berbagai penelitian terdahulu	Tiga *hidden layer* memberikan keseimbangan terbaik antara akurasi dan efisiensi waktu; lebih banyak *layer* meningkatkan risiko *overfitting*.

2.2 Dasar Teori

2.2.1 Persamaan linear diskriminan

Diskriminan merupakan sebuah fungsi yang menerima vektor input **x** dan mengelompokkannya ke dalam K kelas. Kelas tersebut disimbolkan sebagai C_k, dengan $k = 1, 2, 3, ..., K$. Kelas-kelas ini akan dipisahkan oleh satu atau lebih *decision boundary* yang berupa *hyperplane* [13]. Tujuan dari proses klasifikasi adalah untuk menetapkan setiap masukan tepat ke kelas-nya masing-masing. Persamaan linear diskriminan dituliskan sebagai berikut:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{2-1}$$

Dalam persamaan tersebut, **w** disebut sebagai vektor bobot, sedangkan $\mathbf{w}^T$ menyatakan transpose dari vektor bobot. Sementara itu, w_0 disebut juga sebagai bias dan dapat dilambangkan dengan variabel b. Pada permasalahan klasifikasi biner sebuah ma-

sukan **x** diklasifikasikan ke dalam kelas C_1 jika $y(\mathbf{x}) \geq 0$. Namun, jika $y(\mathbf{x}) \leq 0$ maka **x** dikategorikan ke dalam kelas C_2. Kedua kelas ini akan dipisahkan oleh sebuah *decision boundary*. *Decision boundary* sendiri didefinisikan oleh persamaan $y(\mathbf{x}) = 0$. *Decision boundary* akan merepresentasikan sebuah *hyperplane* berdimensi (D - 1) pada masukan berdimensi D. Sebagai contoh apabila masukan memiliki 2 *feature* (D = 2), maka data tersebut dapat direpresentasikan oleh bidang kartesius dengan dua sumbu. Pada kondisi ini, *decision boundary* akan berdimensi satu dan berbentuk garis yang memisahkan kedua kelas pada *input space*. *Decision boundary* ini berkaitan erat dengan vektor bobot **w** dan bias w_0. Sebagai ilustrasi, misalkan terdapat dua titik $\mathbf{x}_A$ dan $\mathbf{x}_B$ yang keduanya berada pada *decision boundary* sehingga $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$. Dengan demikian berlaku hubungan berikut:

$$\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0 \tag{2-2}$$

Persamaan tersebut menunjukan bahwa vektor bobot **w** tegak lurus terhadap setiap vektor yang terletak pada *decision boundary* [13]. Oleh karena itu, vektor **w** menentukan orientasi dari *decision boundary*. Apabila **x** berada pada *decision boundary*, maka $y(\mathbf{x}) = 0$. Jarak normal dari titik $(0, 0)$ ke *decision boundary* dapat dituliskan sebagai berikut :

$$\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||} \tag{2-3}$$

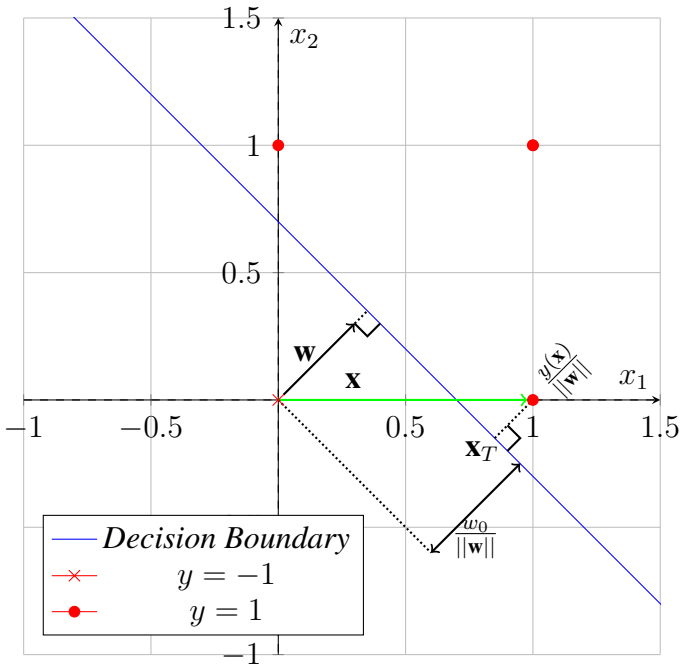
Dapat dilihat bahwa w_0 akan menentukan posisi dari *decision boundary*. Sebagai contoh diberikan data masukan dan keluaran dari fungsi OR seperti pada tabel 2.2.

Tabel 2.2. Masukan dan Keluaran fungsi OR

x_0	x_1	output	y
0	0	0	-1
0	1	1	1
1	0	1	1
1	1	1	1

Pada data tersebut berisi empat buah *instance* dan memiliki 2 buah *feature*. Karena memiliki 2 buah *feature* maka masukan berdimensi 2 atau D = 2. Masukan akan dibedakan menjadi 2 kelas yaitu C_1 untuk $y \geq 0$ dan kelas C_2 untuk $y \leq 0$. Untuk memisahkan kedua kelas tersebut akan dipilih nilai **w** dan w_0. *Decision boundary* yang dibentuk akan memiliki satu dimensi.

Dari Gambar 2.1 dapat dilihat bahwa terdapat *decision boundary* yang memisahkan kedua kelas. Vektor **w** tegak lurus dengan *decision boundary*, dan jarak antara *decision boundary* dan titik origin adalah $\frac{w_0}{||\mathbf{w}||}$. Selanjutnya, untuk mengetahui jarak antara sebuah



Gambar 2.1. Ilustrasi bagaimana fungsi OR dapat dipisahkan menjadi 2 kelas.

titik dengan *decision boundary*, digunakan Persamaan 2-4.

$$r = \frac{y(\mathbf{x})}{||\mathbf{w}||} \tag{2-4}$$

Hal ini dapat dibuktikan dengan memisalkan $\mathbf{x}_T$ sebagai proyeksi dari titik $\mathbf{x}$ pada *decision boundary*. Dengan demikian, vektor $\mathbf{x}$ dapat dituliskan seperti pada Persamaan 2-5.

$$\mathbf{x} = \mathbf{x}_T + r\frac{\mathbf{w}}{||\mathbf{w}||} \tag{2-5}$$

Dengan mengalikan Persamaan 2-5 dengan $\mathbf{w}^T$ dan menambahkan w_0, diperoleh Persamaan 2-6.

$$\mathbf{w}^T\mathbf{x} + w_0 = \mathbf{w}^T\mathbf{x}_T + w_0 + r\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} \tag{2-6}$$

Dengan mensubtitusikan Persamaan 2-1, kondisi $\mathbf{w}^T\mathbf{x}_T + w_0 = 0$ untuk titik pada *decision boundary*, serta identitas $\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} = ||\mathbf{w}||$ ke dalam Persamaan 2-6, diperoleh Persamaan 2-4.

Setelah penjelasan mengenai hubungan antara vektor bobot dengan *decision boundary*, sering kali dilakukan penyederhanaan notasi agar memudahkan manipulasi matriks. Salah satu cara yang umum digunakan adalah menggabungkan bias ke dalam vektor bobot. Penggabungan ini tidak mengubah makna persamaan, tetapi memungkinkan seluruh operasi dituliskan dalam bentuk perkalian vektor. Hal ini dilakukan dengan menambahkan sebuah *dummy input* $x_0 = 1$, lalu mendefinisikan $\hat{w} = (w_0, \mathbf{w})$ dan $\hat{x} = (x_0, \mathbf{x})$,

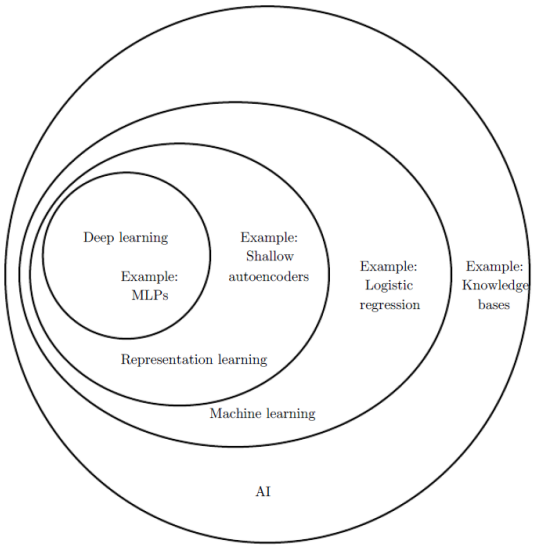
sehingga fungsi dapat dituliskan kembali sebagai Persamaan 2-7.

$$y(\mathbf{x}) = \hat{\mathbf{w}}^T \hat{\mathbf{x}} \tag{2-7}$$

Decision boundary dalam sebuah model klasifikasi saling terikat melalui vektor bobot, sehingga penentuan bobot tersebut akan sangat memengaruhi hasil klasifikasi. Untuk mendapatkan bobot yang optimal, salah satu pendekatan yang umum digunakan adalah metode *machine learning*. Dalam konteks ini, Jaringan Syaraf Tiruan menjadi salah satu teknik yang efektif karena mampu mempelajari pola dan hubungan kompleks antar *feature* secara otomatis.

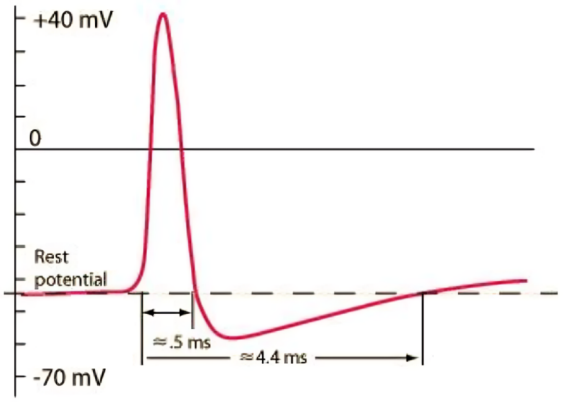
2.2.2 Jaringan Syaraf Tiruan

Jaringan Syaraf Tiruan (JST) merupakan salah satu metode dalam *machine learning* yang digunakan untuk mempelajari pola dari data. *Machine learning* sendiri adalah sebuah subset dari Kecerdasan Artifisial (KA) yang berfokus pada pengembangan algoritma yang dapat belajar dari data dan meningkatkan performanya seiring waktu tanpa harus diprogram secara eksplisit. Ketiga istilah tersebut memiliki bidang ilmu yang saling beririsan. Untuk memahami hubungan antara bidang ilmu tersebut, hubungan ketiganya dapat dilihat pada Gambar 2.2.



Gambar 2.2. Diagram Venn ini menggambarkan hubungan antara bidang ilmu [1]

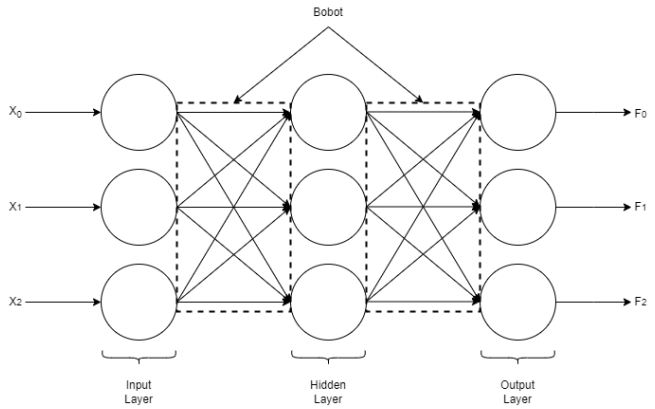
Jaringan Syaraf Tiruan pada awalnya terinspirasi dari jaringan syaraf biologis pada otak manusia [5]. Pada otak manusia *neuron-neuron* berkomunikasi satu sama lain melalui suatu sinyal listrik. Sinyal tersebut diteruskan dari satu *neuron* ke *neuron* lainnya melalui mekanisme yang dikenal sebagai *action potensial*. Seperti pada Gambar 2.3. *Neuron-neuron* yang saling terhubung ini memiliki tingkat kekuatan koneksi tertentu atau *synaptic strength*.



Gambar 2.3. *Action Potential* pada jaringan syaraf

Action Potential (AP) adalah sinyal listrik yang bergerak sepanjang akson *neuron*. Pada kondisi awal, membran *neuron* berada pada kondisi *resting potential*. Namun, ketika *neuron* menerima stimulus dan mencapai threshold, rangkaian proses akan terjadi. Proses ini akan membangkitkan sinyal listrik yang cepat dan sesaat. Sinyal tersebut kemudian merambat sepanjang akson dan menjadi stimulus baru untuk *neuron* berikutnya.

Pada jaringan syaraf tiruan, hubungan antar *neuron* dimodelkan secara matematis. Struktur jaringan umumnya tersusun dalam beberapa berlapis, dan di antara setiap lapisan terdapat bobot yang menentukan seberapa kuat suatu sinyal diteruskan ke lapisan berikutnya [5]. Jaringan syaraf tiruan menerima sinyal masukan melalui *input layer*, memprosesnya pada satu atau lebih *hidden layer*, dan menghasilkan keluaran pada *output layer*. Kedalaman sebuah jaringan ditentukan oleh jumlah *hidden layer* yang dimilikinya; jika terdapat dua atau lebih *hidden layer*, jaringan tersebut dapat dikategorikan sebagai *deep learning* [5]. Arsitektur umum jaringan syaraf tiruan ditunjukkan pada Gambar 2.4.



Gambar 2.4. Jaringan Syaraf Tiruan

Gambar 2.4 menunjukkan arsitektur dasar jaringan syaraf tiruan dengan *input layer*, satu *hidden layer*, serta *output layer*. Setiap garis penghubung antar *neuron* me-

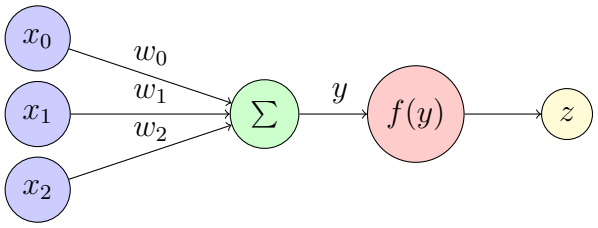
representasikan bobot yang menentukan kuat-lemahnya hubungan antar *neuron*. Melalui struktur berlapis ini, jaringan dapat mempelajari representasi data secara bertahap mulai dari *feature* sederhana hingga kompleks.

Pada jaringan syaraf tiruan, bobot berperan serupa dengan *synaptic strength* pada jaringan syaraf biologis. Nilai bobot akan diperbarui selama proses pelatihan berdasarkan perbedaan antara nilai keluaran model (*predicted value*) dan nilai target yang diharapkan (*expected value*). Dengan demikian, proses pelatihan dapat dipandang sebagai pencarian kombinasi bobot yang menghasilkan performa terbaik [5]. Terdapat berbagai jenis arsitektur dalam jaringan syaraf tiruan, seperti *multilayer perceptron*, *recurrent neural network*, *convolutional neural network*, dan lain-lain. Setiap arsitektur dirancang untuk karakteristik data tertentu; misalnya, *recurrent neural network* efektif untuk data sekuensial seperti sinyal suara, sedangkan *convolutional neural network* unggul dalam pengolahan data berbentuk grid seperti objek gambar.

2.2.3 *Perceptron*

Perceptron merupakan model paling sederhana dari neuron pada awal perkembangan jaringan syaraf tiruan. Model ini terdiri dari satu *input layer* dan satu *output layer*, sehingga sering disebut sebagai *single-layer neural network*. Perceptron menghasilkan keluaran biner, umumnya berupa {0, 1} atau {-1, 1}, dengan menggunakan fungsi aktivasi berupa *threshold function*.

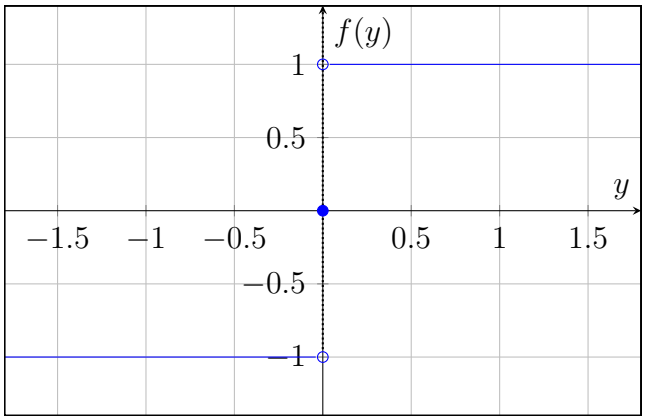
Secara umum, perceptron menerima sejumlah masukan, mengalikannya dengan bobot, lalu menjumlahkannya. Nilai hasil penjumlahan tersebut kemudian dimasukkan ke dalam fungsi aktivasi untuk menentukan kelas keluaran. Arsitektur dasar dari sebuah perceptron dapat dilihat pada Gambar 2.5.



Gambar 2.5. Perceptron yang menerima tiga input dan mengeluarkan satu output

Gambar 2.5 menunjukan perceptron dengan tiga buah masukan yaitu x_0, x_1, dan x_2. bobot-bobot dituliskan dengan simbol w_0, w_1, dan w_2. Kemudian y adalah kombinasi linier antara masukan dan bobotnya seperti pada Persamaan 2-1. Setelah melalui kombinasi linier keluaran diolah terlebih dahulu oleh fungsi $f(y)$. fungsi ini berfungsi untuk menentukan seberapa aktif neuron tersebut. fungsi ini disebut juga sebagai fungsi aktivasi pada jaringan syaraf tiruan modern. Terdapat berbagai macam fungsi aktivasi.

Namun, pada awal perkembangan JST fungsi yang umum dipakai adalah fungsi yang menggunakan *threshold* seperti *step function*.



Gambar 2.6. Output fungsi aktivasi *step function*

Step function adalah salah satu fungsi aktivasi yang non linier dan tidak kontinu. Keluaran dari *step function* dapat dilihat pada Gambar 2.6. *Step function* dapat ditulis seperti pada Persamaan 2-8.

$$f(y) = \begin{cases} -1 & \text{jika } y < 0 \\ 1 & \text{jika } y \geq 0 \end{cases} \tag{2-8}$$

Pembelajaran *perceptron algorithm* menggunakan *perceptron criterion*. yaitu dengan menggunakan Persamaan 2-9 sebagai berikut:

$$E_p(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n y_n \tag{2-9}$$

Untuk menurunkannya, diasumsikan bahwa sebuah vektor bobot $\mathbf{w}$ sedemikian sehingga setiap pola $\mathbf{x}_n$ yang termasuk ke dalam kelas C_1 menghasilkan nilai

$$\mathbf{w}^T \mathbf{x}_n > 0, \tag{2-10}$$

sedangkan pola yang termasuk ke dalam kelas C_2 menghasilkan nilai

$$\mathbf{w}^T \mathbf{x}_n < 0. \tag{2-11}$$

Dengan menggunakan target *expected value* sebagai $y_n \in \{-1, +1\}$, maka pola x_n yang diklasifikasikan dengan benar akan memenuhi persamaan berikut:

$$\mathbf{w}^T \mathbf{x}_n y_n > 0, \tag{2-12}$$

Pada *perceptron criterion*, sebuah pola yang diklasifikasikan dengan benar dianggap tidak memberikan *error*. Sebaliknya, untuk pola yang salah diklasifikasikan, fungsi error berusaha meminimalkan besaran

$$-\mathbf{w}^T \mathbf{x}_n y_n. \tag{2-13}$$

Dengan demikian, Persamaan 2-9 mendefinisikan fungsi *error perceptron* di mana M menyatakan himpunan seluruh pola yang salah diklasifikasikan. Bobot diperbarui menggunakan metode *stochastic gradient descent*. Aturan pembaruan bobot pada iterasi ke-t dituliskan sebagai berikut:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta\, \mathbf{x}_n y_n, \tag{2-14}$$

dengan η menyatakan *learning rate*.

2.2.4 *Multilayer Perceptron*

Multilayer Perceptron (MLP) merupakan salah satu arsitektur jaringan syaraf tiruan *feedforward* yang paling banyak digunakan dalam praktik *machine learning* dan *artificial intelligence*. MLP tersusun atas satu *input layer*, satu atau lebih *hidden layer*, serta satu *output layer*, di mana setiap neuron pada suatu lapisan terhubung penuh (*fully connected*) dengan neuron pada lapisan berikutnya [5] [14] [15]. Arsitektur ini banyak digunakan untuk tugas klasifikasi dan regresi karena kemampuannya dalam memodelkan hubungan non-linear antara data masukan dan keluaran.

Berbeda dengan perceptron tunggal yang hanya mampu menangani permasalahan yang bersifat *linearly separable*, MLP menggunakan fungsi aktivasi non-linear pada *hidden layer* sehingga mampu merepresentasikan pemetaan yang jauh lebih kompleks [16] [17]. Istilah *multilayer perceptron* sendiri sebenarnya dianggap kurang tepat, karena unit pemrosesan yang digunakan bukan lagi perceptron klasik dengan fungsi aktivasi diskrit, melainkan neuron dengan fungsi aktivasi kontinu seperti *sigmoid*, *hyperbolic tangent*, atau *ReLU* [13]. Jika seluruh lapisan dalam suatu jaringan hanya menggunakan transformasi linear, maka jaringan tersebut secara matematis dapat direduksi menjadi satu transformasi linear tunggal, sehingga keberadaan *hidden layer* tidak memberikan keuntungan representasional [13] [18].

Perkembangan MLP menjadi signifikan setelah diperkenalkannya algoritma *backpropagation* oleh Rumelhart, Hinton, dan Williams pada tahun 1986 [19]. Algoritma ini memungkinkan perhitungan gradien *error* terhadap setiap bobot jaringan secara efisien melalui dua tahap utama, yaitu propagasi maju (*forward propagation*) dan propagasi balik (*backpropagation*). Dengan mekanisme ini, kontribusi masing-masing bobot terhadap *error* keluaran dapat ditentukan, sehingga bobot pada *hidden layer* dapat diperbarui se-

cara sistematis menggunakan metode optimasi berbasis *gradient descent* [5] [20]. Tanpa adanya *backpropagation*, pelatihan jaringan dengan banyak *hidden layer* menjadi sangat sulit dilakukan [21].

Dari sudut pandang kemampuan representasi, MLP telah terbukti sebagai *universal function approximator*. Jaringan dengan satu *hidden layer* dan jumlah neuron yang memadai secara teoritis mampu mendekati fungsi kontinu apa pun pada domain terbatas [22]. Namun, dalam praktik, penggunaan beberapa *hidden layer* dengan ukuran yang lebih kecil sering kali lebih efisien dibandingkan satu *hidden layer* berukuran besar, baik dari segi jumlah parameter maupun kemudahan proses pelatihan [18]. Struktur bertingkat ini memungkinkan jaringan mengekstraksi *feature* secara hierarkis, di mana lapisan awal mempelajari representasi sederhana, sedangkan lapisan yang lebih dalam mempelajari representasi yang lebih abstrak [23].

MLP digunakan secara luas dalam berbagai aplikasi, khususnya pada tugas klasifikasi dan regresi. Pada tugas klasifikasi, *output layer* umumnya menggunakan fungsi aktivasi seperti *sigmoid* atau *softmax*, sedangkan pada regresi sering digunakan neuron keluaran tanpa fungsi aktivasi untuk menghasilkan nilai kontinu [15] [14]. Fungsi kerugian yang umum digunakan dalam pelatihan MLP antara lain *mean squared error* (MSE) dan *cross-entropy* [15]. Meskipun memiliki kemampuan representasi yang tinggi, proses pelatihan MLP tetap menghadapi tantangan berupa konvergensi ke *local minima* serta waktu komputasi yang relatif besar, terutama pada jaringan dengan banyak lapisan dan jumlah parameter yang besar [13] [21].

2.2.5 Proses Pelatihan

Proses pelatihan pada Multilayer Perceptron (MLP) bertujuan untuk memperoleh nilai bobot yang optimal sehingga jaringan mampu menghasilkan keluaran yang mendekati nilai target. Proses ini dilakukan secara iteratif melalui beberapa tahap utama, yaitu forward propagation, perhitungan loss function, backpropagation, dan pembaruan bobot menggunakan metode optimasi tertentu.

2.2.5.1 *forward propagation*

Forward propagation merupakan tahap awal dalam proses pelatihan MLP, di mana data masukan dipropagasikan dari *input layer* menuju *output layer*. Pada setiap neuron, sinyal masukan akan dikombinasikan secara linier dengan bobot yang bersesuaian dan ditambahkan dengan bias. Kombinasi linier tersebut kemudian dilewatkan ke fungsi aktivasi untuk menghasilkan keluaran neuron.

Secara umum, keluaran suatu neuron pada *layer* ke-l dapat dituliskan sebagai :

$$\mathbf{y}^{(l)} = f(\sum_i \mathbf{w}_i^{(l)} \mathbf{x}_i^{(l-1)} + w_0^{(l)}) \tag{2-15}$$

di mana $\mathbf{w}_i^{(l)}$ menyatakan vektor bobot, $\mathbf{x}_i^{(l)}$ menyatakan masukan *neuron*, $w_0^{(l)}$ adalah bias, dan f merupakan fungsi aktivasi. Proses ini dilakukan secara berurutan hingga diperoleh keluaran pada *output layer* sebagai *predicted value* dari jaringan.

2.2.5.2 *Loss Function*

Loss function adalah parameter yang digunakan untuk mengukur seberapa baik atau buruk kinerja suatu model dalam memprediksi nilai yang benar dari data yang diberikan. *loss function* merupakan komponen penting dalam proses pelatihan model. Dengan menurunkan nilai *loss function* model akan menjadi lebih baik [1]. Tujuan utama dari *loss function* adalah untuk memberikan umpan balik kepada model terkait seberapa baik model tersebut memperkirakan nilai yang benar dari data yang diamati. Dengan kata lain, *loss function* memberi tahu model seberapa jauh atau dekat prediksi model dengan nilai yang seharusnya.

Mean Square Error *Mean Squared Error* (MSE) merupakan fungsi loss yang paling umum digunakan pada permasalahan regresi. MSE mengukur rata-rata dari kuadrat selisih antara *expected value* dan *predicted value*.

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{2-16}$$

Dimana N adalah jumlah sampel data, y_n adalah *expected value* dari sampel ke-n, $\hat{y}_n$ adalah *predicted value* oleh model untuk sampel ke-n. MSE memberikan penalti kuadrat terhadap *error*, sehingga *error* yang besar berdampak lebih signifikan terhadap nilai loss. Sifat ini membuat MSE sangat sensitif terhadap *outlier*, tetapi juga mendorong model untuk menghasilkan prediksi yang sangat mendekati nilai target. Selain itu, MSE memiliki turunan yang kontinu dan mudah dihitung, sehingga mempermudah optimasi menggunakan *gradient descent*.

Fungsi Loss Lain Selain MSE, terdapat beberapa fungsi loss lain yang umum digunakan dalam berbagai konteks:

1. Binary Cross-Entropy (BCE) digunakan untuk klasifikasi biner, mengukur perbedaan antara probabilitas prediksi dan label sebenarnya.
2. Categorical Cross-Entropy (CCE) digunakan untuk klasifikasi multi-kelas.

3. Huber Loss kombinasi MSE dan MAE, memiliki karakteristik lebih tahan terhadap outlier.
4. Kullback–Leibler Divergence (KL Div) mengukur perbedaan dua distribusi probabilitas.

Namun, karena penelitian ini berfokus pada data numerik dan pengaruh arsitektur jaringan terhadap akurasi, maka MSE dipilih sebagai fungsi loss utama.

2.2.5.3 *Backpropagation*

Backpropagation adalah algoritma yang digunakan untuk melatih jaringan saraf tiruan, khususnya pada model *deep learning*. Algoritma ini memungkinkan jaringan untuk mengoptimalkan nilai bobot dan biasnya sehingga dapat meminimalkan nilai *loss function* [13]. Istilah *backpropagation* berasal dari *backward propagation of errors*. Algoritma ini bekerja dengan menyebarkan *error* dari *output layer* ke lapisan-lapisan sebelumnya.

Proses *backpropagation* terdiri dari dua tahap, yaitu *forward pass* dan *backward pass*. Pada tahap *forward pass*, *input* data dipropagasikan dari *input layer* hingga *output layer* untuk menghasilkan *predicted value*. Kemudian, *error* antara *predicted value* ($\hat{y}$) dan *expected value* (y) dihitung dengan *loss function*.

Pada tahap backward pass, gradien *loss function* terhadap keluaran jaringan dihitung, kemudian dipropagasikan kembali ke seluruh lapisan menggunakan aturan rantai (*chain rule*). Nilai gradien ini menunjukkan seberapa besar kontribusi masing-masing bobot terhadap *error* yang terjadi. Berdasarkan informasi tersebut, bobot dan bias diperbarui menggunakan metode optimisasi, seperti *gradient descent*, agar *error* pada iterasi berikutnya dapat dikurangi. Persamaan pembaruan bobot secara umum dapat dituliskan sebagai berikut:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial L}{\partial \mathbf{w}}$$

di mana $\mathbf{w}$ adalah vektor bobot, η adalah learning rate, dan $\frac{\partial L}{\partial \mathbf{w}}$ adalah gradien dari *loss function* terhadap bobot tersebut. Langkah-langkah ini diulangi untuk setiap *batch* data selama beberapa epoch sampai memenuhi kriteria tertentu. riteria tersebut dapat berupa *loss function* konvergen ke nilai minimum, *predicted value* sudah serupa dengan *expected value*, iterasi mencapai jumlah yang ditentukan, atau dengan membandingkan bobot sebelum dan sesudah pembaruannya dengan *relative error*.

2.2.5.4 Gradient Descent

Gradient descent adalah algoritma optimasi yang digunakan untuk meminimalkan *loss function* dalam berbagai model pembelajaran mesin dan jaringan saraf tiruan. Inti

dari algoritma ini adalah memperbarui parameter model secara iteratif untuk mengurangi nilai *loss function* [13]. Proses ini dilakukan dengan mengikuti arah gradien dari *loss function*.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t) \tag{2-17}$$

Dimana (θ_t) adalah parameter pada iterasi ke-t. η adalah laju pembelajaran (*learning rate*). $\nabla_\theta J(\theta_t)$ adalah gradien dari *loss function* J terhadap parameter θ pada iterasi ke-t. Cara kerja *gradient descent* dimulai dengan inisialisasi parameter model (θ) secara acak atau dengan nilai tertentu. Setelah itu, *loss function* $J(\theta)$ dievaluasi berdasarkan parameter saat ini dan data pelatihan. Langkah selanjutnya adalah menghitung gradien dari *loss function* terhadap parameter. Gradien adalah vektor yang menunjukkan arah dan laju perubahan dari *loss function*, yang secara matematis dinyatakan sebagai $\nabla_\theta J(\theta)$. Parameter kemudian diperbarui dengan bergerak ke arah gradien negatif untuk mengurangi nilai *loss function*. Pembaruan parameter dilakukan dengan menggunakan persamaan 2-17, dimana η adalah learning rate yang menentukan ukuran langkah yang diambil menuju minimum *loss function*. Langkah-langkah ini diulangi sampai *loss function* konvergen ke nilai minimum atau perubahan dalam *loss function* menjadi sangat kecil.

Varian Gradient Descent Gradient descent adalah algoritma inti dalam optimasi model pembelajaran mesin. Dengan memilih varian yang sesuai dan menggunakan teknik optimasi, model dapat dilatih dengan lebih efisien dan efektif, mencapai konvergensi yang lebih cepat dan hasil yang lebih baik.

1. Batch Gradient Descent: Menggunakan seluruh dataset untuk menghitung gradien dan memperbarui parameter [1]. Kelebihannya adalah konvergen menuju minimum global dengan lebih stabil, namun memerlukan komputasi yang besar jika dataset sangat besar.
2. Stochastic Gradient Descent (SGD): Menggunakan satu sampel data untuk menghitung gradien dan memperbarui parameter [1]. Kelebihannya adalah lebih cepat dan lebih efisien untuk dataset besar, namun gradien yang dihitung akan memiliki *noise* sehingga jalur konvergensi bisa lebih berliku-liku.
3. Mini-Batch Gradient Descent: Kombinasi antara batch gradient descent dan SGD. Menggunakan batch kecil (mini-batch) dari dataset untuk menghitung gradien dan memperbarui parameter [1]. Pendekatan ini menggabungkan kecepatan dan efisiensi dari SGD dengan kestabilan batch gradient descent.

Untuk meningkatkan stabilitas dan kecepatan konvergensi, SGD dapat dikombinasikan dengan metode momentum. Metode momentum memperhitungkan arah pembaruan bobot sebelumnya sehingga dapat mengurangi osilasi dan membantu jaringan keluar

dari local minima. Selain itu, terdapat metode optimasi lain seperti Adam yang mengadaptasi laju pembelajaran secara otomatis untuk setiap bobot, meskipun tidak selalu digunakan pada setiap penelitian.

Karena dataset yang digunakan pada penelitian ini relatif kecil dan model bersifat linier, maka algoritma Batch Gradient Descent dipilih untuk menjamin konvergensi yang stabil menuju solusi optimal. Penggunaan seluruh dataset pada setiap pembaruan bobot memastikan bahwa arah gradien yang dihitung benar-benar merepresentasikan keseluruhan distribusi data.

2.2.6 Normalisasi

Normalisasi adalah proses mengubah skala data agar setiap *feature* memiliki rentang nilai atau distribusi tertentu yang diinginkan. Nilai *feature* yang bervariasi secara signifikan dapat memengaruhi stabilitas proses pelatihan jaringan saraf tiruan. Pada algoritma *gradient descent*, pembaruan bobot dilakukan berdasarkan *gradien* dari fungsi *loss*. Jika skala antar *feature* berbeda jauh, *gradien* yang dihasilkan juga menjadi tidak seimbang. *feature* dengan nilai besar cenderung menghasilkan pembaruan bobot yang terlalu besar, sedangkan *feature* dengan nilai kecil menghasilkan pembaruan yang lambat. Akibatnya, proses konvergensi dapat menjadi tidak stabil atau bahkan gagal [24].

Sebagai ilustrasi diberikan data seperti pada Tabel 2.3 yang menunjukan harga rumah di Jakarta.

Luas Tanah (m^2)	Jumlah Kamar Tidur	Usia Rumah (tahun)	Jarak ke Pusat Kota (km)	Harga Rumah (IDR)
300	3	10	15	1,500,000,000
500	4	5	10	2,000,000,000
450	3	20	20	1,750,000,000
600	5	8	5	2,500,000,000

Tabel 2.3. Data *dummy* harga rumah di Jakarta

Jika *raw* data ini langsung digunakan ke algoritma pembelajaran seperti regresi linear atau jaringan syaraf tiruan, *feature* "luas tanah" akan mendominasi proses pelatihan karena memiliki skala yang jauh lebih besar dibandingkan "Jumlah Kamar Tidur" hal ini akan menimbulkan bias bahwa "luas tanah" lebih penting dari "Jumlah Kamar Tidur". Untuk mengatasi masalah ini, dapat dilakukan normalisasi pada setiap *feature* sehingga semua *feature* memiliki skala yang serupa. Salah satu metode normalisasi yang umum adalah Min-Max Scaling, yang mengubah skala setiap *feature* ke dalam rentang 0 hingga

1. Persamaan *Min-Max Scaling* (2-18)dituliskan sebagai berikut:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2-18}$$

Selain *Min-max scaling*, terdapat metode normalisasi lain yaitu *Z-score standardization*. *Z-score standardization* adalah metode normalisasi data yang mengubah data sehingga memiliki nilai rata-rata 0 dan standar deviasi 1 [24]. Persamaan *Z-score standardization* (2-19):

$$Z = \frac{X - \mu}{\sigma} \tag{2-19}$$

Dengan normalisasi, setiap *feature* berkontribusi secara proporsional terhadap proses pembelajaran jaringan, sehingga mempercepat konvergensi, menstabilkan pelatihan, dan meningkatkan akurasi prediksi model.

2.2.7 *Relative error*

Relative error adalah ukuran ketidakakuratan dalam pengukuran atau estimasi yang dinyatakan sebagai perbandingan antara *absolute error* dan *true value*. Persamaan dapat ditulis sebagai berikut :

$$\text{Relative Error} = \frac{\text{Absolute Error}}{\text{True Value}} \tag{2-20}$$

Dimana *Absolute Error* pada konteks ini adalah selisih antara *True Value* dan *Measured Value*. Dalam pelatihan jaringan syaraf tiruan, *relative error* sering digunakan sebagai kriteria penghentian untuk menentukan kapan pelatihan harus dihentikan. Penggunaan *relative error* sebagai parameter penghentian memiliki beberapa manfaat penting. Pertama, ini memungkinkan pelatihan dihentikan ketika *relative error* mencapai nilai yang cukup kecil, menunjukkan bahwa model telah mencapai tingkat akurasi yang diinginkan. Kedua, ini membantu mencegah *overfitting*, dimana model menjadi terlalu terlatih pada data pelatihan dan tidak dapat digeneralisasi dengan baik pada data baru. Ketiga, penggunaan *relative error* sebagai kriteria penghentian menghemat waktu dan sumber daya komputasi, karena pelatihan lebih lanjut mungkin tidak memberikan peningkatan signifikan dalam performa model. Terakhir, *relative error* memberikan penilaian kinerja yang bersifat relatif terhadap skala nilai yang diukur, sehingga memudahkan evaluasi dan perbandingan kinerja model pada berbagai dataset atau aplikasi yang berbeda. Dengan demikian, *relative error* sebagai parameter pemberhenti memastikan bahwa pelatihan JST dilakukan secara efisien dan menghasilkan model yang memiliki kinerja baik pada data.

BAB III
METODE PENELITIAN

3.1 Alat dan Bahan Tugas akhir

3.1.1 Alat Tugas akhir

Pengerjaan penelitian ini menggunakan alat dan bahan. alat yang digunakan berupa perangkat keras seperti laptop untuk menjalankan program, software, serta library yang akan digunakan.

1. *Laptop* dengan spesifikasi sebagai berikut :
   - Sistem operasi Windows 10
   - Processor Intel I7-8750H
   - Memori 16GB DDR4
   - NVIDIA GeForce GTX 1050 (4GB)
   - SSD Samsung MZNLN128HAHQ 128GB
   - HDD HGST HTS721010A9E630
2. Bahasa pemrograman python 3.10.12 sebagai interpreter. Digunakan sebagai bahasa pemrograman utama.
3. Visual Studio Code
4. Google Colaboratory sebagai *cloud computing* yang digunakan.
5. Numpy Library untuk melakukan perhitungan matriks pada bahasa pemrograman python. Library utama yang digunakan untuk melakukan perhitungan matriks.

3.1.2 Bahan Tugas akhir

Dataset yang digunakan berupa data tinggi dan lebar suatu lemari, *buffet*, dan *wardrobe*. Dataset berupa data tabular dan dapat dipisahkan secara linear. Dataset ini bersifat *dummy* namun dengan memperhatikan dengan skala aslinya.

3.2 Metode yang Digunakan

Penelitian ini menggunakan pendekatan *kuantitatif* untuk menganalisis pengaruh jumlah *hidden layer* terhadap akurasi dan waktu komputasi pada jaringan saraf tiruan. Evaluasi kinerja model dilakukan dengan mengukur nilai akurasi dan waktu pelatihan, yang selanjutnya dianalisis menggunakan statistik deskriptif dan uji signifikansi.

Penelitian ini dilakukan dengan mengembangkan beberapa model jaringan sa-

raf tiruan yang dilatih menggunakan dataset yang sama. Seluruh proses pengembangan dan pelatihan model diimplementasikan menggunakan bahasa pemrograman *Python* versi 3.10.12 dan dijalankan pada platform *Google Colaboratory* berbasis *cloud computing*.

Sebelum proses pelatihan, dataset diproses terlebih dahulu menggunakan metode normalisasi *Min–Max Scaling* untuk menyamakan rentang nilai setiap *feature*. Data yang telah dinormalisasi kemudian dipropagasikan ke dalam jaringan melalui proses *forward propagation* hingga mencapai *output layer*. Keluaran jaringan digunakan untuk menghitung nilai *loss function*, yang merepresentasikan selisih antara *predicted value* dan *expected value* yang diberikan. Pada penelitian ini, *Mean Squared Error* (MSE) digunakan sebagai *loss function*. Nilai MSE diminimalkan selama proses pelatihan menggunakan metode *Gradient Descent*. Setelah model dibuat, nilai parameter terbaik dicari untuk model tersebut.

Untuk menganalisis pengaruh jumlah *hidden layer*, jaringan saraf tiruan dibangun dengan beberapa konfigurasi arsitektur yang berbeda. Terdapat tiga model utama yang digunakan, yaitu:

1. Model pertama
2. Model kedua
3. Model ketiga

Seluruh model menggunakan fungsi aktivasi *step function* pada *output layer*. Proses pelatihan dilakukan menggunakan metode *Gradient Descent* untuk meminimalkan nilai *Mean Squared Error* (MSE). Selama proses pelatihan, nilai bobot jaringan diperbarui secara iteratif hingga tercapai kondisi konvergensi, yang ditandai dengan penurunan nilai MSE dan stabilitas perubahan bobot.

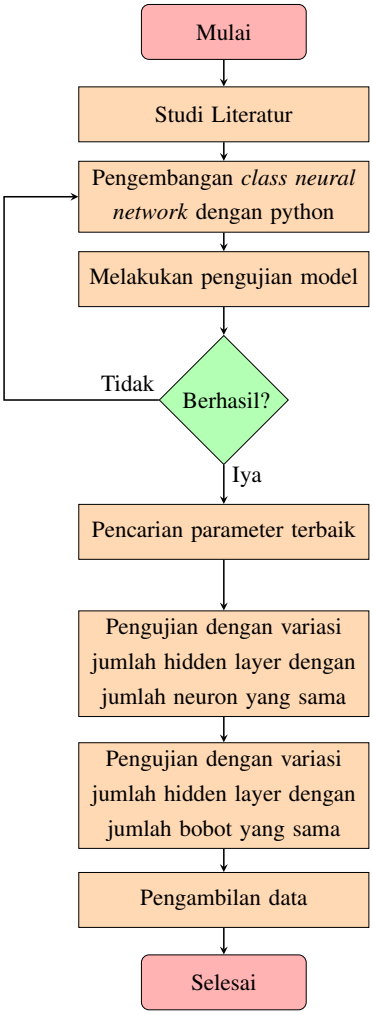
3.2.1 Implementasi Algoritma Pelatihan

3.2.2 *Dataset* yang digunakan

3.2.3 Desain Eksperimen

3.3 Alur Tugas Akhir

Penelitian ini dilakukan secara bertahap dengan alur seperti pada Gambar 3.7. Studi literatur dilakukan pada awal penelitian untuk memahami konsep dan teori yang akan digunakan pada penelitian. Selanjutnya, dilakukan Pengembangan program jaringan syaraf tiruan dengan bahasa pemrograman python pada *google colaboratory*. Setelah program dibuat, maka dilakukan pengujian untuk mengetahui apakah program berhasil atau tidak. Penelitian diakhiri dengan pengambilan data pengujian dan penyusunan laporan akhir.



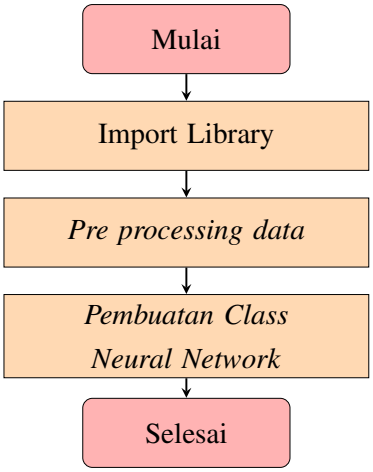
Gambar 3.7. *Flowchart* alur penelitian

3.3.1 Studi Literatur

Studi literatur merupakan tahap awal pada penelitian agar dapat memahami lebih lanjut topik penelitian yang dilakukan. Studi literatur dilakukan dengan mengumpulkan materi dan referensi dari berbagai sumber seperti artikel, jurnal penelitian, forum, dan situs web.

3.3.2 Pengembangan program jaringan syaraf tiruan dengan python

Pengembangan program dilakukan pada *Google Colaboratory* dengan menggunakan bahasa pemrograman python. Diagram alir dari program terlihat pada Gambar 3.8



Gambar 3.8. Diagram alir program

3.3.2.1 Import Library

```
1 import numpy as np
2 import pandas as pd
```

Import library digunakan untuk memasukan library yang dibutuhkan kedalam program. Library yang dibutuhkan adalah library numpy dan library pandas. Numpy digunakan untuk melakukan pengoperasian matriks, sedangkan pandas digunakan untuk mengambil sumber data yang berbentuk file csv.

3.3.2.2 *pre-processing* data

Dataset pada Tabel **??** tersebut masih dalam bentuk data mentah atau *raw*. Data tersebut perlu diolah terlebih dahulu untuk memudahkan proses pelatihan. Beberapa pengolahan data yang diperlukan adalah pemberian label dan juga normalisasi data. Label dibutuhkan terutama pada algoritma *supervised learning* seperti pada model ANN. Label ini disebut juga sebagai *expected value* karena label ini yang diharapkan keluar dari model yang dilatih. Terdapat beberapa metode dalam pemberian label. *One vs All* adalah metode yang akan digunakan. Data akan diberi nilai 1 pada label yang bersesuaian dan -1 pada label lainnya. Kemudian data tersebut dinormalisasikan. Normalisasi dilakukan dengan metode Minmax Scaling.

No	Lebar	Tinggi	C_0	C_1	C_2
1	1.1	1.3	1	-1	-1
2	1.1	1.5	1	-1	-1
3	0.7	1.5	1	-1	-1
4	0.6	1.9	1	-1	-1
5	0.8	2	1	-1	-1
6	1.2	0.6	-1	1	-1
7	1.6	0.5	-1	1	-1
8	1.7	0.7	-1	1	-1
9	2.2	0.4	-1	1	-1
10	2.2	0.5	-1	1	-1
11	2.3	0.8	-1	1	-1
12	1.5	0.5	-1	1	-1
13	2	2	-1	-1	1
14	1.8	2.2	-1	-1	1
15	1.5	1.8	-1	-1	1
16	2.3	1.5	-1	-1	1

Tabel 3.1. Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah diberi label

Tabel 3.1 menunjukan bagaimana pemberian label dilakukan pada kelas lemari. Dimana C_0 diberikan nilai 1 sedangkan C_1 dan C_2 bernilai -1. Pada kelas buffet C_1 diberikan nilai 1 sedangkan C_0 dan C_2 bernilai -1. Hal tersebut dilakukan juga pada kelas Wardrobe. Oleh karena itu metode ini disebut *One vs All*. Setelah dataset diberikan label dengan metode *One vs All*. Kemudian data disimpan kedalam *Google Drive* dalam format csv.

```
1    from google.colab import drive
2    drive.mount('/content/drive/')
3    data = pd.read_csv('/content/drive/MyDrive/skripsi/data/
   datalemari(-1,1).csv')
4    data.head()
```

Sumber data yang tersimpan pada *Google Drive* perlu diakses agar dapat digunakan kedalam bahasa pemrograman *python*. *Google Drive* digunakan untuk memudahkan pengaksesan data oleh *Google Colaboratory* karena masih dalam satu *environment*. Ke-

mudian Library Pandas digunakan untuk menyimpan data file csv menjadi bentuk *Data-Frame* pada python dengan cara mengisi lokasi dari file csv data pada perintah *read_csv*. Setelah data disimpan kedalam program selanjutnya data akan di *pre-processing* terlebih dahulu dengan normalisasi *minmax* dengan Persamaan 2-18.

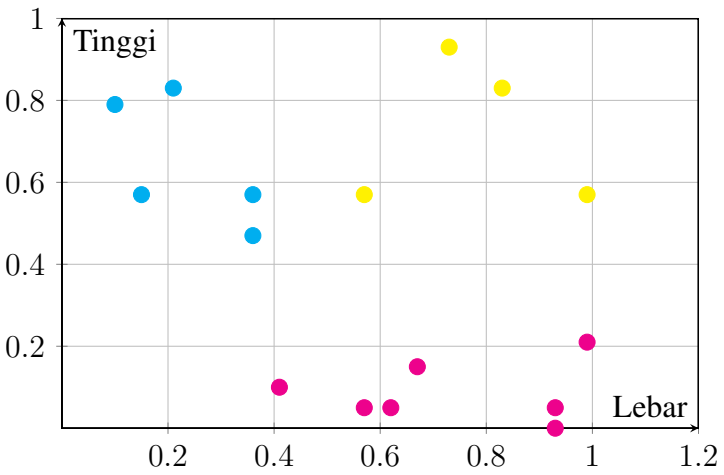
```
1    def minmax_scaling (x_t):
2        min = np.min(x_t)
3        max = np.max(x_t)
4        return ((x_t-min)/(max-min))
5
6    X = minmax_scaling(data.iloc[:,:2].values.T)
7    y = data.iloc[:,2:5].values.T
```

Fungsi diatas menggunakan Library numpy.max dan numpy.min untuk mencari nilai maksimum dan minimum pada array *x_t*. Kemudian disimpan pada variabel X. Sedangkan variabel y akan menyimpan nilai label atau *expected value* dari dataset.

No	Lebar	Tinggi	C_0	C_1	C_2
1	0.36	0.47	1	-1	-1
2	0.36	0.57	1	-1	-1
3	0.15	0.57	1	-1	-1
4	0.10	0.78	1	-1	-1
5	0.21	0.83	1	-1	-1
6	0.41	0.10	-1	1	-1
7	0.62	0.05	-1	1	-1
8	0.67	0.15	-1	1	-1
9	0.93	0.00	-1	1	-1
10	0.93	0.05	-1	1	-1
11	0.99	0.21	-1	1	-1
12	0.57	0.05	-1	1	-1
13	0.83	0.83	-1	-1	1
14	0.73	0.93	-1	-1	1
15	0.57	0.57	-1	-1	1
16	0.99	0.57	-1	-1	1

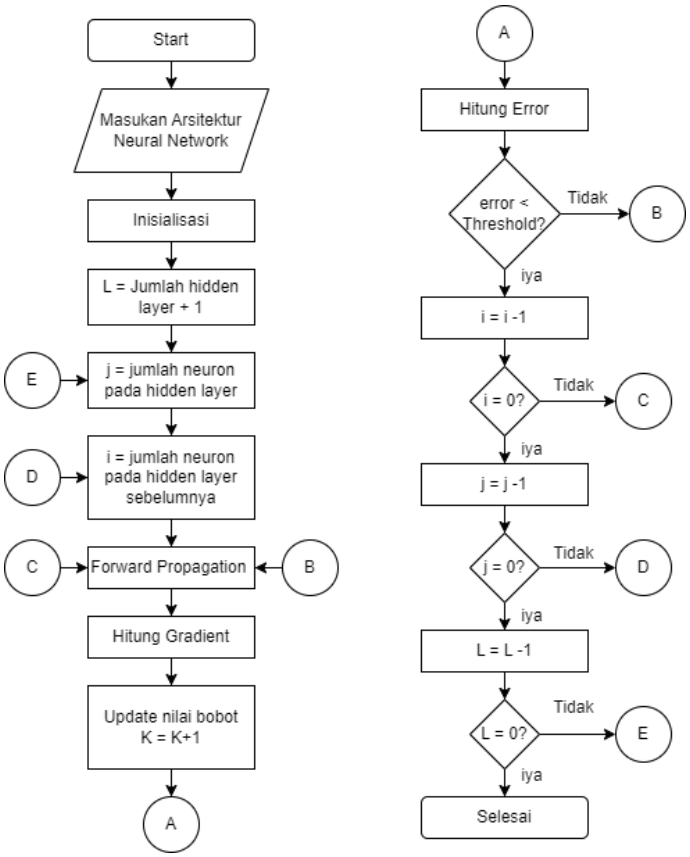
Tabel 3.2. Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah normalisasi

Table 3.2 menunjukan data telah diubah ukurannya menjadi rentang 0 hingga 1 setelah melalui proses normalisasi Minmax Scaling. Gambar 3.9 menunjukkan plot kartesian dari Table 3.2.



Gambar 3.9. Grafik dari dataset setelah melalui Minmax Scaling. *Cyan* adalah lemari, *magenta* adalah buffet, dan kuning adalah wardrobe

3.3.2.3 Pembuatan *Class Neural Network*



Gambar 3.10. Diagram Alir Class Neural Network

Pembuatan *class* ditunjukan agar dengan satu *source code* dapat dilakukan percobaan dengan nilai jaringan tersembunyi yang dapat dirubah-ubah sesuai dengan model yang diinginkan. Pembuatan kelas juga dapat membantu penulis dalam melakukan perbaikan pada program. Diagram alir dari *Class Neural Network* ditunjukkan pada Gambar 3.10.

```
1     def __init__(self, input_size, hidden_size, output_size,
    weight_init=0.5):
2         self.input_size = input_size
3         self.hidden_size = hidden_size
4         self.output_size = output_size
5         self.num_layers = np.concatenate(([input_size],
    hidden_size, [output_size]))
6         self.L = len(self.num_layers)-1
7         self.weight = {}
8         self.Y = {}
9         self.F = {}
10        self.delta = {}
11        self.dJ_dF = {}
12        self.dF_dY = {}
13        self.dY_dF = {}
14        self.dJ_dw = {}
15        self.t = 0
16        self.d2J_dw2 = {}
17        self.accuracy = []
18        self.mse = []
19
20        if weight_init == 'random':
21            for l in range(0, self.L):
22                self.weight[f'w{l}'] = np.random.rand(self.
    num_layers[l+1], self.num_layers[l]+1)
23                self.dJ_dw[f'{l}'] = np.array([])
24
25        else:
26            for l in range(0, self.L):
27                self.weight[f'w{l}'] = weight_init * np.ones((
    self.num_layers[l+1], self.num_layers[l]+1))
28                self.dJ_dw[f'{l}'] = np.array([])
```

Inisialisasi adalah fungsi yang pertama kali dijalankan ketika *Class Neural Network* dipanggil. Inisialisasi akan membuat variabel-variabel yang diperlukan oleh neural network. Termasuk didalamnya adalah bobot awal yang akan digunakan selama

pelatihan. Sebagai contoh pada Gambar 2.4 jaringan tersebut memiliki satu *hidden layer* sehingga jaringan memiliki dua buah bobot maka L akan bernilai 2, bobot tersebut adalah W^1 dan W^2.

$$W^1 = \begin{bmatrix} W^1_{0,0} & W^1_{0,1} & W^1_{0,2} \\ W^1_{1,0} & W^1_{1,1} & W^1_{1,2} \\ W^1_{2,0} & W^1_{2,1} & W^1_{2,2} \end{bmatrix} W^2 = \begin{bmatrix} W^2_{0,0} & W^2_{0,1} & W^2_{0,2} \\ W^2_{1,0} & W^2_{1,1} & W^2_{1,2} \\ W^2_{2,0} & W^2_{2,1} & W^2_{2,2} \end{bmatrix} \tag{3-1}$$

$$W^1 = W^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \tag{3-2}$$

Setelah dibuat variabel untuk bobot maka langkah selanjutnya adalah dengan menentukan nilai bobot awal. Bobot awal ini umumnya dideklarasikan dengan suatu fungsi random. Namun karena fokus pada penelitian ini adalah jumlah dari hidden layer, maka nilai bobot akan dibuat sama, yaitu baris pertama akan bernilai 1 untuk mendistribusikan nilai bias. Kemudian baris lainnya akan bernilai 0.5 sama seperti pada persamaan 3-2.

```
1     def ForwardPropagation(self, X):
2         self.X = np.append(np.ones((1, self.N), X, axis = 0)
3 
4         for l in range (0, self.L):
5             if l == 0 :
6                 self.Y[f'{l}'] = np.dot(self.weight['w0'], self.
    X)
7                 self.F[f'{l}'] = self.tanh(self.Y[f'{l}'])
8                 self.F[f'{l}'] = np.append(np.ones((1,self.N)),
    self.F[f'{l}'], axis = 0)
9             elif l == len(self.hidden_size):
10                self.Y[f'{l}'] = np.dot(self.weight[f'w{l}'],
    self.F[f'{l-1}'])
11                self.F[f'{l}'] = self.tanh(self.Y[f'{l}'])
12                self.Y_out = self.Y[f'{l}']
13                self.F_out = self.F[f'{l}']
14                self.F_step = self.step_function(self.F[f'{l}'])
15            else :
16                self.Y[f'{l}'] = np.dot(self.weight[f'w{l}'],
    self.F[f'{l-1}'])
17                self.F[f'{l}'] = self.tanh(self.Y[f'{l}'])
18                self.F[f'{l}'] = np.append(np.ones((1,self.N)),
```

```
        self.F[f'{l}'], axis = 0)
19          return
```

Setelah proses inisialisasi selesai dan semua variabel yang diperlukan sudah dibuat. Langkah selanjutnya adalah dengan melakukan *Forward Porpagation* dengan menggunakan Persamaan 2-7. Setelah *forward propagation* dilakukan hingga mencapai layer terakhir. Sinyal keluaran pada layer terakhir disebut juga sebagai *predicted value*. *Predicted value* ini bersamaan dengan *expected value* dimasukkan kedalam *loss function* MSE sebagai parameter *error* dari model. Umumnya pada awal pelatihan nilai dari *loss function* akan cukup tinggi. Semakin kecil *loss function* semakin baik pula modelnya. Untuk menurunkan nilai *loss function* dapat menggunakan *Gradient descent* pada persamaan 2-7. Yaitu dengan menggunakan turunan *loss function* terhadap parameter yang ingin dilatih. Karena dataset yang digunakan cukup kecil maka seluruh data akan digunakan kedalam data pelatihan. berikut persamaan 3-3 dan 3-4:

$$\frac{d\text{MSE}}{dw^l_{(j,i)}} = \frac{d\frac{1}{N}\sum_{n=1}^{N}(d_n - y_n)^2}{dw^l_{(j,i)}} \tag{3-3}$$

$$\frac{d^2\text{MSE}}{d(w^l_{(j,i)})^2} = \frac{d^2\frac{1}{N}\sum_{n=1}^{N}(d_n - y_n)^2}{d(w^l_{(j,i)})^2} \tag{3-4}$$

Dimana $w^l_{(j,i)}$ adalah bobot ke-l, yang terhubung dengan neuron j dan neuron i, $l = 1, 2, ..., L$, d_n adalah *expected value* ke-n atau label ke-n, dan y_n adalah *predicted value* ke-n. Setelah mendapatkan gradiennya dengan Persamaan 3-3 dan Persamaan 3-4 nilai tersebut dimasukkan untuk memperbaharui bobot $w^l_{(j,i)}$. Dengan menggunakan aturan rantai maka penggunaannya diterapkan pada fungsi update seperti berikut :

```
1     def update(self, y):
2         n = self.X.shape[1]
3 
4         for l in range(self.L, -1, -1):
5             if l == len(self.hidden_size):
6                 self.dY_dY[f'{l}'] = self.weight[f'w{l}']
7                 self.delta[f'{l}'] = (-2 / n)  * (y-self.Y[f'{l
    }'])
8                 self.dJ_dw[f'{l}'] = np.dot(self.delta[f'{l}'],
    self.Y[f'{l-1}'].T)
9                 temp = self.delta[f'{l}']
10 
11                self.d2J_dw2[f'{l}'] = (2 / n) * np.sum(self.Y[f
    '{l-1}']**2, axis=1)
12                self.d2J_dw2[f'{l}'] = np.outer(np.ones(self.
```

```
output_size), self.d2J_dw2[f'{l}'])
                temp_2 = (2 / n) * np.sum(self.weight[f'w{l
}']**2, axis=0)

            elif l == 0:
                self.delta[f'{l}'] = np.dot(self.dY_dY[f'{l
+1}'].T, temp)
                self.dJ_dw[f'{l}'] = np.dot(self.delta[f'{l}'],
self.X.T)

                self.d2J_dw2[f'{l}'] = np.outer(temp_2[:, np.
newaxis], np.sum(self.X**2, axis =1)[np.newaxis, :])

            else:
                self.dY_dY[f'{l}'] = self.weight[f'w{l}']
                self.delta[f'{l}'] = np.dot(self.dY_dY[f'{l
+1}'].T, temp)
                self.dJ_dw[f'{l}'] = np.dot(self.delta[f'{l}'],
self.Y[f'{l-1}'].T)
                temp = self.delta[f'{l}']

                self.d2J_dw2[f'{l}'] = np.outer(temp_2[:, np.
newaxis], np.sum(self.Y[f'{l-1}']**2, axis=1)[np.newaxis, :])
                temp_2 = np.dot(temp_2, self.weight[f'w{l}']**2)

        return
```

Setelah mendapatkan *gradient* langkah selanjutnya adalah *backpropagation* yaitu dengan mendistribusikan *gradient* tersebut ke semua bobot yang ada dan melakukan pembaharuan bobot. Pembaharuan bobot menggunakan Persamaan 2-17. Pembaharuan bobot dilakukan satu persatu yaitu dimulai dengan bobot terdepan yaitu bobot $w^l_{j,i}$. Pelatihan dilakukan pada satu bobot tersebut secara berulang hingga memenuhi kriteria *threshold* pada *relative error*, jika syarat sudah terpenuhi pelatihan pada bobot tersebut berhenti dan dilanjutkan pada bobot selanjutnya. Hal ini dilakukan berulang hingga semua bobot telah dilatih atau jika nilai *predicted value* sudah serupa dengan nilai *expected value*. Penggunaan i, j, dan l untuk memastikan agar semua bobot disetiap layer telah melalui proses pelatihan. Sedangkan t adalah banyak iterasi yang dilakukan karena tiap bobot akan melakukan jumlah pelatihan yang berbeda. t akan digunakan untuk membandingkan waktu komputasi yang dilaksanakan. Penggunaan algoritam tersebut akan disimpan pada fungsi *backpropagation* didalam *class Neural Network* seperti berikut:

```
def Backpropagation(self, X, y, learning_rate=0.1):
```

```
        self.ForwardPropagation(X)
        break_all_loop = False
        for l in range(self.L, -1, -1):
          for j in range(self.num_layers[w+1]):
            for i in range(self.num_layers[w]):
              while True:
                self.update(y)
                self.t +=1

                self.accuracy.append(self.accuracy_func(y, self.
F))
                self.mse.append(self.mse_func(y, self.Y_out))

                delta_t = learning_rate * (self.dJ_dw[f'{l}'][j,
 i]/self.d2J_dw2[f'{l}'][j,i])
                w_n = self.weight[f'w{l}'][j, i] - delta_t

                error = self.error(w_n, self.weight[f'w{l}'][j,
i])

                self.weight[f'w{l}'][j, i] = w_n
                self.ForwardPropagation(X)

                if np.sum(np.abs(y-self.F)) == 0:
                  self.accuracy.append(self.accuracy_func(y,
self.F))
                  self.mse.append(self.mse_func(y, self.Y_out))
                  break_all_loop = True
                  break
                if error < 0.1 :
                  break

              if break_all_loop == True :
                break
            if break_all_loop == True :
              break
          if break_all_loop == True :
            break

        return
```

3.3.3 Pengujian program jaringan syaraf tiruan

Pengujian program jaringan syaraf tiruan dilakukan untuk mengetahui apakah program sudah sesuai dengan perintah yang diharapkan. Salah satu faktor penentu program tersebut telah berjalan dengan baik adalah dengan memperhatikan nilai *loss function*-nya. Pelatihan jaringan syaraf tiruan pada program yang dibuat menggunakan metode *gradient descent*, sehingga nilai dari *loss function* dari program yang benar akan berkurang seiring dengan bertambahnya iterasi pada model tersebut.

3.3.4 Pengujian dengan satu hidden layer dengan variasi jumlah neuron

```
1  nn1 = {}
2  nn1_mse_iterasi = {}
3  nn1_accuracy_iterasi = {}
4  for h_n in range (1, 11):
5    print(f'Hidden Neuron{h_n}')
6    nn1[f'{h_n}'] = NeuralNetwork(2, [h_n], 3, weight_init
   =0.5)
7    nn1[f'{h_n}'].Backpropagation(X, y)
8    nn1_mse_iterasi[f'{h_n}'] = []
9    nn1_accuracy_iterasi[f'{h_n}'] = []
10   for t in range (1, 10000):
11     old_mse=nn1[f'{h_n}'].mse[-1]
12     nn1_mse_iterasi[f'{h_n}'].append(old_mse)
13     nn1_accuracy_iterasi[f'{h_n}'].append(nn1[f'{h_n}'].
   accuracy[-1])
14     nn1[f'{h_n}'].Backpropagation(X, y)
15     nn1[f'{h_n}'].ForwardPropagation(X)
16     new_mse=nn1[f'{h_n}'].mse[-1]
17     if new_mse == old_mse :
18       print(f'error epoch ke-{t} :',np.sum((y-nn1[f'{h_n}'].
   Y_out)**2)/16)
19       break
20     if t % 100 == 0 :
21       print(f'error epoch ke-{t} :',np.sum((y-nn1[f'{h_n}'].
   Y_out)**2)/16)
22     if np.sum(np.abs(y-nn1[f'{h_n}'].F)) == 0:
23       nn1_mse_iterasi[f'{h_n}'].append(new_mse)
24       nn1_accuracy_iterasi[f'{h_n}'].append(nn1[f'{h_n}'].
   accuracy[-1])
25       print(f'error epoch ke-{t} :',np.sum((y-nn1[f'{h_n}'].
   Y_out)**2)/16)
```

```
26              break
```

Pada *Source Code* program akan membuat *Class Neural Network* dengan satu hidden layer namun dengan hidden neuron yang bervariasi. Variasi hidden neuron dilakukan dari 1 hingga 10 hidden neuron. Didalam *Source Code* tersebut juga menyimpan beberapa variabel yang akan dianalisis lebih lanjut seperti nilai dari MSE dan juga akurasinya pada iterasi ke t.

3.3.5 Pengujian dengan variasi jumlah hidden layer dengan jumlah neuron yang sama

```
1     nn_vhl = {}
2     nn_vhl_mse_iterasi = {}
3     nn_vhl_accuracy_iterasi = {}
4     for h_n in range (3, 7):
5         start_hidden_layers = 1
6         last_hidden_layers = 4
7         hidden_neuron = []
8         for h_l in range (start_hidden_layers,
    last_hidden_layers+1):
9             print(f'number of hidden layer : {h_l}')
10            print(f'number of neuron each hidden layer : {h_n}')
11            hidden_neuron.append(h_n)
12            nn_vhl[f'{h_n}{h_l}'] = NeuralNetwork(2,
    hidden_neuron, 3, weight_init= 0.3)
13            nn_vhl[f'{h_n}{h_l}'].backward(X, y)
14            nn_vhl_mse_iterasi[f'{h_n}{h_l}'] = []
15            nn_vhl_accuracy_iterasi[f'{h_n}{h_l}'] = []
16            for i in range (10000):
17            old_mse = nn_vhl[f'{h_n}{h_l}'].mse[-1]
18            nn_vhl_mse_iterasi[f'{h_n}{h_l}'].append(old_mse)
19            nn_vhl_accuracy_iterasi[f'{h_n}{h_l}'].append(nn_vhl
    [f'{h_n}{h_l}'].accuracy[-1])
20            nn_vhl[f'{h_n}{h_l}'].backward(X, y, learning_rate
    =0.1)
21            nn_vhl[f'{h_n}{h_l}'].forward(X)
22            new_mse = nn_vhl[f'{h_n}{h_l}'].mse[-1]
23            nn_vhl[f'{h_n}{h_l}'].backward(X, y, 0.1)
24            nn_vhl[f'{h_n}{h_l}'].forward(X)
25            if new_mse == old_mse :
26              print(f'error epoch ke-{i} :',np.sum((y-nn_vhl[f'{
    h_n}{h_l}'].Y_out)**2)/16)
```

```
27                 break
28             if i % 100 == 0 :
29                 print(f'error epoch ke-{i} :',np.sum((y-nn_vhl[f'{
    h_n}{h_l}'].Y_out)**2)/16)
30             if np.sum(np.abs(y-nn_vhl[f'{h_n}{h_l}'].F)) == 0:
31                 nn_vhl_mse_iterasi[f'{h_n}{h_l}'].append(new_mse)
32                 nn_vhl_accuracy_iterasi[f'{h_n}{h_l}'].append(
    nn_vhl[f'{h_n}{h_l}'].accuracy[-1])
33                 print(f'error epoch ke-{i} :',np.sum((y-nn_vhl[f'{
    h_n}{h_l}'].Y_out)**2)/16)
34                 break
```

Pada pengujian ini yaitu dengan melakukan variasi jumlah hidden layer degan jumlah neuron yang sama pada tiap hidden layernya. variasi jumlah hidden layer dilakukan dari satu hidden layer hingga empat hidden layer. Sedangkan jumlah Hidden Neuron yang dibuat sama dilakukan dari 3 Hidden Neuron hingga 6 Hidden Neuron. Variabel h_l akan digunakan untuk menyimpan hidden layer yang digunakan. Variabel h_n digunakan untuk menyimpan jumlah hidden neuron yang digunakan

3.3.6 Pengujian dengan variasi jumlah hidden layer dengan jumlah bobot yang sama

Pada pengujian ini diperlukan jumlah bobot yang sesuai untuk bisa digunakan pada jumlah hidden layer yang bervariasi. Dengan menggunakan faktor-faktor darijumlah bobot yang akan digunakan, ditemukan bobot yang dapat digunakan adalah 72, 105, 144, dan 189.

```
1  nn_34643 = NeuralNetwork(2, [4, 6, 4], 3, weight_init=0.3)
2  nn_34643_mse_iterasi = []
3  nn_34643_accuracy_iterasi = []
4  nn_34643.Backpropagation(X, y, learning_rate=0.1)
5  for i in range (1, 10000):
6    old_mse=nn_34643.mse[-1]
7    nn_34643_mse_iterasi.append(old_mse)
8    nn_34643_accuracy_iterasi.append(nn_344443.accuracy[-1])
9    nn_34643.Backpropagation(X, y, learning_rate=0.1)
10   nn_34643.ForwardPropagation(X)
11   new_mse = nn_34643.mse[-1]
12   if new_mse == old_mse :
13     print(f'error epoch ke-{i} :',np.sum((y-nn_34643.Y_out)
    **2)/16)
14     break
```

```
15        if i % 100 == 0 :
16          print(f'error epoch ke-{i} :',np.sum((y-nn_34643.Y_out)
    **2)/16)
17        if np.sum(np.abs(y-nn_34643.F)) == 0:
18          nn_34643_mse_iterasi.append(new_mse)
19          nn_34643_accuracy_iterasi.append(nn_34643.accuracy[-1])
20          print(f'error epoch ke-{i} :',np.sum((y-nn_34643.Y_out)
    **2)/16)
21          break
```

Pada *Source Code* diatas menunjukkan bagaimana untuk mencari jumlah bobot yang dapat digunakan. Untuk jumlah bobot sebesar 72 maka arsitektur yang akan dibuat adalah 3-4-6-4-3, 3-6-6-3, dan 3-12-3. Untuk jumlah bobot sebesar 105 maka arsitektur yang akan dibuat adalah 3-5-5-5-5-3 dan 3-7-4-8-3. Untuk jumlah bobot sebesar 144 maka arsitektur yang akan dibuat adalah 3-6-6-6-6-3, 3-3-21-3-3, 3-6-14-3, dan 3-24-3, Untuk jumlah bobot sebesar 72 maka arsitektur yang akan dibuat adalah 3-7-7-7-7-3, 3-10-6-11-3, dan 3-6-19-3. Dengan Kombinasi tersebut maka akan dihasilkan jumlah bobot yang sama.

3.3.7 Pengambilan data

Setelah program sudah diyakini benar maka langkah selanjutnya adalah mengambil data yang ingin dianalisis. Dalam penelitian ini data yang diambil adalah jumlah *hidden layer*, akurasi, dan waktu komputasi.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Optimal untuk Parameter

Berikut ini adalah yang perlu diperhatikan untuk mengisi bab hasil dan pembahasan:

1. Setiap rumusan masalah boleh memiliki lebih dari 1 tujuan.
2. Setiap subbab harus spesifik menjawab setiap tujuan yang dituliskan.
3. Setiap rumusan masalah boleh dijawab dengan 1 subbab atau lebih.

Berikut ini adalah contoh sub bab untuk menjelaskan tujuan penelitian.

4.2 Pembahasan Tujuan 1 dengan Hasil Penelitian 1 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas)

Sub bab pertama adalah membahas tujuan penelitian pertama dengan hasil penelitian ke-1. Dapat ditambahkan beberapa sub bab jika diperlukan.

4.3 Pembahasan Tujuan 1 dengan Hasil Penelitian 2 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas)

Sub bab kedua adalah membahas tujuan penelitian pertama dengan hasil penelitian ke-2. Sub bab ini merupakan contoh tambahan sub bab pertama.

4.4 Pembahasan Tujuan 2 dengan Hasil Penelitian 3 (Ubah Judul Sesuai dengan Hal yang Hendak dibahas)

Sub bab ketiga adalah membahas tujuan penelitian kedua. Dapat ditambahkan beberapa sub bab jika diperlukan.

4.5 Perbandingan Hasil Penelitian dengan Hasil Terdahulu

Pembahasan penutup dapat menjelaskan mengenai kelebihan hasil pengembangan / penelitian dan kekurangan dibandingkan dengan skripsi atau penelitian terdahulu atau perbandingan terhadap produk lain yang ada di pasaran. Penulis dapat menggunakan tabel untuk membandingkan secara gamblang dan menjelaskannya.

BAB V
TAMBAHAN (OPSIONAL)

Anda boleh menambahkan Bab jika diperlukan. Jumlah Bab tidak harus sesuai dengan *template*.

Bab tambahan ini diperlukan jika hasil penelitian untuk menjawab tujuan cukup panjang atau terdiri dari banyak sub bab. Mahasiswa boleh menjawab 1 tujuan penelitian dengan 1 bab.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Kesimpulan dapat diawali dengan apa yang dilakukan dengan tugas akhir ini lalu dilanjutkan dengan poin-poin yang menjawab tujuan penelitian, apakah tujuan sudah tercapai atau belum, tentunya berdasarkan data ataupun hasil dari Bab pembahasan sebelumnya. Dalam beberapa hal, kesimpulan dapat juga berisi tentang temuan/*findings* yang Anda dapatkan setelah melakukan pengamatan dan atau analisis terhadap hasil penelitian.

Kesimpulan menjawab seberapa jauh rumusan masalah tercapai berdasarkan hasil penelitian. Semua rumusan masalah harus disimpulkan berdasarkan data penelitian.

6.2 Saran

Saran berisi hal-hal yang bisa dilanjutkan dari penelitian atau skripsi ini, yang belum dilakukan karena batasan permasalahan. Saran bukan berisi saran kepada sistem atau pengguna, tetapi saran diberikan kepada aspek penelitian yang dapat dikembangkan dan ditambahkan di penelitian atau skripsi selanjutnya.

Catatan: Mahasiswa perlu melihat sinkronisasi antara rumusan masalah, tujuan, metode, hasil penelitian, dan kesimpulan.

DAFTAR PUSTAKA

[1] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[2] E. Rich and K. Knight, "Artificial intelligence," 1991. [Online]. Available: https://www.scirp.org/reference/referencespapers?referenceid=2414899

[3] R. Waters and K. Shubber, "Google invests $300mn in artificial intelligence start-up anthropic," Feb. 2023. [Online]. Available: https://www.ft.com/content/583ead66-467c-4bd5-84d0-ed5df7b5bf9c

[4] J. Islam and Y. Zhang, "Early diagnosis of alzheimer's disease: A neuroimaging study with deep learning architectures," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1962–19 622.

[5] J. D. Kelleher, *Deep Learning*, ser. MIT Press Essential Knowledge series. London, England: MIT Press, Sep. 2019.

[6] M. Adriani, A. Purwarianti *et al.*, "Strategi nasional kecerdasan artifisial 2020-2045," Desember 2020. [Online]. Available: https://ai-innovation.id/images/gallery/ebook/stranas-ka.pdf

[7] J. Echanobe, I. del Campo, and M. V. Martínez, "Design and optimization of a neural network-based driver recognition system by means of a multiobjective genetic algorithm," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 3745–3750.

[8] A. Surkan and J. Singleton, "Neural networks for bond rating improved by multiple hidden layers," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 157–162 vol.2.

[9] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 136–141, 1993.

[10] G. Panchal, A. Ganatra, Y. P. Kosta, and D. Panchal, "Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers," *International Journal of Computer Theory and Engineering*, p. 332–337, 2011. [Online]. Available: http://dx.doi.org/10.7763/IJCTE.2011.V3.328

[11] A. J. Thomas, M. Petridis, S. D. Walters, S. M. Gheytassi, and R. E. Morgan, *Two Hidden Layers are Usually Better than One*. Springer International Publishing, 2017. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-65172-9_24

[12] M. Uzair and N. Jamil, "Effects of hidden layers on the efficiency of neural networks," in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 2020, pp. 1–6.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. MIT Press Essential Knowledge series. London, England: MIT Press, August 2006.

[14] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.

[15] A. Publishing, *Python Machine Learning for Beginners*, 10 2020.

[16] G. Bonaccorso, *Mastering Machine Learning Algorithms*, 2nd ed. Birmingham, England: Packt Publishing, Jan. 2020.

[17] L. D. Knowings, *Building Neural Networks from Scratch with Python*, 2 2024.

[18] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 7 2022.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 10 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[20] G. F. Luger, *Artificial intelligence*, 6th ed. Upper Saddle River, NJ: Pearson, Feb. 2008.

[21] M. Negnevitsky, *Artificial Intelligence A Guide to Intelligent Systems*. Pearson Education, 2005.

[22] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, 2003.

[23] H. Nelson, *Essential math for AI*. O'Reilly Media, 1 2023.

[24] J. Langr and V. Bok, *GANs in action*. New York, NY: Manning Publications, Nov. 2019.

[25] L. E. Nugroho, "E-book as a platform for exploratory learning interactions," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 11, no. 01, pp. 62–65, 2016. [Online]. Available: http://www.online-journals.org/index.php/i-jet/article/view/5011

[26] P. I. Santosa, "User?s preference of web page length," *International Journal of Research and Reviews in Computer Science*, pp. 180–185, 2011.

[27] N. A. Setiawan, "Fuzzy decision support system for coronary artery disease diagnosis based on rough set theory," *International Journal of Rough Sets and Data Analysis (IJRSDA)*, vol. 1, no. 1, pp. 65–80, 2014.

[28] C. P. Wibowo, P. Thumwarin, and T. Matsuura, "On-line signature verification based on forward and backward variances of signature," in *Information and Communication Technology, Electronic and Electrical Engineering (JICTEE), 2014 4th Joint International Conference on*. IEEE, 2014, pp. 1–5.

[29] D. A. Marenda, A. Nasikun, and C. P. Wibowo, "Digitory, a smart way of learning islamic history in digital era," *arXiv preprint arXiv:1607.07790*, 2016.

[30] S. Wibirama, S. Tungjitkusolmun, and C. Pintavirooj, "Dual-camera acquisition for accurate measurement of three-dimensional eye movements," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 8, no. 3, pp. 238–246, 2013.

[31] C. P. Wibowo, "Clustering seasonal performances of soccer teams based on situational score line," *Communications in Science and Technology*, vol. 1, no. 1, 2016.

Catatan: Daftar pustaka adalah apa yang dirujuk atau disitasi, bukan apa yang telah dibaca, jika tidak ada dalam sitasi maka tidak perlu dituliskan dalam daftar pustaka.

LAMPIRAN

L.1 Isi Lampiran

Lampiran bersifat opsional bergantung hasil kesepakatan dengan pembimbing dapat berupa:

1. Bukti pelaksanaan Kuesioner seperti pertanyaan kuesioner, resume jawaban responden, dan dokumentasi kuesioner.
2. Spesifikasi Aplikasi atau Sistem yang dikembangkan meliputi spesifikasi teknis aplikasi, tautan unduh aplikasi, manual penggunaan aplikasi, hingga screenshot aplikasi.
3. Cuplikan kode yang sekiranya penting dan ditambahkan.
4. Tabel yang terlalu panjang yang masih diperlukan tetapi tidak memungkinkan untuk ditayangkan di bagian utama skripsi.
5. Gambar-gambar pendukung yang tidak terlalu penting untuk ditampilkan di bagian utama. Akan tetapi, mendukung argumentasi/pengamatan/analisis.
6. Penurunan rumus-rumus atau pembuktian suatu teorema yang terlalu panjang dan terlalu teknis sehingga Anda berasumsi bahwa pembaca biasa tidak akan menelaah lebih lanjut. Hal ini digunakan untuk memberikan kesempatan bagi pembaca tingkat lanjut untuk melihat proses penurunan rumus-rumus ini.

LAMPIRAN

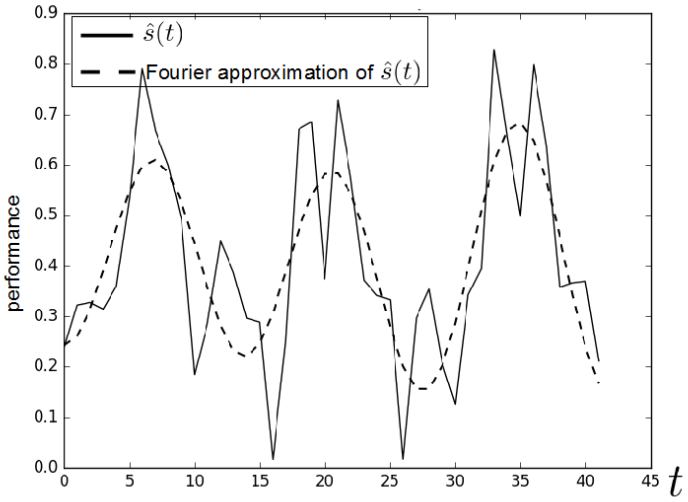
L.2 Panduan Latex

L.2.1 Syntax Dasar

L.2.1.1 Penggunaan Sitasi

Contoh penggunaan sitasi [25, 26] [27] [28] [29] [30, 31]

L.2.1.2 Penulisan Gambar



Gambar L.1. Contoh gambar.

Contoh gambar terlihat pada Gambar L.1. Gambar diambil dari [31].

L.2.1.3 Penulisan Tabel

Tabel L.1. Tabel ini

ID	Tinggi Badan (cm)	Berat Badan (kg)
A23	173	62
A25	185	78
A10	162	70

Contoh penulisan tabel bisa dilihat pada Tabel L.1.

L.2.1.4 Penulisan formula

Contoh penulisan formula

$$L_{\psi_z} = \{t_i \mid v_z(t_i) \leq \psi_z\} \tag{1}$$

Contoh penulisan secara *inline*: $PV = nRT$. Untuk kasus-kasus tertentu, kita membutuhkan perintah "mathit" dalam penulisan formula untuk menghindari adanya jeda saat penulisan formula.

Contoh formula **tanpa** menggunakan "mathit": $PVA = RTD$

Contoh formula **dengan** menggunakan "mathit": $PVA = RTD$

L.2.1.5 Contoh list

Berikut contoh penggunaan list

1. First item
2. Second item
3. Third item

L.2.2 Blok Beda Halaman

L.2.2.1 Membuat algoritma terpisah

Untuk membuat algoritma terpisah seperti pada contoh berikut, kita dapat memanfaatkan perintah *algstore* dan *algrestore* yang terdapat pada paket *algcompatible*. Pada dasarnya, kita membuat dua blok algoritma dimana blok pertama kita simpan menggunakan *algstore* dan kemudian di-restore menggunakan *algrestore* pada algoritma kedua. Perintah tersebut dimaksudkan agar terdapat kesinamungan antara kedua blok yang sejatinya adalah satu blok.

Algorithm 1 Contoh algorima

```
1: procedure CREATESET(v)
2:     Create new set containing v
3: end procedure
```

Pada blok algoritma kedua, tidak perlu ditambahkan caption dan label, karena sudah menjadi satu bagian dalam blok pertama. Pembagian algoritma menjadi dua bagian ini berguna jika kita ingin menjelaskan bagian-bagian dari sebuah algoritma, maupun untuk memisah algoritma panjang dalam beberapa halaman.

```
4: procedure CONCATSET(v)
5:     Create new set containing v
6: end procedure
```

L.2.2.2 Membuat tabel terpisah

Untuk membuat tabel panjang yang melebihi satu halaman, kita dapat mengganti kombinasi *table* + *tabular* menjadi *longtable* dengan contoh sebagai berikut.

Tabel L.2. Contoh tabel panjang

header 1	header 2
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar
foo	bar

L.2.2.3 Menulis formula terpisah halaman

Terkadang kita butuh untuk menuliskan rangkaian formula dalam jumlah besar sehingga melewati batas satu halaman. Solusi yang digunakan bisa saja dengan memindahkan satu blok formula tersebut pada halaman yang baru atau memisah rangkaian formula menjadi dua bagian untuk masing-masing halaman. Cara yang pertama mungkin akan menghasilkan alur yang berbeda karena ruang kosong pada halaman pertama akan diisi oleh teks selanjutnya. Sehingga di sini kita dapat memanfaatkan *align* yang sudah diatur dengan mode *allowdisplaybreaks*. Penggunakan *align* ini memungkinkan satu rangkaian formula terpisah berbeda halaman.

Contoh sederhana dapat digambarkan sebagai berikut.

$$\begin{aligned} x &= y^2 \\ x &= y^3 \\ a + b &= c \\ x &= y - 2 \\ a + b &= d + e \\ x^2 + 3 &= y \\ a(x) &= 2x \end{aligned} \tag{2}$$

$$
\begin{aligned}
b_i &= 5x \\
10x^2 &= 9x \\
2x^2 + 3x + 2 &= 0 \\
5x - 2 &= 0 \\
d &= \log x \\
y &= \sin x
\end{aligned}
$$

LAMPIRAN

L.3 Format Penulisan Referensi

Penulisan referensi mengikuti aturan standar yang sudah ditentukan. Untuk internasionalisasi DTETI, maka penulisan referensi akan mengikuti standar yang ditetapkan oleh IEEE (*International Electronics and Electrical Engineers*). Aturan penulisan ini bisa diunduh di http://www.ieee.org/documents/ieeecitationref.pdf. Gunakan Mendeley sebagai *reference manager* dan *export* data ke format Bibtex untuk digunakan di Latex.

Berikut ini adalah sampel penulisan dalam format IEEE:

L.3.1 Book

Basic Format:

[1] J. K. Author, "Title of chapter in the book," in Title of His Published Book, xth ed. City of Publisher, Country: Abbrev. of Publisher, year, ch. x, sec. x, pp. xxx–xxx.

Examples:

[1] B. Klaus and P. Horn, Robot Vision. Cambridge, MA: MIT Press, 1986.

[2] L. Stein, "Random patterns," in Computers and You, J. S. Brake, Ed. New York: Wiley, 1994, pp. 55-70.

[3] R. L. Myer, "Parametric oscillators and nonlinear materials," in Nonlinear Optics, vol. 4, P. G. Harper and B. S. Wherret, Eds. San Francisco, CA: Academic, 1977, pp. 47-160.

[4] M. Abramowitz and I. A. Stegun, Eds., Handbook of Mathematical Functions (Applied Mathematics Series 55). Washington, DC: NBS, 1964, pp. 32-33.

[5] E. F. Moore, "Gedanken-experiments on sequential machines," in Automata Studies (Ann. of Mathematical Studies, no. 1), C. E. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton Univ. Press, 1965, pp. 129-153.

[6] Westinghouse Electric Corporation (Staff of Technology and Science, Aerospace Div.), Integrated Electronic Systems. Englewood Cliffs, NJ: Prentice-Hall, 1970.

[7] M. Gorkii, "Optimal design," Dokl. Akad. Nauk SSSR, vol. 12, pp. 111-122, 1961 (Transl.: in L. Pontryagin, Ed., The Mathematical Theory of Optimal Processes. New York: Interscience, 1962, ch. 2, sec. 3, pp. 127-135).

[8] G. O. Young, "Synthetic structure of industrial plastics," in Plastics, vol. 3,

Polymers of Hexadromicon, J. Peters, Ed., 2nd ed. New York: McGraw-Hill, 1964, pp. 15-64.

L.3.2 Handbook

Basic Format:

[1] Name of Manual/Handbook, x ed., Abbrev. Name of Co., City of Co., Abbrev. State, year, pp. xx-xx.

Examples:

[1] Transmission Systems for Communications, 3rd ed., Western Electric Co., Winston Salem, NC, 1985, pp. 44-60.

[2] Motorola Semiconductor Data Manual, Motorola Semiconductor Products Inc., Phoenix, AZ, 1989.

[3] RCA Receiving Tube Manual, Radio Corp. of America, Electronic Components and Devices, Harrison, NJ, Tech. Ser. RC-23, 1992.

Conference/Prosiding

Basic Format:

[1] J. K. Author, "Title of paper," in Unabbreviated Name of Conf., City of Conf., Abbrev. State (if given), year, pp.xxx-xxx.

Examples:

[1] J. K. Author [two authors: J. K. Author and A. N. Writer] [three or more authors: J. K. Author et al.], "Title of Article," in [Title of Conf. Record as], [copyright year] © [IEEE or applicable copyright holder of the Conference Record]. doi: [DOI number]

Sumber Online/Internet

Basic Format:

[1] J. K. Author. (year, month day). Title (edition) [Type of medium]. Available: http://www.(URL)

Examples:

[1] J. Jones. (1991, May 10). Networks (2nd ed.) [Online]. Available: http://www.atm.com

Skripsi, Tesis dan Disertasi

Basic Format:

[1] J. K. Author, "Title of thesis," M.S. thesis, Abbrev. Dept., Abbrev. Univ., City of Univ., Abbrev. State, year.

[2] J. K. Author, “Title of dissertation,” Ph.D. dissertation, Abbrev. Dept., Abbrev. Univ., City of Univ., Abbrev. State, year.

Examples:

[1] J. O. Williams, “Narrow-band analyzer,” Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993. [2] N. Kawasaki, “Parametric study of thermal and chemical nonequilibrium nozzle flow,” M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993

LAMPIRAN

L.4 Contoh Source Code

L.4.1 Sample algorithm

Algorithm 2 Kruskal's Algorithm

```
1:  procedure MAKESET(v)
2:      Create new set containing v
3:  end procedure
4:
5:  function FINDSET(v)
6:      return a set containing v
7:  end function
8:
9:  procedure UNION(u,v)
10:     Unites the set that contain u and v into a new set
11: end procedure
12:
13: function KRUSKAL(V, E, w)
14:     A ← {}
15:     for each vertex v in V do
16:         MakeSet(v)
17:     end for
18:     Arrange E in increasing costs, ordered by w
19:     for each (u,v) taken from the sorted list do
20:         if FindSet(u) ≠ FindSet(v) then
21:             A ← A ∪ {(u, v)}
22:             Union(u, v)
23:         end if
24:     end for
25:     return A
26: end function
```

L.4.2 Sample Python code

```
import numpy as np

def incmatrix(genl1,genl2):
  m = len(genl1)
  n = len(genl2)
  M = None #to become the incidence matrix
  VT = np.zeros((n*m,1), int)  #dummy variable

  #compute the bitwise xor matrix
  M1 = bitxormatrix(genl1)
  M2 = np.triu(bitxormatrix(genl2),1)

  for i in range(m-1):
    for j in range(i+1, m):
      [r,c] = np.where(M2 == M1[i,j])
      for k in range(len(r)):
        VT[(i)*n + r[k]] = 1;
        VT[(i)*n + c[k]] = 1;
        VT[(j)*n + r[k]] = 1;
        VT[(j)*n + c[k]] = 1;

  if M is None:
    M = np.copy(VT)
  else:
    M = np.concatenate((M, VT), 1)

  VT = np.zeros((n*m,1), int)

  return M
```

L.4.3 Sample Matlab code

```
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

  %convert elements into usigned integers
  A = uint8(A);
  B = uint8(B);

  m1 = length(A);
  m2 = length(B);
  X = uint8(zeros(m1, m2));
  for n1=1:m1
    for n2=1:m2
      X(n1, n2) = bitxor(A(n1), B(n2));
    end
  end
```