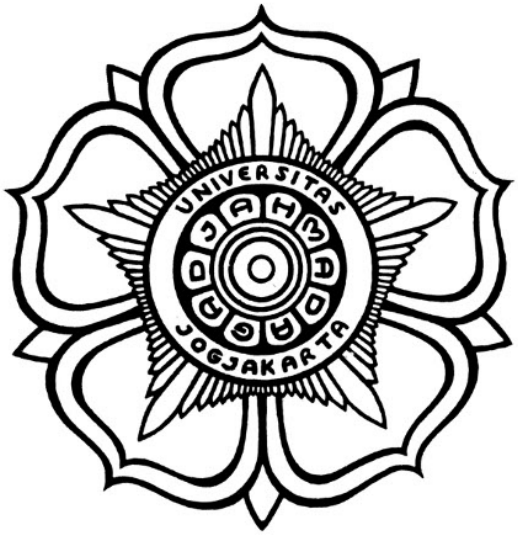
PENGARUH JUMLAH *HIDDEN LAYER* TERHADAP KINERJA AKURASI DAN WAKTU PEMBELAJARAN PADA JARINGAN SYARAF TIRUAN

SKRIPSI



Disusun oleh:

Muhamad Alif Ridha
19/439613/TK/48343

PROGRAM STUDI TEKNIK ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI
FAKULTAS TEKNIK UNIVERSITAS GADJAH MADA
YOGYAKARTA
2023

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

DAFTAR TABEL

DAFTAR GAMBAR

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

ABSTRACT

Keywords : Artificial Neural Network, Hidden Layer

BAB I
PENDAHULUAN

1.1 Latar Belakang

Belakangan ini penggunaan kecerdasan artifisial (KA) semakin berkembang dan menjadi perhatian global. Data dari laporan terbaru menunjukkan bahwa investasi global dalam teknologi KA terus meningkat, dengan perkiraan mencapai lebih dari 300 miliar dolar AS pada tahun 2023 [2]. Perusahaan-perusahaan besar seperti Google, Amazon, dan Facebook ikut mengintegrasikan KA ke dalam produk dan layanan mereka untuk meningkatkan efisiensi dan pengalaman pengguna [3]. Selain itu, KA juga telah merambah ke berbagai sektor, termasuk kesehatan, keuangan, otomotif, dan manufaktur. Misalnya, dalam industri kesehatan, KA digunakan untuk menganalisis data medis dan membantu dalam diagnosis penyakit [4]. Di sektor keuangan, KA digunakan untuk mengoptimalkan portofolio investasi dan mendeteksi keadaan obligasi dimasa depan.

Indonesia pun tidak ingin melewatkan kesempatan ini, melalui Badan Pengkajian dan Penerapan Teknologi (BPPT). Indonesia membentuk Strategi Nasional Kecerdasan Artifisial Indonesia 2020 - 2045 (STRANAS KA) [5]. Pembentukan STRANAS KA ini merupakan perwujudan Visi Indonesia Emas 2045. Untuk itu pengembangan kecerdasan artifisial akan dikaji dan diteliti untuk mengembangkan sektor-sektor yang sedang diprioritaskan. Saat ini metode kecerdasan artifisial yang sedang ramai digunakan dan dikembangkan adalah Jaringan Syaraf Tiruan (JST).

Jaringan syaraf tiruan menjadi sorotan utama dalam perkembangan kecerdasan artifisial. Model-model jaringan syaraf tiruan telah menjadi fondasi bagi berbagai aplikasi KA, seperti pengenalan wajah, penerjemah bahasa, dan *driver assistance system* [6]. Keberhasilan jaringan syaraf tiruan dalam menangani tugas-tugas kompleks ini dapat dilihat dari peningkatan akurasi dan kecepatan dalam pengolahan data. Hal ini didukung dengan semakin berkembangnya teknologi yang membuat daya komputasi menjadi lebih baik dan meningkatnya penggunaan *cloud* sehingga semakin memudahkan pengumpulan data dan mengurangi resiko kehilangan data.

Secara garis besar jaringan syaraf tiruan adalah model matematis yang terdiri dari serangkaian *neuron* yang terhubung satu sama lain. Setiap lapisan dalam jaringan memiliki peran khusus dalam pengolahan informasi. Salah satunya adalah *hidden layer* 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 langsung dapat terlihat didalam data. Pemilihan jumlah dan ukurannya memerlukan pengaturan yang baik untuk mendapatkan hasil yang diinginkan. Secara umum jika jumlah *hidden layer* terlalu sedikit, jaringan mungkin akan gagal mempelajari representasi yang cukup kompleks dari data, sementara jika terlalu banyak, dapat terjadi *overfitting*, yaitu jaringan menjadi terlalu cocok dengan data yang dilatih dan kinerjanya akan menurun jika dipertemukan 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

Penentuan jumlah dan ukuran *hidden layer* pada suatu jaringan syaraf tiruan masih memerlukan *tuning* agar mendapatkan hasil yang dinginkan. Namun efek pengurangan ataupun penambahan pada *hidden layer* masih ambigu dan diperlukan penelitian lebih lanjut. Oleh karena itu, maka disusun rumusan masalah seperti berikut:

1. Jumlah *hidden layer* pada jaringan syaraf tiruan tidak memiliki jumlah yang pasti.
2. Belum ada informasi mengenai efek apa yang terjadi pada pengurangan dan penambahan *hidden layer* pada jaringan syaraf tiruan.
3. Belum ada informasi mengenai perngaruh jumlah *hidden layer* terhadap waktu komputasi dan akurasi.

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk menganalisis pengaruh jumlah *hidden layer* dengan melakukan variasi terhadap jumlah *hidden layer* sehingga akan terlihat efek yang diberikan.

1.4 Batasan Penelitian

Batasan permasalahan pada skripsi ini adalah pada data, arsitektur yang digunakan, dan variabel pengaruh yang dianalisis. Data yang digunakan adalah data yang *linearly separable* artinya data dapat dipisahkan secara linier. Data memiliki 16 *instance*, 2 fitur,

dan tiga kelas. Kemudian batasan pada arsitektur yang digunakan. Arsitektur yang akan digunakan adalah arsitektur *Multilayer Perceptron*. Pada penelitian ini tidak akan dibahas arsitektur lain seperti *Convolutional Neural Network*, *Reccurent Neural Network*, dll. Kemudian pengaruh yang dianalisis difokuskan pada waktu komputasi dan akurasinya.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah untuk mengetahui pengaruh jumlah dan ukuran *hidden layer* terhadap akurasi dan waktu komputasi. Bagi akademisi dapat digunakan untuk memahami lebih lanjut mengenai jaringan syaraf tiruan dan membuka penelitian baru terkait jaringan syaraf tiruan. Bagi praktisi dapat digunakan sebagai pedoman untuk menentukan jumlah dan ukuran *hidden layer* yang akan digunakan.

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 dari jumlah lapisan tersembunyi sudah banyak dilakukan. Berikut beberapa artikel atau jurnal terkait.

Alvin J. Surkan [7], pada penelitiannya membuat sebuah JST untuk memprediksi rating obligasi dari *moody's or standard and poor's*, data tersebut berisi tujuh buah fitur dan 126 pola obligasi sebagai *instance*. Moody's memisahkan rating obligasi menjadi tujuh kelas yaitu [Aaa, Aal, Aa2, Aa3, A1 , A2, A3]. Dimana [Aaa] adalah kualitas tinggi, [Aa1, Aa2, Aa3] adalah kualitas menengah, dan [A1, A2, A3] adalah kualitas rendah. Neural network yang dibuat akan digunakan untuk membedakan kelas kualitas tinggi [Aaa] dan kelas kualitas rendah [A1, A2, A3]. JST dibuat dengan tiga model arsitektur. satu dari tiga model menggunakan satu lapisan tersembunyi dan dua jenis lainnya menggunakan dua lapisan tersembunyi dengan jumlah hidden neuron yang berbeda, dengan masing - masing arsitektur secara berurutan [7-14-2], [7-5-10-2], dan [7-10-5-2]. hasil menunjukan dua hidden layer lebih baik dari satu hidden layer dari akurasi yang diberikan.

Jacques de villiers [8], pada penelitiannya menggunakan data *distribution of distribution* untuk mengetahui pengaruh variasi dari data terhadap JST satu *hidden layer* dan JST dua *hidden layer*. Hasil penelitian yang didapatkan adalah tidak ada peningkatan performa yang signifikan antara keduanya, rata-rata satu *hidden layer* memiliki klasifikasi yang lebih baik daripada dua *hidden layer*, dan pelatihan dengan dua *hidden layer* lebih cepat jika neuron pada kedua *hidden layer* kurang lebih sama. Sedangkan peningkatan jumlah sampel dapat memberikan akurasi yang lebih baik.

Pada penelitian [9] menggunakan *Hopfield Neural Networks* dan *Akaike Criterion Information* untuk pemilihan jumlah hidden layer dan jumlah neuronnya. Pemilihan parameter diharapkan dapat menghindari *overfitting* dan *underfitting*. Hasil yang didapatkan berupa peningkatan *correct classification rate* dan penurunan galat seiring dengan penambahan jumlah lapisan tersembunyi bersamaan dengan peningkatan *training time*. Penulis menekankan jika penggunaan yang membutuhkan akurasi tinggi maka peningkatan jumlah lapisan tersembunyi dapat ditambahkan dan jika pada penggunaan yang memerlukan waktu yang cepat maka satu lapisan tersembunyi sudah mencukupi.

Pada penelitian [10] dibuat jaringan syaraf tiruan dengan *The Levenberg-Marquardt algorithm*. Penellitian menggunakan 10 set data yang berbeda. 9 dari 10 data set menunjukan hasil yang lebih baik dalam hal akurasi pada dua lapisan tersembunyi dibandingkan

satu lapisan tersembunyi, tetapi jumlah peningkatan akurasi berbeda - beda pada tiap data set dan sangat bergantung pada data set yang digunakan.

Pada artikel [11] berisi hasil dari artikel - artikel yang membahas pengaruh jumlah lapisan tersembunyi terhadap *time complexity* dan akurasi. Dari hasil penelitiannya sulit untuk menentukan jumlah lapisan tersembunyi yang tepat hanya dengan melihat lapisan input dan lapisan outpunya saja. Meskipun banyak metode yang dapat digunakan untuk mencari jumlah lapisan tersembunyi, namun aproksimasinya akan bergantung pada data yang digunakan. Jika akurasi yang tinggi sangat dibutuhkan maka dapat menggunakan jaringan syaraf tiruan yang *deep*. Namun jika *time complexity* yang diinginkan maka jaringan syaraf tiruan yang *deep* tidaklah cocok. Kemudian penambahan *neuron* atau lapisan tersembunyi yang berlebih dapat menyebabkan *overfitting*. Oleh karena itu disarankan untuk menganalisa sampel data yang digunakan agar dapat mengaproksimasi jumlah lapisan tersembunyi.

No	**Nama Penelitian**	**Metode**	**Hasil Pembahasan**
1	*Neural Networks For Bond Rating Improved By Multiple Hidden Layers*	*Multilayer Perceptron*	Dua *hidden layer* lebih baik dari satu *hidden layer* dari segi akurasi yang diberikan.
2	*Backpropagation Neural Nets with One and Two Hidden Layers*	*Multilayer Perceptron*	Tidak ada perbedaan signifikan antara satu *hidden layer* dan dua *hidden layer*
3	*Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers*	*Multilayer Perceptron*	Peningkatan jumlah lapisan tersembunyi atau jumlah neuron tersembunyi memberikan hasil yang lebih baik.
4	*Two Hidden Layers are Usually Better than One*	*Multilayer Perceptron*	Dua *hidden layer* lebih unggul dari satu *hidden layer*

5	*Effects of Hidden Layers on the Efficiency of Neural networks*	*Multilayer Perceptron*	Dengan jumlah lapisan tersembunyi yang sesuai dapat mengurangi waktu pelatihan dan tetap menghasilkan akurasi yang baik. Banyak metode yang dapat digunakan untuk mengaproksimasi jumlah lapisan tersembunyi, namun hal itu tetap bergantung dengan jenis data.

Tabel 2.1. Perbandingan Penelitian Terdahulu

Pada penelitian ini akan dibuat jaringan syaraf tiruan dengan jumlah lapisan tersembunyi dan *neuron* yang beragam, kemudian akan dibandingkan akurasi dan juga waktu komputasinya. Data yang digunakan adalah data tinggi dan lebar suatu barang. Dari data tersebut akan dibagi menjadi tiga kategori yaitu *wardrobe*, lemari, dan *buffet*.

2.2 Dasar Teori

2.2.1 Persamaan linear diskriminan

Diskriminan merupakan sebuah fungsi yang menerima vektor input $\vec{x}$ dan mengelompokkannya kedalam K kelas, Kelas tersebut disimbolkan sebagai C_k. Dimana $k = 1, 2, 3, ..., K$. Kelas-kelas ini akan dipisahkan oleh sebuah atau lebih *decision boundary* yang berupa *hyperplane* [12]. Tujuan dari klasifikasi adalah untuk menetapkan setiap masukan tepat ke kelas-nya masing-masing. Persamaan linear diskriminan dituliskan sebagai berikut:

$$y = \sum_{i=1}^{I} w_i^T x_i + w_0 \tag{2-1}$$

Atau dalam vektor dapat ditulis:

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

Dalam persamaan tersebut, $\vec{w}^T$ disebut sebagai vektor bobot, sementara w_0 disebut juga sebagai bias dan terkadang dilambangkan dengan variabel b. Bias negatif terkadang disebut juga sebagai *threshold* atau batasan. Sebuah vektor masukan $\vec{x}$ diklasifikasikan ke dalam kelas C_1 jika $y(\vec{x}) \geq 0$, dan ke dalam kelas C_2 jika sebaliknya.

Decision Boundary didefinisikan oleh persamaan $y(\vec{x}) = 0$, yang merepresentasikan sebuah *hyperplane* berdimensi (D - 1) pada masukan berdimensi D. Sebagai contoh jika masukan memiliki 2 *feature* artinya D = 2, maka data masukan dapat direpresentasikan oleh bidang kartesian dengan 2 sumbu. Sedangkan *decision boundary* akan berdimensi 1 dan berupa garis yang memisahkan daerah-daerah pada *input space*. *Decision boundary* ini berkaitan erat dengan $\vec{w}$ dan w_0. Misalkan dua titik $\vec{x}_A$ dan $\vec{x}_B$ yang keduanya berada pada *decision boundary* sehingga $y(\vec{x}_A) = y(\vec{x}_B) = 0$, maka berlaku :

$$\vec{w}^T(\vec{x}_A - \vec{x}_B) = 0 \tag{2-3}$$

Sehingga vektor $\vec{w}$ tegak lurus terhadap setiap vektor yang berada dalam *decision boundary* [12]. Maka vektor $\vec{w}$ akan menentukan orientasi dari *decision boundary*. Kemudian jika $\vec{x}$ berada pada *boundary decision*, maka $y(\vec{x}) = 0$. Jarak normal dari titik (0,0) ke *decision boundary* dapat ditulis :

$$\frac{\vec{w}^T\vec{x}}{|\vec{w}|} = -\frac{w_0}{|\vec{w}|} \tag{2-4}$$

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.

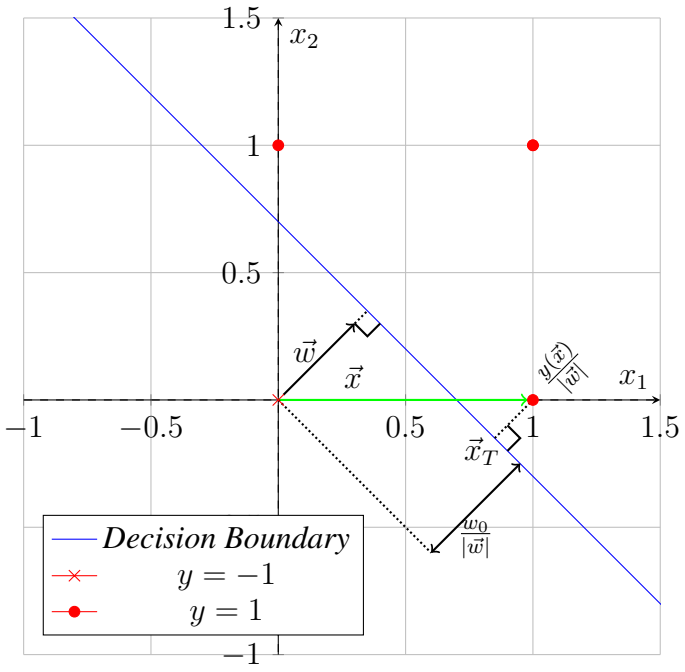
Tabel 2.2. Masukan dan Keluaran fungsi OR

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 $\vec{w}$ dan w_0. *Decision boundary* yang akan dibentuk akan memiliki dimensi D-1.

Dari gambar 2.1 dapat dilihat terdapat garis biru yang memisahkan kedua label. Terlihat vektor $\vec{w}$ tegak lurus dengan *decision boundary*, dan jarak antara *decision boundary* dan titik origin adalah $\frac{w_0}{|\vec{w}|}$. Kemudian untuk mengetahui jarak antara salah satu titik dengan *decision boundary* melalui persamaan 2-5.

$$r = \frac{y(\vec{x})}{|\vec{w}|} \tag{2-5}$$



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

Hal ini dapat dibuktikan dengan memisalkan $\vec{x}_T$ sebagai proyeksi dari titik $\vec{x}$ pada *decision boundary*. Sehingga vektor $\vec{x}$ dapat ditulis seperti pada persamaan 2-6.

$$\vec{x} = \vec{x}_T + r\frac{\vec{w}}{|\vec{w}|} \tag{2-6}$$

Dengan mengkalikan persamaan 2-6 dengan $\vec{w}^T$ dan menambahkan w_0 didapatkan persamaan 2-7.

$$\vec{w}^T\vec{x} + w_0 = \vec{w}^T\vec{x}_T + w_0 + r\frac{\vec{w}^T\vec{w}}{||\vec{w}||} \tag{2-7}$$

Karena $y(\vec{x}) = \vec{w}^T\vec{x} + w_0$, $\vec{w}^T\vec{x}_T + w_0 = 0$, dan $\frac{\vec{w}^T\vec{w}}{|\vec{w}|} = |\vec{w}|$. Maka didapatkan persamaan 2-5. Untuk memudahkan notasi penulisan terkadang diperkenalkan sebuah *dummy 'input'* berupa $x_0 = 1$ lalu didefinisikan $\hat{w} = (w_0, \vec{w})$ dan $\hat{x} = (x_0, \vec{x})$ agar dapat ditulis persamaan 2-8.

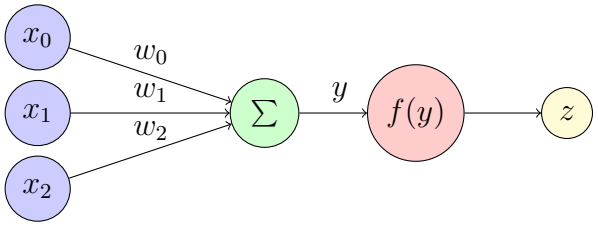
$$y(\vec{x}) = \hat{w}^T\hat{x} \tag{2-8}$$

Terlihat pada Gambar 2.1 *decision boundary* berhasil memisahkan kedua kelas. Untuk mencari *decision boundary* yang tepat dapat melalui beberapa metode seperti *Support vector machine*, eliminasi gauss, dan regresi linier. Sedangkan metode yang serupa dapat diterapkan dengan jaringan syaraf tiruan.

2.2.2 *Perceptron Algorithm*

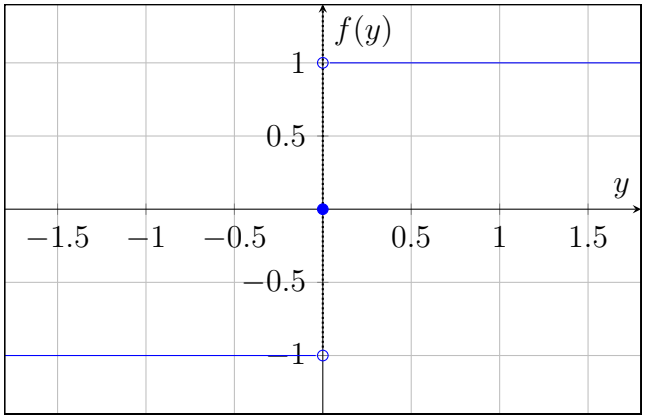
Perceptron algorithm adalah algoritma pembelajaran *machine learning* pada *artificial neural network* [3]. *Perceptron* menerima beberapa masukan dan menjumlahkan-

nya. Sebelum keluar dari neuron, isyarat yang dijumlahkan tersebut akan diolah terlebih dahulu dengan memasukkannya ke fungsi aktivasi. Untuk lebih jelas Perhatikan Gambar 2.2.



Gambar 2.2. Perceptron yang menerima tiga input dan mengeluarkan satu output

Gambar 2.2 menunjukan unit komputasi terkecil pada *artificial neural network*. Perceptron menerima tiga buah masukan yaitu x_0, x_1, dan x_2. Sedangkan garis-garis penghubung adalah bobot yang masing-masing terhubung dengan tiap-tiap input, 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 aktivasi $f(y)$. fungsi aktivasi ini berfungsi untuk menentukan seberapa aktif neuron tersebut. fungsi aktivasi juga memungkinkan jaringan syaraf tiruan dapat mengenali pola masukan dan keluaran yang *non-linear*. Terdapat berbagai macam fungsi aktivasi salah satunya adalah *step function*.



Gambar 2.3. 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 2.3. *Step function* dapat ditulis seperti pada Persamaan 2-9.

$$f(y) = \begin{cases} -1 & \text{jika } y < 0 \\ 1 & \text{jika } y \geq 0 \end{cases} \tag{2-9}$$

Pembelajaran *perceptron algorithm* menggunakan *perceptron criterion*. *Perceptron cri-*

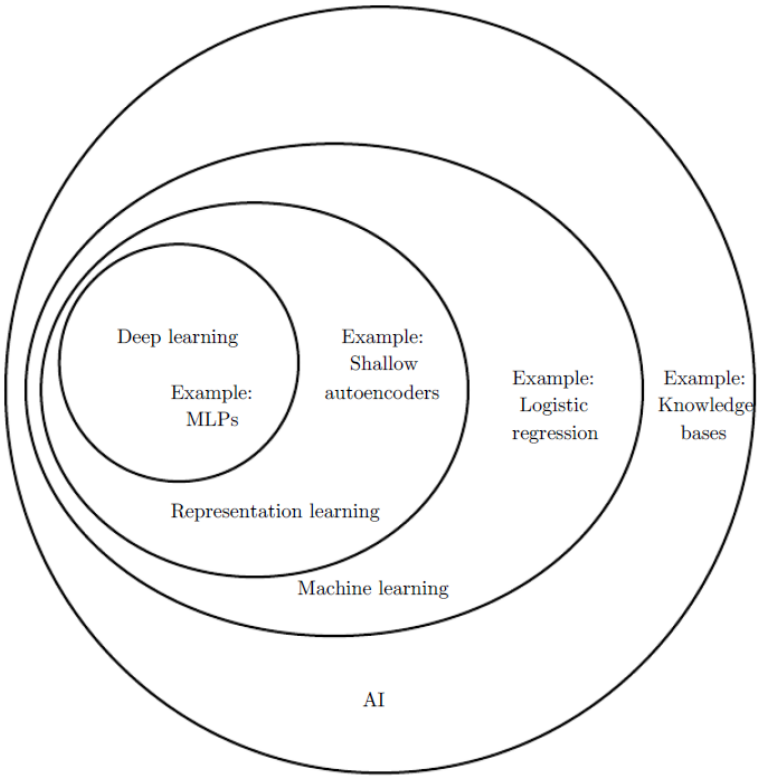
terion berupaya menemukan hyperplane yang memisahkan dua kelas titik data. yaitu dengan menggunakan persamaan:

$$E_p(w) = -\sum_{n \in M} w^T x_n y_n \tag{2-10}$$

Persamaan 2-10 berupaya menggunakan *misclassified pattern* untuk melakukan pembelajaran pada *perceptron*, dimana M adalah jumlah *misclassified pattern*.

2.2.3 *Artificial Neural Network*

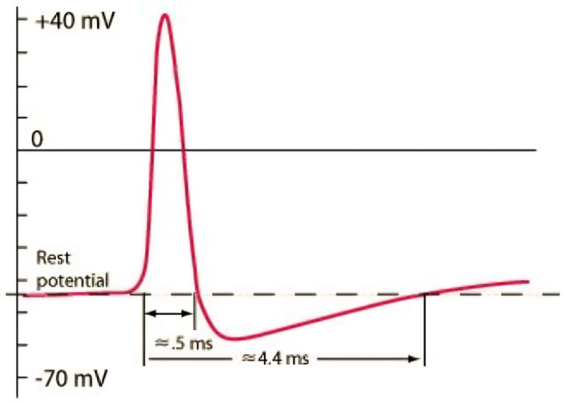
Dalam perkembangan *artificial neural network* kerap kali terdapat istilah-istilah yang terdengar serupa tetapi memiliki arti yang berbeda seperti *artificial neural network, machine learning*, dan *deep learning*. Ketiga istilah tersebut memiliki bidang ilmu yang beririsan. *Artificial Neural Network* adalah metode komputasi dari subbidang ilmu *machine learning*, *machine learning* adalah subbidang dari *Artificial Intelligence*, dan *deep learning* merupakan bagian dari *artificial neural network*. Untuk memahami hubungan antara bidang ilmu, dapat dilihat pada Gambar 2.4.



Gambar 2.4. Diagram Venn ini menggambarkan bahwa *deep learning* adalah bagian dari *representation learning*, yang merupakan bagian dari *machine learning*, yang mana digunakan dalam berbagai pendekatan AI, meskipun tidak semuanya. Di dalam setiap bagian diagram, terdapat contohnya. [1]

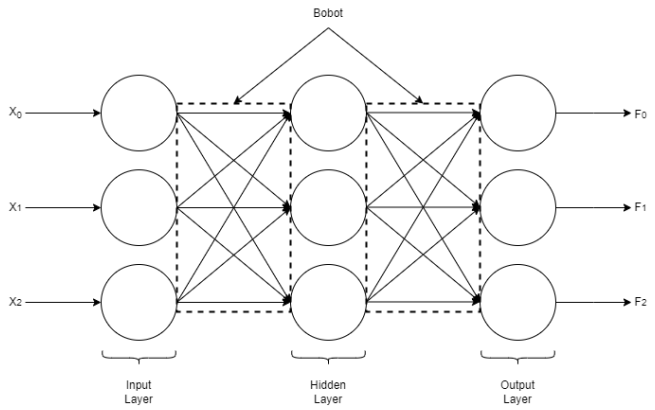
Artificial neural networks terinspirasi dari jaringan syaraf biologis pada otak ma-

nusia [3]. Pada otak manusia *neuron-neuron* berkomunikasi satu sama lain melalui sinyal listrik dan ikatan kimia. Sedangkan pada jaringan syaraf tiruan keduanya digantikan dengan model matematika. Sinyal ini diteruskan pada tiap *neuron* dengan sesuatu yang disebut sebagai *action potensial*. Seperti pada Gambar 2.5. *Neuron-neuron* yang saling terhubung ini memiliki *"strength"*.



Gambar 2.5. *Action Potential* pada jaringan syaraf

Jika pada jaringan syaraf manusia setiap *neuron* dapat terhubung dengan *neuron* yang lain, maka pada jaringan syaraf tiruan *neuron-neuron* tersusun teratur membentuk lapisan-lapisan [3]. Tiap-tiap lapisan ini terhubung satu sama lain oleh bobot. Jaringan syaraf tiruan akan menerima sinyal masukan melalui lapisan masukan setelah itu akan diteruskan ke lapisan tersembunyi dan diproses. Langkah selanjutnya tergantung dengan jumlah lapisan tersembunyi, sinyal keluaran dari lapisan tersembunyi dapat diteruskan ke lapisan tersembunyi lainnya atau diteruskan ke lapisan keluaran. Pada lapisan keluaran, sinyal akan diproses dan akan mengeluarkan sinyal keluaran. Kedalaman dari sebuah *neural networks* dapat dilihat dari jumlah lapisan tersembunyi yang dimilikinya. *Neural networks* yang memiliki dua atau lebih lapisan tersembunyi dapat dikatakan sebagai *deep learning* [3]. Arsitektur dari *neural network* digambarkan pada Gambar 2.6.



Gambar 2.6. Jaringan Syaraf Tiruan

Pada gambar 2.6 peran dari bobot untuk menunjukkan seberapa kuat hubungan satu *neuron* dengan *neuron* yang lainnya seperti halnya *"strength"* yang ada pada jaringan syaraf. Saat melakukan pelatihan, nilai dari bobot tersebut akan diperbaharui berdasarkan hubungan antara target yang ingin dicapai *expected value* dan sinyal keluarannya *predicted value*. Pelatihan pada *neural network* dapat diartikan sebagai pencarian nilai bobot terbaik [3]. Terdapat berbagai macam arsitektur pada model *artificial neural network*. *multilayer perceptron, reccurent neural networks, convolutional neural networks* dan lainnya. Setiap arsitektur memiliki karakter yang berbeda dan memiliki kecocokan yang berbeda terhadap tipe data tertentu. Contohnya *reccurent neural network* yang baik pada data kontinu seperti isyarat suara sehingga dapat digunakan pada pengenalan suara dan *convolutional neural network* yang baik pada data berbentuk grid sehingga cocok digunakan pada pengenalan gambar.

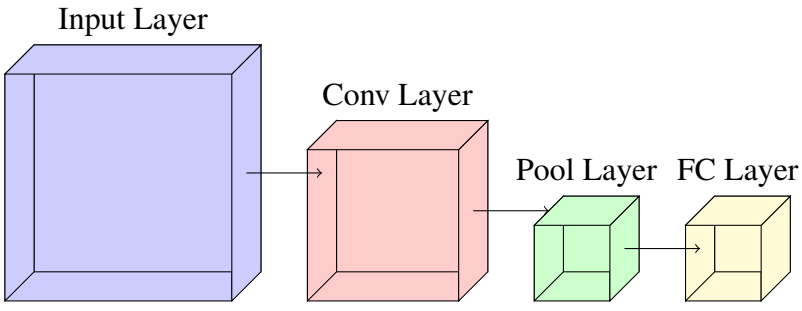
2.2.3.1 Arsitektur *Artificial Neural Network*

Arsitektur dari jaringan syaraf tiruan mempengaruhi pelatihan dan hasil yang keluar. Pengembangan jaringan syaraf tiruan meliputi pemilihan jenis arsitektur, pemilihan jumlah lapisan tersembunyi ,dan pemilihan jumlah *neuron* pada tiap lapisan [3]. *Neuron* pada lapisan masukan akan mengikuti jumlah *feature* yang diberikan oleh data ditambah 1 untuk merepresentasikan *bias*. *Neuron* pada lapisan keluaran akan mengikuti jumlah kelas K dari data. Sedangkan jenis arsitektur, jumlah *neuron*, dan jumlah lapisan pada lapisan tersembunyi disesuaikan dengan model yang dibutuhkan.

1. *Multilayer Perceptron*

   Gambar 2.6 adalah contoh jenis arsitektur *neural network*. Jenis arsitektur ini merupakan yang pertama kali muncul dibanding dengan jenis lainnya. Arsitektur memiliki alur informasi yang bergerak maju atau disebut juga *feedforward neural network* [12]. Dapat digunakan untuk jenis data yang berbentuk tabular dan penyelesaian masalah regresi atau klasifikasi sederhana.

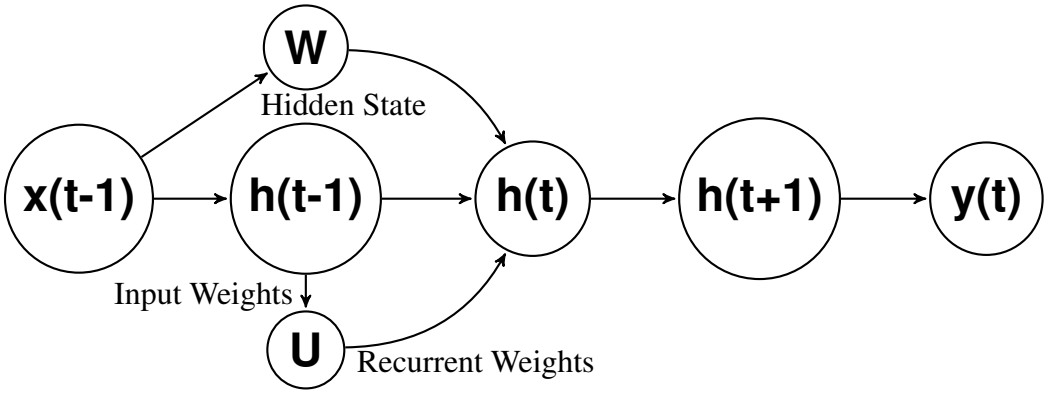
2. *Convolutional Neural Network*



Gambar 2.7. Arsitektur Convolutional Neural Network

Gambar 2.7 adalah arsitektur *Convolutional Neural Network* (CNN). CNN adalah jenis jaringan syaraf tiruan yang dirancang khusus untuk mengolah data yang memiliki struktur grid, seperti gambar [12]. CNN sangat populer dalam tugas-tugas dari bidang *computer vision* seperti pengenalan gambar, deteksi objek, dan segmentasi gambar. CNN terdiri dari beberapa jenis lapisan, masing-masing dengan peran yang spesifik dalam proses pengolahan dan analisis data. Conv Layer berfungsi untuk mengekstrak fitur dari input melalui operasi konvolusi. Pool Layer berfungsi mengurangi dimensi spasial dari fitur untuk mengurangi komputasi dan mencegah overfitting. FC Layer berguna untuk melakukan klasifikasi berdasarkan fitur yang diekstraksi oleh lapisan sebelumnya. CNN juga termasuk kedalam jenis *feedforward neural network.*

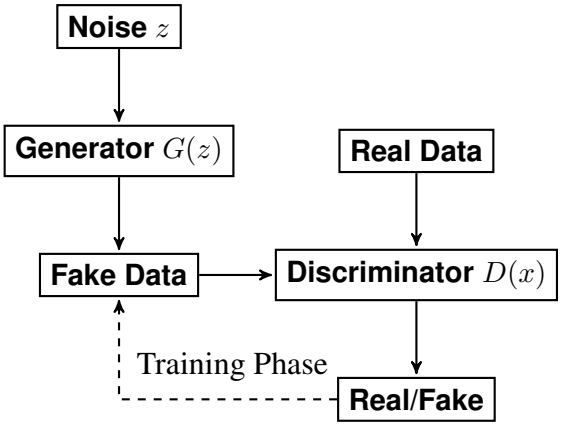
3. *Reccurent Neural Network*



Gambar 2.8. Arsitektur *Reccurent Neural Network*

Gambar 2.8 menunjukkan jenis arsitektur *Reccurent Neural Network* (RNN). RNN adalah jenis jaringan syaraf tiruan yang dirancang untuk mengolah informasi data sekuensial [1]. RNN memiliki kemampuan untuk mengingat informasi dari input sebelumnya dan menggunakan informasi ini untuk mempengaruhi output saat ini. Karena itu RNN sangat berguna untuk tugas-tugas yang melibatkan data sekuensial, seperti *natural language processing* (NLP), pengenalan suara, dan analisis data yang terikat waktu.

4. *Generative Adversial Network*



Gambar 2.9. Arsitektur *Generative Adversarial Network*

Generative Adversarial Network (GAN) adalah sebuah arsitektur jaringan saraf tiruan yang terdiri dari dua model utama: *Generator* dan *Discriminator* [13]. Ide utama dibalik GAN adalah untuk melatih *Generator* agar dapat menghasilkan data yang sangat mirip dengan data asli yang diberikan, sementara *Discriminator* dilatih untuk membedakan antara data palsu yang dihasilkan oleh *Generator* dan data asli. Jika diskriminator berhasil membedakan data asli dan data yang dibuat oleh generator maka generator dilatih kembali untuk menghasilkan data palsu yang lebih baik. Sedangkan jika diskriminator gagal membedakannya maka diskriminator dilatih kembali hingga dapat membedakan kedua jenis data. Dengan ini kedua model saling berkompetisi untuk menghasilkan model terbaik. GAN biasa digunakan untuk menghasilkan model *Generator* yag serupa dengan data aslinya. Seperti pada *Fake Image Generator*.

2.2.4 Normalisasi

Normalisasi adalah proses mengubah skala data sehingga memiliki karakteristik tertentu yang lebih diinginkan. Nilai suatu data sangat berpengaruh pada pelatihan *neural network*. Algoritma *gradient descent* pada *neural network* bekerja dengan memperbarui bobot berdasarkan *gradien* dari *loss function*. Jika fitur-fitur memiliki skala yang berbeda-beda, *gradien* yang dihitung bisa menjadi sangat besar atau sangat kecil, menyebabkan pembaruan bobot yang tidak seimbang dan mengakibatkan konvergensi yang lambat atau bahkan tidak stabil [13]. Untuk fitur dengan nilai yang sangat besar, gradien mungkin menjadi sangat besar, menyebabkan langkah-langkah pembaruan bobot yang terlalu besar. Sebaliknya, untuk fitur dengan nilai yang sangat kecil, gradien mungkin menjadi sangat kecil, menyebabkan langkah-langkah pembaruan yang terlalu kecil. Kedua situasi ini dapat membuat algoritma gradient descent tidak efisien atau bahkan gagal konvergen. Sebagai contoh diberikan data seperti pada Tabel 2.3. Misalkan dataset tersebut akan digunakan untuk memprediksi harga rumah.

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 data mentah ini langsung digunakan ke algoritma pembelajaran mesin (misalnya, regresi linear), fitur-fitur dengan skala yang lebih besar (seperti luas tanah) akan mendominasi proses pelatihan model. Misalnya, nilai "Luas Tanah" berkisar antara 300 hingga 600, sementara "Jumlah Kamar Tidur" hanya berkisar antara 3 hingga 5. Algoritma mungkin lebih condong ke fitur "Luas Tanah" karena perbedaan skala yang signifikan, meskipun "Jumlah Kamar Tidur" mungkin juga sangat penting dalam menentukan harga rumah.

Untuk mengatasi masalah ini, dapat dilakukan normalisasi pada setiap fitur sehingga semua fitur memiliki skala yang serupa. Salah satu metode normalisasi yang umum adalah Min-Max Scaling, yang mengubah skala setiap fitur ke dalam rentang 0 hingga 1. Persamaan *Min-Max Scaling* (2-11):

$$x_{norm} = \frac{x - x_{min}}{x - x_{max}} \tag{2-11}$$

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 [13]. *Z-score standardization* cenderung lebih tahan terhadap *outlier* dibandingkan dengan *min-max scaling*. Karena z-score didasarkan pada nilai rata-rata dan nilai standar deviasi, *outlier* tidak secara drastis mengubah skala data secara keseluruhan. Persamaan *Z-score standardization* (2-12):

$$Z = \frac{X - \mu}{\sigma} \tag{2-12}$$

Dengan X adalah nilai asli data, μ adalah nilai rata-rata dari data, dan σ adalah standar deviasi dari data. Secara garis besar dengan normalisasi, algoritma pembelajaran mesin memberikan bobot yang proporsional kepada setiap fitur berdasarkan informasi yang dikandungnya, bukan berdasarkan skala aslinya. Ini membantu meningkatkan kinerja model dan menghasilkan prediksi yang lebih akurat.

2.2.5 *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.

2.2.5.1 *Mean Square Error*

Mean Squared Error (MSE) adalah parameter yang digunakan untuk mengukur rata-rata kuadrat kesalahan antara nilai yang diprediksi oleh model dan nilai sebenarnya [1].

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 \tag{2-13}$$

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.

2.2.5.2 *Binary Cross-Entropy*

Binary Cross-Entropy (BCE) adalah *loss function* yang digunakan dalam permasalan klasifikasi dua kelas atau biner. BCE mengukur perbedaan antara distribusi probabilitas yang diprediksi oleh model dan distribusi probabilitas sebenarnya dari data [12].

$$L(y, \hat{y}) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right) \tag{2-14}$$

Dimana y adalah *expected value* (0 atau 1). $\hat{y}$ adalah probabilitas yang *predicted value* oleh model bahwa label adalah 1 (nilai antara 0 dan 1). Untuk sejumlah N sampel data, BCE dinyatakan sebagai rata-rata dari semua sampel:

$$L(Y, \hat{Y}) = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n \log(\hat{y}_n) + (1-y_n)\log(1-\hat{y}_n)\right) \tag{2-15}$$

2.2.5.3 *Categorical Cross-Entropy Loss*

Categorical Cross-Entropy (CCE) adalah *loss function* yang serupa dengan BCE namun digunakan pada masalah klasifikasi multi-kelas [1]. CCE mengukur perbedaan antara distribusi probabilitas yang diprediksi oleh model dan distribusi probabilitas sebe-

narnya dari data.

$$L(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{2-16}$$

Dimana K adalah jumlah kelas. y_k adalah *expected value* untuk kelas ke-k (dinyatakan sebagai one-hot encoding). $\hat{y}_k$ adalah probabilitas *predicted* oleh model untuk kelas ke-k (nilai antara 0 dan 1). Untuk sejumlah N sampel data, CCE dinyatakan sebagai rata-rata dari semua sampel:

$$L(Y, \hat{Y}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log(\hat{y}_{nk}) \tag{2-17}$$

2.2.5.4 *Huber Loss*

Huber Loss (HL) adalah *loss function* yang digunakan dalam permasalahan regresi. HL menggabungkan persamaan dari *Mean Squared Error* (MSE) dan *Mean Absolute Error* (MAE) untuk memberikan galat yang lebih *robust* terhadap outliers [14]. Untuk satu sampel data, HL didefinisikan sebagai:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{jika } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{jika } |y - \hat{y}| > \delta \end{cases} \tag{2-18}$$

Dimana y adalah *expected value*. $\hat{y}$ adalah *predicted value* oleh model. δ adalah ambang batas yang menentukan peralihan antara MSE dan MAE. Untuk sejumlah N sampel data, Huber Loss dinyatakan sebagai rata-rata dari semua sampel:

$$L_\delta(Y, \hat{Y}) = \frac{1}{N}\sum_{n=1}^{N} L_\delta(y_n, \hat{y}_n) \tag{2-19}$$

2.2.5.5 Kullback-Leibler Divergence (KL Divergence)

Kullback-Leibler Divergence (KL Divergence) adalah ukuran yang digunakan dalam teori informasi dan statistik untuk mengukur seberapa banyak satu distribusi probabilitas berbeda dari distribusi probabilitas lain [14]. Untuk dua distribusi probabilitas P dan Q, di mana P adalah distribusi *expected value* dan Q adalah distribusi dari *predicted value*, KL Divergence dari Q ke P didefinisikan sebagai:

$$D_{KL}(P \parallel Q) = \sum_{x} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{2-20}$$

Jika P dan Q adalah distribusi kontinu, maka rumusnya adalah:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx \tag{2-21}$$

KL Divergence tidak simetris, artinya $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$. KL Divergence selalu non-negatif, yaitu $D_{KL}(P \parallel Q) \geq 0$, dan nol hanya jika $P = Q$ secara identik. KL Divergence dapat dianggap sebagai jumlah informasi tambahan yang diperlukan untuk mengkodekan distribusi P menggunakan distribusi Q.

2.2.6 *Activasion Function*

Fungsi aktivasi adalah komponen penting dalam jaringan saraf tiruan yang menentukan keluaran dari suatu *neuron* atau node berdasarkan masukan yang diterima. Fungsi aktivasi memperkenalkan non-linearitas ke dalam model, yang memungkinkan jaringan saraf untuk mempelajari dan memodelkan data yang kompleks, seperti gambar, teks, dan audio. Tanpa fungsi aktivasi, jaringan saraf akan menjadi kombinasi linear dari input, yang membatasi kemampuannya untuk memecahkan masalah non-linear [12]. Fungsi *sigmoid* dan *step function* sering digunakan dalam jaringan saraf lapis satu dan pada output untuk tugas klasifikasi biner (0, 1).

2.2.6.1 *Step Function*

Step function adalah fungsi yang merubah nilai secara ke 0 dan 1 pada titik-titik tertentu, dan tetap konstan di antara titik-titik tersebut. *Step function* biasanya digunakan dalam permasalahan dimana variabel output hanya dapat mengambil nilai-nilai diskrit tertentu, yang membuatnya berbeda dari fungsi kontinu yang berubah secara halus.

$$f(x) = \begin{cases} 0 & \text{jika } x < 0 \\ 1 & \text{jika } x \geq 0 \end{cases} \tag{2-22}$$

Tetapi fungsi ini memiliki kekurangan, Karena $f(x)$ bersifat diskrit. Turunan dari *step function* tidak berguna pada titik mana pun untuk pelatihan. Tepat pada setiap x mendekati 0, turunan tidak terdefinisi dan turunan pada titik lainnya bernilai 0. Oleh karena itu, turunan dari *loss function* yang menggunakan *step function* tidak memberikan informasi tentang bagaimana memperbarui parameter model [1].

2.2.6.2 *Sigmoid Function*

Fungsi sigmoid adalah fungsi matematis yang memiliki bentuk kurva berbentuk "S" atau sigmoid. Fungsi ini sering digunakan dalam konteks jaringan saraf tiruan dan statistik untuk mengaktifkan neuron. Fungsi sigmoid mengubah input menjadi nilai antara 0 dan 1 [3], membuatnya sangat berguna dalam masalah klasifikasi biner. Fungsi sigmoid didefinisikan sebagai:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2-23}$$

Dimana e adalah basis dari logaritma natural (sekitar 2.71828), dan x adalah input fungsi tersebut. Output dari fungsi sigmoid selalu berada di antara 0 dan 1, yaitu $0 < \sigma(x) < 1$. Fungsi sigmoid memperkenalkan non-linearitas ke model, yang memungkinkan jaringan saraf tiruan untuk belajar dan mewakili hubungan yang kompleks dalam data. Saat x mendekati positif tak terhingga ($+\infty$), output dari fungsi sigmoid mendekati 1. Sebaliknya, saat x mendekati negatif tak terhingga ($-\infty$), output mendekati 0. Pada $x = 0$, nilai fungsi sigmoid adalah 0.5. Ini berarti input nol memberikan output tengah-tengah antara 0 dan 1. Turunan dari fungsi sigmoid penting dalam algoritma pembelajaran, seperti *backpropagation*. Turunan dari fungsi sigmoid adalah:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{2-24}$$

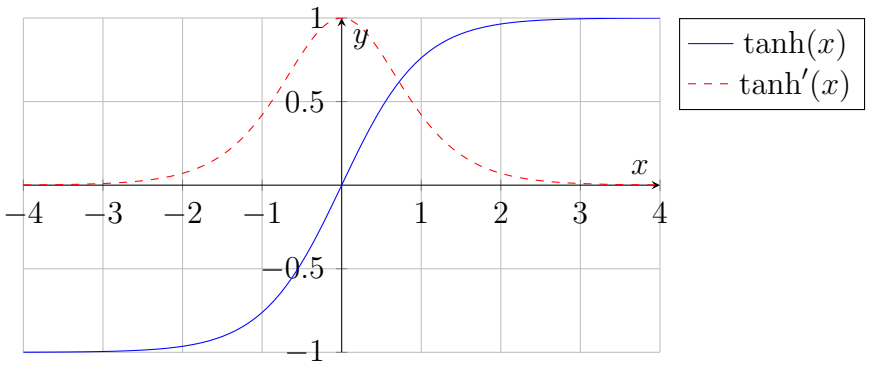
Fungsi sigmoid sering digunakan sebagai fungsi aktivasi dalam jaringan saraf tiruan, terutama di lapisan tersembunyi dan keluaran. Dalam permasalahan regresi, fungsi sigmoid digunakan untuk model probabilitas. Fungsi sigmoid memperkenalkan non-linearitas yang penting untuk menangkap hubungan kompleks dalam data. Outputnya berada dalam rentang terbatas [0,1], yang berguna untuk interpretasi sebagai probabilitas. Namun, dalam jaringan *deep learning*, gradien yang dihitung selama *backpropagation* dapat menjadi sangat kecil, menyebabkan pembelajaran menjadi sangat lambat atau bahkan terhenti. Selain itu, output yang selalu positif dapat menyebabkan ketidakseimbangan dalam pembaruan bobot selama pelatihan jaringan saraf tiruan. Fungsi sigmoid adalah komponen penting dalam banyak model pembelajaran mesin, dan pemahamannya sangat penting untuk bekerja dengan jaringan saraf tiruan dan algoritma terkait.

2.2.6.3 *Hyperbolic Tangent Function* (tanh)

Fungsi tangens hiperbolik, atau *hyperbolic tangent* (tanh), adalah salah satu fungsi aktivasi yang umum digunakan dalam jaringan saraf tiruan. Fungsi ini memetakan nilai input dari seluruh bilangan real ke rentang antara -1 dan 1 [3].

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2-25}$$

Dimana $\sinh(x)$ adalah fungsi sinus hiperbolik. $\cosh(x)$ adalah fungsi kosinus hiperbolik.



Gambar 2.10. *Graph* untuk fungsi tanh(x)

Sifat-sifat Fungsi tanh adalah Range $\tanh(x)$ menghasilkan nilai dalam rentang $[-1, 1]$. Fungsi ini adalah fungsi ganjil, yaitu $\tanh(-x) = -\tanh(x)$. Turunan dari fungsi tanh adalah:

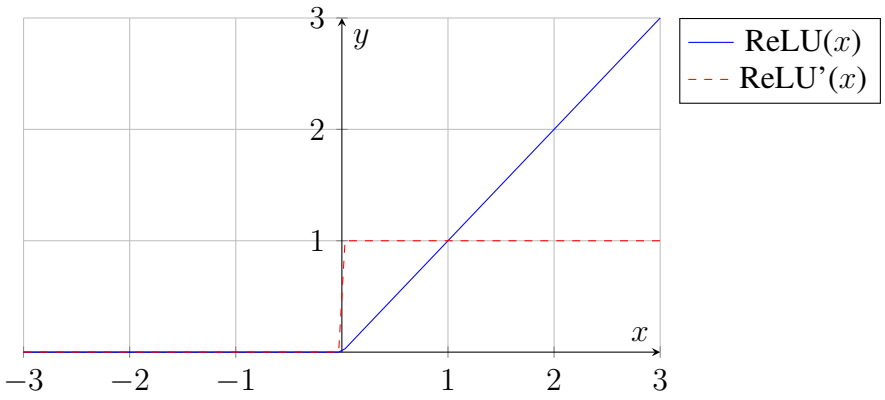
$$\tanh'(x) = 1 - \tanh^2(x) \tag{2-26}$$

2.2.6.4 *Rectified Linear Unit* (ReLU)

Fungsi ReLU adalah salah satu fungsi aktivasi yang paling populer dan banyak digunakan dalam jaringan saraf tiruan. Fungsi ReLU mengubah semua nilai negatif menjadi nol dan mempertahankan nilai positif [3]. Fungsi ini populer karena kecepatan konvergensinya yang tinggi dalam *deep learning*. Kelebihan utamanya dalam mengatasi masalah *vanishing gradient* dan membuatnya sangat berguna dalam *deep learning*. ReLU juga membantu dalam konvergensi yang lebih cepat saat pelatihan model, yang sangat penting dalam aplikasi-aplikasi pembelajaran mesin yang membutuhkan waktu pelatihan yang efisien. Fungsi ReLU didefinisikan sebagai:

$$\text{ReLU}(x) = \max(0, x) \tag{2-27}$$

Rentang keluaran ReLU adalah $[0, \infty)$. Fungsi ini linear ketika x lebih besar dari atau sama dengan 0. ReLU bisa menghasilkan keluaran nol untuk nilai input negatif. ReLU membantu mengatasi masalah *vanishing gradient*. Fungsi ReLU menghasilkan keluaran nol untuk input negatif, dapat menciptakan banyak kekosongan didalam jaringan. efeknya *neuron-neuron* bisa mati selama pelatihan dan hanya menghasilkan nol. Fungsi ReLU juga tidak memiliki turunan yang terdefinisi dititik nol.



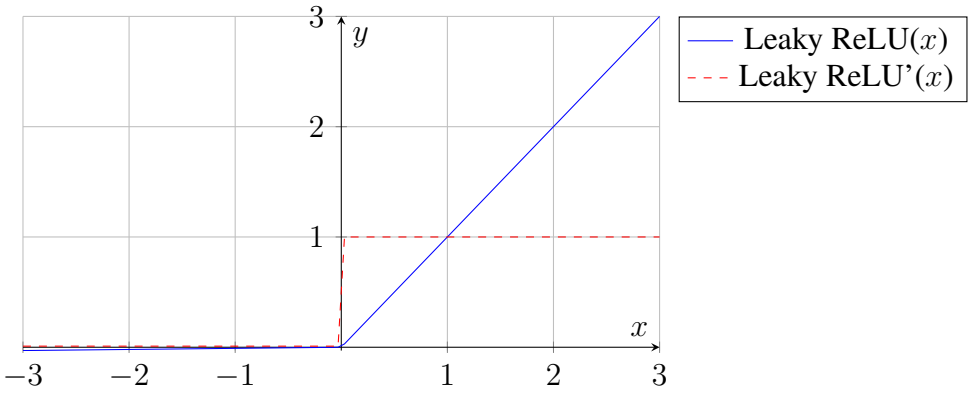
Gambar 2.11. *Graph* untuk fungsi ReLU

2.2.6.5 *Leaky ReLU*

Fungsi *Leaky ReLU* adalah variasi dari fungsi aktivasi ReLU yang mengatasi masalah "*dying ReLU*" dimana neuron menjadi tidak aktif dengan memberikan kemiringan kecil untuk nilai negatifnya [3]. Leaky ReLU sering digunakan sebagai fungsi aktivasi dalam lapisan tersembunyi dari jaringan saraf tiruan, terutama ketika ada masalah dengan neuron yang mati selama pelatihan. Dengan menjaga gradien untuk nilai negatif, *Leaky ReLU* membantu memastikan bahwa neuron tetap aktif, sehingga meningkatkan stabilitas dan efisiensi pelatihan model. Leaky ReLU didefinisikan sebagai:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{jika } x \geq 0 \\ \alpha x & \text{jika } x < 0 \end{cases} \tag{2-28}$$

Dimana α adalah parameter kecil yang menentukan kemiringan untuk nilai negatif. Biasanya, α diatur ke nilai kecil seperti 0.01. Dengan memberikan gradien kecil untuk nilai negatif, Leaky ReLU membantu menjaga neuron tetap aktif selama pelatihan. Seperti ReLU, fungsi Leaky ReLU juga sederhana untuk dihitung dan sangat efisien dari segi komputasi. Namun pemilihan nilai α bisa menjadi tantangan tersendiri.



Gambar 2.12. *Graph* untuk Leaky ReLU

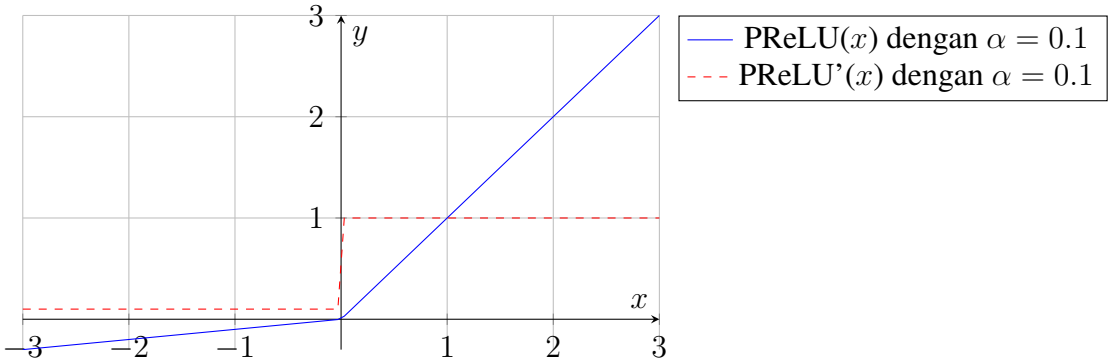
Rentang keluaran *Leaky ReLU* adalah $(-\infty, \infty)$. Fungsi ini linear untuk $x \geq 0$ dan memiliki kemiringan kecil (yang ditentukan oleh α) untuk $x < 0$. Fungsi *Leaky ReLU* memiliki gradien yang tidak nol untuk nilai negatif.

2.2.6.6 *Parametric ReLU* (PReLU)

Fungsi PReLU adalah variasi dari fungsi aktivasi ReLU yang memperkenalkan parameter α yang dapat dipelajari untuk nilai negatif [1]. Fungsi PReLU sering digunakan dalam *deep learning*, terutama ketika ada kebutuhan untuk meningkatkan fleksibilitas dan kinerja fungsi aktivasi. Dengan parameter yang dapat dipelajari, PReLU memberikan kemampuan pada model untuk menyesuaikan model lebih baik terhadap data, yang bisa menghasilkan kinerja yang lebih baik dalam berbagai tugas pembelajaran mesin. Fungsi PReLU didefinisikan sebagai:

$$\text{PReLU}(x) = \begin{cases} x & \text{jika } x \geq 0 \\ \alpha x & \text{jika } x < 0 \end{cases} \tag{2-29}$$

Dimana α adalah parameter yang dapat dipelajari. Dengan α sebagai parameter yang dapat dipelajari, fungsi PReLU dapat menyesuaikan dengan data selama pelatihan. PReLU membantu menjaga neuron tetap aktif dengan memiliki gradien yang tidak nol untuk nilai negatif. Nilai α dioptimalkan selama pelatihan, sehingga model dapat menyesuaikan fungsi aktivasi untuk kinerja terbaik. Namun memperkenalkan parameter yang dapat dipelajari akan menambah kompleksitas pelatihan. Potensi risiko *overfitting* dengan menambahkan lebih banyak parameter yang dapat dipelajari.



Gambar 2.13. *Graph* untuk Parametric ReLU

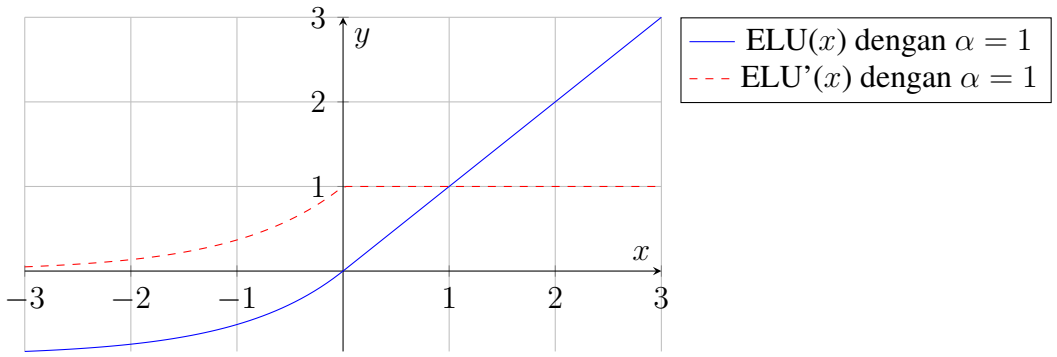
Rentang keluaran PReLU adalah $(-\infty, \infty)$. Fungsi ini linear untuk $x \geq 0$ dan linear dengan kemiringan α untuk $x < 0$. Fungsi PReLU memiliki gradien yang tidak nol untuk nilai negatif.

2.2.6.7 *Exponential Linear Unit* (ELU)

Fungsi ELU adalah fungsi aktivasi yang memberikan nilai negatif untuk input negatif dan memiliki turunan yang lebih halus, yang membantu dalam mempercepat pelatihan dan mengatasi masalah "*dying ReLU*". Fungsi ELU juga cukup sering digunakan dalam *deep learning*, terutama ketika ada kebutuhan untuk mengatasi masalah "*dying ReLU*" dan mempercepat konvergensi [14]. Dengan memiliki saturasi negatif dan gradien halus, ELU dapat membantu model untuk belajar lebih efisien dan mencapai kinerja yang lebih baik dalam berbagai tugas pembelajaran mesin. Fungsi ELU didefinisikan sebagai:

$$\text{ELU}(x) = \begin{cases} x & \text{jika } x \geq 0 \\ \alpha(e^x - 1) & \text{jika } x < 0 \end{cases} \tag{2-30}$$

Dimana α adalah parameter positif yang menentukan kemiringan untuk nilai negatif. ELU memberikan nilai negatif untuk input negatif, membantu menjaga neuron tetap aktif. ELU memiliki saturasi negatif yang membantu mempercepat konvergensi. Turunan dari ELU lebih halus dibandingkan dengan ReLU. Menghitung eksponensial untuk nilai negatif bisa lebih mahal secara komputasi. Pemilihan nilai α bisa mempengaruhi performa model.

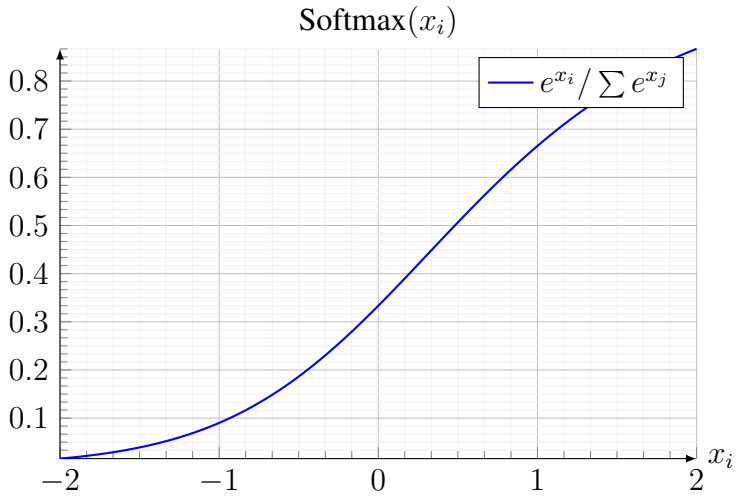


Gambar 2.14. *Graph* untuk ELU

Rentang keluaran ELU adalah $[-\alpha, \infty)$. ELU memiliki saturasi negatif untuk nilai x negatif yang besar. Fungsi ini linear untuk $x \geq 0$.

2.2.6.8 Softmax

Fungsi ini mengubah nilai menjadi distribusi probabilitas [12]. Fungsi ini biasanya digunakan di lapisan output untuk tugas klasifikasi.



Gambar 2.15. *Graph* untuk fungsi softmax

Fungsi ini memastikan bahwa output jaringan adalah distribusi probabilitas, yang sangat berguna untuk menentukan kelas yang paling mungkin dari sebuah input. Softmax juga memungkinkan penggunaan *loss function* seperti *categorical cross-entropy*, yang menghitung kesalahan antara distribusi prediksi dan distribusi target. Fungsi softmax untuk elemen i dari vektor input $\mathbf{z}$ didefinisikan sebagai:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2-31}$$

2.2.7 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* [12]. Proses ini dilakukan dengan mengikuti arah gradien dari *loss function*.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t) \tag{2-32}$$

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 terbesar 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 2-32, 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.

2.2.7.1 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 contoh data untuk menghitung gradien dan memperbarui parameter [1]. Kelebihannya adalah lebih cepat dan lebih efisien untuk dataset besar, namun gradien yang dihitung lebih bising 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.

2.2.7.2 Optimasi Gradient Descent

Untuk meningkatkan kinerja dan konvergensi gradient descent, beberapa teknik dan varian optimasi dapat digunakan:

1. *Momentum*

   Momentum adalah teknik yang digunakan dalam algoritma optimasi gradient descent untuk mempercepat konvergensi dan mengatasi masalah terjebak pada lokal minima. Prinsip dasar momentum adalah memperhitungkan gradien sebelumnya untuk menentukan arah dan jarak perpindahan parameter dalam proses optimasi [1]. Dengan momentum, pembaruan parameter dilakukan sebagai berikut:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta J(\theta_t) \tag{2-33}$$

$$\theta_{t+1} = \theta_t - \eta v_t \tag{2-34}$$

Dimana v_t adalah variabel momentum pada iterasi ke-t. Koefisien momentum β biasanya dipilih antara 0.5 dan 0.9. Bagian dari gradien sebelumnya yang dibawa ke iterasi berikutnya ditunjukkan oleh v_{t-1}. Semakin besar nilai β, semakin besar pengaruh gradien sebelumnya terhadap perbaruan parameter. Gradien dari iterasi saat ini diberikan oleh $(1-\beta)\nabla_\theta J(\theta_t)$. Dengan menggunakan momentum, pembaruan parameter menjadi lebih stabil dan cepat. Gradien sebelumnya membantu memperlancar jalur perpindahan parameter, yang memungkinkan algoritma optimasi mencapai konvergensi dengan lebih efisien.

2. *AdaGrad*

   Adagrad digunakan untuk merubah nilai *learning rate* menyesuaikan dengan gradien yang didapat. Adagrad sangat berguna dalam situasi dimana fitur jarang muncul atau terdapat perbedaan skala yang signifikan antara fitur-fitur dalam dataset. Algoritma ini secara otomatis menyesuaikan *learning rate* berdasarkan frekuensi pembaharuan parameter, sehingga fitur yang jarang muncul mendapatkan pembaruan yang lebih besar. Adagrad mengurangi kebutuhan untuk melakukan pencarian *learning rate* yang optimal [1]. Seiring waktu, laju pembelajaran bisa menjadi sangat kecil karena penjumlahan kuadrat gradien yang terus bertambah, sehingga pembaruan parameter menjadi sangat lambat. Adagrad kurang efektif jika terdapat perubahan skala yang sangat besar antara fitur. Dalam algoritma Adagrad, parameter diperbarui dengan cara sebagai berikut:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_{\theta_t} J(\theta_t) \tag{2-35}$$

$$G_{t,ii} = G_{t-1,ii} + (\nabla_{\theta_t} J(\theta_t))^2 \tag{2-36}$$

   Dimana θ_t adalah parameter pada iterasi ke-t. η adalah laju pembelajaran awal (learning rate). $\nabla_{\theta_t} J(\theta_t)$ adalah gradien dari fungsi biaya J terhadap parameter θ pada iterasi ke-t. G_t adalah matriks diagonal yang menyimpan jumlah kuadrat dari gradien untuk setiap parameter hingga iterasi ke-t. ϵ adalah nilai kecil (biasanya 10^{-8}) untuk mencegah pembagian dengan nol. Sedangkan $G_{t,ii}$ diperbarui dengan Persamaan 2-36. Dimana $G_{t,ii}$ menyimpan jumlah kuadrat gradien dari iterasi sebelumnya hingga iterasi ke-t untuk parameter θ_t.

3. *RMSProp*

   RMSprop ini adalah perbaikan dari Adagrad dan dirancang untuk mengatasi beberapa kelemahan dari Adagrad, khususnya masalah peluruhan laju pembelajaran yang terlalu cepat [1]. RMSprop memperbaiki Adagrad dengan memperkenalkan faktor peluruhan eksponensial untuk menyimpan rata-rata kuadrat gradien sebelumnya. Dengan cara ini, RMSprop menjaga *learning rate* tetap besar dan stabil

sepanjang proses pelatihan. Dalam RMSprop, parameter diperbarui dengan cara sebagai berikut:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla_{\theta_t} J(\theta_t) \tag{2-37}$$

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)(\nabla_{\theta_t} J(\theta_t))^2 \tag{2-38}$$

Dimana θ_t adalah parameter pada iterasi ke-t. η adalah laju pembelajaran (learning rate). $\nabla_{\theta_t} J(\theta_t)$ adalah gradien dari fungsi biaya J terhadap parameter θ pada iterasi ke-t. $E[g^2]_t$ adalah rata-rata eksponensial dari kuadrat gradien. ϵ adalah nilai kecil (biasanya 10^{-8}) untuk mencegah pembagian dengan nol. γ adalah faktor peluruhan, biasanya dipilih antara 0.9 dan 0.99. $E[g^2]_t$ menyimpan rata-rata eksponensial dari kuadrat gradien pada iterasi ke-t. Dengan menggunakan rata-rata eksponensial dari kuadrat gradien, RMSprop menjaga laju pembelajaran tetap besar dan stabil sepanjang proses pelatihan. RMSprop menyesuaikan *learning rate* untuk setiap parameter, sehingga efektif dalam menangani fitur yang jarang muncul dan data dengan skala yang berbeda. Memilih nilai yang tepat untuk γ dan η bisa memerlukan beberapa eksperimen. Menyimpan dan memperbarui rata-rata eksponensial dari kuadrat gradien juga menambah beban komputasi.

4. *Adam (Adaptive Moment Estimation)*

   Adam adalah algoritma optimasi yang populer digunakan dalam pelatihan jaringan syaraf tiruan. Algoritma ini menggabungkan keuntungan dari dua algoritma optimasi lainnya, yaitu RMSprop dan Momentum. Adam secara dinamis menyesuaikan *learning rate* untuk setiap parameter berdasarkan estimasi momen pertama (rata-rata eksponensial dari gradien) dan momen kedua (rata-rata eksponensial dari kuadrat gradien) [1]. Adam menggunakan dua perkiraan momen, yaitu momen pertama (mean) dan momen kedua (variance), untuk mengendalikan pembaruan parameter. Dengan ini, Adam dapat mengatasi masalah peluruhan *learning rate* yang cepat dan juga menangani gradien yang jarang muncul dengan lebih efektif. Pembaharuan parameter dengan menggunakan persamaan 2-39

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2-39}$$

   Dimana θ_t adalah parameter yang dilatih pada iterasi ke-t. η adalah *learning rate*. $\hat{m}_t$ adalah momen pertama. $\hat{v}_t$ adalah momen kedua. ϵ adalah nilai kecil (biasanya 10^{-8}) untuk mencegah pembagian dengan nol. Momen pertama dan kedua diperbaharui dengan persamaan 2-42 dan 2-43.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \nabla_{\theta_t} J(\theta_t) \tag{2-40}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_{\theta_t} J(\theta_t))^2 \tag{2-41}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2-42}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2-43}$$

Dimana $\nabla_{\theta_t} J(\theta_t)$ adalah gradien dari fungsi biaya J terhadap parameter θ pada iterasi ke-t. m_t adalah momen pertama (rata-rata eksponensial dari gradien). v_t adalah momen kedua (rata-rata eksponensial dari kuadrat gradien). β_1 dan β_2 adalah faktor peluruhan untuk momen pertama dan kedua, biasanya dipilih sebagai $\beta_1 = 0.9$ dan $\beta_2 = 0.999$. Dengan menggabungkan RMSprop dan Momentum, Adam sering kali mencapai konvergensi lebih cepat dan lebih stabil dibandingkan dengan algoritma optimasi lainnya. Adam biasanya memerlukan sedikit penyetelan hyperparameter, membuatnya lebih mudah digunakan dalam berbagai masalah optimasi. Namun menyimpan momen pertama dan kedua membutuhkan lebih banyak memori dibandingkan dengan algoritma seperti Stochastic Gradient Descent (SGD). Meskipun Adam biasanya bekerja dengan baik tanpa banyak penyetelan, dalam beberapa kasus tertentu, pemilihan hyperparameter bisa mempengaruhi kinerjanya secara signifikan.

2.2.8 *Backpropagation*

Backpropagation adalah algoritma yang digunakan untuk melatih jaringan saraf tiruan, terutama *deep learning*. Algoritma ini merupakan bagian penting dari proses pembelajaran yang memungkinkan jaringan untuk mengoptimalkan bobot dan biasnya sehingga dapat meminimalkan *loss function* [12]. *Backpropagation* adalah singkatan dari *backward propagation of errors*. Algoritma ini bekerja dengan menghitung gradien dari *loss function* terhadap setiap bobot dalam jaringan, kemudian menggunakan gradien ini untuk memperbarui bobot-bobot tersebut.

Proses *backpropagation* terdiri dari dua fase utama: *forward pass* dan *backward pass*. Pada fase *forward pass*, input data dijalankan melalui jaringan untuk menghasilkan *predicted value*. Kemudian, pada fase *backward pass*, kesalahan antara *predicted value* dan *expected value* dihitung dengan *loss function*, dan gradien dari *loss function* terhadap bobot-bobot dihitung untuk memperbarui bobot-bobot tersebut.

Langkah-langkah dalam *backpropagation* dimulai dengan inisialisasi bobot dan bias secara acak atau dengan nilai tertentu. Pada *forward pass*, input data diberikan ke lapisan input jaringan. Setiap neuron dalam lapisan tersembunyi dan lapisan output menghitung nilai aktivasinya berdasarkan input yang diterima dan fungsi aktivasi yang digunakan. Output dari lapisan terakhir dihitung dan digunakan untuk menghitung *loss function*. Setelah itu, *loss function* $L(y, \hat{y})$ dihitung berdasarkan perbedaan antara *expe-*

cted value (y) dan *predicted value* ($\hat{y}$). Pada backward pass, gradien dari *loss function* terhadap output dari lapisan terakhir dihitung. Kesalahan ini kemudian dipropagasi mundur melalui jaringan, lapisan demi lapisan, dengan menggunakan aturan rantai (chain rule) untuk menghitung gradien dari *loss function* terhadap setiap bobot. Bobot diperbarui dengan menggunakan gradien yang dihitung selama backward pass. Formula pembaruan bobot adalah:

$$w_{ij} := w_{ij} - \eta \frac{\partial L}{\partial w_{ij}}$$

Dimana w_{ij} adalah bobot antara neuron i dan neuron j, η adalah learning rate, dan $\frac{\partial L}{\partial w_{ij}}$ adalah gradien dari *loss function* terhadap bobot tersebut. Langkah-langkah ini diulangi untuk setiap batch data selama beberapa epoch sampai memenuhi kriteria tertentu. Kriteria tersebut dapat berupa *loss function* konvergen ke nilai minimum, *predicted value* sudah serupa dengan *expected value*, iterasi mencapai jumlah yang ditentukan, atau dengan menggunakan *relative error*.

2.2.9 *Relative error*

Relative error adalah ukuran ketidakakuratan dalam pengukuran atau estimasi yang dinyatakan sebagai perbandingan antara kesalahan absolut dan nilai sebenarnya. Persamaan dapat ditulis sebagai berikut :

$$\text{Relative Error} = \frac{\text{Absolute Error}}{\text{True Value}} \tag{2-44}$$

Dimana *Absolute Error* 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.

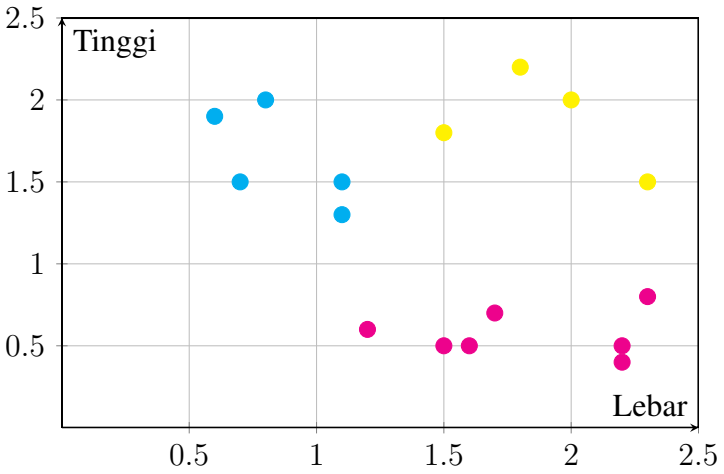
1. *Laptop* dengan spesifikasi sebagai berikut :
    - Sistem operasi Windows 10
    - Processor Intel I7-8750H
    - Memori 16GB DDR4
    - NVIDIA GeForce GTX 1050 (4GB)
    - SSD Samsung MZNLN128HAHQ 128GB
    - HDD HGST HTS721010A9E630
2. Bahasa pemrograman python 3.10.12 sebagai interpreter. Digunakan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

3.1.2 Bahan Tugas akhir

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

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

Dataset yang digunakan berupa data tinggi dan lebar suatu lemari, *buffet*, dan *wardrobe*. Dataset berupa data tabular dan dapat dipisahkan secara linear. Dataset berjumlah 16 *instance*, 2 *feature*, dan 3 kelas. Dataset dapat dilihat pada Tabel 3.1.



Gambar 3.1. Grafik dari dataset pada Tabel 3.1. *Cyan* adalah lemari, *magenta* adalah buffet, dan kuning adalah wardrobe

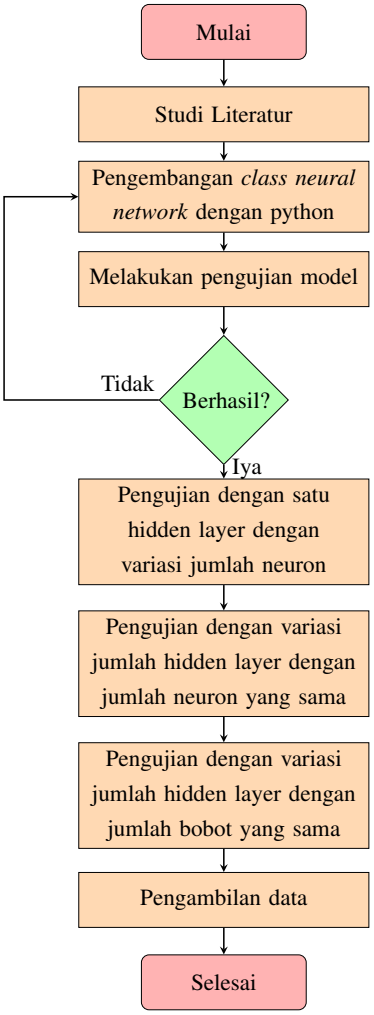
Pada Gambar 3.1 terlihat bahwa terdapat jarak diantara ketiga kelas sehingga dataset dapat dikatakan *linearly separable*.

3.2 Metode yang Digunakan

Penelitian dilakukan dengan mengembangkan model jaringan syaraf tiruan yang dilatih dengan dataset yang ada. Pengembangan jaringan syaraf tiruan dilakukan dengan bahasa pemrograman python versi 3.10.12 yang dijalankan pada *Cloud Computing* Google Colaboratory. Dataset akan diproses terlebih dahulu dengan metode normalisasi *minmax scalling*. Dataset yang telah dinormalisasi diteruskan ke lapisan berikutnya melalui *forward bias* hingga mencapai lapisan keluaran. Lapisan keluaran kemudian digunakan untuk menghitung *lost function* atau perhitungan error antara lapisan output dan label dari data yang diberikan. *Lost function* yang digunakan adalah *Mean Square Error* (MSE), nilai dari MSE ini seiring dengan pelatihan neural network akan berkurang dengan metode *Gradient Descent*. Jaringan syaraf tiruan akan menggunakan jumlah *hidden layer* yang bervariasi untuk melihat perubahan pada waktu komputasi dan akurasinya. Terdapat tiga model yang akan digunakan yaitu model pertama dengan satu hidden layer dengan variasi neuron-nya, model kedua dengan neuron yang sama namun dengan jumlah hidden layer yang bervariasi, dan model ketiga dengan jumlah bobot yang sama namun dengan jumlah hidden layer yang berbeda. Ketiga model akan menggunakan fungsi aktivasi *step function* pada *output layer*. Pelatihan model menggunakan metode *gradient descent* untuk menurunkan nilai dari *Mean Square Error* sebagai *lost function*. Nilai dari bobot akan beradaptasi seiring dengan penurunan nilai *Mean Square Error*.

3.3 Alur Penelitian

Penelitian ini dilakukan secara bertahap dengan alur seperti pada Gambar 3.2. 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.



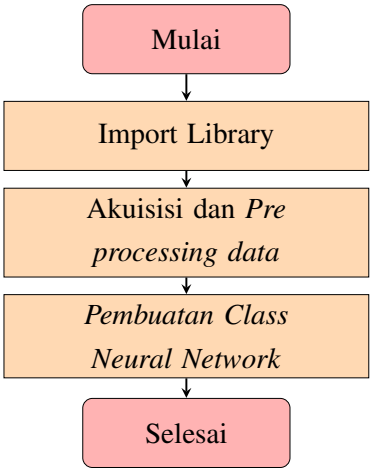
Gambar 3.2. *Flowchart* alur penelitian

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 3.3



Gambar 3.3. 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 Akuisisi dan *pre-processing* data

Dataset pada Tabel 3.1 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 dan dirubah skalanya. 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 3.2. Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah diberi label

Tabel 3.2 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 2-11.

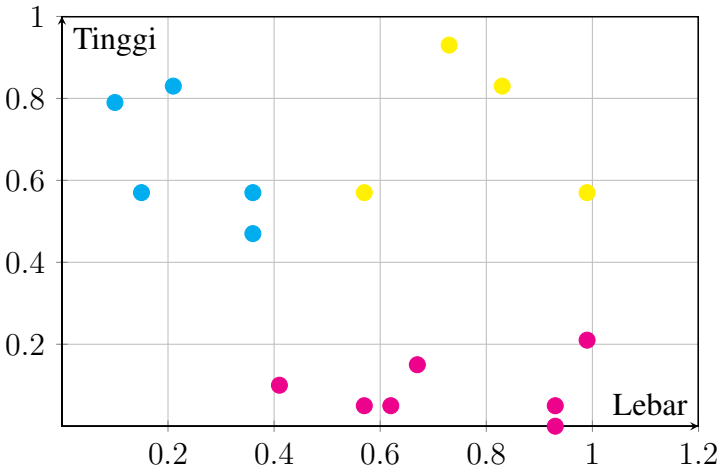
```
1    def minmax_scaling (x_t):
2        min = np.min(x_t)
3        max = np.max(x_t)
4        return ((x_t-min)/(max-min))
5
6    X = minmax_scaling(data.iloc[:,:2].values.T)
7    y = data.iloc[:,2:5].values.T
```

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

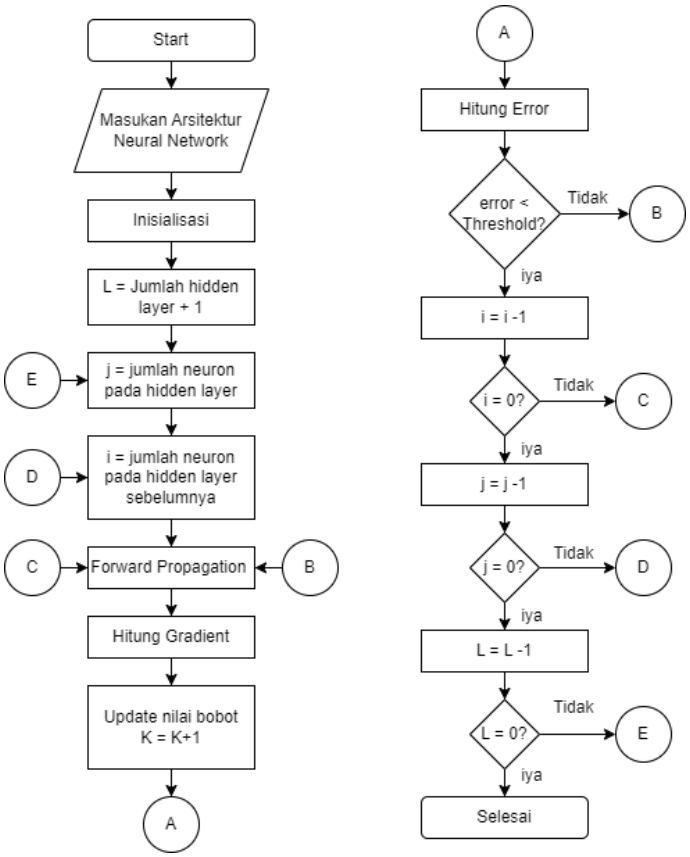
Tabel 3.3. Dataset ukuran lebar dan tinggi dari lemari, buffet, dan wardrobe setelah normalisasi

Table 3.3 menunjukan data telah diubah ukurannya menjadi rentang 0 hingga 1 setelah melalui proses normalisasi Minmax Scaling. Gambar 3.4 menunjukkan plot kartesian dari Table 3.3.



Gambar 3.4. 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 3.5. 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 3.5.

```
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 2.6 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 2-8. 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 *lost function* MSE sebagai parameter *error* dari model. Umumnya pada awal pelatihan nilai dari *lost function* akan cukup tinggi. Semakin kecil *lost function* semakin baik pula modelnya. Untuk menurunkan nilai *lost function* dapat menggunakan *Gradient descent* pada persamaan 2-8. Yaitu dengan menggunakan turunan *lost function* terhadap parameter yang ingin dilatih. Karena dataset yang digunakan cukup kecil maka seluruh data akan digunakan dan disebut dengan *stocasthic gradient descent*. 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}'])
13                  temp_2 = (2 / n) * np.sum(self.weight[f'w{l
    }']**2, axis=0)
14
15              elif l == 0:
16                  self.delta[f'{l}'] = np.dot(self.dY_dY[f'{l
    +1}'].T, temp)
17                  self.dJ_dw[f'{l}'] = np.dot(self.delta[f'{l}'],
    self.X.T)
18
19                  self.d2J_dw2[f'{l}'] = np.outer(temp_2[:, np.
    newaxis], np.sum(self.X**2, axis =1)[np.newaxis, :])
20
21              else:
22                  self.dY_dY[f'{l}'] = self.weight[f'w{l}']
23                  self.delta[f'{l}'] = np.dot(self.dY_dY[f'{l
    +1}'].T, temp)
24                  self.dJ_dw[f'{l}'] = np.dot(self.delta[f'{l}'],
    self.Y[f'{l-1}'].T)
25                  temp = self.delta[f'{l}']
26
27                  self.d2J_dw2[f'{l}'] = np.outer(temp_2[:, np.
    newaxis], np.sum(self.Y[f'{l-1}']**2, axis=1)[np.newaxis, :])
28                  temp_2 = np.dot(temp_2, self.weight[f'w{l}']**2)
29
30          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 2-32. 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:

```
1   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 *lost function*-nya. Pelatihan jaringan syaraf tiruan pada program yang dibuat menggunakan metode *gradient descent*, sehingga nilai dari *lost 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
```

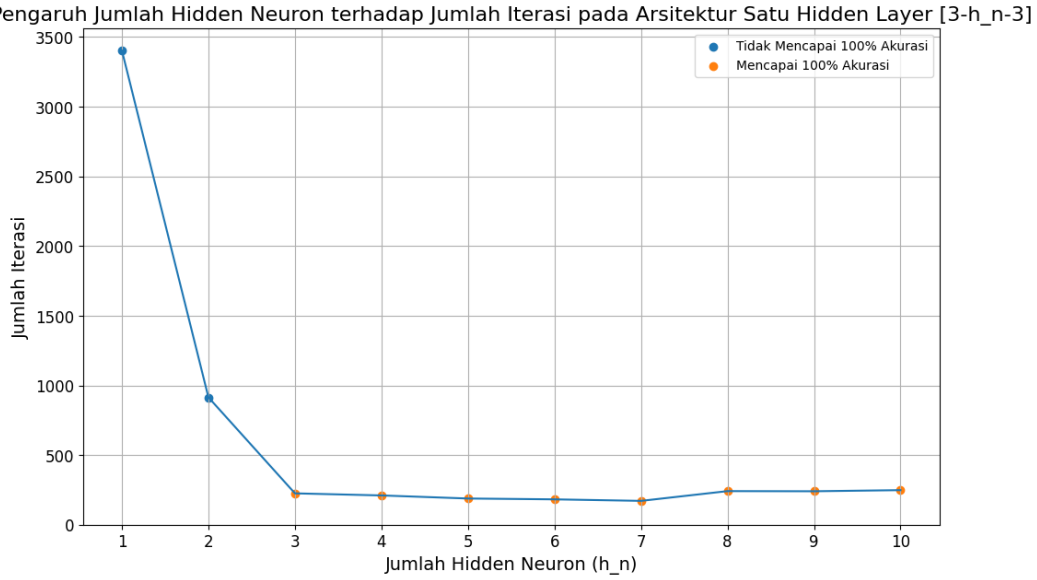
Pada *Source Code* diatas menunjukkan bagaimana untuk mencari jumlah bobot yang dapat digunakan. Untuk jumlah bobot sebesar 72 maka arsitektur yang akan dibuat adalah 3-4-6-4-3, 3-6-6-3, dan 3-12-3. Untuk jumlah bobot sebesar 105 maka arsitektur yang akan dibuat adalah 3-5-5-5-5-3 dan 3-7-4-8-3. Untuk 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.

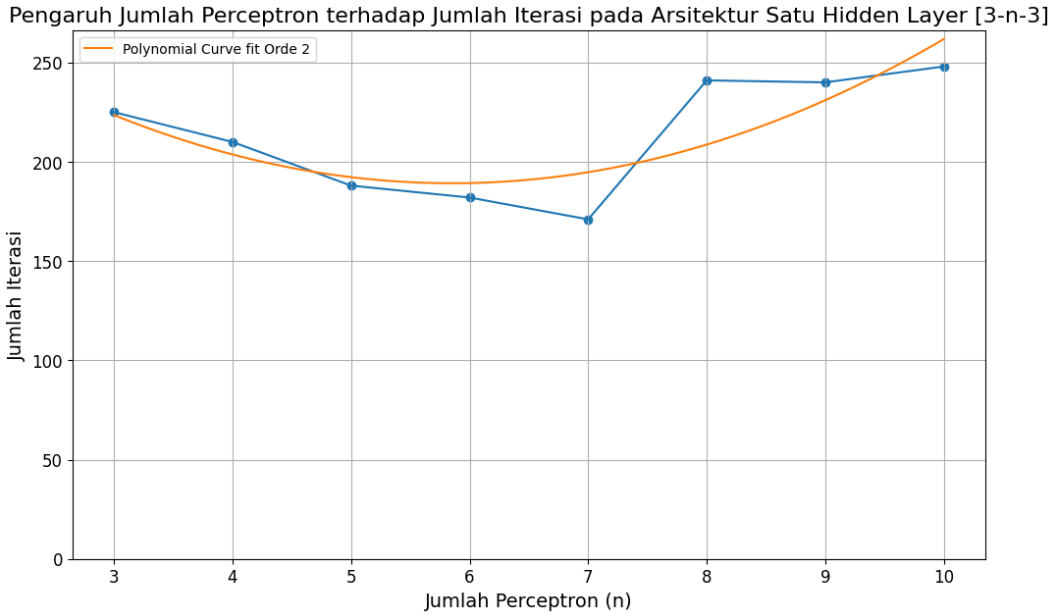
BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Pengujian dengan satu hidden layer dengan variasi jumlah neuron



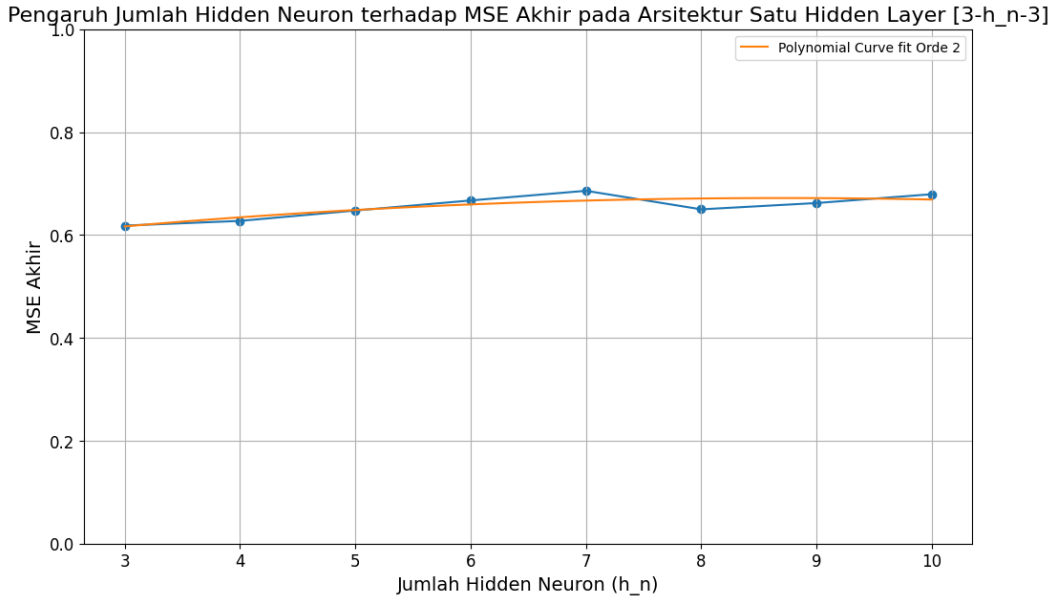
Gambar 4.1. Grafik akurasi terhadap jumlah iterasi

Pada gambar 4.1 terlihat pengaruh jumlah hidden neuron pada 1 hidden layer dengan arsitektur [3-h_n-3]. Terlihat pada konfigurasi dengan 1 dan 2 hidden neuron tidak mencapai akurasi 100%. Sedangkan pada konfigurasu 3 hingga 10 hidden neuron berhasil mencapai 100% akurasi. Untuk melihat trend nya dapat menggunakan *Polynomial Curve Fit*.



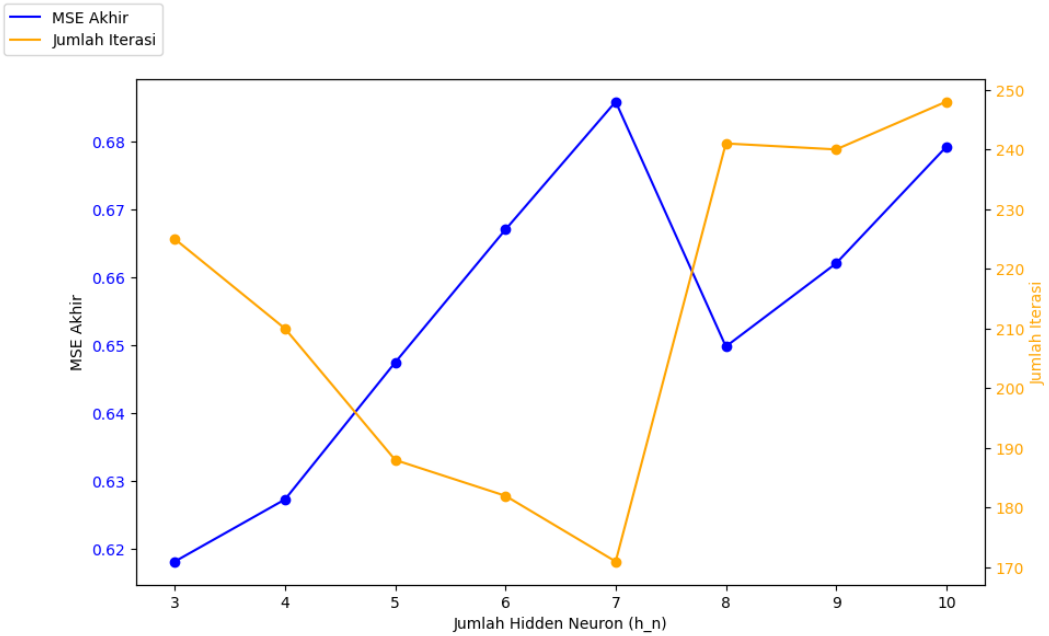
Gambar 4.2. Grafik akurasi terhadap jumlah iterasi

Pada gambar 4.2 akan difokuskan pada jaringan yang berhasil mencapai 100% akurasi. Terlihat dengan menggunakan *Polynomial Curve Fit* orde 2 bahwa iterasi turun seiring degan penambahan jumlah hidden neuron hingga titik tertentu, kemudian meningkat kembali.



Gambar 4.3. Grafik akurasi terhadap jumlah iterasi

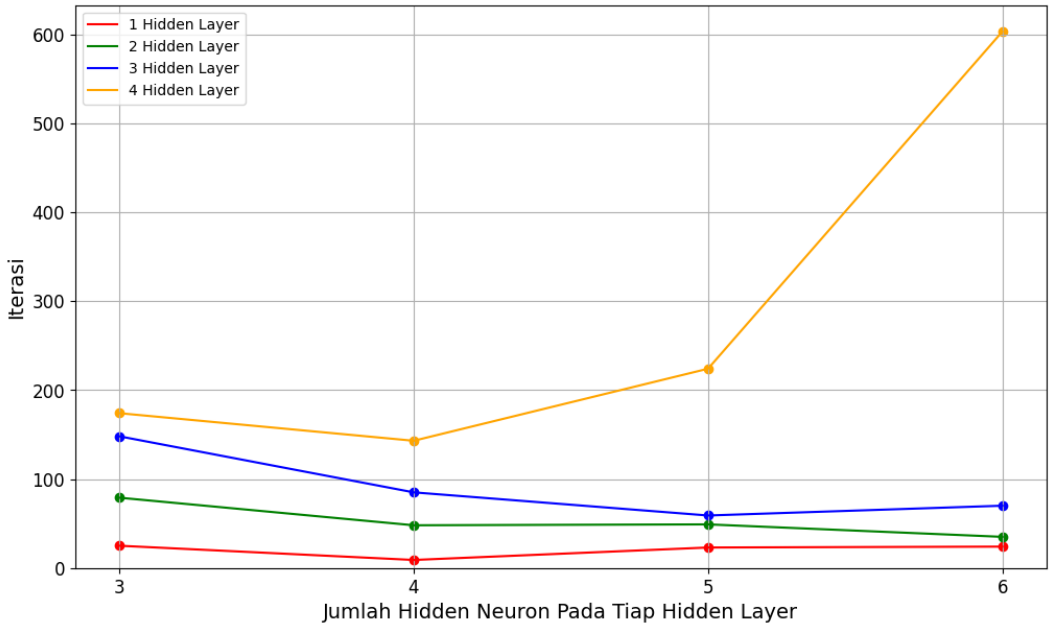
Pada gambar 4.3 terlihat bahwa nilai MSE akhir meningkat seiring dengan penurunan jumlah iterasi yang diperlukan untuk mencapai 100% akurasi. Hal ini akan lebih terlihat jika kedua plot dibuat dalam satu figur seperti pada Gambar 4.4.



Gambar 4.4. Grafik akurasi terhadap jumlah iterasi

4.2 Hasil Pengujian dengan variasi jumlah hidden layer dengan jumlah neuron yang sama

Pelatihan dilakukan dengan jumlah hidden layer yang bervariasi mulai dari 1 hingga 4 hidden layer dan 3 hingga 6 Hidden Neuron. Nilai awal bobot dibuat serupa untuk semua variasi yaitu sesuai dengan Persamaan 3-2. Nilai Threshold dibuat 0.1 dan nilai *learning rate* dibuat 0.1. Detail pelatihan lebih lanjut dijelaskan pada Halaman 37. Data hasil yang diambil berupa grafik jumlah Hidden Neuron dengan jumlah iterasinya.

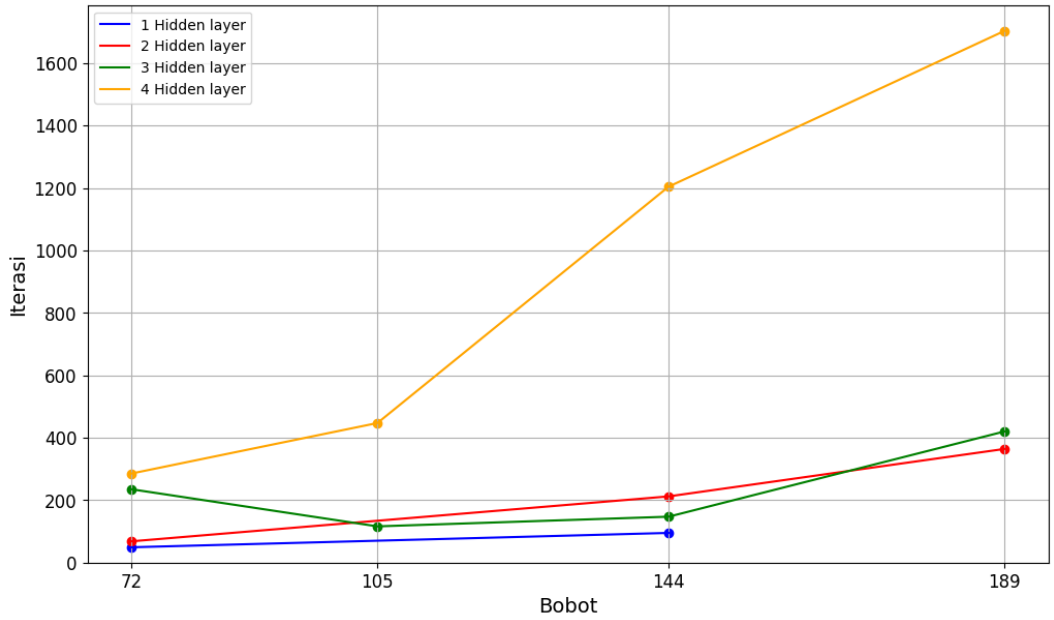


Gambar 4.5. Grafik akurasi terhadap jumlah iterasi

Pada Gambar 4.5 terlihat bahwa semua konfigurasi dari 1 hidden layer hingga 4 hidden layer mencapai 100% akurasi. Terlihat bahwa semakin banyak *hidden layer* semakin banyak pula iterasi yang diperlukan. Dan dengan konfigurasi 4 *Hidden Layer* dengan 6 *Hidden Neuron* di setiap *Hidden layer*nya memiliki jumlah iterasi terbanyak. Pada konfigurasi tersebut akan memiliki 144 bobot.

4.3 Hasil Pengujian dengan variasi jumlah hidden layer dengan jumlah bobot yang sama

Pengujian menggunakan bobot yang sama dengan variasi jumlah Hidden Layer yang berbeda. Pemilihan bobot yang benar diperlukan karena tidak semua jumlah bobot dapat dibuat pada beberapa jumlah *Hidden layer* tertentu.



Gambar 4.6. Grafik akurasi terhadap jumlah iterasi

Sebagai contoh adalah pada Gambar 4.6, terlihat bahwa bobot 105 tidak memiliki konfigurasi pada 1 dan 2 Hidden layer. Pada Jumlah bobot 189 juga tidak ditemukan konfigurasi untuk 1 Hidden layer. Berdasarkan gambar tersebut 4 hidden layer selalu memiliki jumlah iterasi terbanyak dibanding dengan konfigurasi *Hidden Layer* lebih sedikit. Sedangkan dengan konfigurasi 3 *hidden layer* memiliki jumlah iterasi yang lebih tinggi dari 1 atau 2 hidden layer kecuali pada jumlah bobot 144 dan selalu lebih rendah dari 4 jumlah hidden layer.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, dapat ditarik beberapa kesimpulan sebagai berikut:

1. Penggunaan arsitektur dengan 1 hidden layer menunjukkan bahwa semakin banyak neuron pada hidden layer semakin sedikit iterasi yang diperlukan sampai pada 7 hidden neuron dan meningkat kembali.
2. Penggunaan arsitektur dengan jumlah neuron yang sama pada tiap layer menunjukkan semakin banyak hidden layer semakin lama pula waktu iterasi yang dibutuhkan. Hal ini terjadi pada konfigurasi 3, 4, 5, dan 6 hidden layer.
3. Penggunaan arsitektur dengan jumlah bobot yang sama menunjukkan 4 hidden layer selalu memiliki waktu komputasi yang lebih lama dibandingkan dengan jumlah hidden layer yang lebih sedikit. Satu hidden layer selalu memiliki waktu komputasi yang lebih sedikit dibanding dengan jumlah hidden layer lebih banyak.
4. Penggunaan arsitektur dengan jumlah hidden neuron yang lebih sedikit dari neuron pada layer masukan selalu gagal mencapai 100%. Hal ini berlaku pada jumlah hidden neuron pada hidden layer ke berapapun.

5.2 Saran

Dari penelitian yang dilakukan, masih terdapat banyak kekurangan. Penelitian ini hanya berfokus pada aristektur Multilayer Perceptron tanpa fungsi aktifasi pada lapisan tersembunyi dan dilakukan pada data yang dapat dipisahkan secara linear.

1. Dapat menggunakan jaringan dengan arsitektur yang lebih beragam seperti Convolutiona Neural Network, Reccurent Neural Network, dll.
2. Melakukan penelitian dengan data yang lebih beragam.Misalnya saja dengan data yang tidak dapat dipisahkan secara linear atau dengan data yang memiliki outlier yang bervariasi

DAFTAR PUSTAKA

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

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

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

[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] 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

[6] 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.

[7] 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.

[8] 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.

[9] 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

[10] 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, p. 279–290. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-65172-9_24

[11] 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.

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

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

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