

Tugas Besar 1 IF3270 Pembelajaran Mesin

Feedforward Neural Network



Disusun oleh:

Immanuel Sebastian Girsang	(13522058)
Venantius Sean Adi Nugroho	(13522078)
Julian Chandra Sutadi	(13522080)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2025

DAFTAR ISI

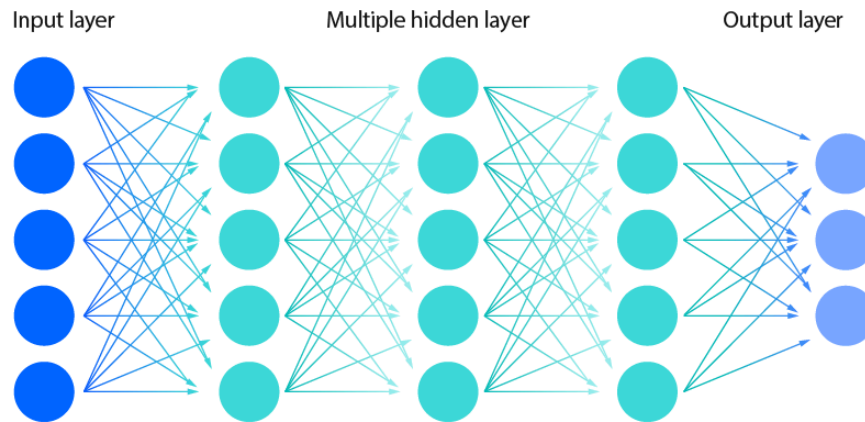
BAB I.....	3
DESKRIPSI PERSOALAN.....	3
1.1. Deskripsi Tugas.....	3
Gambar 1. Visualisasi dari Feed Forward Neural Network.....	3
1.1.1. Definisi FFNN.....	3
1.1.2. Inisialisasi bobot.....	4
1.1.3. Fungsi Aktivasi.....	5
1.1.4. Fungsi Loss.....	6
1.1.5. Gradient Descent.....	7
BAB II.....	8
PEMBAHASAN.....	8
2.1. Deskripsi kelas beserta deskripsi atribut dan methodnya.....	8
2.1.1. Kelas Value.....	8
2.1.2. Kelas Module.....	11
2.1.3. Kelas Layer.....	11
2.1.4. Kelas FFNN.....	12
2.1.5. Kelas ZeroInit.....	14
2.1.6. Kelas RandomUniform.....	14
2.1.7. Kelas RandomNormal.....	15
2.1.8. Kelas XavierInit.....	16
2.1.9. Kelas Helnit.....	16
2.1.10. Kelas LossFunction.....	17
2.1.11. Kelas MSELoss.....	17
2.1.12. Kelas BinaryCrossEntropyLoss.....	17
2.1.13. Kelas CategoricalCrossEntropyLoss.....	17
2.2. Penjelasan Forward Propagation.....	18
2.3. Penjelasan Backward Propagation dan Weight Update.....	18
2.4. Hasil Pengujian.....	19
2.4.1. Pengaruh Depth dan Width.....	19
2.4.1.1. Depth sebagai variabel bebas, width konstan = 100 neuron tiap layer.....	19
2.4.1.1.1. Jumlah hidden layer = 2.....	19
2.4.1.1.2. Jumlah hidden layer = 3.....	19
2.4.1.1.3. Jumlah hidden layer = 4.....	20
2.4.1.2. Width sebagai variabel bebas, depth konstan = 2 hidden layer.....	21
2.4.1.2.1. Jumlah neuron tiap layer = 50.....	21
2.4.1.2.2. Jumlah neuron tiap layer = 100.....	21
2.4.1.2.3. Jumlah neuron tiap layer = 200.....	22
2.4.2. Pengaruh Fungsi Aktivasi.....	22

2.4.2.1. Linear Activation Function.....	22
2.4.2.2. ReLU Activation Function.....	24
2.4.2.3. Sigmoid Activation Function.....	26
2.4.2.4. Tanh Activation Function.....	27
2.4.2.5. ELU Activation Function.....	29
2.4.2.6. Leaky ReLU Activation Function.....	30
2.4.3. Pengaruh Learning Rate.....	32
2.4.3.1. Learning rate = 0.01.....	32
2.4.3.2. Learning rate = 0.1.....	33
2.4.3.3. Learning rate = 0.5.....	35
2.4.4. Pengaruh Inisialisasi Bobot.....	36
2.4.4.1. Inisialisasi nol.....	37
2.4.4.2. Inisialisasi acak dengan distribusi uniform.....	38
2.4.4.3. Inisialisasi acak dengan distribusi normal.....	39
2.4.4.4. Inisialisasi Xavier.....	41
2.4.4.5. Inisialisasi He.....	43
2.4.5. Perbandingan dengan Library sklearn.....	44
2.4.6. Perbandingan Regularisasi.....	45
2.4.6.1. Tanpa regularisasi.....	45
2.4.6.2. Regularisasi L1.....	46
2.4.6.3. Regularisasi L2.....	48
2.5. Analisis.....	50
2.5.1. Pengaruh Depth terhadap Akurasi.....	50
2.5.2. Pengaruh Width terhadap Akurasi.....	50
2.5.3. Pengaruh Learning Rate terhadap Akurasi.....	51
2.5.4. Pengaruh Inisialisasi Bobot terhadap Akurasi.....	51
2.5.5. Pengaruh Fungsi Aktivasi terhadap Akurasi.....	52
2.5.6. Perbandingan dengan Library Scikit-Learn.....	53
2.5.7. Pengaruh Regularisasi terhadap Akurasi.....	53
BAB III.....	55
KESIMPULAN DAN SARAN.....	55
PEMBAGIAN TUGAS.....	57
REFERENSI.....	58

BAB I

DESKRIPSI PERSOALAN

1.1. Deskripsi Tugas



Gambar 1. Visualisasi dari Feed Forward Neural Network

Dalam tugas besar ini, mahasiswa akan membuat *Feed Forward Neural Network* (FFNN) dari *scratch*. Model ini nantinya akan digunakan untuk memprediksi data MNIST yaitu data dari gambar tulisan tangan manusia dari digit 0 sampai 9. Model yang diimplementasikan nantinya akan digunakan untuk memprediksi digit yang ada pada suatu gambar tertentu.

1.1.1. Definisi FFNN

Feed Forward Neural Network merupakan salah satu model dalam pembelajaran mesin yang terinspirasi dari jaringan saraf manusia. Model jaringan saraf ini disebut dengan Feed Forward karena aliran informasi hanya bisa bergerak ke satu arah yaitu dari lapisan *input* menuju lapisan *output* melalui berbagai *hidden layer* tanpa adanya *feedback*. Di setiap lapisan, terdapat beberapa *neuron* yang berfungsi untuk menerima input dari neuron lainnya, serta melakukan "aktivasi" terhadap nilai tersebut. Bobot ini kemudian akan digunakan untuk *neuron-neuron* di layer selanjutnya. Terdapat 3 lapisan dalam FFNN :

1. Lapisan Input (*Input layer*)

Lapisan ini menerima data mentah dari luar sistem. Biasanya, setiap *neuron* dalam lapisan ini mewakili satu fitur dalam *dataset*. Di lapisan ini, tidak ada operasi ataupun transformasi yang dilakukan. Lapisan ini hanya bertugas meneruskan *input* ke lapisan selanjutnya.

2. Lapisan Tersembunyi (*Hidden layer*)

Secara strategi, lapisan ini bertugas untuk mengekstraksi “pola” dari data yang ada. Setiap neuron dalam lapisan tersembunyi ini memiliki *weight* dan juga *bias* serta fungsi aktivasi. Fungsi aktivasi digunakan untuk memperkenalkan nonlinearitas pada data yang ada.

3. Lapisan Output (*Output layer*)

Lapisan ini menghasilkan hasil akhir dari jaringan saraf. Fungsi aktivasi di lapisan ini bergantung pada *task* yang sedang dilakukan. Beberapa fungsi aktivasi yang

1. **Regresi** menggunakan fungsi aktivasi *linear*
2. **Klasifikasi biner** menggunakan fungsi aktivasi *sigmoid*
3. **Klasifikasi multikelas** menggunakan fungsi aktivasi *softmax*

Secara singkat, FFNN bekerja dengan cara melalui proses *forward propagation*, di mana input diberikan ke jaringan dan diteruskan ke lapisan berikutnya hingga menghasilkan output akhir. Berikut adalah langkah-langkah utama dalam proses ini:

1. Inisialisasi Bobot dan Bias

Setiap koneksi antar neuron memiliki bobot awal yang diinisialisasi secara acak. Terdapat beberapa *method* yang bisa digunakan untuk bias juga diberikan pada setiap neuron untuk meningkatkan fleksibilitas model.

2. Kalkulasi Aktivasi Tiap Neuron

Hasil dari perkalian antara bobot dan juga *input* di setiap neuron akan di “aktivasi” dengan cara memasukkan nilai tersebut ke dalam fungsi aktivasi yang digunakan.

3. Prediksi Hasil

Setelah propagasi selesai, lapisan output memberikan hasil akhir, yang bisa berupa nilai numerik (regresi) atau kelas (klasifikasi).

4. Evaluasi dengan Fungsi Loss

Perbedaan antara output yang dihasilkan dan label yang diharapkan dihitung menggunakan *fungsi loss*.

5. Optimasi dengan Backpropagation & Gradient Descent

Bobot diperbarui dengan menurunkan error menggunakan algoritma *gradient descent*, di mana turunan *fungsi loss* terhadap setiap bobot dihitung dan dilakukan pembaruan secara iteratif.

1.1.2. Inisialisasi bobot

Inisialisasi bobot adalah proses memberikan nilai awal pada bobot sebelum model mulai dilatih. Jika inisialisasi dilakukan dengan tidak tepat, model bisa mengalami *vanishing gradient* atau *exploding gradient*, yang membuat jaringan sulit untuk belajar secara efektif. Beberapa metode inisialisasi yang digunakan di tugas besar kali ini adalah:

1. Zero Initialization

Inisialisasi dengan membuat semua nilai *weight* dan *bias* menjadi 0.

2. Random Normal Initialization

Random normal initialization dilakukan dengan menerima input *variance* dan juga *mean* untuk nantinya digunakan sebagai standar dalam membagi nilai *weight* dan *bias* secara random namun tetap terdistribusi normal dengan *mean* serta *variance* yang dimasukkan pengguna.

3. Random Uniform Initialization

Random uniform initialization dilakukan dengan menerima input *upper bound* dan juga *lower bound* untuk nantinya digunakan sebagai standar dalam membagi nilai *weight* dan *bias* secara random namun tetap terdistribusi *uniform*.

4. Xavier Initialization

Xavier *initialization* dilakukan dengan menggunakan rumus yang dibuat oleh Xavier Glorot dengan cara menggunakan informasi mengenai berapa input yang akan diterima oleh *input* dan juga berapa *output* yang akan diberikan olehnya. Rumus yang digunakan apabila klasifikasi yang dilakukan adalah *multiclass* sebagai berikut:

$$x = \sqrt{\frac{6}{\text{jumlah output} + \text{input output}}}$$

Apabila klasifikasi yang dilakukan bersifat biner, maka rumus yang dilakukan adalah sebagai berikut:

$$x = \sqrt{\frac{2}{\text{jumlah output} + \text{input output}}}$$

Dari sini, kita akan mendistribusikan bias dan bobot dari range $[-x, +x]$ secara uniform di layer-layer kita.

5. He Initialization

He Initialization merupakan inisialisasi yang dibuat secara spesifik untuk Fungsi aktivasi ReLU dengan merandom angka secara distribusi normal dengan mean = 0 dan juga variansi dengan rumus:

$$\frac{2}{n}$$

Dengan n adalah jumlah input dari layer yang sedang diinisialisasi.

1.1.3. Fungsi Aktivasi

Fungsi aktivasi adalah fungsi non-linear yang menentukan apakah neuron akan diaktifkan berdasarkan input yang diterimanya. Tanpa fungsi aktivasi, jaringan hanya akan merepresentasikan fungsi linear, yang membatasi kemampuannya dalam mempelajari pola kompleks. Beberapa fungsi yang akan digunakan di tugas besar ini adalah:

1. Linear

$$Linear(x) = x$$

2. Sigmoid

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

3. Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

4. ReLU

$$ReLU(x) = \max(0, x)$$

5. Leaky ReLU

$$Leaky_ReLU(x, \alpha) = x \text{ if } x > 0, \alpha * x \text{ otherwise}$$

6. ELU

$$Elu(x, \alpha) = \alpha * (e^x - 1) \text{ if } x < 0, x \text{ otherwise}$$

7. Softmax

$$\text{Untuk vector } \vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n, \\ softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

1.1.4. Fungsi Loss

Fungsi loss digunakan untuk mengukur seberapa baik model memprediksi output yang diharapkan. Model dilatih untuk meminimalkan fungsi loss. Beberapa fungsi *loss* yang akan digunakan di tugas besar kali ini adalah:

1. Mean Square Error

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

2. Binary Cross Entropy

$$\mathcal{L}_{BCE} = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$$

y_i = Actual binary label (0 or 1)

$\hat{y}_i$ = Predicted value of y_i

n = Batch size

3. Categorical Cross Entropy

$$\mathcal{L}_{CCE} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C (y_{ij} \log \hat{y}_{ij})$$

y_{ij} = Actual value of instance i for class j

$\hat{y}_{ij}$ = Predicted value of y_{ij}

C = Number of classes

n = Batch size

1.1.5. Gradient Descent

Gradient Descent adalah algoritma optimasi yang digunakan untuk memperbarui bobot dalam jaringan saraf berdasarkan turunan dari fungsi loss. Pada setiap *epochnya*, bobot yang ada di setiap lapisan akan di *update* untuk meminimalkan *loss function* dengan cara mengurangnya sebesar *learning rate* tertentu dari gradiennya terhadap *loss function*.

BAB II

PEMBAHASAN

2.1. Deskripsi kelas beserta deskripsi atribut dan methodnya

2.1.1. Kelas Value

Deskripsi Kelas	
Kelas ini berfungsi sebagai wrapper dari struktur data data torch.Tensor yang terlibat di dalam perhitungan FFNN dan memiliki fitur auto-differentiation dengan melakukan pembangunan graf. Definisi operasi-operasi dengan turunan yang khas di-override di kelas ini. Sehingga, setiap kali melakukan operasi, yang terjadi adalah pengembalian objek Value yang baru dengan informasi operasi yang harus dilakukan saat suatu nilai yang memiliki objek tersebut di dalam graf terarahnya ingin mencari nilai turunan parsial dari objek pemanggil terhadap semua operan yang terlibat. Struktur kelas ini mengambil inspirasi dari library micrograd dalam implementasi auto differentiation, serta library autodidact yang digunakan sebagai bahan ajar di University of Toronto untuk mendapatkan referensi perhitungan vector-Jacobian product dan broadcasting jika yang terlibat dalam operasi adalah struktur data dengan dimensi lebih dari 1. Fungsi-fungsi aktivasi didefinisikan pada kelas ini. Jika operasinya merupakan komposisi dari operasi dasar yang sudah terdefinisi pada kelas ini, maka metode untuk implementasi fungsi tersebut tidak akan mendefinisikan ulang atribut <code>_backward</code> dan langsung mengembalikan nilai yang didapatkan dari operasi saja.	
Atribut	
data	Menyimpan data berupa objek torch.Tensor.
requires_grad	Boolean flag yang menandakan apakah objek perlu nilai grad dihitung saat backward propagation dilakukan.
grad	Menyimpan nilai gradien objek, yaitu nilai turunan parsial dari objek Value yang memanggil metode <code>backward()</code> terhadap objek ini.
_backward	Merupakan lambda function yang dapat dipanggil ketika pemanggilan <code>backward()</code> dilakukan oleh objek lain pada elemen lebih akhir dari topological graph operasi-operasi yang terjadi.

<code>_prev</code>	Menyimpan child nodes, yaitu operand dari operasi yang terdefinisi pada kelas Value yang menghasilkan objek yang diacu.
<code>_op</code>	String yang menandakan operasi yang menyebabkan objek terbuat.
Metode	
<code>__init__(self, data, requires_grad=False, _children=(), _op="")</code>	Konstruktor objek.
<code>__repr__(self)</code>	Memberi format yang akan muncul jika objek dicetak.
<code>T(self)</code>	Melakukan transpose matriks.
<code>__add__(self, other)</code>	Override metode penjumlahan ketika operan sebelah kiri adalah objek Value.
<code>__mul__(self, other)</code>	Override metode perkalian antareleman ketika operan sebelah kiri adalah objek Value.
<code>__matmul__(self, other)</code>	Override metode perkalian matriks ketika operan sebelah kiri adalah objek Value.
<code>__truediv__(self, other)</code>	Override metode pembagian ketika operan sebelah kiri adalah objek Value.
<code>__pow__(self, other)</code>	Override metode pemangkatan untuk operan objek Value terhadap int atau float.
<code>exp(self)</code>	Pemangkatan bilangan euler terhadap objek Value.
<code>log(self)</code>	Perhitungan logaritma terhadap objek Value.
<code>abs(self)</code>	Perhitungan nilai mutlak terhadap objek Value.
<code>sum(self)</code>	Mencari jumlah semua elemen atribut data.
<code>mean(self)</code>	Mencari rata-rata semua elemen atribut data.
<code>linear(self)</code>	Fungsi aktivasi, mengembalikan diri sendiri.
<code>relu(self)</code>	Fungsi aktivasi, mengembalikan nilai

	maksimum dari nol dan tiap elemen atribut data dalam dimensi atribut data.
<code>tanh(self)</code>	Fungsi aktivasi, mengembalikan nilai tanh dari tiap elemen atribut data dalam dimensi atribut data.
<code>sigmoid(self)</code>	Fungsi aktivasi, mengembalikan nilai sigmoid dari tiap elemen atribut data dalam dimensi atribut data.
<code>leaky_relu(self, negative_slope=0.01)</code>	Fungsi aktivasi, mengembalikan nilai Leaky ReLU dari tiap elemen atribut data dalam dimensi atribut data.
<code>elu(self, alpha=1.0)</code>	Fungsi aktivasi, mengembalikan nilai ELU dari tiap elemen atribut data dalam dimensi atribut data.
<code>softmax(self)</code>	Fungsi aktivasi, mengembalikan nilai proporsi eksponensial dari tiap elemen atribut data dalam dimensi atribut data.
<code>clamp(self, min_val, max_val)</code>	Memotong nilai tiap elemen atribut data yang lebih kecil dari <code>min_val</code> menjadi <code>min_val</code> dan yang lebih besar dari <code>max_val</code> menjadi <code>max_val</code> .
<code>backward(self)</code>	Membangun topological graph, kemudian dari elemen terakhir pada graf memanggil fungsi yang disimpan pada atribut <code>_backward</code> elemen tersebut sampai pada elemen pertama.
<code>__neg__(self)</code>	Override ketika suatu objek Value memiliki prefiks '- '.
<code>__radd__(self, other)</code>	Override metode penjumlahan ketika operan sebelah kanan adalah objek Value.
<code>__sub__(self, other)</code>	Override metode pengurangan ketika operan sebelah kiri adalah objek Value.
<code>__rsub__(self, other)</code>	Override metode pengurangan ketika operan sebelah kanan adalah objek Value.
<code>__rmul__(self, other)</code>	Override metode perkalian antarelemen ketika operan sebelah kanan adalah objek Value.

<code>__rtruediv__(self, other)</code>	Override metode pembagian ketika operan sebelah kanan adalah objek Value.
<code>_reduce_grad(self, grad, target_shape)</code>	Melakukan "reverse broadcasting." Setiap kali melakukan operasi antara dua tensor dengan ukuran yang berbeda namun kompatibel untuk broadcasting, maka torch akan secara otomatis melakukan broadcasting. Namun, karena auto differentiation diimplementasikan secara mandiri, metode ini yang memastikan reduksi nilai dari operasi bersama nilai gradien objek pemanggil metode <code>_backward</code> dilakukan dengan benar.

2.1.2. Kelas Module

Deskripsi Kelas	
Kelas general yang akan diturunkan oleh kelas Layer dan kelas FFNN, sesuai API PyTorch.	
Metode	
<code>zero_grad(self)</code>	Membuat semua nilai yang diberikan metode parameters menjadi nol.
<code>parameters(self)</code>	Mengembalikan semua parameter objek.

2.1.3. Kelas Layer

Deskripsi Kelas	
Unit terkecil Feed Forward Neural Network. Untuk mendukung paralelisasi, semua weights dan bias model disimpan dalam bentuk matriks (2D-tensor), sehingga, tidak terdapat kelas neuron.	
Atribut	
<code>activation</code>	Fungsi aktivasi layer.
<code>W</code>	Bobot layer.
<code>b</code>	Bias layer.
<code>input_shape</code>	Dimensi masukan layer.

output_shape	Dimensi keluaran layer.
init_method	Metode inisialisasi bobot dan bias layer.
Metode	
<code>__init__(self, output_shape, activation, init_method, input_shape)</code>	Menginisialisasi bentuk layer, termasuk berapa neuron (output_shape), fungsi aktivasi, ingin diinisialisasi dengan cara apa, dan dimensi masukkan.
parameters(self)	Implementasi metode yang sama dari kelas Module.
<code>__call__(self, X)</code>	Menghitung nilai net dan output aktivasi berdasarkan input X.

2.1.4. Kelas FFNN

Deskripsi Kelas	
Kelas ini menyediakan API untuk Feed Forward Neural Network dan merupakan implementasi dari kelas Module yang terinspirasi dari API TensorFlow, PyTorch, dan micrograd.	
Atribut	
random_seed	Nilai random untuk seeding pengacakan agar eksperimen dapat diulang.
learning_rate	Konstanta yang mengatur proporsi perubahan bobot saat melakukan optimisasi.
verbose	Verbose bernilai satu artinya detil akan ditampilkan saat pelatihan.
epochs	Besaran ini menyatakan berapa kali seluruh dataset ingin dikunjungi secara keseluruhan ketika pelatihan model.
batch_size	Menyatakan banyaknya baris dari dataset latih yang ingin dihitung sebelum melakukan pembaruan nilai bobot/bias.
layers	List dari objek Layer.
loss_function	Objek fungsi loss yang akan digunakan untuk perhitungan loss.

losses	List yang berisi sejarah perhitungan loss terhadap dataset latih pada bagian akhir tiap epoch.
val_losses	List yang berisi sejarah perhitungan loss terhadap dataset latih pada bagian akhir tiap epoch.
regularization	String yang menyatakan tipe regularisasi yang akan digunakan.
reg_coef	Koefisien (menyatakan alpha atau lambda, tergantung atribut regularization) yang mengatur kekuatan regularisasi loss.
Metode	
<code>__init__(self, layers_list, random_seed, learning_rate, verbose, epochs, batch_size)</code>	Konstruktor kelas FFNN, menerima list objek Layer yang akan menjadi hidden layers dan output layer dari Feed Forward Neural Network.
<code>parameters(self)</code>	Implementasi metode yang sama dari kelas Module.
<code>compile(self, loss_function, regularization, reg_coef)</code>	Tanpa perlu membuat objek baru, mendefinisikan loss function dan metode regularisasi dan koefisien regularisasinya (bila ada).
<code>fit(self, X, y, X_val, y_val)</code>	Melakukan pelatihan (memanggil metode untuk forward dan backward propagation) dari dataset X dan target y yang diberikan dan menerima validation set X_val dan y_val. Jika X_val dan y_val diberikan, maka setiap epoch akan dilakukan perhitungan loss validation set.
<code>predict(self, X)</code>	Memanggil <code>__call__</code> untuk setiap layer dengan input yang diberikan user untuk hidden layer pertama dan output dari hidden layer sebelumnya untuk hidden layer kedua sampai output layer.
<code>score(self, X_test, y_test)</code>	Menghitung akurasi dari prediksi yang dilakukan. Hanya mendukung task klasifikasi.
<code>evaluate_model(model, X_test, y_test)</code>	Serupa dengan metode score.

<code>plot_loss(model)</code>	Melakukan plot sejarah nilai training loss dan validation loss model.
<code>plot_weight_distribution(model, layer_shown)</code>	Menampilkan distribusi weight model sesuai list of integer yaitu indeks layer yang diberikan.
<code>plot_weight_gradient_distribution(model, layer_shown)</code>	Menampilkan distribusi gradien bobot model sesuai list of integer yaitu indeks layer yang diberikan.
<code>save(self, filepath)</code>	Menyimpan objek model secara persistent ke dalam file.
<code>load(self, filepath)</code>	Menginstansiasi objek model dari model yang telah disimpan secara persistent pada file.
<code>plot_network(self)</code>	Memvisualisasikan neural network. Jika melakukan hover pada neuron, akan ditampilkan bobot, bias, dan gradien saat itu.

2.1.5. Kelas ZeroInit

Deskripsi Kelas	
Kelas ini merupakan kelas yang nantinya akan digunakan untuk melakukan inisialisasi bobot terhadap <i>layer</i> yang ada. ZeroInit menggunakan strategi untuk mengisi semua bobot dan juga bias dengan nilai 0.	
Atribut	
Metode	
<code>initialize(self, layer)</code>	Metode ini digunakan untuk melakukan inisialisasi nilai 0 ke bobot dan bias yang ada di layer <i>input</i> .

2.1.6. Kelas RandomUniform

Deskripsi Kelas
Kelas ini merupakan kelas yang nantinya akan digunakan untuk melakukan inisialisasi bobot terhadap <i>layer</i> yang ada. RandomUniform menggunakan strategi untuk mengisi semua bobot dan juga bias dengan nilai <i>random</i> tapi tetap terdistribusi secara <i>uniform</i> dengan masukan berupa <i>upper_bound</i> dan <i>lower_bound</i> .

Atribut	
seed	<i>Seed</i> digunakan untuk memastikan <i>reproducibility</i> dari fungsi-fungsi <i>random</i> yang digunakan di dalam kelas ini.
lower_bound	<i>lower_bound</i> digunakan untuk menentukan angka paling kecil yang bisa <i>dirandom</i> .
upper_bound	<i>upper_bound</i> digunakan untuk menentukan angka paling besar yang bisa <i>dirandom</i> .
Metode	
initialize(self, layer)	Metode ini digunakan untuk melakukan inisialisasi nilai antara <i>lower bound</i> dengan <i>upper bound</i> secara <i>uniform</i> ke seluruh bobot dan bias yang ada di layer masukan.

2.1.7. Kelas RandomNormal

Deskripsi Kelas	
Kelas ini merupakan kelas yang nantinya akan digunakan untuk melakukan inisialisasi bobot terhadap <i>layer</i> yang ada. RandomNormal menggunakan strategi untuk mengisi semua bobot dengan nilai <i>random</i> tapi tetap terdistribusi secara <i>normal</i> dengan masukan berupa <i>mean</i> dan <i>variance</i> .	
Atribut	
seed	<i>Seed</i> digunakan untuk memastikan <i>reproducibility</i> dari fungsi-fungsi <i>random</i> yang digunakan di dalam kelas ini.
mean	Mean merupakan rata-rata dari inisialisasi bobot yang ingin digunakan untuk membentuk distribusi normal.
variance	Variance merupakan variansi dari inisialisasi bobot yang ingin digunakan untuk membentuk distribusi normal.
Metode	
initialize(self, layer)	Metode ini digunakan untuk melakukan inisialisasi nilai antara <i>lower bound</i> dengan <i>upper bound</i> secara <i>uniform</i> ke seluruh bobot dan bias yang ada di layer masukan.

2.1.8. Kelas XavierInit

Deskripsi Kelas	
Kelas ini merupakan kelas yang nantinya akan digunakan untuk melakukan inisialisasi bobot terhadap layer yang ada. Xavier (Versi MLP) menggunakan strategi untuk mengisi semua bobot dan juga bias dengan nilai sesuai rumus yang telah dijelaskan sebelumnya. Nilai bobot akan dipilih dari <i>range</i> $[+x, -x]$ dan akan didistribusikan secara uniform di seluruh network.	
Atribut	
seed	<i>Seed</i> digunakan untuk memastikan <i>reproducibility</i> dari fungsi-fungsi <i>random</i> yang digunakan di dalam kelas ini.
multiclass	<i>Multiclass</i> digunakan untuk menentukan apakah klasifikasi yang sedang digunakan adalah permasalahan <i>multiclass</i> atau <i>binary</i> (bisa dari melihat <i>activation function</i> output layer)
Metode	
initialize(self, layer)	Metode ini digunakan untuk melakukan inisialisasi bobot dengan metode Xavier di layer yang menjadi masukan.

2.1.9. Kelas Helnit

Deskripsi Kelas	
Kelas ini merupakan kelas yang nantinya akan digunakan untuk melakukan inisialisasi bobot terhadap layer yang ada. He menggunakan strategi untuk mengisi semua bobot dan juga bias dengan nilai sesuai rumus yang telah dijelaskan sebelumnya.	
Atribut	
seed	<i>Seed</i> digunakan untuk memastikan <i>reproducibility</i> dari fungsi-fungsi <i>random</i> yang digunakan di dalam kelas ini.
Metode	
initialize(self, layer)	Metode ini digunakan untuk melakukan inisialisasi bobot dengan metode He di layer yang menjadi masukan.

2.1.10. Kelas LossFunction

Deskripsi Kelas	
Kelas umum yang harus diimplementasikan oleh kelas yang mengimplementasikan fungsi-fungsi loss yang akan digunakan.	
Metode	
<code>__call__(self, y_pred, y_true)</code>	Secara internal metode ini memanggil metode forward.
<code>forward(self, y_pred, y_true)</code>	Metode ini akan raise <code>NotImplementedError</code>

2.1.11. Kelas MSELoss

Deskripsi Kelas	
Kelas ini mengimplementasikan Mean Squared Error Loss untuk permasalahan multi label yang setiap labelnya adalah binary classification, single label binary classification, dan regression.	
Metode	
<code>forward(self, y_pred, y_true)</code>	Menghitung rata-rata dari selisih kuadrat label prediksi dan label sebenarnya. Menerima input dengan dimensi (batch_size,), (batch_size, 1), atau (batch_size, n_label).

2.1.12. Kelas BinaryCrossEntropyLoss

Deskripsi Kelas	
Kelas ini mengimplementasikan Binary Cross Entropy Loss untuk permasalahan multi label yang setiap labelnya adalah binary classification dan single label binary classification.	
Metode	
<code>forward(self, y_pred, y_true)</code>	Menghitung cross entropy dan menghitung rata-ratanya, baik terhadap seluruh baris atau terhadap seluruh label. Menerima input dengan dimensi (batch_size,), (batch_size, 1), atau (batch_size, n_label).

2.1.13. Kelas CategoricalCrossEntropyLoss

Deskripsi Kelas

Kelas ini mengimplementasikan Categorical Cross Entropy Loss untuk permasalahan multiclass classification.	
Metode	
forward(self, y_pred, y_true)	Menghitung categorical cross entropy dan menghitung rata-ratanya terhadap semua baris data.

2.2. Penjelasan Forward Propagation

Secara umum, forward propagation untuk dilakukan dengan sekuens perhitungan sebagai berikut

1. Input layer menerima instans data X (batch_size, n_features)
2. Untuk setiap layer,
 - a. Hitung nilai $Z = X @ W.T + b$, dengan W dan b adalah bobot dan bias layer dengan dimensi(output_size, input_size) dan (1, output_size)
 - b. Jika layer memiliki fungsi aktivasi f, maka layer akan mengeluarkan output f(Z).
 - c. Jika bukan layer terakhir, jadikan nilai f(Z) sebagai input layer setelahnya
3. Hasil forward propagation adalah output (batch_size, n_label) dari layer terakhir.

Perhitungan ini melibatkan beberapa kelas, yaitu:

1. Kelas FFNN yang menginisiasi forward propagation dan mengirim output suatu layer ke layer lain
2. Kelas Layer yang memanggil perhitungan fungsi aktivasi
3. Kelas Value yang melakukan perhitungan fungsi aktivasi

2.3. Penjelasan Backward Propagation dan Weight Update

Implementasi backward propagation dilakukan pada kelas Value pada modul autodiff. Backward propagation dilakukan setiap kali pemrosesan forward propagation suatu batch selesai dilakukan. Tahapan backward propagation dan weight update adalah sebagai berikut:

1. Hitung nilai loss dari nilai prediksi hasil forward propagation dan true label. Nilai loss bisa saja memiliki regularization term.
2. Panggil metode backward() terhadap nilai loss yang merupakan objek Value. Di dalam metode backward(),
 - a. Bangun topological graph dari semua operasi.
 - b. Untuk setiap objek Value elemen topological graph, dari elemen paling belakang
 - i. Panggil metode _backward() yang disimpan pada atribut _backward. Metode ini akan mengisi nilai atribut grad dari child node objek pemanggil _backward() dengan memanfaatkan aturan rantai, secara spesifik menambahkan nilai gradien child node dengan nilai

- vector-Jacobian product.
3. Update nilai bobot/bias sebesar negatif gradien dengan proporsi learning rate.

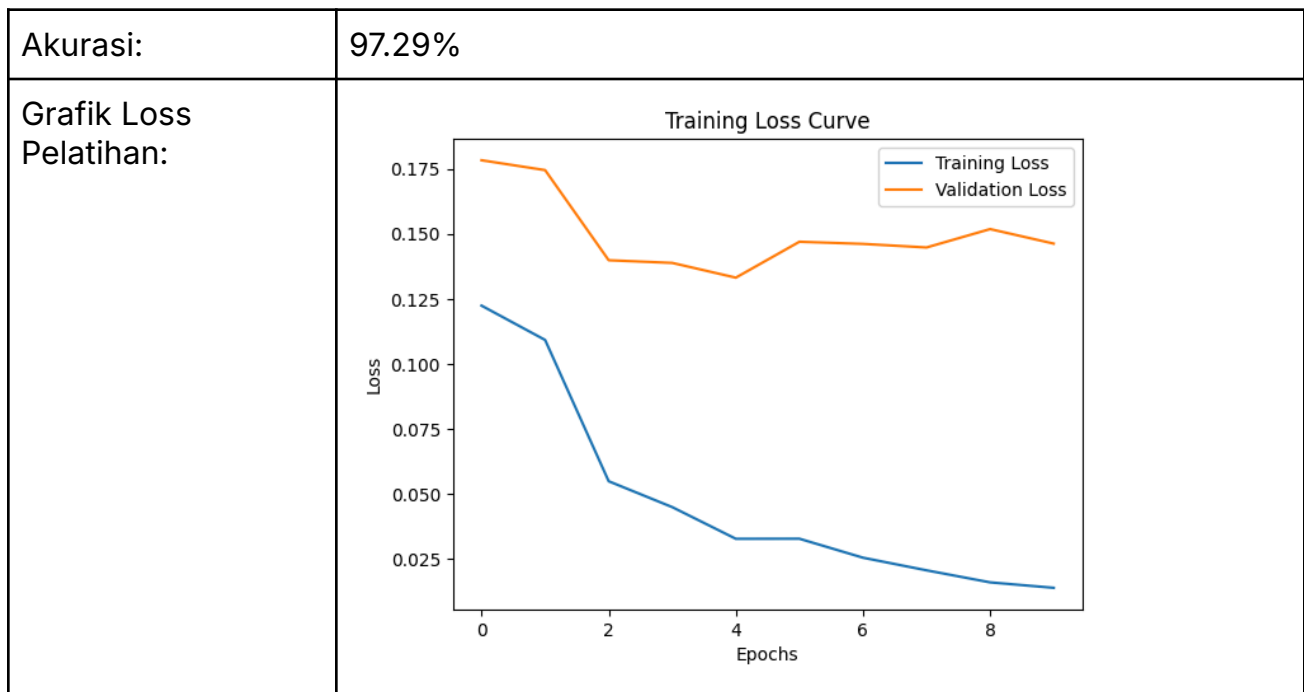
2.4. Hasil Pengujian

2.4.1. Pengaruh *Depth* dan *Width*

Pada pengujian ini, fungsi aktivasi yang digunakan adalah ReLU (kecuali output layer) dengan metode inisialisasi Xavier dan epoch 10.

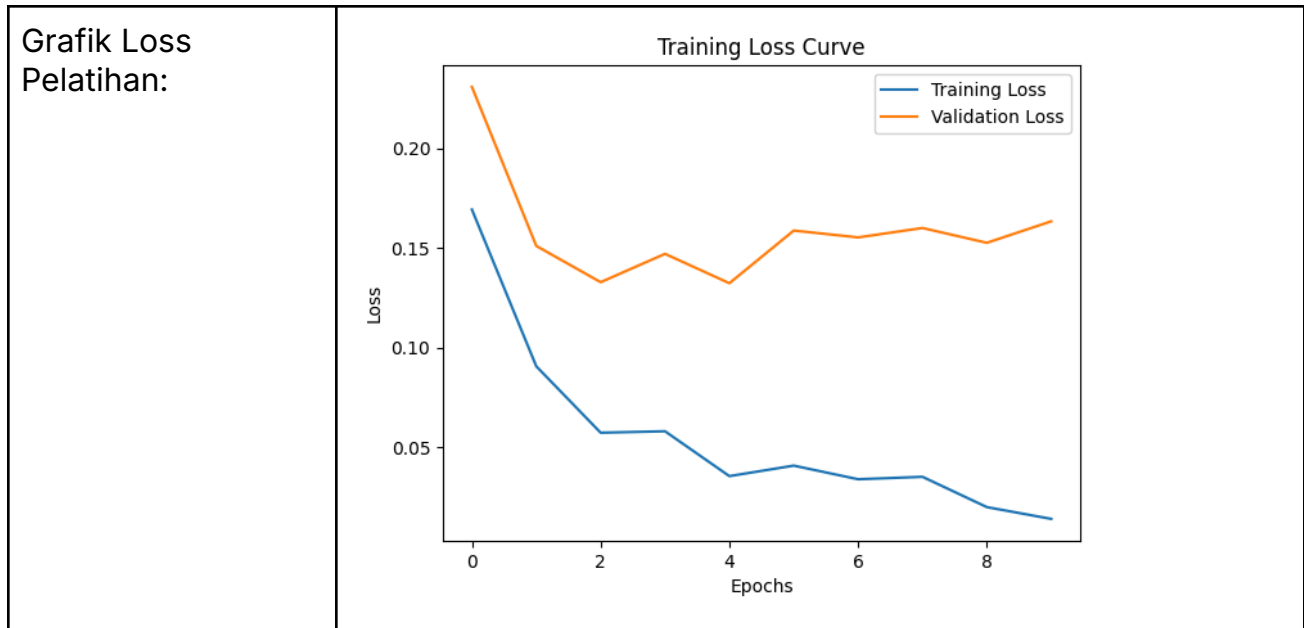
2.4.1.1. Depth sebagai variabel bebas, width konstan = 100 neuron tiap layer

2.4.1.1.1. Jumlah hidden layer = 2

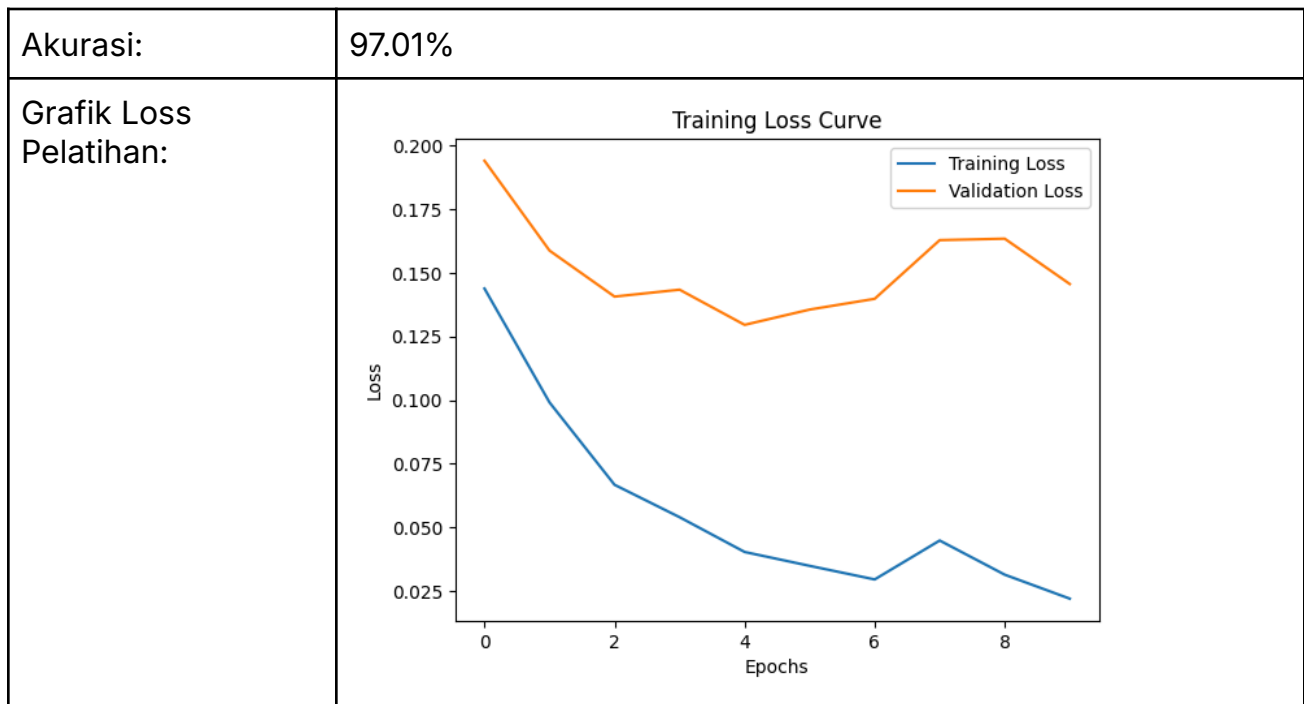


2.4.1.1.2. Jumlah hidden layer = 3

Akurasi:	97.06%
----------	--------

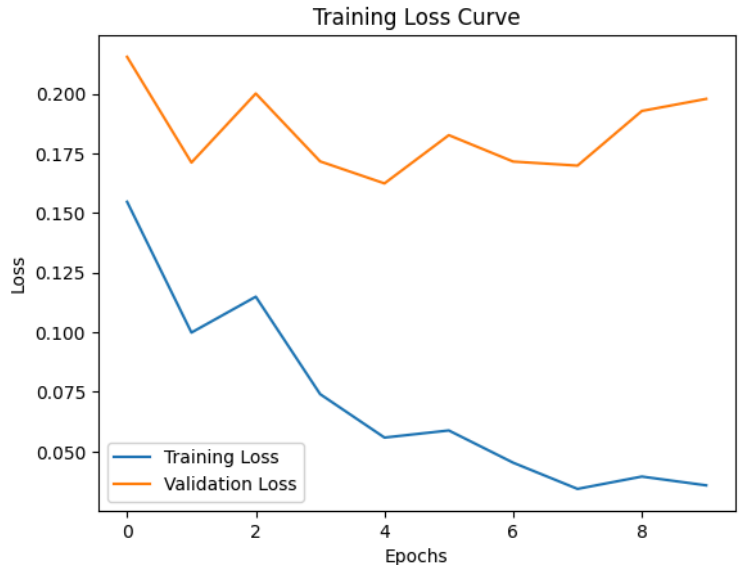


2.4.1.1.3. Jumlah hidden layer = 4

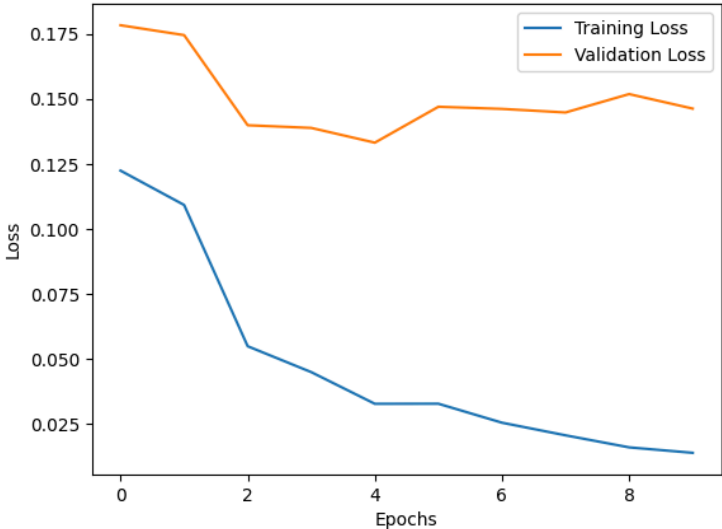


2.4.1.2. Width sebagai variabel bebas, depth konstan = 2 hidden layer

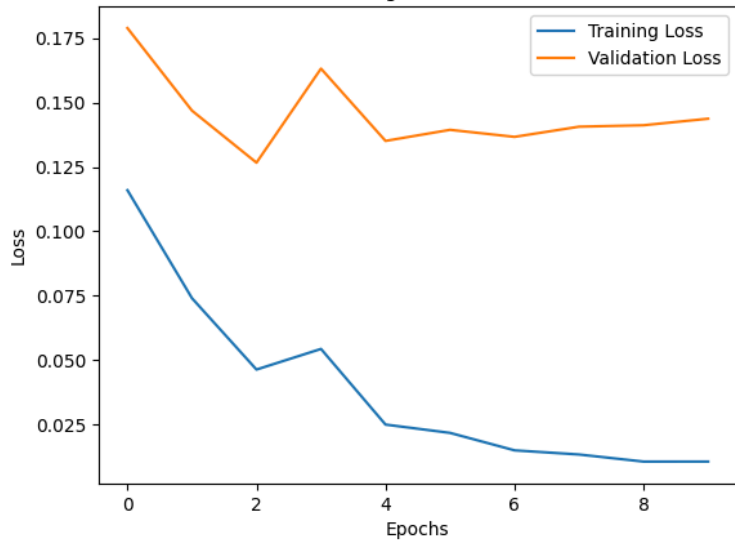
2.4.1.2.1. Jumlah neuron tiap layer = 50

Akurasi:	96.39%																																	
Grafik Loss Pelatihan:	Training Loss Curve  Loss Epochs Training Loss Validation Loss <table><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.155</td><td>0.210</td></tr><tr><td>1</td><td>0.100</td><td>0.170</td></tr><tr><td>2</td><td>0.115</td><td>0.200</td></tr><tr><td>3</td><td>0.075</td><td>0.170</td></tr><tr><td>4</td><td>0.055</td><td>0.165</td></tr><tr><td>5</td><td>0.058</td><td>0.185</td></tr><tr><td>6</td><td>0.045</td><td>0.170</td></tr><tr><td>7</td><td>0.040</td><td>0.170</td></tr><tr><td>8</td><td>0.042</td><td>0.195</td></tr><tr><td>9</td><td>0.040</td><td>0.200</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.155	0.210	1	0.100	0.170	2	0.115	0.200	3	0.075	0.170	4	0.055	0.165	5	0.058	0.185	6	0.045	0.170	7	0.040	0.170	8	0.042	0.195	9	0.040	0.200
Epochs	Training Loss	Validation Loss																																
0	0.155	0.210																																
1	0.100	0.170																																
2	0.115	0.200																																
3	0.075	0.170																																
4	0.055	0.165																																
5	0.058	0.185																																
6	0.045	0.170																																
7	0.040	0.170																																
8	0.042	0.195																																
9	0.040	0.200																																

2.4.1.2.2. Jumlah neuron tiap layer = 100

Akurasi:	97.29%																																	
Grafik Loss Pelatihan:	Training Loss Curve <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.123</td><td>0.178</td></tr><tr><td>1</td><td>0.110</td><td>0.175</td></tr><tr><td>2</td><td>0.055</td><td>0.140</td></tr><tr><td>3</td><td>0.045</td><td>0.140</td></tr><tr><td>4</td><td>0.032</td><td>0.135</td></tr><tr><td>5</td><td>0.032</td><td>0.147</td></tr><tr><td>6</td><td>0.025</td><td>0.146</td></tr><tr><td>7</td><td>0.020</td><td>0.145</td></tr><tr><td>8</td><td>0.015</td><td>0.152</td></tr><tr><td>9</td><td>0.013</td><td>0.147</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.123	0.178	1	0.110	0.175	2	0.055	0.140	3	0.045	0.140	4	0.032	0.135	5	0.032	0.147	6	0.025	0.146	7	0.020	0.145	8	0.015	0.152	9	0.013	0.147
Epochs	Training Loss	Validation Loss																																
0	0.123	0.178																																
1	0.110	0.175																																
2	0.055	0.140																																
3	0.045	0.140																																
4	0.032	0.135																																
5	0.032	0.147																																
6	0.025	0.146																																
7	0.020	0.145																																
8	0.015	0.152																																
9	0.013	0.147																																

2.4.1.2.3. Jumlah neuron tiap layer = 200

Akurasi:	97.54%																																				
Grafik Loss Pelatihan:	Training Loss Curve  The graph displays the training and validation loss over 10 epochs. The training loss (blue line) starts at approximately 0.115 and decreases steadily to about 0.01. The validation loss (orange line) starts at approximately 0.175, drops to 0.125 at epoch 2, rises to 0.165 at epoch 3, and then fluctuates between 0.135 and 0.145 for the remaining epochs. <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.115</td><td>0.175</td></tr><tr><td>1</td><td>0.075</td><td>0.145</td></tr><tr><td>2</td><td>0.045</td><td>0.125</td></tr><tr><td>3</td><td>0.055</td><td>0.165</td></tr><tr><td>4</td><td>0.025</td><td>0.135</td></tr><tr><td>5</td><td>0.02</td><td>0.14</td></tr><tr><td>6</td><td>0.015</td><td>0.135</td></tr><tr><td>7</td><td>0.012</td><td>0.14</td></tr><tr><td>8</td><td>0.01</td><td>0.14</td></tr><tr><td>9</td><td>0.01</td><td>0.145</td></tr><tr><td>10</td><td>0.01</td><td>0.145</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.115	0.175	1	0.075	0.145	2	0.045	0.125	3	0.055	0.165	4	0.025	0.135	5	0.02	0.14	6	0.015	0.135	7	0.012	0.14	8	0.01	0.14	9	0.01	0.145	10	0.01	0.145
Epochs	Training Loss	Validation Loss																																			
0	0.115	0.175																																			
1	0.075	0.145																																			
2	0.045	0.125																																			
3	0.055	0.165																																			
4	0.025	0.135																																			
5	0.02	0.14																																			
6	0.015	0.135																																			
7	0.012	0.14																																			
8	0.01	0.14																																			
9	0.01	0.145																																			
10	0.01	0.145																																			

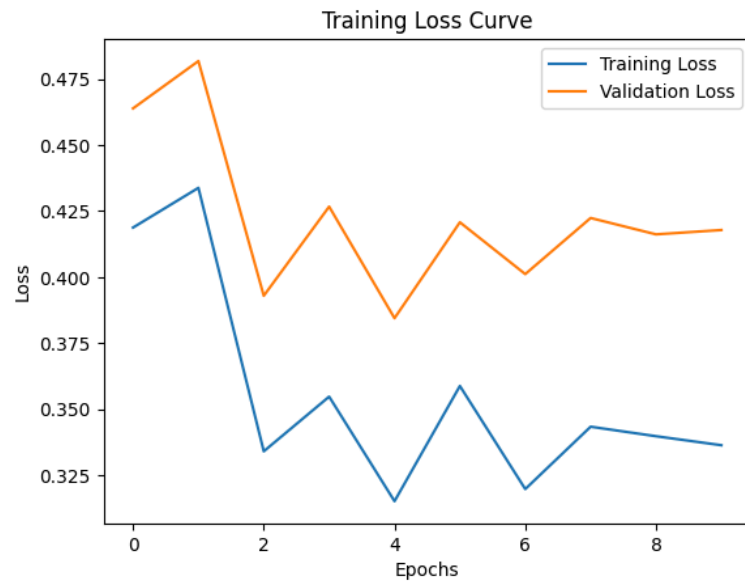
2.4.2. Pengaruh Fungsi Aktivasi

Pada pengujian ini, digunakan 2 hidden layer dengan 100 neuron tiap layer dan metode inisialisasi yaitu Xavier dengan 10 epoch. Adapula layer 3 yang merupakan output layer.

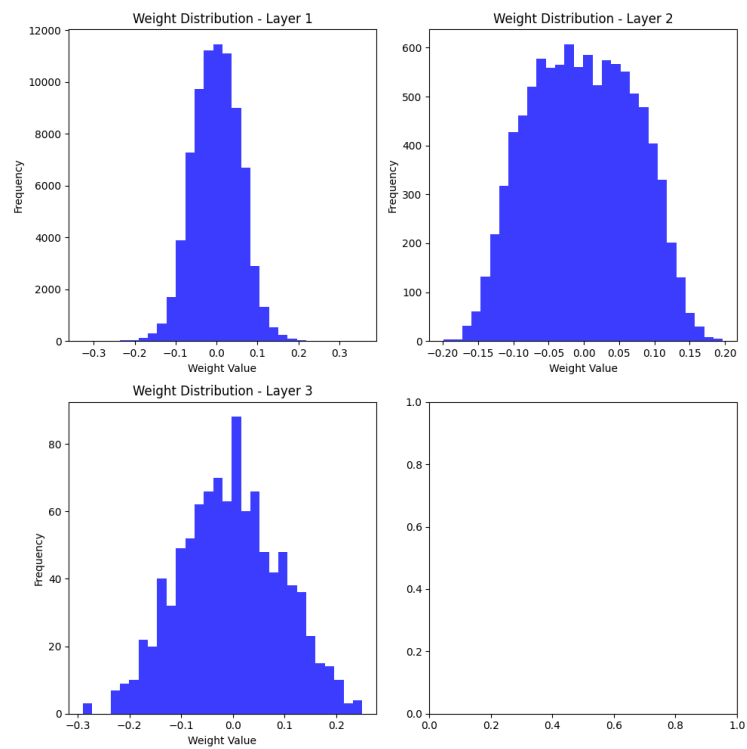
2.4.2.1. Linear Activation Function

Akurasi:	89.55%
----------	--------

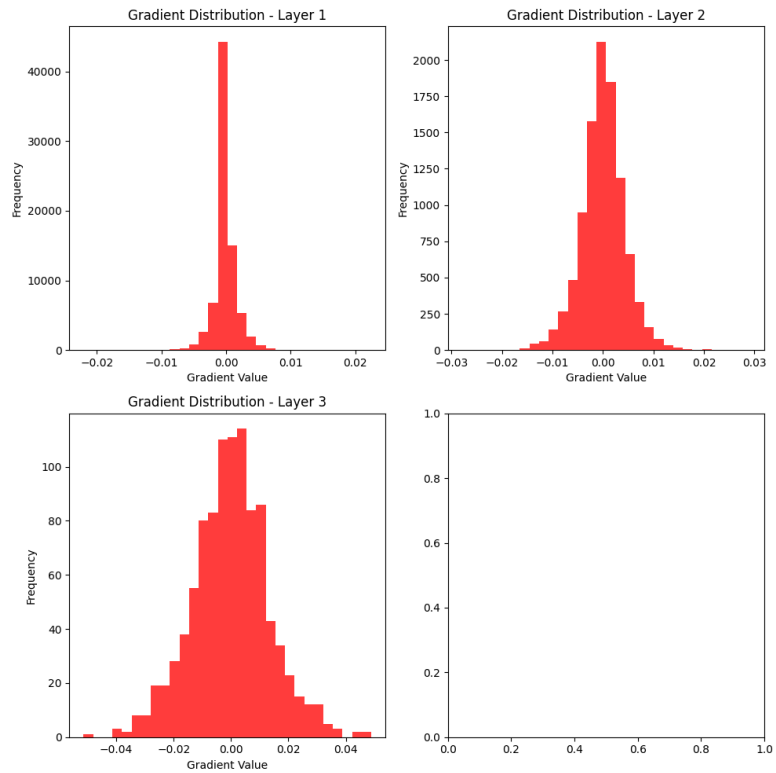
Grafik Loss Pelatihan:



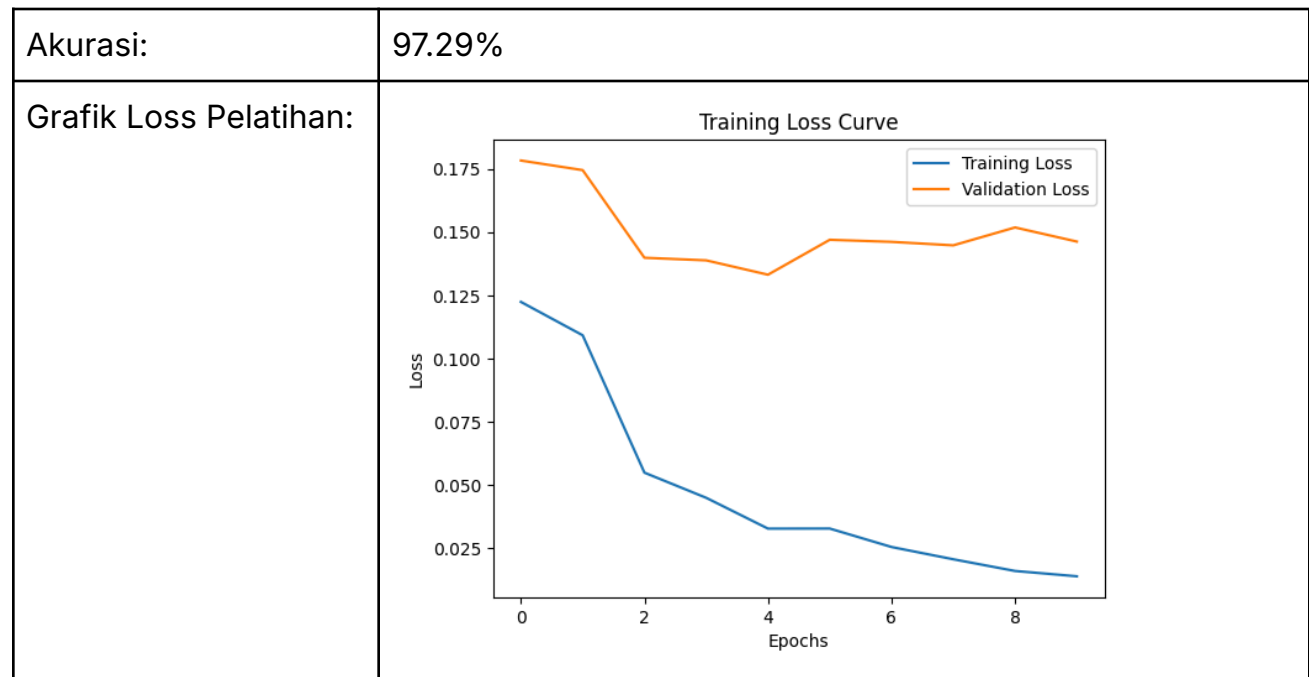
Distribusi Bobot:



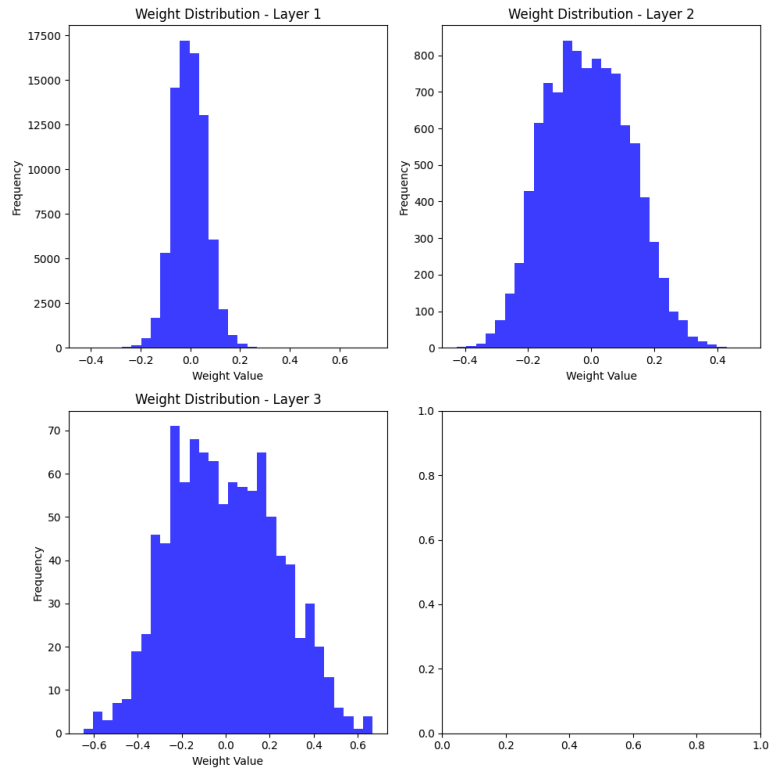
Distribusi Gradien Bobot:



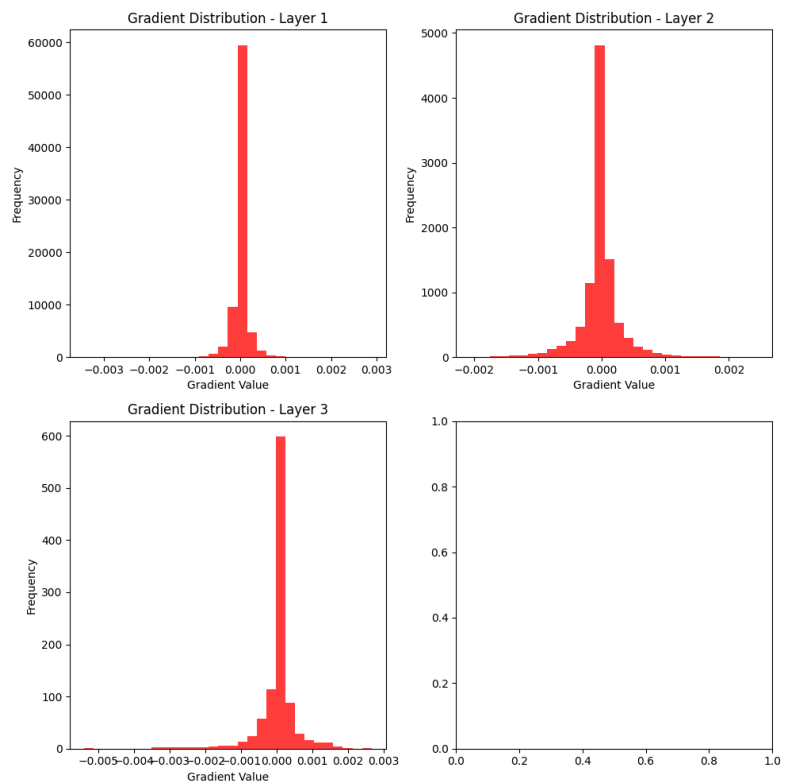
2.4.2.2. ReLU Activation Function



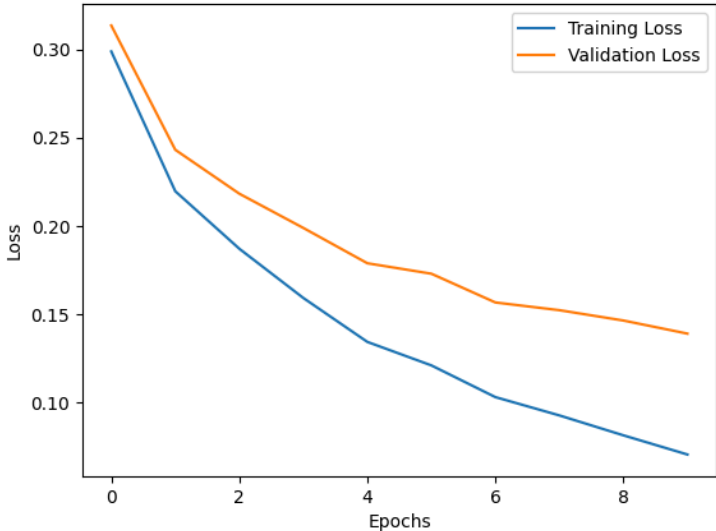
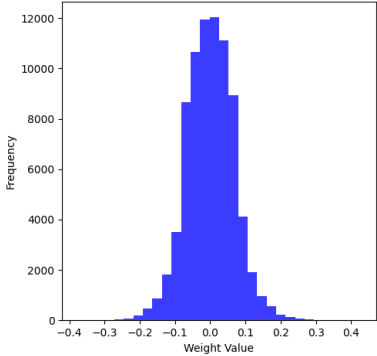
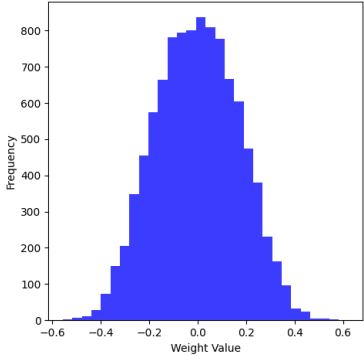
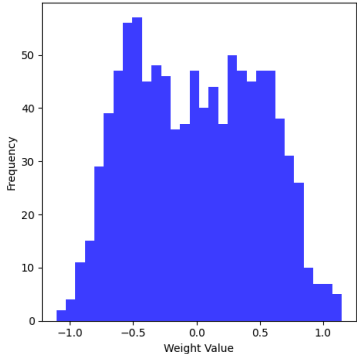
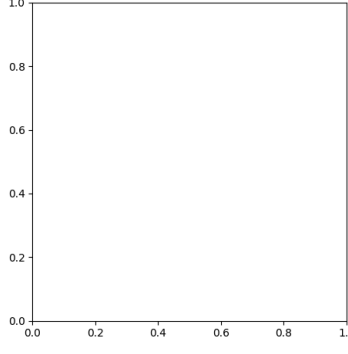
Distribusi Bobot:



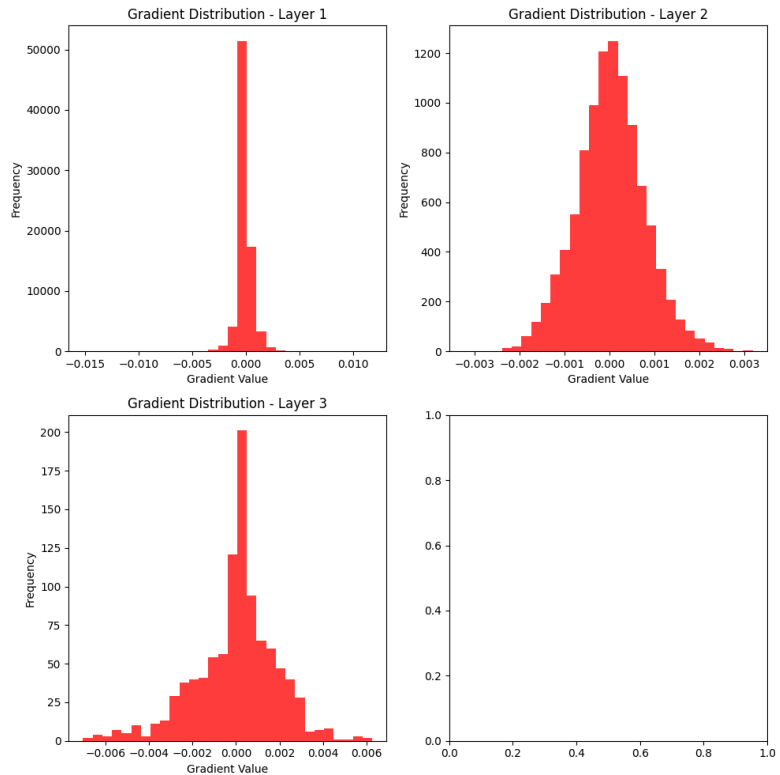
Distribusi Gradien Bobot:



2.4.2.3. Sigmoid Activation Function

Akurasi:	96.16%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.30</td><td>0.32</td></tr><tr><td>1</td><td>0.22</td><td>0.24</td></tr><tr><td>2</td><td>0.19</td><td>0.22</td></tr><tr><td>3</td><td>0.17</td><td>0.20</td></tr><tr><td>4</td><td>0.14</td><td>0.18</td></tr><tr><td>5</td><td>0.13</td><td>0.17</td></tr><tr><td>6</td><td>0.11</td><td>0.16</td></tr><tr><td>7</td><td>0.10</td><td>0.15</td></tr><tr><td>8</td><td>0.09</td><td>0.15</td></tr><tr><td>9</td><td>0.07</td><td>0.14</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.30	0.32	1	0.22	0.24	2	0.19	0.22	3	0.17	0.20	4	0.14	0.18	5	0.13	0.17	6	0.11	0.16	7	0.10	0.15	8	0.09	0.15	9	0.07	0.14
Epochs	Training Loss	Validation Loss																																
0	0.30	0.32																																
1	0.22	0.24																																
2	0.19	0.22																																
3	0.17	0.20																																
4	0.14	0.18																																
5	0.13	0.17																																
6	0.11	0.16																																
7	0.10	0.15																																
8	0.09	0.15																																
9	0.07	0.14																																
Distribusi Bobot:	Weight Distribution - Layer 1  Weight Distribution - Layer 2  Weight Distribution - Layer 3  																																	

Distribusi Gradien Bobot:

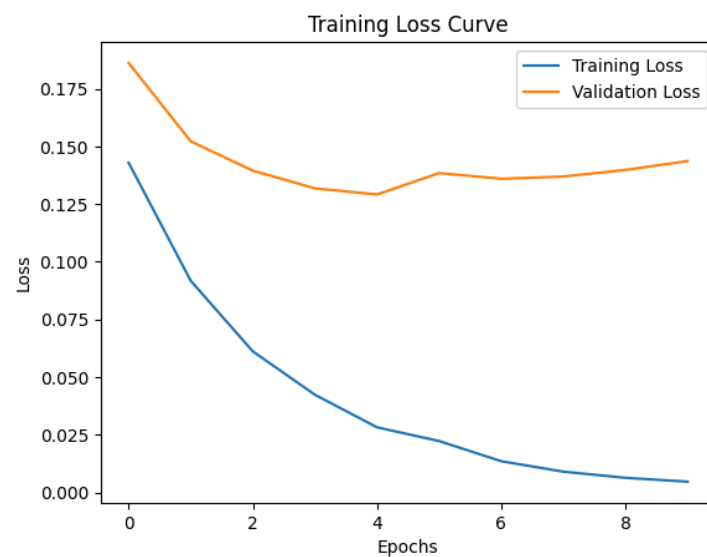


2.4.2.4. Tanh Activation Function

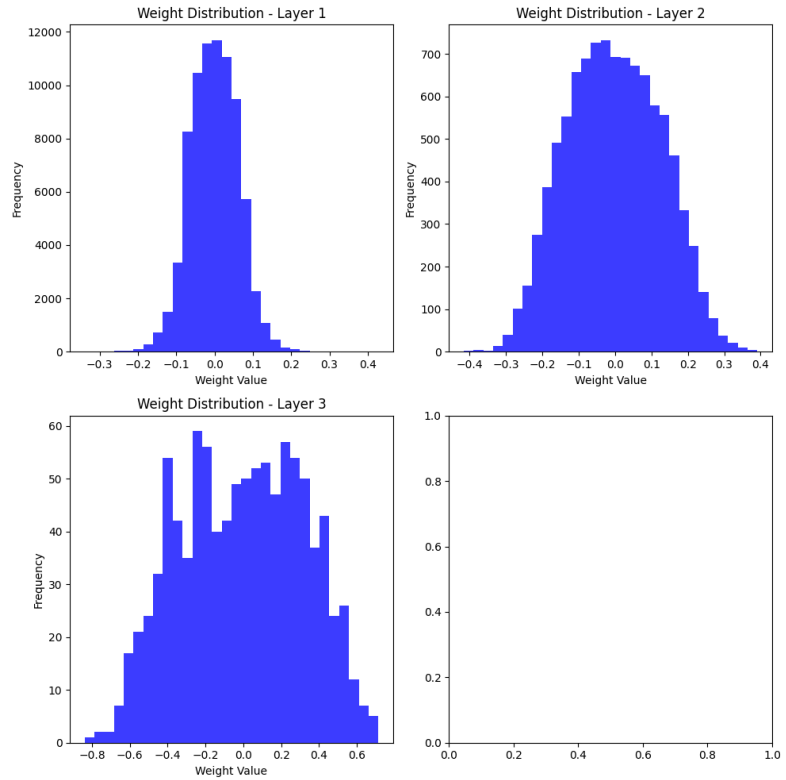
Akurasi:

96.57%

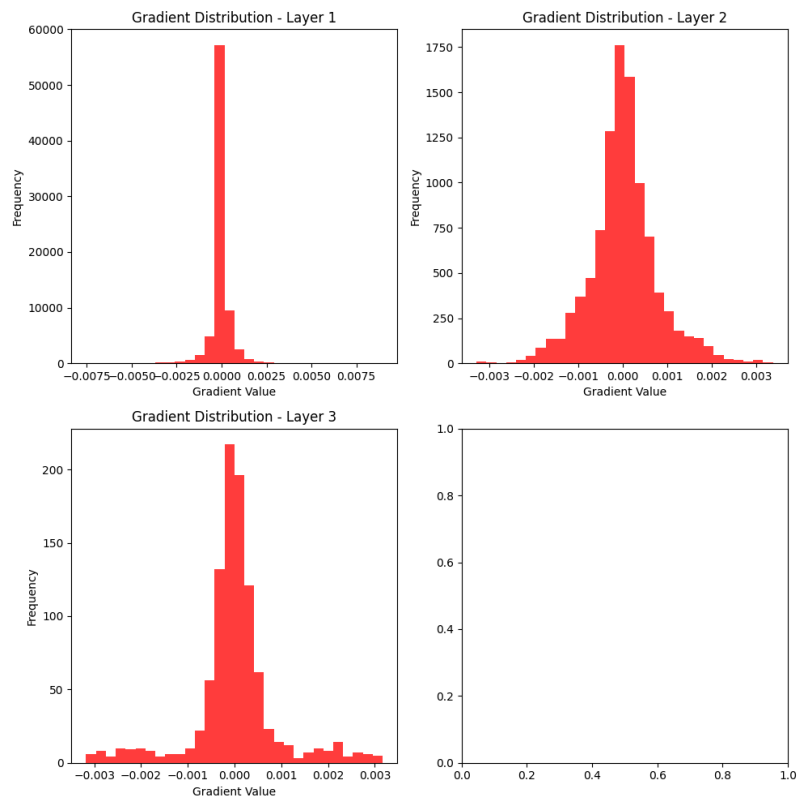
Grafik Loss Pelatihan:



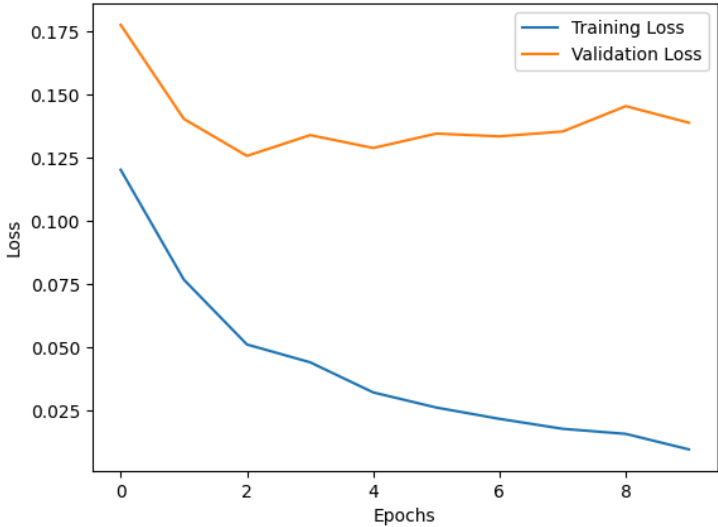
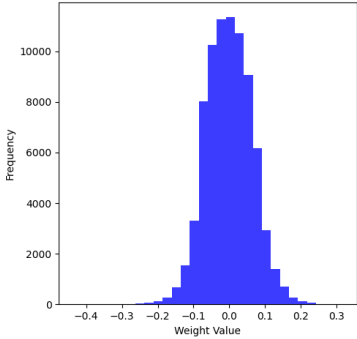
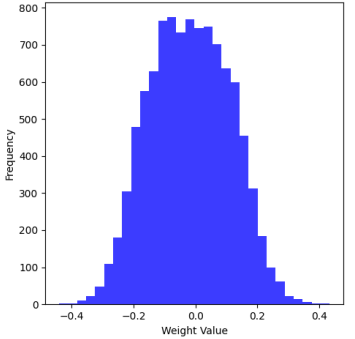
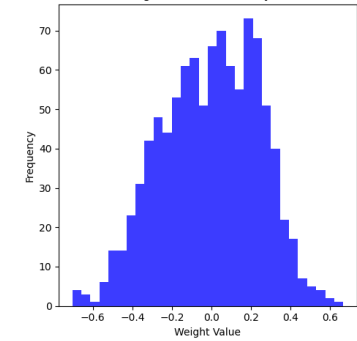
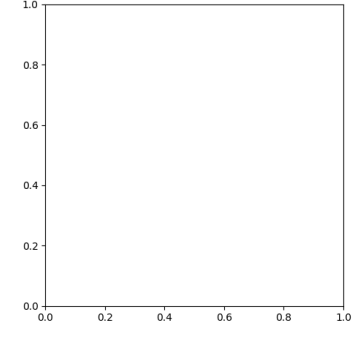
Distribusi Bobot:



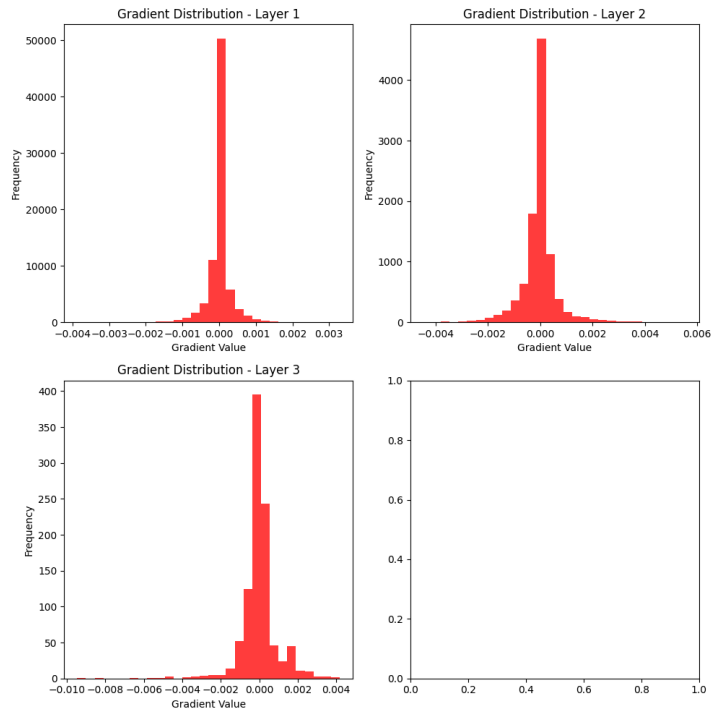
Distribusi Gradien Bobot:



2.4.2.5. ELU Activation Function

Akurasi:	96.27%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.120</td><td>0.180</td></tr><tr><td>1</td><td>0.075</td><td>0.140</td></tr><tr><td>2</td><td>0.050</td><td>0.125</td></tr><tr><td>3</td><td>0.045</td><td>0.135</td></tr><tr><td>4</td><td>0.030</td><td>0.130</td></tr><tr><td>5</td><td>0.025</td><td>0.135</td></tr><tr><td>6</td><td>0.020</td><td>0.135</td></tr><tr><td>7</td><td>0.018</td><td>0.135</td></tr><tr><td>8</td><td>0.015</td><td>0.145</td></tr><tr><td>9</td><td>0.010</td><td>0.140</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.120	0.180	1	0.075	0.140	2	0.050	0.125	3	0.045	0.135	4	0.030	0.130	5	0.025	0.135	6	0.020	0.135	7	0.018	0.135	8	0.015	0.145	9	0.010	0.140
Epochs	Training Loss	Validation Loss																																
0	0.120	0.180																																
1	0.075	0.140																																
2	0.050	0.125																																
3	0.045	0.135																																
4	0.030	0.130																																
5	0.025	0.135																																
6	0.020	0.135																																
7	0.018	0.135																																
8	0.015	0.145																																
9	0.010	0.140																																
Distribusi Bobot:	Weight Distribution - Layer 1  Weight Distribution - Layer 2  Weight Distribution - Layer 3  																																	

Distribusi Gradien Bobot:

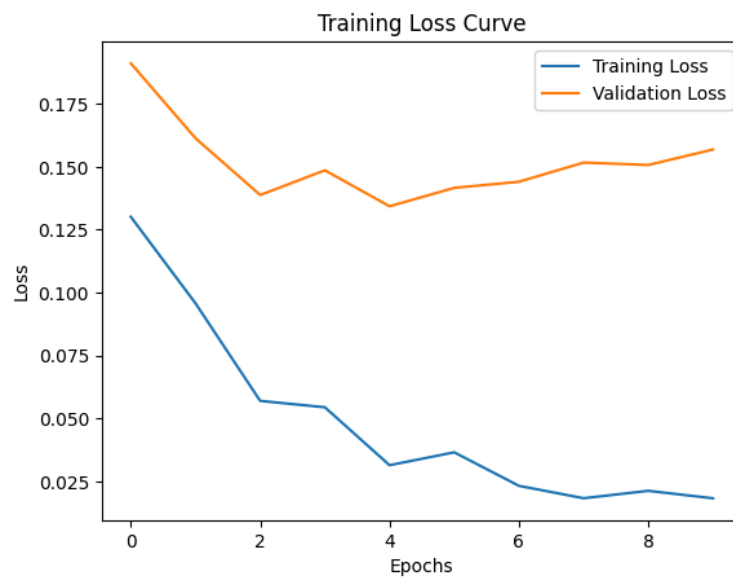


2.4.2.6. Leaky ReLU Activation Function

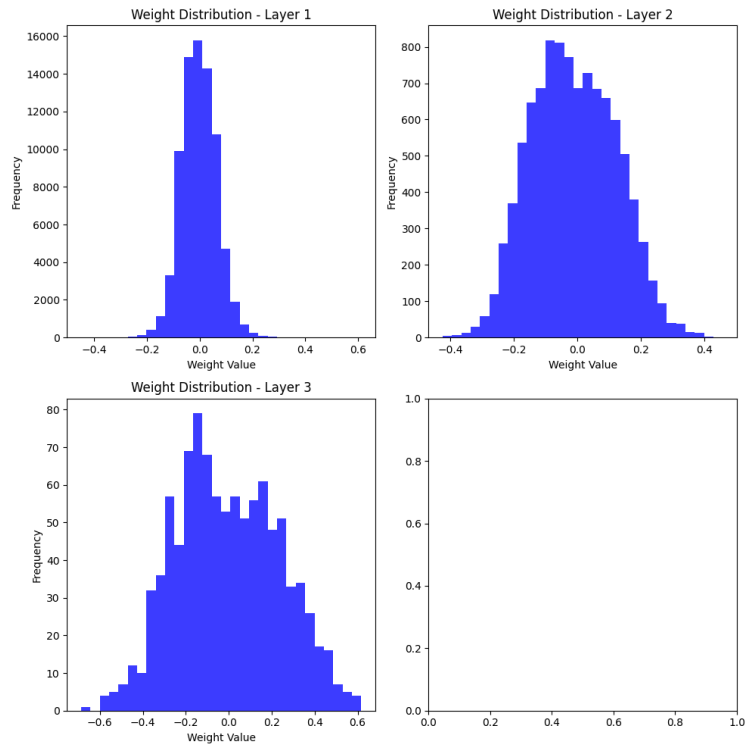
Akurasi:

97.18%

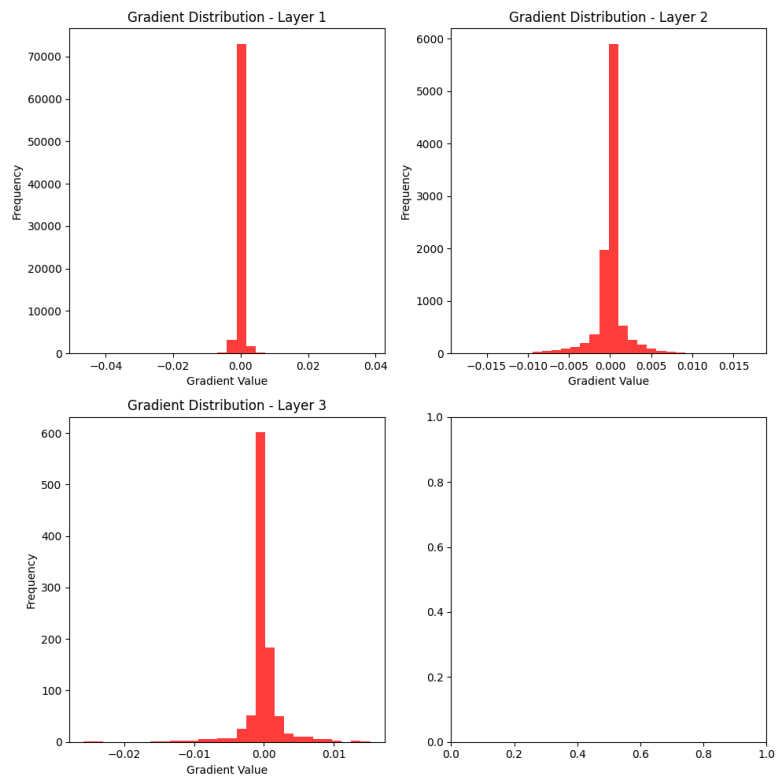
Grafik Loss Pelatihan:



Distribusi Bobot:



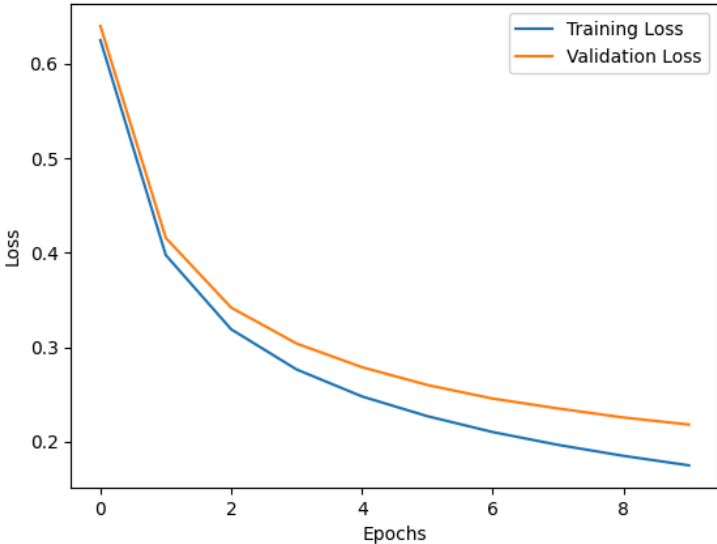
Distribusi Gradien Bobot:



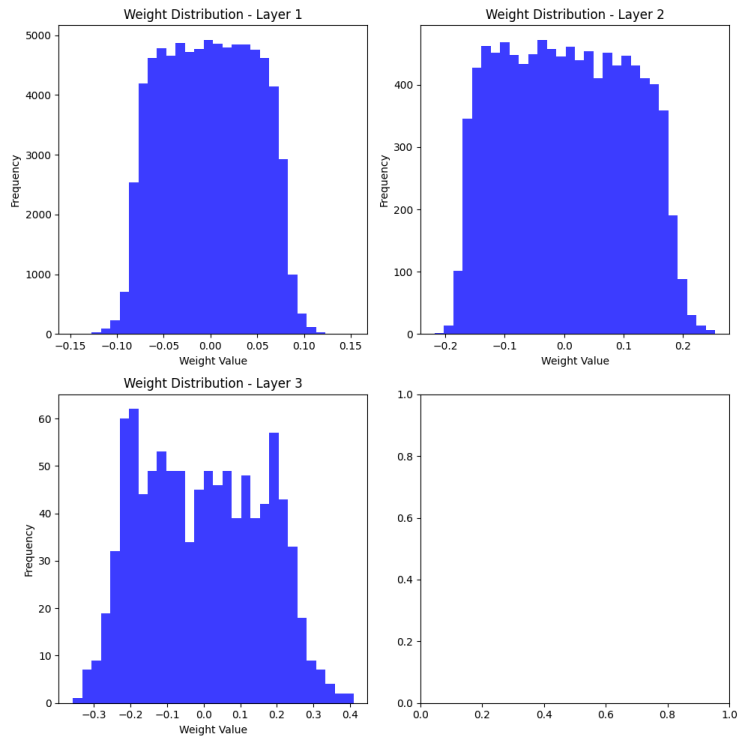
2.4.3. Pengaruh Learning Rate

Pada pengujian ini digunakan 2 hidden layer dengan 100 neuron tiap layernya dan fungsi aktivasi ReLU. Layer 3 merupakan output layer yang memiliki fungsi aktivasi soft max.

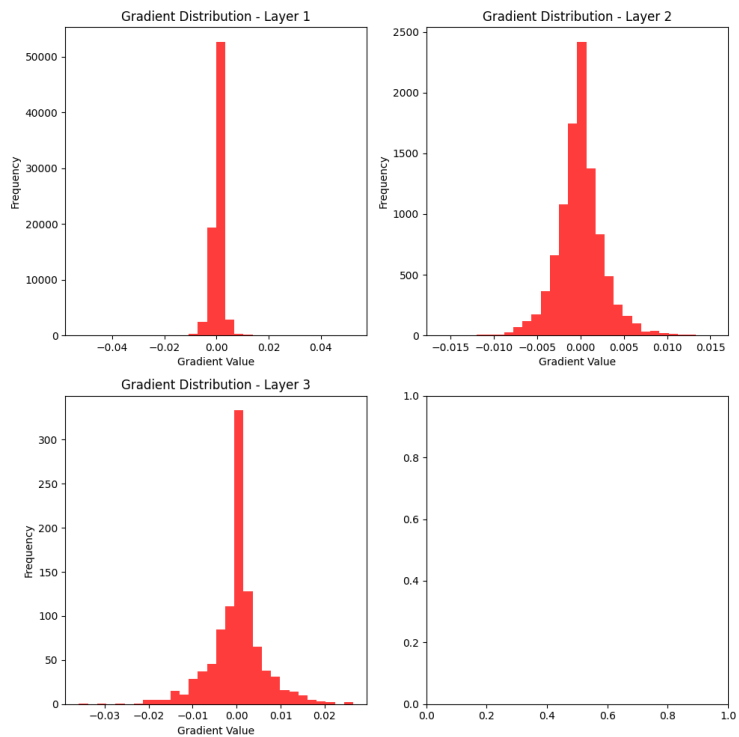
2.4.3.1. Learning rate = 0.01

Akurasi:	93.68%																																	
Grafik Loss Pelatihan:	Training Loss Curve <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.62</td><td>0.64</td></tr><tr><td>1</td><td>0.40</td><td>0.42</td></tr><tr><td>2</td><td>0.32</td><td>0.35</td></tr><tr><td>3</td><td>0.28</td><td>0.31</td></tr><tr><td>4</td><td>0.25</td><td>0.28</td></tr><tr><td>5</td><td>0.23</td><td>0.26</td></tr><tr><td>6</td><td>0.21</td><td>0.25</td></tr><tr><td>7</td><td>0.20</td><td>0.24</td></tr><tr><td>8</td><td>0.19</td><td>0.23</td></tr><tr><td>9</td><td>0.18</td><td>0.22</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.62	0.64	1	0.40	0.42	2	0.32	0.35	3	0.28	0.31	4	0.25	0.28	5	0.23	0.26	6	0.21	0.25	7	0.20	0.24	8	0.19	0.23	9	0.18	0.22
Epochs	Training Loss	Validation Loss																																
0	0.62	0.64																																
1	0.40	0.42																																
2	0.32	0.35																																
3	0.28	0.31																																
4	0.25	0.28																																
5	0.23	0.26																																
6	0.21	0.25																																
7	0.20	0.24																																
8	0.19	0.23																																
9	0.18	0.22																																

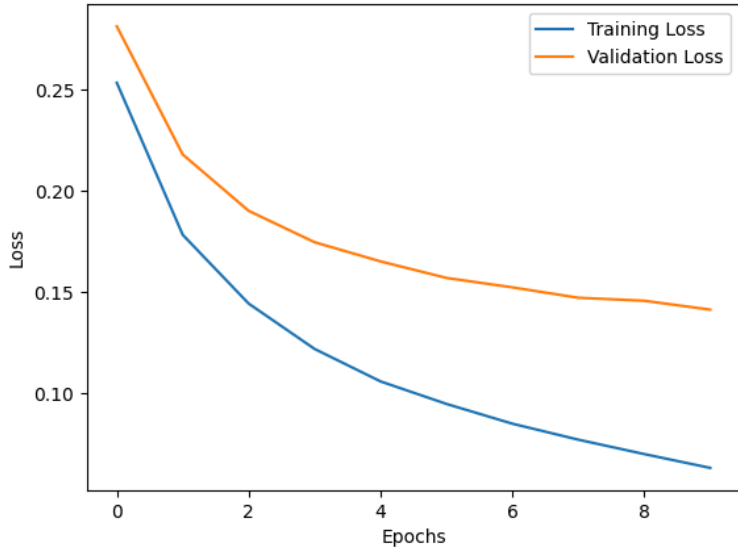
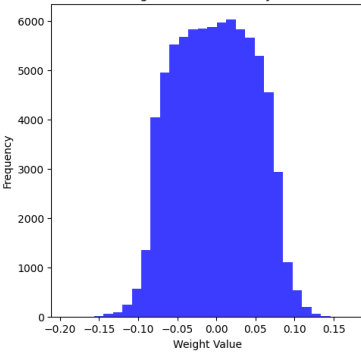
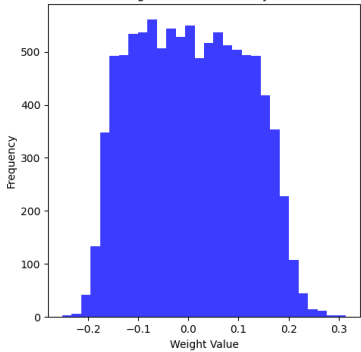
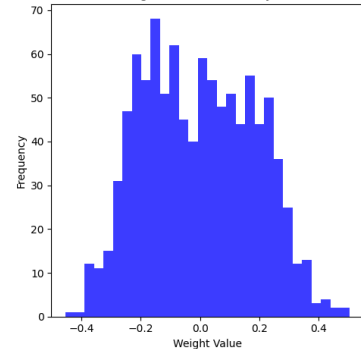
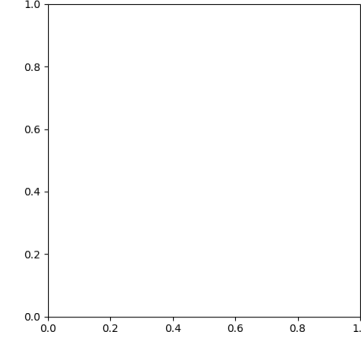
Distribusi Bobot:



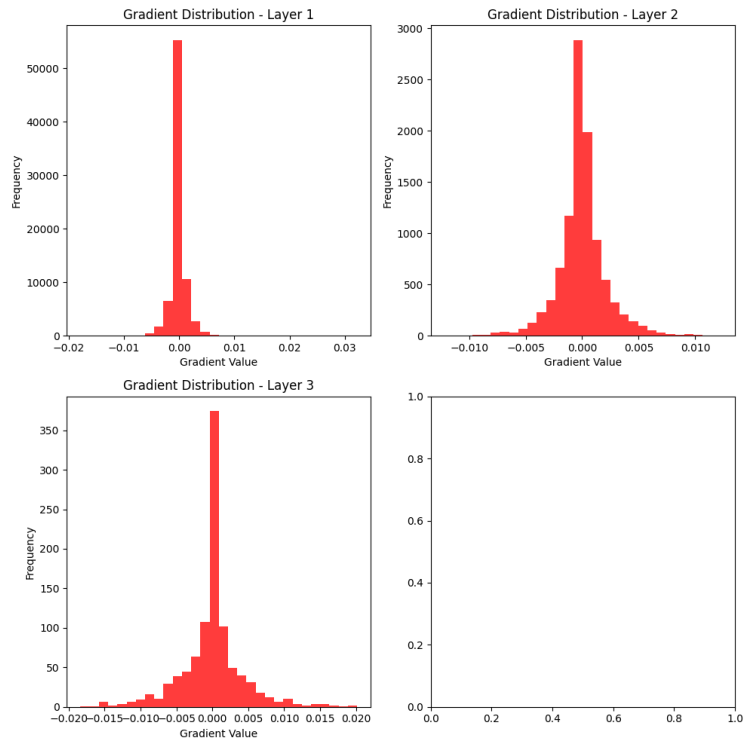
Distribusi Gradien Bobot:



2.4.3.2. Learning rate = 0.1

Akurasi:	96.30%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.26</td><td>0.28</td></tr><tr><td>1</td><td>0.18</td><td>0.22</td></tr><tr><td>2</td><td>0.15</td><td>0.19</td></tr><tr><td>3</td><td>0.13</td><td>0.18</td></tr><tr><td>4</td><td>0.11</td><td>0.17</td></tr><tr><td>5</td><td>0.10</td><td>0.16</td></tr><tr><td>6</td><td>0.09</td><td>0.15</td></tr><tr><td>7</td><td>0.08</td><td>0.15</td></tr><tr><td>8</td><td>0.07</td><td>0.14</td></tr><tr><td>9</td><td>0.07</td><td>0.14</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.26	0.28	1	0.18	0.22	2	0.15	0.19	3	0.13	0.18	4	0.11	0.17	5	0.10	0.16	6	0.09	0.15	7	0.08	0.15	8	0.07	0.14	9	0.07	0.14
Epochs	Training Loss	Validation Loss																																
0	0.26	0.28																																
1	0.18	0.22																																
2	0.15	0.19																																
3	0.13	0.18																																
4	0.11	0.17																																
5	0.10	0.16																																
6	0.09	0.15																																
7	0.08	0.15																																
8	0.07	0.14																																
9	0.07	0.14																																
Distribusi Bobot:	Weight Distribution - Layer 1  Weight Distribution - Layer 2  Weight Distribution - Layer 3  																																	

Distribusi Gradien Bobot:

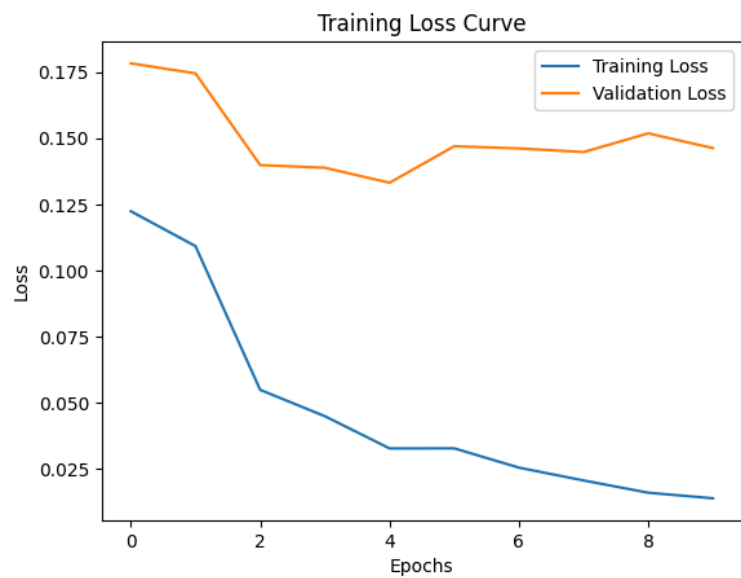


2.4.3.3. Learning rate = 0.5

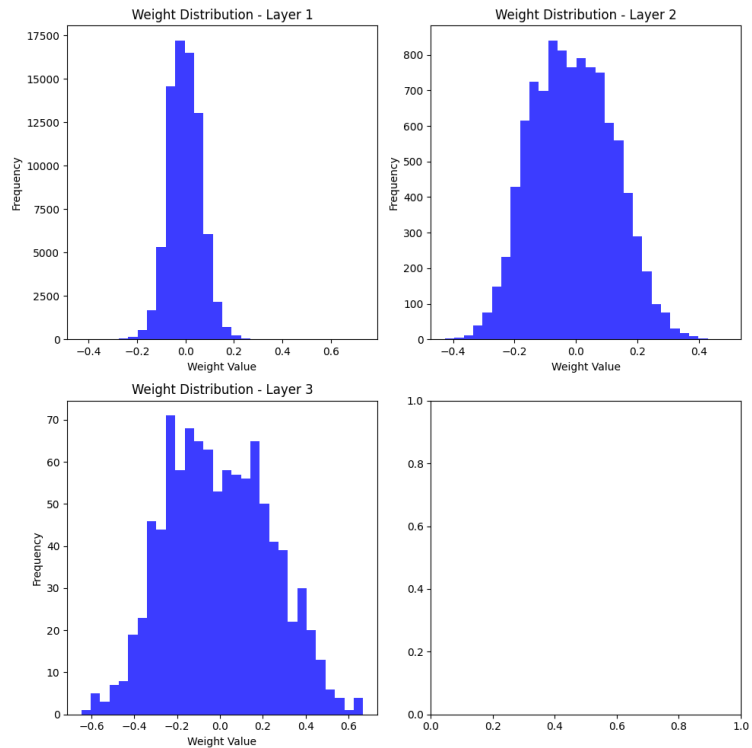
Akurasi:

97.29%

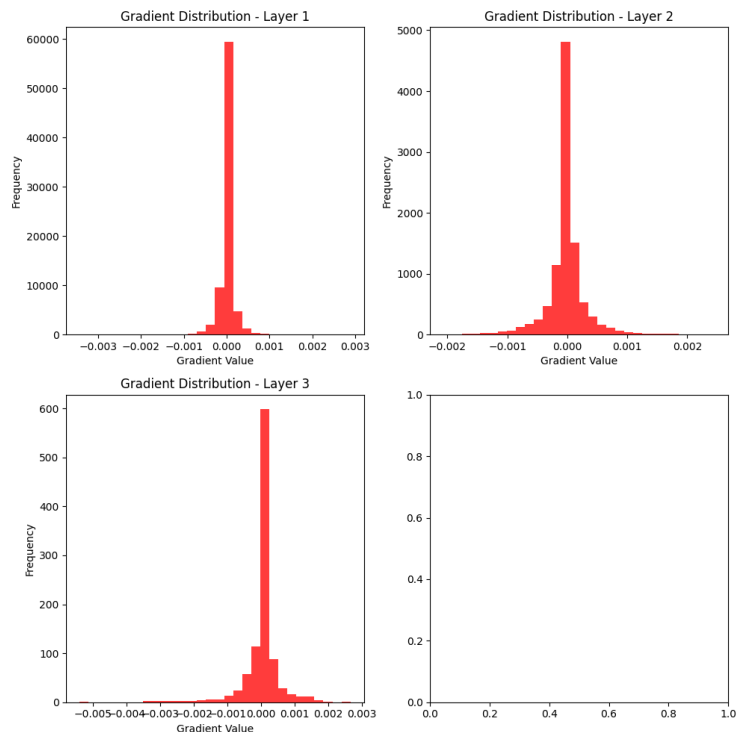
Grafik Loss Pelatihan:



Distribusi Bobot:



Distribusi Gradien Bobot:

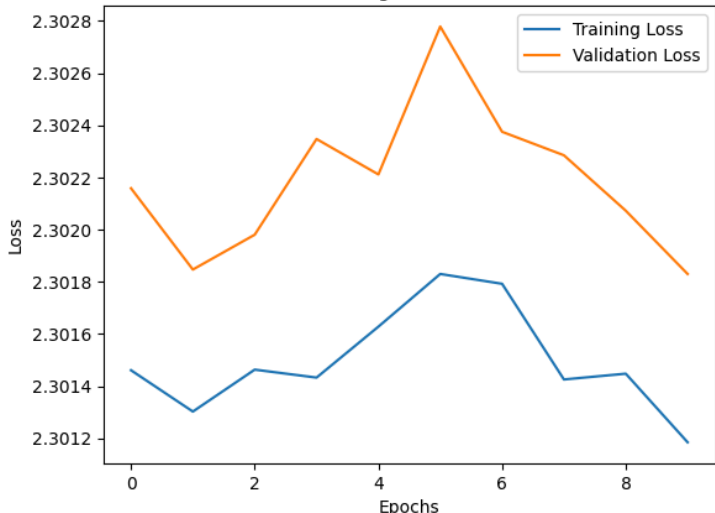
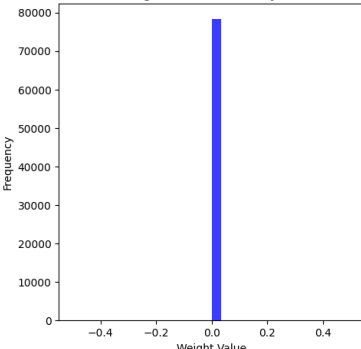
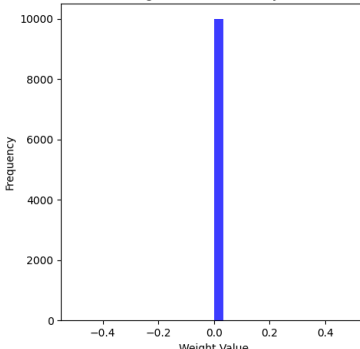
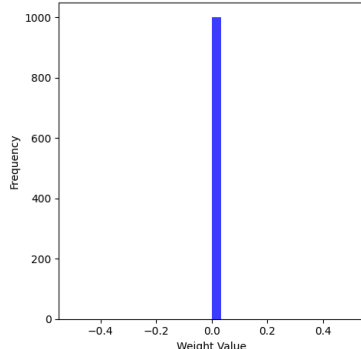
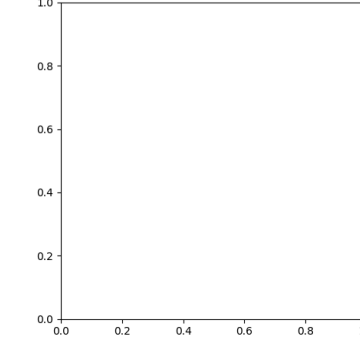


2.4.4. Pengaruh Inisialisasi Bobot

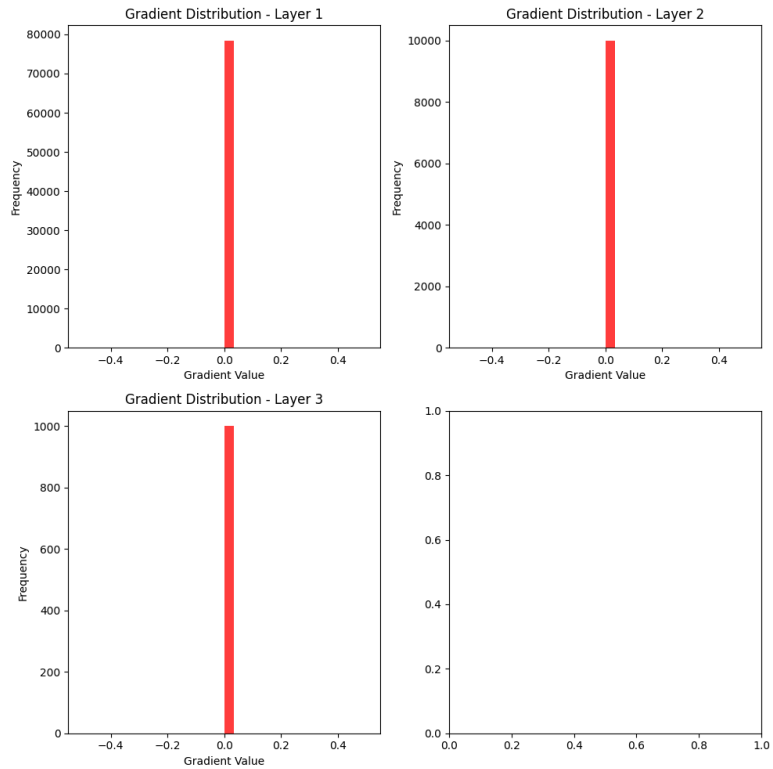
Pada pengujian ini digunakan 2 buah hidden layer dengan 100 neuron tiap layer, tiap hidden layer menggunakan ReLU sebagai fungsi aktivasi

kecuali output layer yang menggunakan soft max, learning rate yang digunakan adalah 0.5.

2.4.4.1. Inisialisasi nol

Akurasi:	11.11%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table><caption>Training Loss Curve Data (Estimated)</caption><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>2.30145</td><td>2.30215</td></tr><tr><td>1</td><td>2.30130</td><td>2.30185</td></tr><tr><td>2</td><td>2.30145</td><td>2.30200</td></tr><tr><td>3</td><td>2.30145</td><td>2.30235</td></tr><tr><td>4</td><td>2.30160</td><td>2.30220</td></tr><tr><td>5</td><td>2.30185</td><td>2.30275</td></tr><tr><td>6</td><td>2.30180</td><td>2.30235</td></tr><tr><td>7</td><td>2.30145</td><td>2.30225</td></tr><tr><td>8</td><td>2.30145</td><td>2.30205</td></tr><tr><td>9</td><td>2.30120</td><td>2.30185</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	2.30145	2.30215	1	2.30130	2.30185	2	2.30145	2.30200	3	2.30145	2.30235	4	2.30160	2.30220	5	2.30185	2.30275	6	2.30180	2.30235	7	2.30145	2.30225	8	2.30145	2.30205	9	2.30120	2.30185
Epochs	Training Loss	Validation Loss																																
0	2.30145	2.30215																																
1	2.30130	2.30185																																
2	2.30145	2.30200																																
3	2.30145	2.30235																																
4	2.30160	2.30220																																
5	2.30185	2.30275																																
6	2.30180	2.30235																																
7	2.30145	2.30225																																
8	2.30145	2.30205																																
9	2.30120	2.30185																																
Distribusi Bobot:	Weight Distribution - Layer 1  Weight Distribution - Layer 2  Weight Distribution - Layer 3  																																	

Distribusi Gradien Bobot:

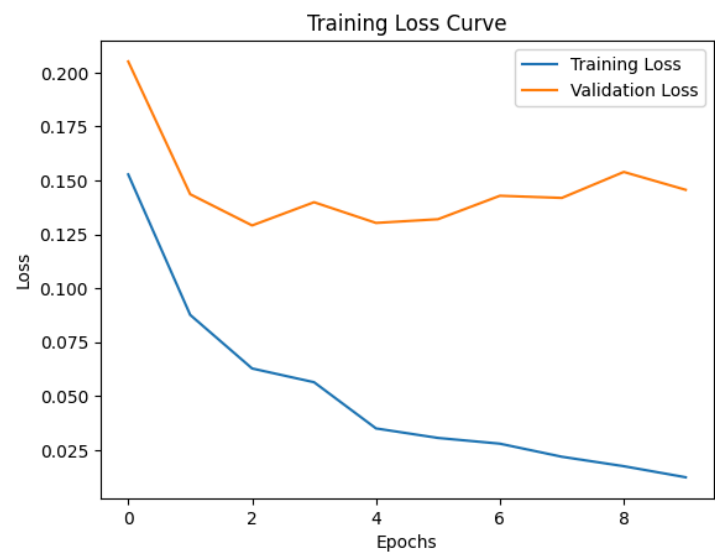


2.4.4.2. Inisialisasi acak dengan distribusi uniform

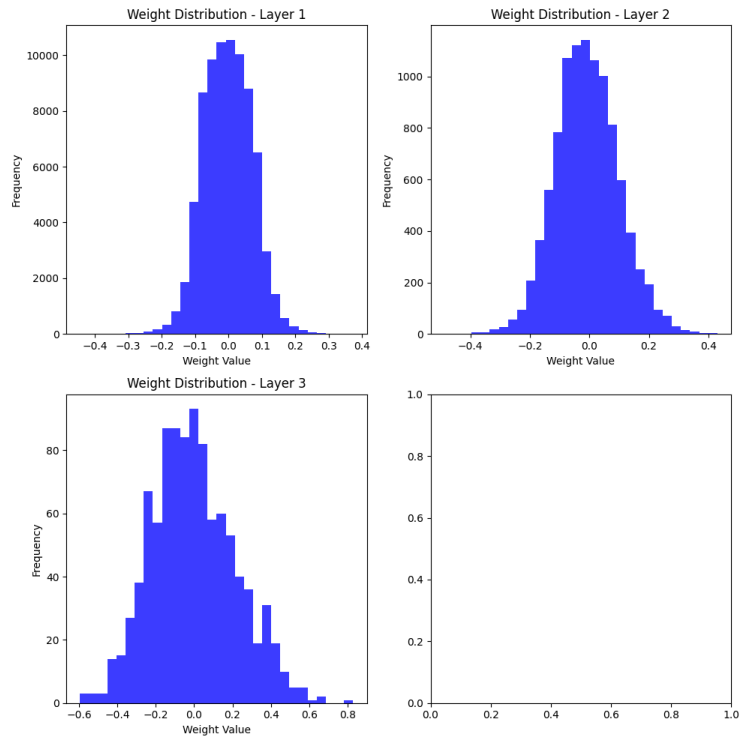
Akurasi:

97.29%

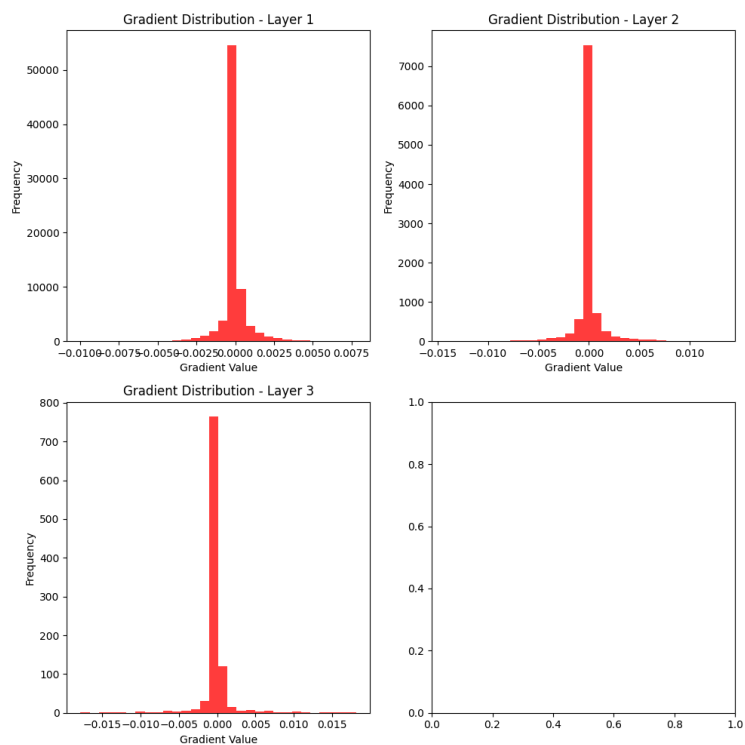
Grafik Loss Pelatihan:



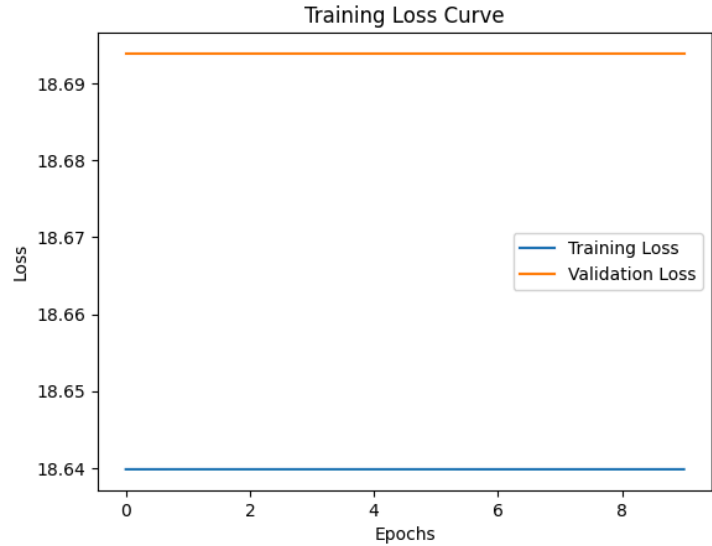
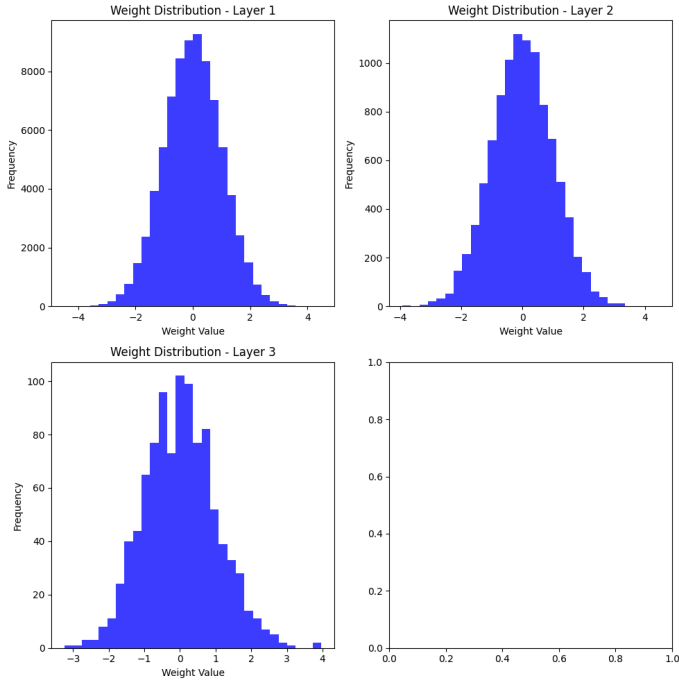
Distribusi Bobot:



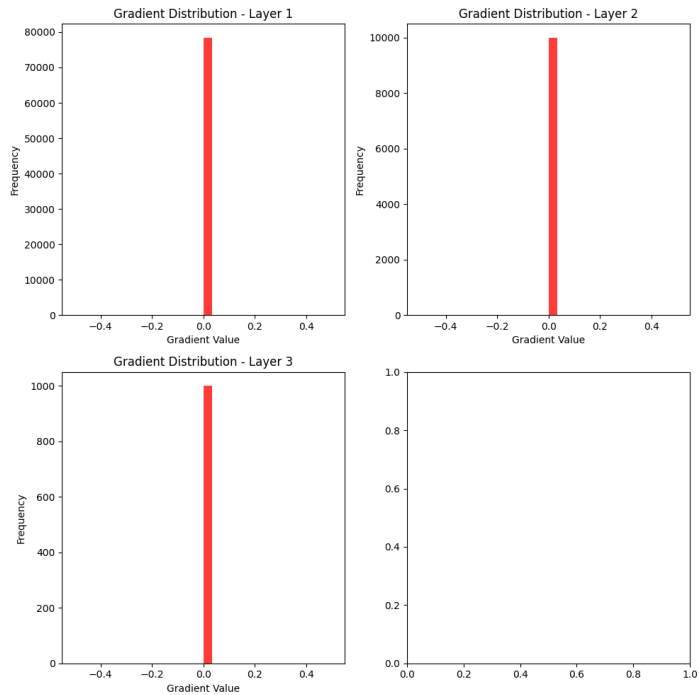
Distribusi Gradien Bobot:



2.4.4.3. Inisialisasi acak dengan distribusi normal

Akurasi:	9.79%
Grafik Loss Pelatihan:	 The Training Loss Curve displays two horizontal lines. The Training Loss (blue line) is constant at 18.64 across all 10 epochs. The Validation Loss (orange line) is constant at 18.69 across all 10 epochs. The x-axis is labeled 'Epochs' and ranges from 0 to 10. The y-axis is labeled 'Loss' and ranges from 18.64 to 18.69.
Distribusi Bobot:	 The Weight Distribution plots show the frequency of weight values for three layers. Layer 1 and Layer 2 show normal distributions centered at 0. Layer 3 shows a bimodal distribution centered at 0. A fourth plot for Layer 4 is empty. Weight Distribution - Layer 1: The x-axis is 'Weight Value' ranging from -4 to 4. The y-axis is 'Frequency' ranging from 0 to 8000. The distribution is centered at 0. Weight Distribution - Layer 2: The x-axis is 'Weight Value' ranging from -4 to 4. The y-axis is 'Frequency' ranging from 0 to 1000. The distribution is centered at 0. Weight Distribution - Layer 3: The x-axis is 'Weight Value' ranging from -3 to 4. The y-axis is 'Frequency' ranging from 0 to 100. The distribution is bimodal, centered at 0. Weight Distribution - Layer 4: The x-axis is 'Weight Value' ranging from 0.0 to 1.0. The y-axis is 'Frequency' ranging from 0.0 to 1.0. The plot is empty.

Distribusi Gradien
Bobot:

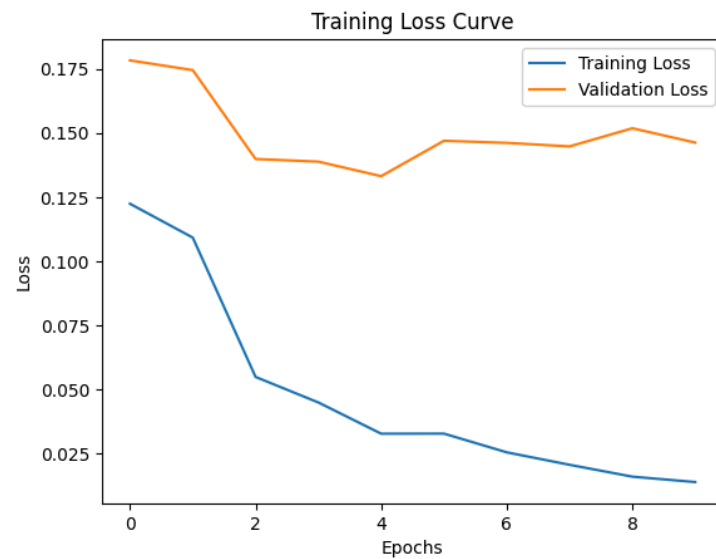


2.4.4.4. Inisialisasi Xavier

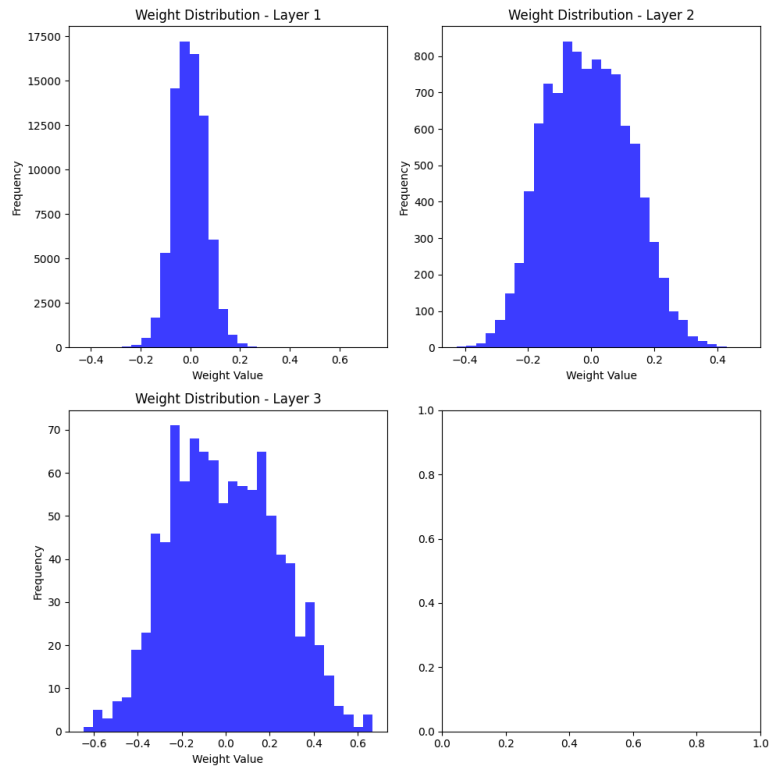
Akurasi:

97.29%

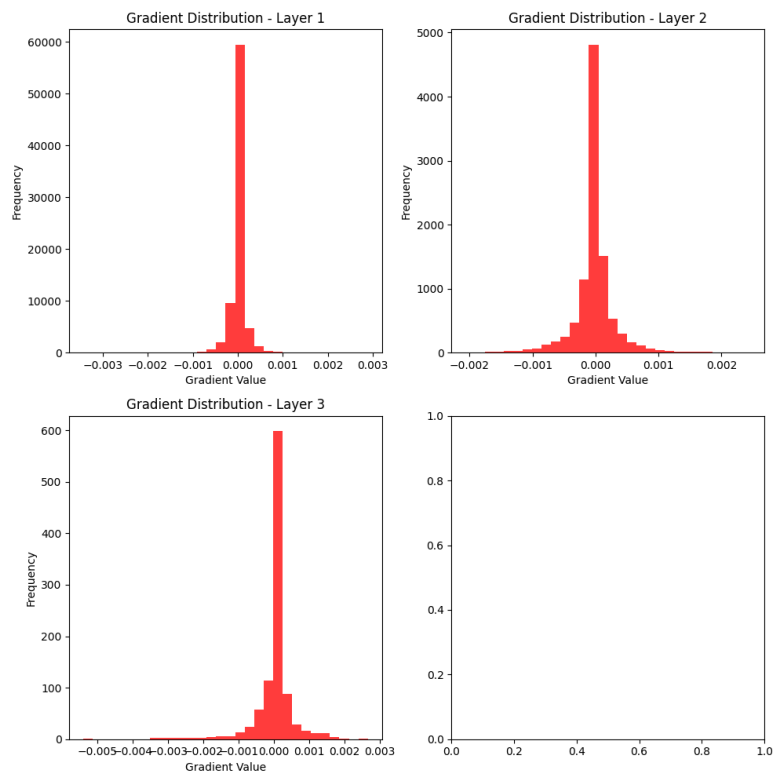
Grafik Loss Pelatihan:



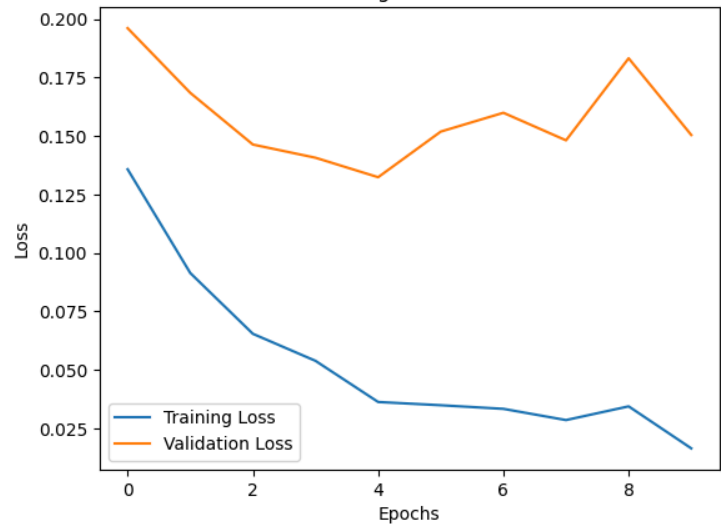
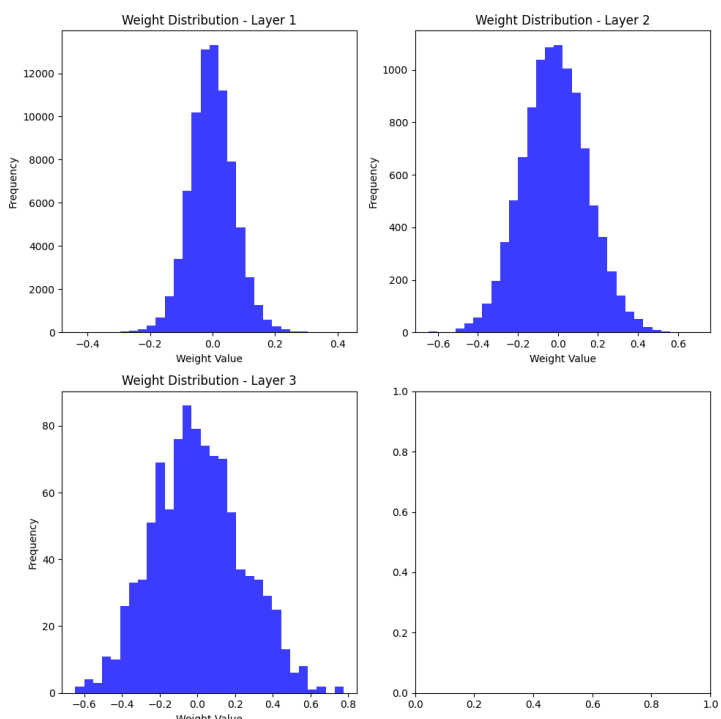
Distribusi Bobot:



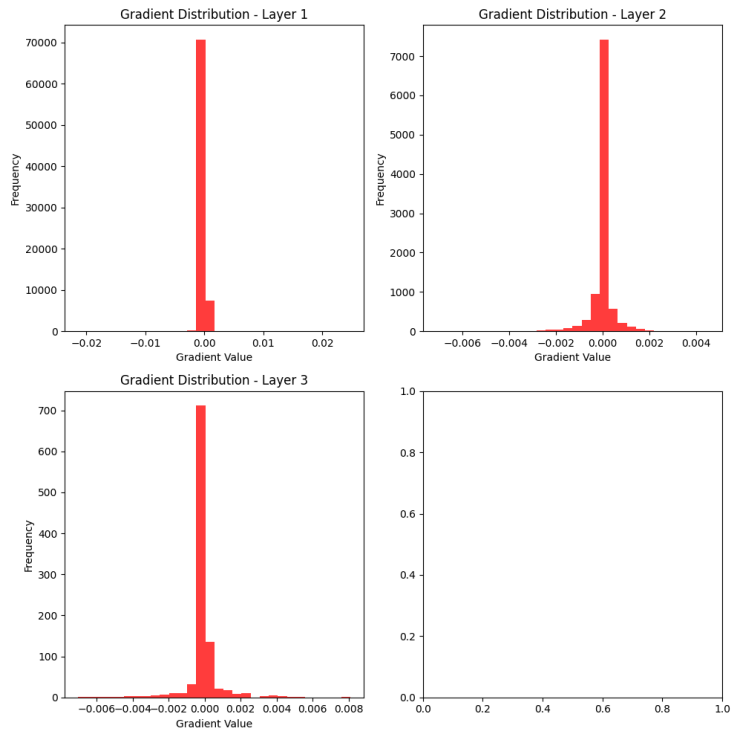
Distribusi Gradien Bobot:



2.4.4.5. Inisialisasi He

Akurasi:	97.24%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table><caption>Approximate data for Training Loss Curve</caption><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr><tr><td>0</td><td>0.135</td><td>0.195</td></tr><tr><td>1</td><td>0.090</td><td>0.165</td></tr><tr><td>2</td><td>0.065</td><td>0.145</td></tr><tr><td>3</td><td>0.055</td><td>0.140</td></tr><tr><td>4</td><td>0.035</td><td>0.130</td></tr><tr><td>5</td><td>0.030</td><td>0.155</td></tr><tr><td>6</td><td>0.028</td><td>0.160</td></tr><tr><td>7</td><td>0.025</td><td>0.145</td></tr><tr><td>8</td><td>0.030</td><td>0.185</td></tr><tr><td>9</td><td>0.015</td><td>0.150</td></tr></table>	Epochs	Training Loss	Validation Loss	0	0.135	0.195	1	0.090	0.165	2	0.065	0.145	3	0.055	0.140	4	0.035	0.130	5	0.030	0.155	6	0.028	0.160	7	0.025	0.145	8	0.030	0.185	9	0.015	0.150
Epochs	Training Loss	Validation Loss																																
0	0.135	0.195																																
1	0.090	0.165																																
2	0.065	0.145																																
3	0.055	0.140																																
4	0.035	0.130																																
5	0.030	0.155																																
6	0.028	0.160																																
7	0.025	0.145																																
8	0.030	0.185																																
9	0.015	0.150																																
Distribusi Bobot:	 <table><caption>Approximate data for Weight Distribution Histograms</caption><tr><th>Layer</th><th>Weight Value Range</th><th>Frequency Range</th></tr><tr><td>Layer 1</td><td>-0.4 to 0.4</td><td>0 to 12000</td></tr><tr><td>Layer 2</td><td>-0.6 to 0.6</td><td>0 to 1000</td></tr><tr><td>Layer 3</td><td>-0.6 to 0.8</td><td>0 to 80</td></tr><tr><td>Layer 4</td><td>0.0 to 1.0</td><td>0.0 to 1.0</td></tr></table>	Layer	Weight Value Range	Frequency Range	Layer 1	-0.4 to 0.4	0 to 12000	Layer 2	-0.6 to 0.6	0 to 1000	Layer 3	-0.6 to 0.8	0 to 80	Layer 4	0.0 to 1.0	0.0 to 1.0																		
Layer	Weight Value Range	Frequency Range																																
Layer 1	-0.4 to 0.4	0 to 12000																																
Layer 2	-0.6 to 0.6	0 to 1000																																
Layer 3	-0.6 to 0.8	0 to 80																																
Layer 4	0.0 to 1.0	0.0 to 1.0																																

Distribusi Gradien Bobot:



2.4.5. Perbandingan dengan Library sklearn

Pada perbandingan performa model dengan model dari sklearn. Data set dan *hyperparameter* yang digunakan akan sama untuk kedua model. Hal-hal yang perlu diperhatikan:

1. Random state atau seed yang diberikan untuk kedua model sama. Namun, tidak ada jaminan bahwa semua struktur internal menggunakan cara pengacakan yang sama.
2. Ditetapkan momentum dan alpha ke 0.0 agar perbandingannya adil. Nilai default akan digunakan untuk hyperparameter lainnya.
3. Menurut literasi, sklearn menggunakan softmax sebagai fungsi aktivasi output layer pada masalah multikelas, hal tersebut juga digunakan untuk model yang dibuat.
4. Ukuran batch (batch size) ditetapkan sebagai 200 untuk kedua model karena nilai tersebut merupakan nilai default pada sklearn ($\min(200, n_samples)$).
5. SKlearn menggunakan inisialisasi Xavier uniform secara default. Konfigurasi yang digunakan untuk sklearn (dicontoh juga untuk model kami) adalah sebagai berikut:



```
model = MLPClassifier(hidden_layer_sizes=(128, 64),  
activation='relu', max_iter=10, solver="sgd",  
verbose=1, random_state=42, learning_rate_init=0.5,  
momentum=0.0,  
alpha=0.0)
```

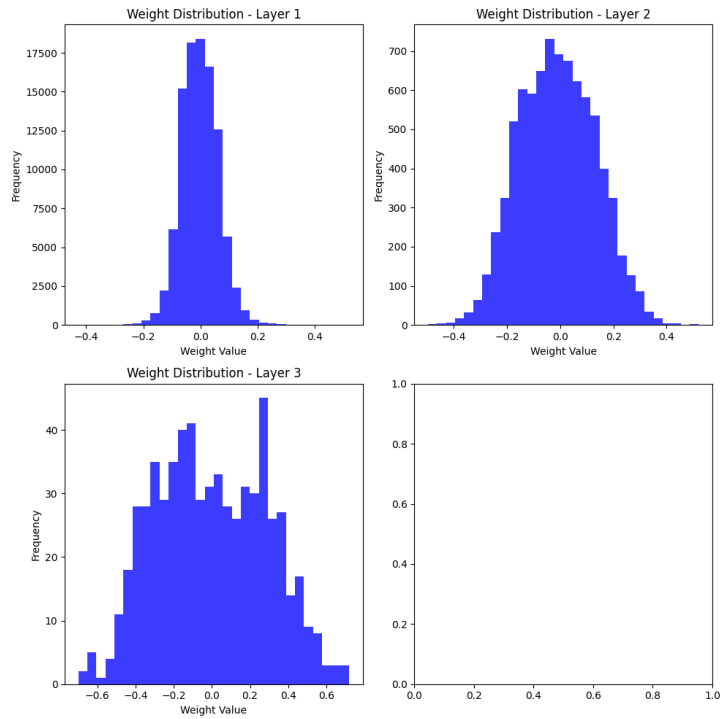
Dari hasil percobaan didapat akurasi model sklearn adalah 97.27%
sedangkan akurasi dari model yang dibuat adalah 97.15%

2.4.6. Perbandingan Regularisasi

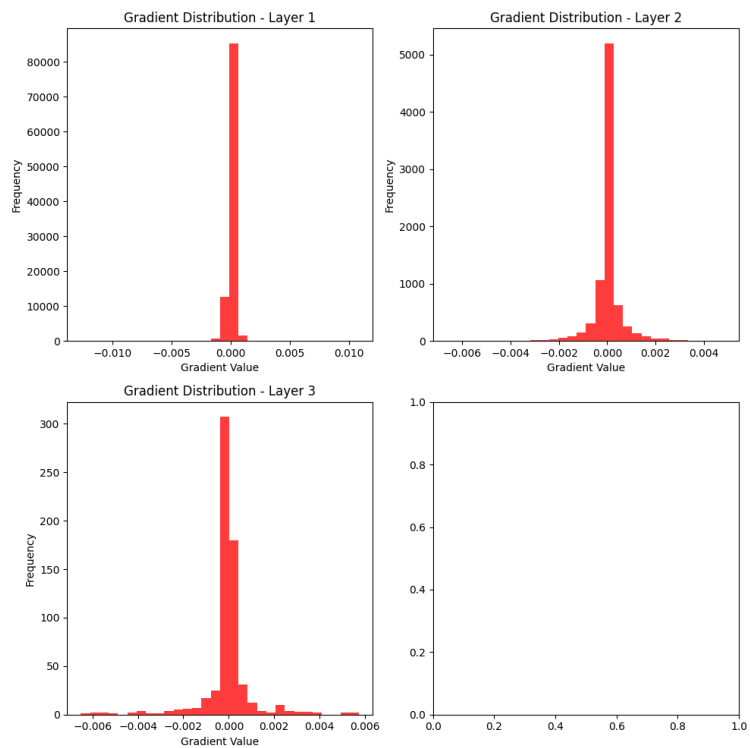
2.4.6.1. Tanpa regularisasi

Akurasi:	97.15%																																	
Grafik Loss Pelatihan:	Training Loss Curve Loss Epochs Training Loss Validation Loss <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.140</td><td>0.210</td></tr><tr><td>1</td><td>0.105</td><td>0.185</td></tr><tr><td>2</td><td>0.060</td><td>0.155</td></tr><tr><td>3</td><td>0.060</td><td>0.165</td></tr><tr><td>4</td><td>0.035</td><td>0.140</td></tr><tr><td>5</td><td>0.035</td><td>0.155</td></tr><tr><td>6</td><td>0.025</td><td>0.155</td></tr><tr><td>7</td><td>0.020</td><td>0.150</td></tr><tr><td>8</td><td>0.018</td><td>0.155</td></tr><tr><td>9</td><td>0.010</td><td>0.150</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.140	0.210	1	0.105	0.185	2	0.060	0.155	3	0.060	0.165	4	0.035	0.140	5	0.035	0.155	6	0.025	0.155	7	0.020	0.150	8	0.018	0.155	9	0.010	0.150
Epochs	Training Loss	Validation Loss																																
0	0.140	0.210																																
1	0.105	0.185																																
2	0.060	0.155																																
3	0.060	0.165																																
4	0.035	0.140																																
5	0.035	0.155																																
6	0.025	0.155																																
7	0.020	0.150																																
8	0.018	0.155																																
9	0.010	0.150																																

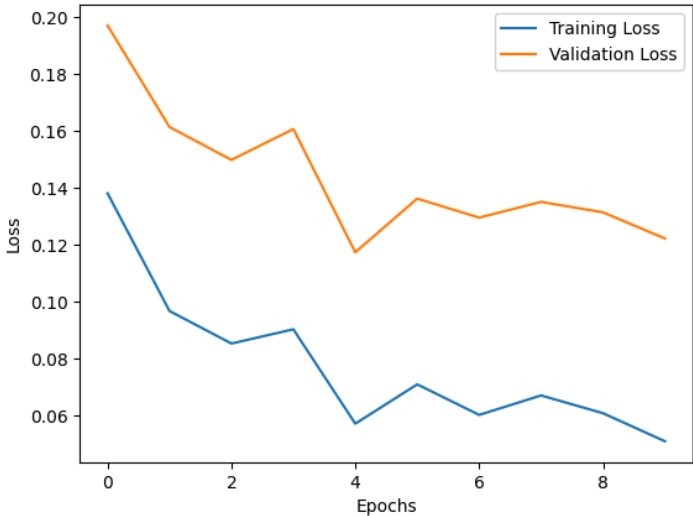
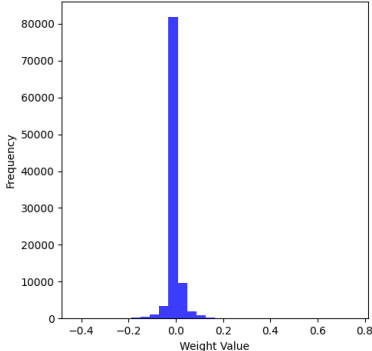
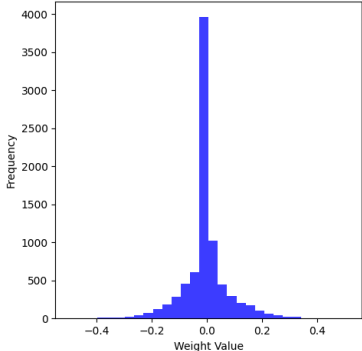
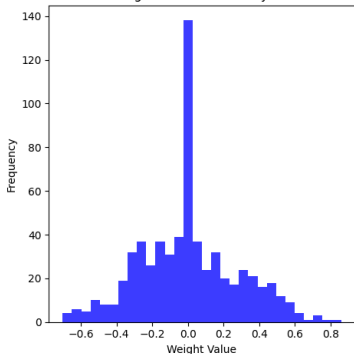
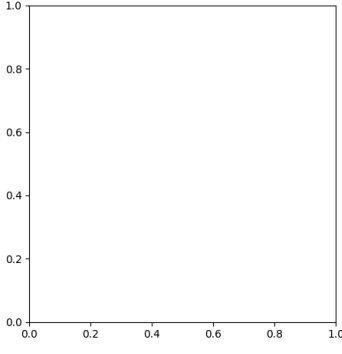
Distribusi Bobot:



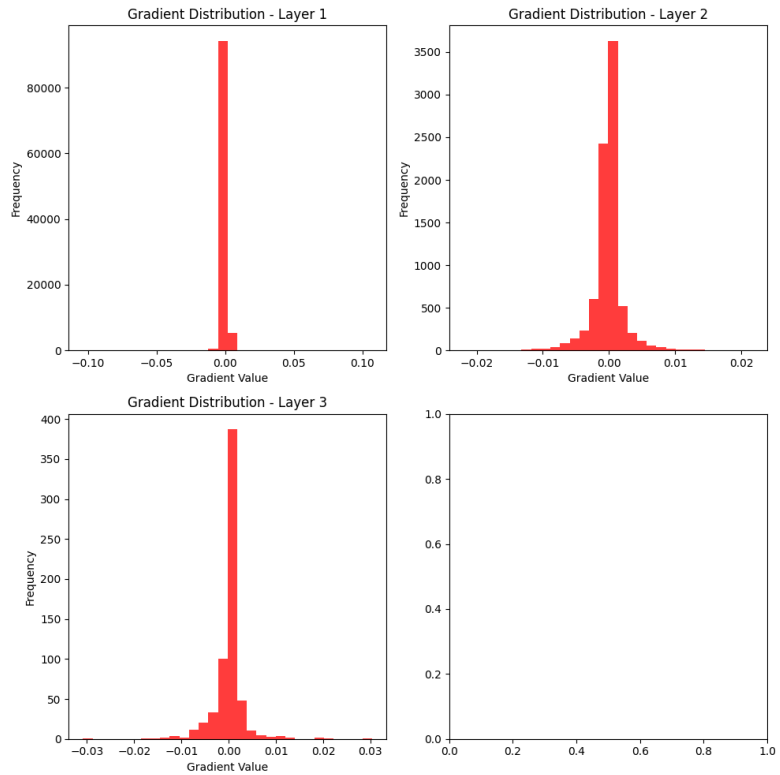
Distribusi Gradien Bobot:



2.4.6.2. Regularisasi L1

Akurasi:	96.93%																																	
Grafik Loss Pelatihan:	Training Loss Curve  <table border="1"><thead><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>0.14</td><td>0.19</td></tr><tr><td>1</td><td>0.09</td><td>0.16</td></tr><tr><td>2</td><td>0.08</td><td>0.15</td></tr><tr><td>3</td><td>0.09</td><td>0.16</td></tr><tr><td>4</td><td>0.06</td><td>0.12</td></tr><tr><td>5</td><td>0.07</td><td>0.13</td></tr><tr><td>6</td><td>0.06</td><td>0.13</td></tr><tr><td>7</td><td>0.07</td><td>0.13</td></tr><tr><td>8</td><td>0.06</td><td>0.13</td></tr><tr><td>9</td><td>0.05</td><td>0.12</td></tr></tbody></table>	Epochs	Training Loss	Validation Loss	0	0.14	0.19	1	0.09	0.16	2	0.08	0.15	3	0.09	0.16	4	0.06	0.12	5	0.07	0.13	6	0.06	0.13	7	0.07	0.13	8	0.06	0.13	9	0.05	0.12
Epochs	Training Loss	Validation Loss																																
0	0.14	0.19																																
1	0.09	0.16																																
2	0.08	0.15																																
3	0.09	0.16																																
4	0.06	0.12																																
5	0.07	0.13																																
6	0.06	0.13																																
7	0.07	0.13																																
8	0.06	0.13																																
9	0.05	0.12																																
Distribusi Bobot:	Weight Distribution - Layer 1  Weight Distribution - Layer 2  Weight Distribution - Layer 3  																																	

Distribusi Gradien Bobot:

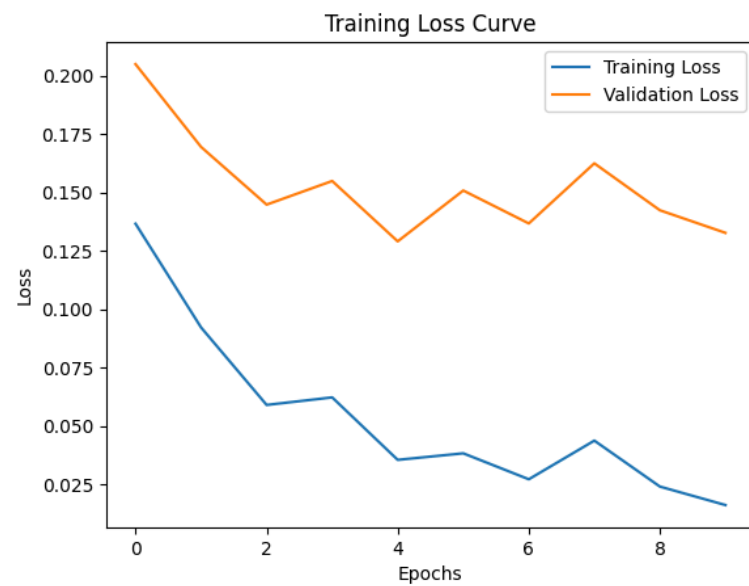


2.4.6.3. Regularisasi L2

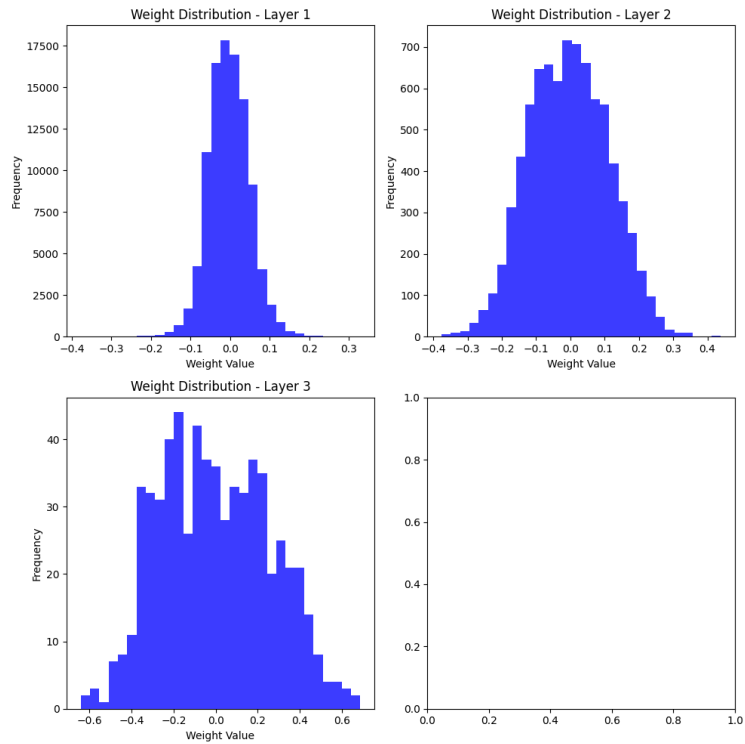
Akurasi:

97.15%

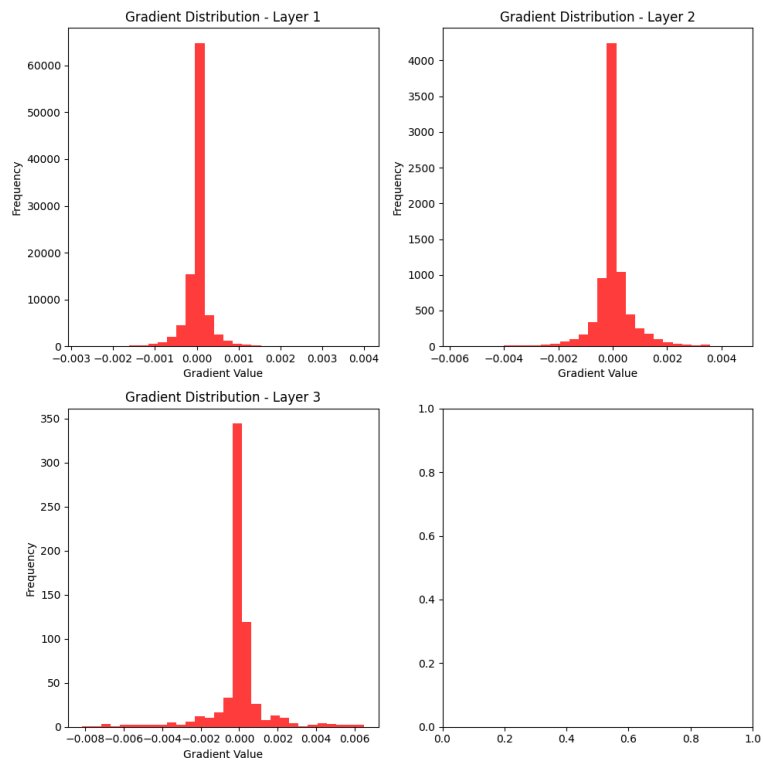
Grafik Loss Pelatihan:



Distribusi Bobot:



Distribusi Gradien Bobot:



2.5. Analisis

2.5.1. Pengaruh *Depth* terhadap Akurasi

No	Width	Depth	Akurasi
1.	100	2	97.29%
2.	100	3	97.06%
3.	100	4	97.01%

Dapat diamati bahwa peningkatan depth (kedalaman jaringan) dapat meningkatkan akurasi model. Hal ini terjadi karena jaringan yang lebih dalam memiliki kapasitas representasi yang lebih besar, memungkinkan model untuk menangkap pola yang lebih kompleks dalam data. Dengan lebih banyak lapisan, model dapat membangun representasi fitur yang lebih abstrak dan hierarkis, yang sering kali meningkatkan performa dalam tugas-tugas seperti klasifikasi gambar, pemrosesan bahasa alami, dan lainnya.

Namun, penambahan depth yang berlebihan juga memiliki risiko, terutama dalam bentuk overfitting. Jaringan yang terlalu dalam dapat belajar pola spesifik dari data latih, termasuk noise, sehingga performa pada data uji menurun. Selain itu, jaringan dalam juga rentan terhadap masalah vanishing/exploding gradients, yang dapat menyebabkan kesulitan dalam konvergensi saat pelatihan. Oleh karena itu, teknik regulasi seperti dropout, batch normalization, dan early stopping sering digunakan untuk mengatasi masalah ini dan menjaga keseimbangan antara bias dan varians dalam model.

2.5.2. Pengaruh *Width* terhadap Akurasi

No	Width	Depth	Akurasi
1.	50	2	96.39%
2.	100	2	97.29%
3.	200	2	97.54%

Dapat diamati bahwa peningkatan width atau banyak neuron pada tiap layer dapat menambah akurasi dari model kami. Penyebab hal tersebut adalah karena dengan neuron yang lebih banyak, kapasitas model akan meningkat, memungkinkan model untuk mempelajari detail yang lebih detail. Adanya data ini mungkin juga membuktikan bahwa pada $\text{width} \leq 200$, masih terjadi underfitting.

2.5.3. Pengaruh *Learning Rate* terhadap Akurasi

No	Learning rate	Akurasi
1.	0.01	93.68%
2.	0.1	96.30%
3.	0.5	97.29%

Penggunaan learning rate berkaitan dengan seberapa cepat fungsi loss mencapai local minimum. Learning rate yang cukup besar berarti fungsi loss akan bergerak menuju local minimum dengan cepat dan sebaliknya jika learning rate yang digunakan kecil. Jika learning rate yang digunakan terlalu kecil, maka model akan sampai ke global minimum terlalu lama. Untuk kasus fungsi yang memiliki banyak local minima, maka ada kemungkinan bahwa fungsi loss gagal keluar dari cekungan kurva karena tidak bisa mengambil langkah yang besar. Namun, jika fungsi loss terlalu besar, terdapat resiko overshooting atau divergensi global optimum.

Dapat dilihat bahwa penggunaan learning rate yang paling kecil (0.01) memberikan akurasi yang paling kecil pula. Hal ini dapat dipahami bahwa di dalam pelatihan dengan pengulangan yang relatif sedikit (epoch=10, batch_size=200), resiko loss function tidak mencapai global optimum menjadi lebih besar. Selanjutnya, jika learning rate yang digunakan adalah 0.1 dan 0.5, akurasi akan berturut-turut meningkat. Artinya, learning rate sebesar 0.5 dikatakan masih dalam batas ideal untuk kasus ini jika ditinjau terisolasi dari hyperparameter lain. Sebab, peningkatan hyperparameter learning rate sampai dengan 0.5 masih menunjukkan peningkatan akurasi sehingga kemungkinan belum ada overshooting.

2.5.4. Pengaruh Inisialisasi Bobot terhadap Akurasi

No	Metode Inisialisasi	Akurasi
1.	Inisialisasi Nol	11.11%
2.	Inisialisasi acak dengan distribusi uniform	97.29%
3.	Inisialisasi acak dengan distribusi normal	9.79%
4.	Inisialisasi Xavier	97.29%
5.	Inisialisasi He	97.24%

Perbedaan performa inisialisasi bobot pada feed-forward neural network (FFNN) dapat dijelaskan melalui pengaruhnya terhadap *gradient flow* dan *activation statistics* selama

pelatihan. Inisialisasi normal acak (random normal) menjadi yang terburuk karena variansnya tidak dikontrol, menyebabkan gradien tidak stabil atau hilang. Inisialisasi nol (zero initialization) gagal karena semua neuron di lapisan yang sama memperbarui bobot secara identik, menghambat pembelajaran fitur yang beragam. Sementara itu, inisialisasi He bekerja cukup baik untuk aktivasi ReLU (pada kasus ini penulis menggunakan ReLU) tetapi kurang optimal jika digunakan dengan sigmoid/tanh. Performa terbaik ditunjukkan oleh inisialisasi seragam (uniform) dan Xavier karena keduanya menyesuaikan skala bobot berdasarkan jumlah neuron, menjaga stabilitas sinyal selama *forward* dan *back propagation*.

Hasil ini sesuai dengan teori: inisialisasi yang diskalakan dengan benar (seperti Xavier atau seragam) mempertahankan variansi aktivasi dan gradien, sehingga mempercepat konvergensi. Inisialisasi Xavier khususnya cocok untuk aktivasi sigmoid/tanh, sedangkan He lebih ditujukan untuk ReLU. Sebaliknya, inisialisasi acak tanpa skala atau nol menyebabkan masalah simetri dan ketidakstabilan, mengurangi akurasi model. Pemilihan metode inisialisasi yang tepat harus mempertimbangkan jenis aktivasi dan kedalaman jaringan untuk menghindari *vanishing/exploding gradients*.

2.5.5. Pengaruh Fungsi Aktivasi terhadap Akurasi

No	Fungsi Aktivasi	Akurasi
1.	Linear	89.55%
2.	ReLU	97.29%
3.	Sigmoid	96.16%
4.	Tanh	96.57%
5.	ELU	96.27%
6.	Leaky_relu	97.18%

ReLU bekerja sangat baik pada dataset MNIST karena kemampuannya menangani pola yang kompleks dengan lebih efisien dibandingkan fungsi aktivasi lain seperti sigmoid dan tanh. MNIST adalah dataset gambar digit yang ditulis menggunakan tangan sehingga mengandung banyak variasi bentuk dan pola. Agar FFNN bisa mengenali pola ini dengan baik, ia harus bisa menangkap hubungan non-linear dalam data. ReLU sangat cocok untuk ini karena ia hanya meneruskan nilai positif dan membiarkan nilai negatif menjadi nol, sehingga jaringan dapat belajar lebih cepat dan lebih stabil. Tidak seperti sigmoid dan tanh yang membatasi output mereka dalam rentang kecil, ReLU membiarkan nilai positif tetap besar, sehingga jaringan bisa lebih fleksibel dalam memahami variasi data.

Keunggulan lain dari ReLU adalah kemampuannya mengatasi masalah vanishing gradient, yang sering menjadi kendala saat menggunakan sigmoid atau tanh. Dalam sigmoid, jika nilai input sangat besar atau sangat kecil, perubahan bobot menjadi sangat kecil,

membuat jaringan sulit belajar, terutama di lapisan yang lebih dalam. Tanh memang sedikit lebih baik, tetapi masih mengalami masalah yang sama. ReLU menghindari hal ini karena gradiennya tetap besar untuk nilai positif, sehingga pembelajaran tetap berjalan efektif bahkan dalam jaringan yang dalam.

Selain itu, ReLU juga membuat jaringan lebih efisien dan tidak mudah overfitting. Karena nilai negatif dipetakan ke nol, tidak semua neuron akan aktif setiap saat, sehingga jaringan menjadi lebih sederhana dan lebih hemat komputasi. Ini berbeda dengan sigmoid dan tanh, yang selalu mengaktifkan semua neuron, sehingga kadang-kadang justru membuat jaringan belajar hal-hal yang tidak terlalu penting dan berisiko overfitting.

2.5.6. Perbandingan dengan Library Scikit-Learn

No	Model yang Digunakan	Akurasi
1.	Buatan Penulis	97.15%
2.	Scikit-Learn	96.27%

Model yang diimplementasikan sudah cukup akurat. Hal tersebut bisa dibuktikan dengan betapa dekatnya akurasi yang dihasilkan model penulis dan model yang disediakan oleh Scikit-Learn. Penulis mencoba melihat secara detail model scikit learn dan kemungkinan perbedaan pada akurasi dan loss dapat disebabkan oleh inisialisasi bobot yang oleh Scikit-Learn dilakukan untuk seluruh model sekaligus dan bukan pada tiap layer seperti yang dilakukan pada program ini.

2.5.7. Pengaruh Regularisasi terhadap Akurasi

No	Metode Regularisasi	Akurasi
1.	Tanpa Regularisasi	97.15%
2.	L1	96.93%
3.	L2	97.15%

Secara umum, metode regularisasi digunakan untuk mencegah overfitting dengan cara menambahkan penalti terhadap loss yang besar. Artinya, setiap kali melakukan backward propagation, besaran yang seharusnya digunakan untuk mengupdate suatu bobot akan berkurang karena adanya penalti. Metode regularisasi L1 akan mendorong model untuk memiliki bobot yang "sparse," yaitu memiliki banyak bobot dengan nilai 0. Sedangkan, metode regularisasi L2 akan mendorong model memiliki bobot yang kecil namun bukan mendorong sparsity.

Oleh karena itu, berdasarkan tabel pengaruh regularisasi terhadap akurasi yang diberikan, dapat dilihat bahwa akurasi bila pelatihan model menggunakan regularisasi L1 menurun dibandingkan jika tanpa regularisasi. Hal ini dapat diduga disebabkan oleh sparsity

sehingga ada fitur-fitur yang menjadi tidak terdeteksi. Melihat akurasi pelatihan model menggunakan regularisasi L2 seakan menandakan bahwa regularisasi tidak memperbaiki kemampuan klasifikasi model.

Namun, karena kemampuan regularisasi masing-masing metode regularisasi, meski hasil akurasi yang diberikan sama saja atau bahkan lebih buruk, selisih dari training loss dan validation loss memiliki perbedaan seperti yang dapat dilihat pada grafik sejarah loss. Model yang dilatih dengan metode regularisasi tidak mengalami overfitting yang besar bila dibandingkan dengan model yang dilatih tanpa metode regularisasi. Hal ini dapat kita lihat pada grafik sejarah loss data latih dan data validasi pelatihan model tanpa regularisasi. Pada suatu epoch tertentu, loss dari data latih mengalami penurunan namun loss dari data validasi malah mengalami kenaikan.

BAB III

KESIMPULAN DAN SARAN

3.1 Kesimpulan

Pada laporan ini, kami mengimplementasikan Feedforward Neural Network yang dapat digunakan untuk permasalahan regresi dan klasifikasi, diuji terhadap dataset klasifikasi digit tulis tangan mnist 784. Program yang dikembangkan dapat memberikan performa yang baik, hampir sebanding dengan performa library ternama seperti Scikit-Learn. Berdasarkan pengujian, perubahan pada hyperparameter mengakibatkan perubahan yang terduga dan seharusnya pada performa model, mengindikasikan *correctness* dari program yang telah dibuat. Pengujian model terhadap beberapa hyperparameter juga mengindikasikan pentingnya pemilihan arsitektur model dan layer (kedalaman, banyak neuron, fungsi aktivasi, inisialisasi) maupun hyperparameter model lainnya dalam membuat suatu model pembelajaran mesin yang baik.

3.2 Saran

Berikut beberapa saran untuk melakukan pengembangan Feedforward Neural Network dari scratch:

- 1) Pahami library yang terlibat. Pada proyek ini, digunakan library PyTorch untuk mendukung vektorisasi operasi. Library ini digunakan karena pada mulanya library ini dianggap dapat mendeteksi secara otomatis keberadaan GPU. Namun, karena ternyata hal itu tidak dapat dilakukan secara otomatis, dan mengingat fitur automatic differentiation milik PyTorch tidak boleh digunakan pada tugas ini, ada baiknya bila menggunakan library numpy saja agar proyek tidak memiliki terlalu banyak *dependencies*.
- 2) Jika mengacu pada referensi (micrograd), kelas yang bertanggung jawab untuk automatic differentiation hanya dapat memiliki satu tipe data dan tidak mendukung operasi dimensi tinggi. Oleh karena itu, seperti yang dilakukan pada tugas ini, lebih baik agar kelas yang melakukan operasi automatic differentiation setidaknya mendukung operasi dimensi dua. Namun, akan lebih baik jika memahami terkait broadcasting dan bagaimana mengatasi efek broadcasting pada backward pass, serta vector-Jacobian product sebelum mencoba mengimplementasikannya.

- 3) Terapkan prinsip pengembangan perangkat lunak agar pengembangan menjadi lebih mudah: high cohesion - loose coupling, SOLID, modularitas, KISS.
- 4) Lakukan pengujian sejak awal agar kebenaran program terjamin. Pada saat bagian awal mengembangkan program ini, terdapat permasalahan pada isu broadcasting dan perhitungan gradien kelas Value yang tidak terlihat karena pengujian yang dilakukan tidak lengkap (i.e. hanya pada data skalar, bukan vektor/matriks)

PEMBAGIAN TUGAS

Kegiatan	Nama (NIM)
Value (Auto Differentiation)	Julian Chandra Sutadi (13522080)
Activation Functions	Immanuel Sebastian Girsang (13522058) Julian Chandra Sutadi (13522080)
Stochastic Gradient Descent	Venantius Sean Ardi Nugroho (13522078)
FFNN	Immanuel Sebastian Girsang (13522058) Venantius Sean Ardi Nugroho (13522078) Julian Chandra Sutadi (13522080)
FFNN Visualization	Venantius Sean Ardi Nugroho (13522078)
Initializers	Immanuel Sebastian Girsang (13522058) Venantius Sean Ardi Nugroho (13522078)
Loss	Julian Chandra Sutadi (13522080)
Regularization	Immanuel Sebastian Girsang (13522058) Julian Chandra Sutadi (13522080)
Experimentation	Immanuel Sebastian Girsang (13522058) Venantius Sean Ardi Nugroho (13522078)
Laporan	Immanuel Sebastian Girsang (13522058) Venantius Sean Ardi Nugroho (13522078) Julian Chandra Sutadi (13522080)

REFERENSI

<https://github.com/karpathy/micrograd/blob/master/micrograd/engine.py>

https://github.com/mattij/autodidact/blob/master/autograd/numpy/numpy_vjps.py

<https://douglasorr.github.io/2021-11-autodiff/article.html>

▶ Softmax - Pullback/vJp rule

<https://builtin.com/data-science/l2-regularization>