

INFORMATICS ENGINEERING STUDY PROGRAM  
SCHOOL OF ELECTRICAL ENGINEERING DAN INFORMATICS  
INSTITUT TEKNOLOGI BANDUNG



# TUGAS BESAR 1

## Pembelajaran Mesin Feedforward Neural Network

**Dibuat oleh:**

Kelompok 33

**Anggota:**

1. 13522013 - Denise Felicia Tiowanni
2. 13522083 - Evelyn Yosiana
3. 13522103 - Steven Tjhia

**IF3270 - Pembelajaran Mesin**

**2025**

# Daftar Isi

<b>Daftar Isi.....</b>	<b>1</b>
<b>Bab I. Deskripsi Persoalan.....</b>	<b>2</b>
1.1. Latar Belakang.....	2
1.2. Ruang Lingkup Implementasi.....	2
<b>Bab II. Pembahasan.....</b>	<b>4</b>
2.1. Penjelasan Implementasi.....	4
2.1.1. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya.....	4
2.1.1.1. FFNN dengan Automatic Differentiation.....	4
2.1.1.2. FFNN Model Biasa (Tanpa AutoDiff).....	8
2.1.2. Penjelasan Forward Propagation.....	12
2.1.3. Penjelasan Backward Propagation dan Weight Update.....	13
b) FFNN Model Biasa (Tanpa AutoDiff).....	14
2.2. Hasil pengujian.....	15
2.2.1. Pengaruh Depth dan Width (Basic FFNN).....	15
2.2.2. Pengaruh Fungsi Aktivasi (Basic FFNN) → dengan RMSNorm.....	22
2.2.3. Pengaruh Fungsi Aktivasi (Basic FFNN) → tanpa RMSNorm.....	33
2.2.4. Pengaruh Learning Rate (Basic FFNN).....	45
2.2.5. Pengaruh Inisialisasi Bobot (Basic FFNN).....	52
2.2.6. Perbandingan dengan Library Sklearn.....	65
2.2.7. Perbandingan dengan dan tanpa Regularisasi.....	68
2.2.7.1. Tanpa Regularisasi.....	69
2.2.7.2. Dengan Regularisasi L1.....	71
2.2.7.3. Dengan Regularisasi L2.....	73
2.2.7.4. Dengan Regularisasi L1 dan L2.....	75
<b>Bab III. Kesimpulan dan Saran.....</b>	<b>78</b>
3.1. Kesimpulan.....	78
3.2. Saran.....	78
<b>Pembagian Tugas.....</b>	<b>79</b>
<b>Referensi.....</b>	<b>80</b>

# Bab I. Deskripsi Persoalan

## 1.1. Latar Belakang

Feedforward Neural Network (FFNN) merupakan salah satu arsitektur dasar dalam dunia machine learning yang terinspirasi dari cara kerja otak manusia. Dalam model ini, data akan masuk melalui satu lapisan input, lalu secara bertahap diteruskan melalui beberapa lapisan tersembunyi (hidden layers), hingga akhirnya menghasilkan suatu output. Informasi selalu mengalir ke satu arah, yaitu dari input ke output, tanpa adanya putaran balik atau siklus.

Setiap neuron dalam sebuah lapisan terhubung dengan semua neuron di lapisan berikutnya. Koneksi antar neuron ini memungkinkan FFNN untuk belajar mengenali pola-pola tertentu dalam data dan membuat prediksi atau keputusan yang diinginkan. Dengan struktur ini, FFNN dapat menyelesaikan berbagai tugas seperti klasifikasi gambar, prediksi nilai, atau pengenalan pola.

Pada mata kuliah Pembelajaran Mesin IF3270 ini, mahasiswa diminta untuk mengimplementasi Feedforward Neural Network secara mandiri (*from scratch*), meliputi proses forward propagation, backward propagation, penghitungan gradien, update bobot, serta visualisasi dan analisis performa model terhadap beberapa hyperparameter yang berpengaruh pada kualitas prediksi.

## 1.2. Ruang Lingkup Implementasi

Adapun ruang lingkup implementasi tugas besar ini mencakup:

- Implementasi modul FFNN yang dapat disesuaikan dengan parameter jumlah neuron di setiap layer (input, hidden, dan output).
- Implementasi berbagai fungsi aktivasi, yaitu Linear, ReLU, Sigmoid, tanh, dan Softmax.
- Implementasi berbagai fungsi loss yang mencakup Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy.
- Metode inisialisasi bobot:
  - Zero initialization

- Random uniform (dengan parameter lower bound, upper bound, seed)
- Random normal (dengan parameter mean, variance, seed)
- Implementasi fitur tambahan meliputi:
  - Penyimpanan bobot dan gradien bobot (termasuk bias)
  - Visualisasi struktur jaringan, bobot, dan gradien dalam bentuk graf
  - Visualisasi distribusi bobot dan gradien bobot dari tiap layer
  - Fasilitas untuk menyimpan (save) dan memuat (load) model secara persisten
- Pengujian model terhadap dataset MNIST dengan memperhatikan pengaruh variasi hyperparameter yang telah ditentukan.
- Melakukan analisis perbandingan performa model implementasi sendiri dengan model dari library sklearn (MLPClassifier).

# Bab II. Pembahasan

## 2.1. Penjelasan Implementasi

### 2.1.1. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya

Dalam penggerjaan tugas besar ini, kami mengimplementasikan dua jenis model Feedforward Neural Network (FFNN), yaitu:

#### 2.1.1.1. FFNN dengan Automatic Differentiation

Model ini dibangun menggunakan kelas VarValue, yang memungkinkan perhitungan turunan dilakukan secara otomatis dengan membentuk computational graph. Implementasi ini bertujuan untuk memahami secara mendalam proses forward dan backward propagation serta bagaimana turunan dari loss ditelusuri kembali hingga parameter.

##### a. Kelas VarValue

Kelas VarValue adalah representasi dari nilai variabel yang mendukung automatic differentiation. Kelas ini memungkinkan kita untuk:

- Menyimpan nilai dan variabel turunan
- Melakukan operasi matematika dasar
- Melacak dependensi antar variabel
- Melakukan perhitungan rantai (chain rule) secara otomatis

Berikut adalah atribut dalam kelas VarValue:

Atribut	Deskripsi
value	Nilai aktual dari variabel (tipe float atau int)
varname	Nama variabel, digunakan untuk pelacakan turunan
children	Tuple berisi variabel-variabel yang menjadi input ke operasi ini
derivative_to	Dictionary yang menyimpan turunan terhadap setiap variabel yang terlibat dalam grafik komputasi

dan berikut adalah method dalam kelas VarValue:

<b>Method</b>	<b>Deskripsi</b>
<code>__init__</code>	Konstruktor untuk membuat objek VarValue
<code>relu()</code>	Fungsi aktivasi ReLU
<code>ln() / log()</code>	Logaritma natural dari nilai variabel
<code>exp()</code>	Eksponensial dari nilai variabel
<code>__add__</code>	Operasi penjumlahan (+)
<code>__sub__</code>	Operasi pengurangan (-)
<code>__mul__</code>	Operasi perkalian (*)
<code>__truediv__</code>	Operasi pembagian (/)
<code>__pow__</code>	Operasi pangkat (**)
<code>__neg__</code>	Unary minus (negasi)
<code>__chain_rule()</code>	Menerapkan rantai turunan berdasarkan children
<code>__radd__</code> , <code>__rmul__</code> , <code>__rsub__</code> , <code>__rtruediv__</code> , <code>__rpow__</code>	Versi reverse dari operasi untuk mendukung angka + VarValue
<code>__eq__</code> , <code>__lt__</code> , <code>__gt__</code>	Operator perbandingan

### b. Kelas Layer

Kelas Layer merepresentasikan sebuah lapisan (layer) pada jaringan Neural Network. Kelas ini menangani proses:

- Inisialisasi bobot dan bias
- Propagasi maju (forward propagation)
- Propagasi mundur (backward propagation) dengan perhitungan turunan otomatis
- Pembaruan bobot dan bias

Berikut adalah atribut dalam kelas Layer:

Atribut	Deskripsi
n_neurons	Jumlah neuron pada layer ini
init	Metode inisialisasi bobot (zero, uniform, normal, xavier, he)
activation	Fungsi aktivasi yang digunakan (linear, relu, sigmoid, tanh, softmax)
weights	Matriks bobot
biases	Vektor bias
learning_rate	Laju pembelajaran yang akan digunakan untuk update
grad_weights	Gradien bobot setelah backward pass
grad_biases	Gradien bias setelah backward pass
current_input_batch	Batch input saat ini untuk layer ini
net	Nilai hasil dot product + bias sebelum aktivasi
out	Output dari layer setelah fungsi aktivasi

dan berikut adalah method dalam kelas Layer:

Method	Deskripsi
__init__()	Inisialisasi layer beserta parameter bobot dan bias
forward(current_input_batch)	Melakukan forward propagation dengan input yang diberikan
backward(err)	Melakukan backward propagation berdasarkan turunan loss (err)
clean_derivative()	Menghapus informasi turunan di semua variabel untuk siklus baru
__update_weights_dEdW(dEdW)	Melakukan update bobot menggunakan gradien
__update_bias	Melakukan update bias menggunakan gradien

s_dEdB(dEdB)	
__update_weights_err_term(err_term)	<b>(Belum digunakan)</b> update bobot dengan error term
__update_biasess_err_term(err_term)	<b>(Belum digunakan)</b> update bias dengan error term

### c. Kelas FFNN

Kelas FFNN (Feedforward Neural Network) adalah implementasi FFNN yang dapat dilatih dengan algoritma backpropagation. Kelas ini mendukung berbagai fungsi aktivasi, loss function, dan metode visualisasi bobot serta arsitektur.

Berikut adalah atribut dalam kelas FFNN:

Atribut	Deskripsi
loss	Jenis fungsi loss yang digunakan (mse, bce, cce)
batch_size	Jumlah data dalam setiap batch
learning_rate	Laju pembelajaran untuk update bobot
epochs	Jumlah iterasi pelatihan
verbose	Flag untuk menampilkan progress bar saat training
layers	List berisi layer-layer jaringan (objek Layer)
weights	Placeholder untuk menyimpan bobot seluruh jaringan
bias	Placeholder untuk menyimpan bias seluruh jaringan
x, y	Data latih (input dan target yang telah di-one-hot encode)
onehot_encoder	Encoder untuk mengubah target menjadi one-hot vector

dan berikut adalah method dalam kelas FFNN:

Method	Deskripsi
<code>__init__()</code>	Konstruktor: mengatur parameter umum model
<code>__loss(out, target)</code>	Fungsi loss (mse, bce, atau cce)
<code>build_layers(*layers)</code>	Menyusun struktur jaringan dengan layer-layer yang diberikan
<code>fit(x, y, validation_data=None)</code>	Melatih model menggunakan data input dan label
<code>predict(x_predict)</code>	Melakukan prediksi terhadap input baru
<code>visualize()</code>	Visualisasi arsitektur jaringan saraf menggunakan plotly
<code>save(filename)</code>	Menyimpan parameter model ke file pickle
<code>load(filename)</code>	Memuat parameter model dari file pickle
<code>plot_weights_distribution(layers_to_plot)</code>	Menampilkan distribusi bobot layer-layer tertentu dalam bentuk boxplot
<code>plot_gradients_distribution(layers_to_plot)</code>	Menampilkan distribusi gradien layer-layer tertentu dalam bentuk boxplot
<code>plot_loss_history(history)</code>	Menampilkan grafik loss (training dan validasi) selama proses training

#### 2.1.1.2. FFNN Model Biasa (Tanpa AutoDiff)

Karena keterbatasan memori dan sumber daya komputasi pada perangkat yang kami gunakan (terutama saat memproses data dalam jumlah besar seperti MNIST), kami juga membangun versi model biasa tanpa VarValue, dengan perhitungan gradien dilakukan secara manual atau menggunakan pendekatan diferensiasi numerik.

### a. Kelas Layer

Kelas ini berfungsi secara sama dengan kelas Layer yang terdapat pada model AutoDiff. Hanya saja, berbeda dari versi auto-diff, pada model ini perhitungan dilakukan secara manual menggunakan NumPy, dan tidak menyimpan grafik komputasi.

Berikut adalah atribut dalam kelas Layer:

Atribut	Deskripsi
n_neurons	Jumlah neuron dalam layer
init	Metode inisialisasi bobot, misalnya zero, uniform, normal, xavier, he
activation	Jenis fungsi aktivasi yang digunakan (linear, relu, sigmoid, tanh, softmax)
init_params	Parameter tambahan untuk inisialisasi (misal: mean, variance, lower, upper, seed)
weights	Matriks bobot layer, berukuran (input_dim, n_neurons)
biases	Vektor bias untuk setiap neuron, berukuran (1, n_neurons)

dan berikut adalah method dalam kelas Layer:

Method	Deskripsi
__init__()	Konstruktor layer, menerima jumlah neuron, jenis aktivasi, dan parameter inisialisasi bobot
initialize(input_dim)	Menginisialisasi bobot dan bias sesuai dimensi input dan metode inisialisasi yang dipilih
activate(x)	Mengaplikasikan fungsi aktivasi ke output linear ( $z = x @ \text{weights} + \text{bias}$ ). Mendukung berbagai fungsi aktivasi seperti relu, sigmoid, tanh, softmax, atau linear
activation_derivative(x)	Mengembalikan turunan dari fungsi aktivasi terhadap input x, yang dibutuhkan dalam proses

	backpropagation
--	-----------------

### b. Kelas FFNN

*Fungsi masih sama dengan FFNN dengan AutoDiff.*

Berikut adalah atribut dalam kelas FFNN:

Atribut	Deskripsi
layers	List berisi layer-layer (Layer) yang membentuk arsitektur jaringan
learning_rate	Laju pembelajaran (learning rate) yang digunakan saat update bobot
loss	Jenis loss function yang digunakan (mse, bce, cce)
batch_size	Ukuran batch yang digunakan selama training
epochs	Jumlah epoch (iterasi penuh terhadap data latih)
verbose	Opsi output (0: tanpa output, 1: tampilkan progress bar & loss)
loss_func	Fungsi untuk menghitung nilai loss
loss_derivative	Fungsi untuk menghitung turunan dari fungsi loss
train_losses	Array untuk menyimpan train loss
val_losses	Array untuk menyimpan val loss
weights	Array untuk menyimpan weight
biases	Array untuk menyimpan bias
gradient_weights	Array untuk menyimpan gradient weight
gradient_biases	Array untuk menyimpan gradient bias

dan berikut adalah method dalam kelas Layer:

Method	Deskripsi
--------	-----------

<code>__init__()</code>	Konstruktor kelas FFNN, mengatur parameter training dan fungsi loss
<code>build_layers(*layer_args)</code>	Menyusun arsitektur jaringan berdasarkan urutan layer yang diberikan
<code>_initialize_network(input_dim)</code>	Menginisialisasi bobot dan bias setiap layer berdasarkan dimensi input
<code>forward(X)</code>	Melakukan forward propagation, mengembalikan zs (pre-activation) dan activations dari semua layer
<code>backward(X, y, zs, activations)</code>	Melakukan backpropagation: menghitung dan memperbarui bobot dan bias berdasarkan gradien loss
<code>fit(X, y, X_val=None, y_val=None)</code>	Melatih model terhadap data input dan label. Bisa juga menerima data validasi
<code>predict(X)</code>	Melakukan prediksi terhadap input X dan mengembalikan hasil klasifikasi
<code>save(filename)</code>	Menyimpan state model (layer, parameter, dll) ke file .pkl
<code>load(filename)</code>	Memuat state model dari file .pkl dan mengembalikan instance FFNN
<code>visualize_architecture(output_file, figsize)</code>	Menampilkan dan menyimpan diagram arsitektur jaringan menggunakan networkx
<code>plot_weight_distribution(layers_to_plot, figsize)</code>	Visualisasi distribusi bobot layer tertentu dalam bentuk histogram
<code>plot_biases_distribution(layers_to_plot, figsize)</code>	Visualisasi distribusi bias layer tertentu dalam bentuk histogram
<code>plot_gradient_weight_distribution(layers_to_plot, figsize)</code>	Menghitung dan memvisualisasikan distribusi gradien bobot setelah satu langkah backward
<code>plot_gradient_bias()</code>	Menghitung dan memvisualisasikan distribusi

as_distribution(ayers_to_plot, figsize)	gradien bias setelah satu langkah backward
plot_training_losses()	Memvisualisasikan training dan validation loss perepoch

### c. Fungsi-Fungsi Tambahan

Digunakan untuk menyimpan loss function dan turunannya. Fungsi-fungsi ini tidak berada dalam kelas apa pun, tapi digunakan oleh kelas FFNN saat menentukan loss dan gradiennya.

Fungsi	Deskripsi
<code>mse(y_true, y_pred)</code>	Menghitung Mean Squared Error antara label dan prediksi. Jika label bukan one-hot, otomatis dikonversi
<code>mse_derivative(y_true, y_pred)</code>	Mengembalikan turunan dari MSE terhadap output jaringan
<code>bce(y_true, y_pred)</code>	Menghitung Binary Cross-Entropy Loss. Prediksi akan diklip untuk menghindari log(0)
<code>bce_derivative(y_true, y_pred)</code>	Mengembalikan turunan dari BCE Loss terhadap prediksi
<code>cce(y_true, y_pred)</code>	Menghitung Categorical Cross-Entropy Loss, mendukung input label sebagai integer class (akan di-one-hot-kan otomatis)
<code>cce_derivative(y_true, y_pred)</code>	Mengembalikan turunan dari CCE Loss terhadap output. Dihitung sebagai $y_{pred} - y_{true}$

#### 2.1.2. Penjelasan Forward Propagation

Implementasi forward propagation pada FFNN dengan *Autodiff* sama saja dengan FFNN *Basic*. Forward propagation adalah proses di mana input dari data dimasukkan ke jaringan, dan kemudian dihitung melalui setiap layer hingga menghasilkan output prediksi.

Berikut adalah langkah-langkah forward propagation yang kami implementasikan:

1. Inisialisasi Bobot dan Bias

Saat forward() dipanggil pertama kali pada suatu layer, bobot (weights) dan bias (biases) akan diinisialisasi. Metode inisialisasi disesuaikan dengan parameter init (zero, uniform, normal, xavier, atau he). Proses ini hanya dilakukan sekali per layer. Setelah itu, bobot akan digunakan dan diperbarui selama training.

2. Linear Transformation

Di setiap layer, dilakukan perhitungan linear:

$$\text{net} = X \cdot W + b$$

dimana X adalah input batch, W adalah bobot, dan b adalah bias. Hasil net kemudian disimpan untuk digunakan dalam backward pass.

3. Fungsi Aktivasi

Output dari net kemudian dilewatkan ke fungsi aktivasi sesuai layer (linear, relu, sigmoid, tanh, atau softmax).

4. Output Diteruskan ke Layer Selanjutnya

Output dari layer akan menjadi input untuk layer berikutnya. Ini akan dilakukan berulang untuk semua layer dalam model.

5. Hasil Akhir

Setelah melewati seluruh layer, output yang terakhir pun didapatkan dan dapat digunakan untuk menghitung loss, membandingkan hasil prediksi dengan label aktual, atau diteruskan ke backpropagation.

### 2.1.3. Penjelasan Backward Propagation dan Weight Update

Backward propagation (backpropagation) adalah proses menghitung turunan (gradien) dari fungsi loss terhadap bobot dan bias di setiap layer. Hasil turunan ini digunakan untuk mengupdate bobot guna meminimalkan error pada epoch berikutnya.

#### a) FFNN dengan Autodiff

Berikut adalah langkah-langkah backward propagation yang kami implementasikan pada FFNN dengan *Autodiff*:

1. Validasi Parameter *Error*

Parameter *err* adalah objek *VarValue* hasil dari fungsi loss. Atribut *derivative\_to* pada *VarValue* menyimpan semua turunan loss terhadap setiap variabel (bobot dan bias).

2. Ambil Gradien dari *Computational Graph*

Pertama-tama, kita akan membuat array kosong untuk menyimpan turunan loss terhadap bobot ( $dEdW$ ) dan bias ( $dEdB$ ). Kemudian, kita akan mengambil turunan loss terhadap masing-masing bobot. Apabila turunan tidak ada (misal, gradien = 0), maka default-nya adalah 0. Hal yang sama dilakukan untuk bias.

3. Simpan Gradien

Gradien kemudian akan disimpan untuk keperluan visualisasi yang mungkin diperlukan.

4. Update Bobot dan Bias

Untuk memperbarui bobot dan bias, dilakukan gradient descent update:

$$W = W - \text{learning\_rate} \times \frac{\partial L}{\partial W}$$

$$b = b - \text{learning\_rate} \times \frac{\partial L}{\partial b}$$

*Clipping* digunakan agar nilai perubahan tidak terlalu besar (mencegah *exploding gradients*).

**b) FFNN Model Biasa (Tanpa AutoDiff)**

Berikut adalah langkah-langkah backward propagation yang kami implementasikan pada FFNN model biasa:

1. Menghitung Turunan *Error* terhadap *Net* tiap Layer

Melakukan perhitungan terhadap *error term* yang pada program kami direpresentasikan oleh variabel *delta*.

## 2. Menghitung Gradien Error terhadap Bobot

Gradien dihitung dengan melakukan perkalian (perkalian matriks pada *batch*) antara *delta* dengan *output* dari layer sebelumnya yang pada program kami direpresentasikan oleh variabel *a\_prev*. Karena data masukan dapat berupa *batch*, maka diambil nilai rata-rata dari gradien tersebut.

## 3. Update Bobot dan Bias

Untuk memperbaharui bobot dan bias, dilakukan gradient descent update:

$$W = W - \text{learning\_rate} \times \frac{\partial L}{\partial W}$$
$$b = b - \text{learning\_rate} \times \frac{\partial L}{\partial b}$$

## 2.2. Hasil pengujian

### 2.2.1. Pengaruh Depth dan Width (*Basic FFNN*)

#### a. Depth

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test. Variasi depth yang dilakukan adalah 3, 5, dan 7.

Hyperparameter yang digunakan adalah sebagai berikut:

- Loss: MSE
- Learning Rate: 0.01
- Batch size: 200
- Epochs: 100

Arsitektur yang digunakan adalah sebagai berikut:

Untuk depth = 3

No. Layer	Jumlah Neuron	Initialization	Activation Function
-----------	---------------	----------------	---------------------

1.	64	<i>Uniform</i>	<i>Softmax</i>
2.	32	<i>Uniform</i>	<i>Softmax</i>
3.	10	<i>Uniform</i>	<i>Softmax</i>

Untuk depth = 5

No. Layer	Jumlah Neuron	Initialization	Activation Function
1.	128	<i>Uniform</i>	<i>Softmax</i>
2.	64	<i>Uniform</i>	<i>Softmax</i>
3.	64	<i>Uniform</i>	<i>Softmax</i>
4.	32	<i>Uniform</i>	<i>Softmax</i>
5.	10	<i>Uniform</i>	<i>Softmax</i>

Untuk depth = 7

No. Layer	Jumlah Neuron	Initialization	Activation Function
1.	128	<i>Uniform</i>	<i>Softmax</i>
2.	128	<i>Uniform</i>	<i>Softmax</i>
3.	64	<i>Uniform</i>	<i>Softmax</i>
4.	64	<i>Uniform</i>	<i>Softmax</i>
5.	32	<i>Uniform</i>	<i>Softmax</i>
6.	32	<i>Uniform</i>	<i>Softmax</i>
7.	10	<i>Uniform</i>	<i>Softmax</i>

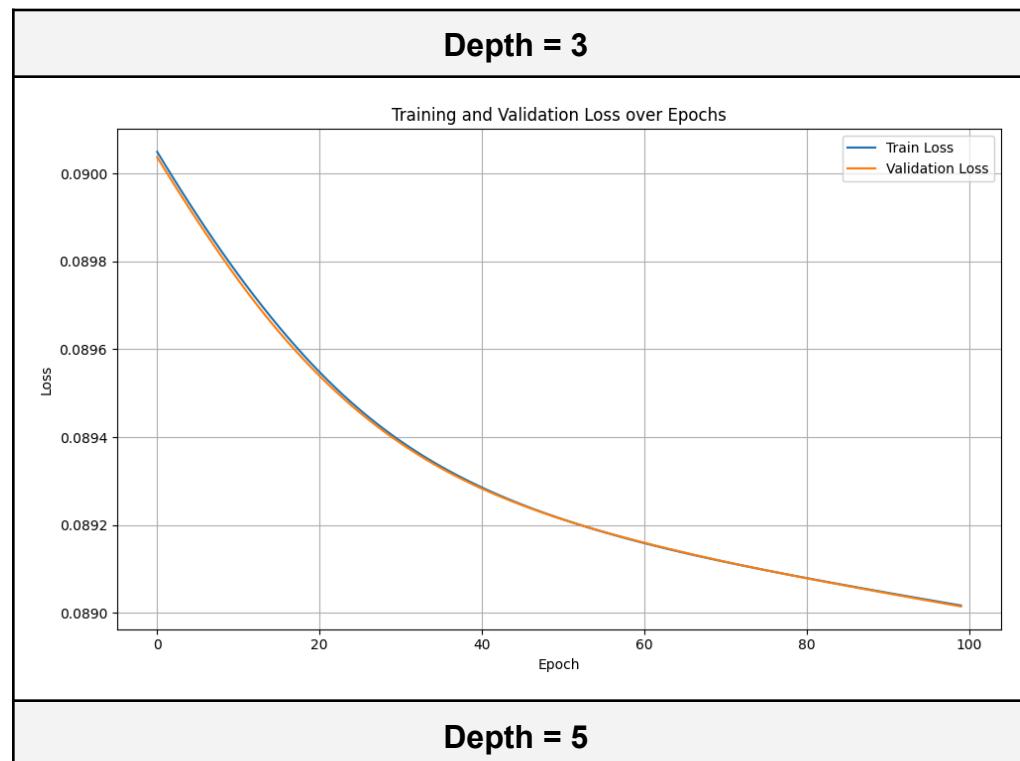
Perbandingan hasil:

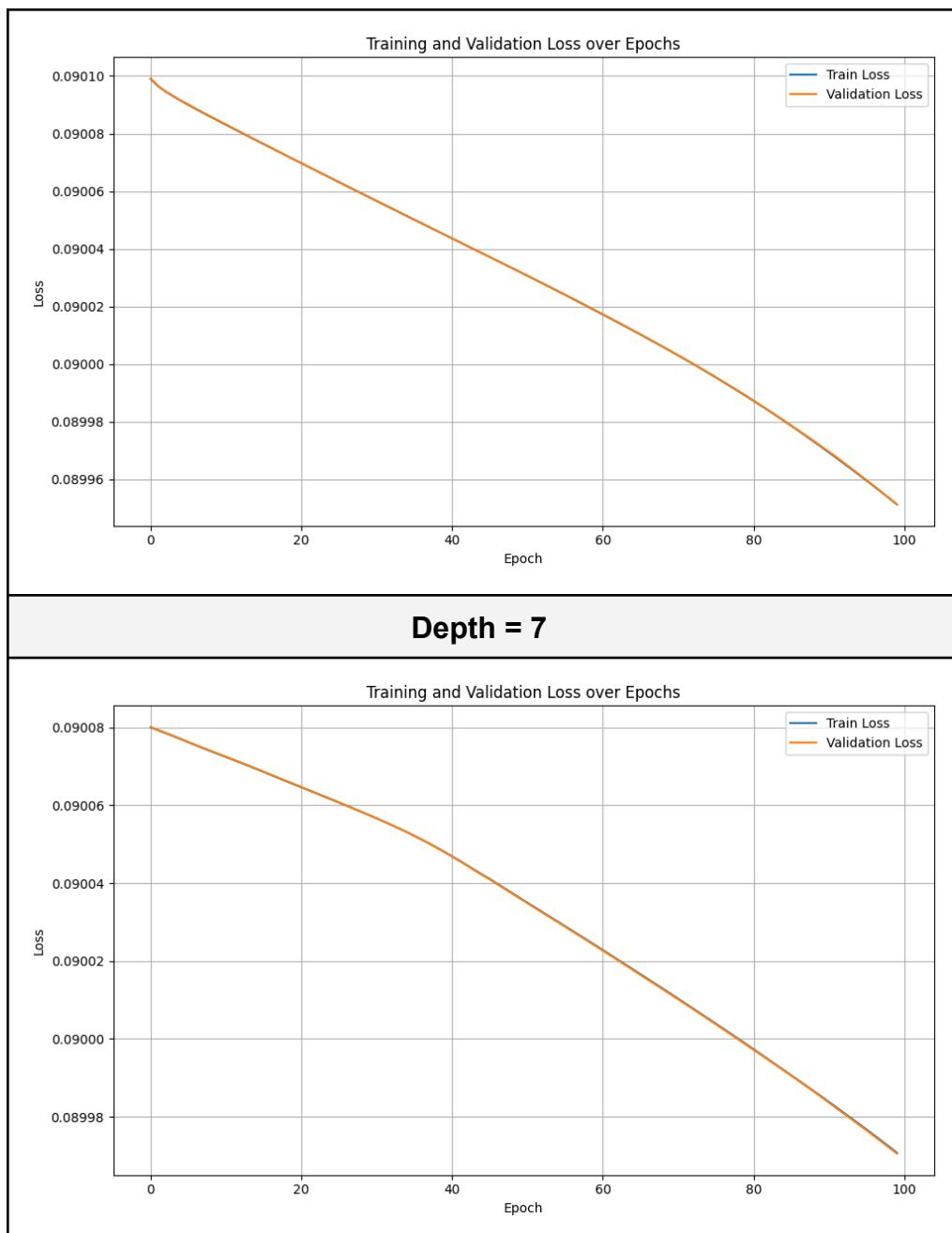
Aspek	Depth = 3	Depth = 5	Depth = 7
Hasil prediksi (10 data)	[4 4 4 4 4 4 4 1 4 4]	[6 7 6 6 7 6 6 6 6 6]	[3 3 3 3 3 3 3 3 3 3]

pertama dari data test)			
Target (10 data pertama dari data test)	[2 9 0 0 4 3 2 9 0 4]		
Akurasi (seluruh 5000 data test)	0.21	0.1818	0.102

Berdasarkan hasil percobaan, didapatkan bahwa model dengan *depth* 3 memiliki akurasi yang paling besar daripada model yang dilatih dengan *depth* 5 dan 7. Tetapi tetap perlu diperhatikan bahwa perbedaannya sangat sedikit dan seluruhnya memiliki akurasi yang terbilang kecil. Hal tersebut menunjukkan bahwa **penambahan kedalaman tidak selalu mendapatkan hasil yang lebih baik** dan tergantung pada aspek lainnya juga (seperti misalnya *activation function* yang digunakan, metode inisialisasi, dll.).

*Loss Curve:*





Berdasarkan hasil di atas dapat terlihat bahwa penurunan *loss* pada model dengan *depth* 3 terjadi lebih cepat daripada model dengan *depth* 5 dan 7. Semakin besar *depth* pada model, penurunan *loss* terjadi dengan lebih lambat. Hal tersebut terjadi karena semakin banyak *depth* pada model maka proses pembaruan bobot yang dilakukan juga lebih banyak sehingga diperlukan lebih banyak penyesuaian lagi. Namun, hal tersebut bisa saja dipengaruhi juga dari

aspek lain seperti metode inisialisasi dan *activation function* yang digunakan.

b. Width

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test. Variasi width yang dilakukan adalah 32, 64, dan 128.

Hyperparameter yang digunakan adalah sebagai berikut:

- Loss: MSE
- Learning Rate: 0.01
- Batch size: 200
- Epochs: 100

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron (berdasarkan width)	Initialization	Activation Function
1.	32/64/128	Uniform	Softmax
2.	32/64/128	Uniform	Softmax
3.	32/64/128	Uniform	Softmax
4.	10	Uniform	Softmax

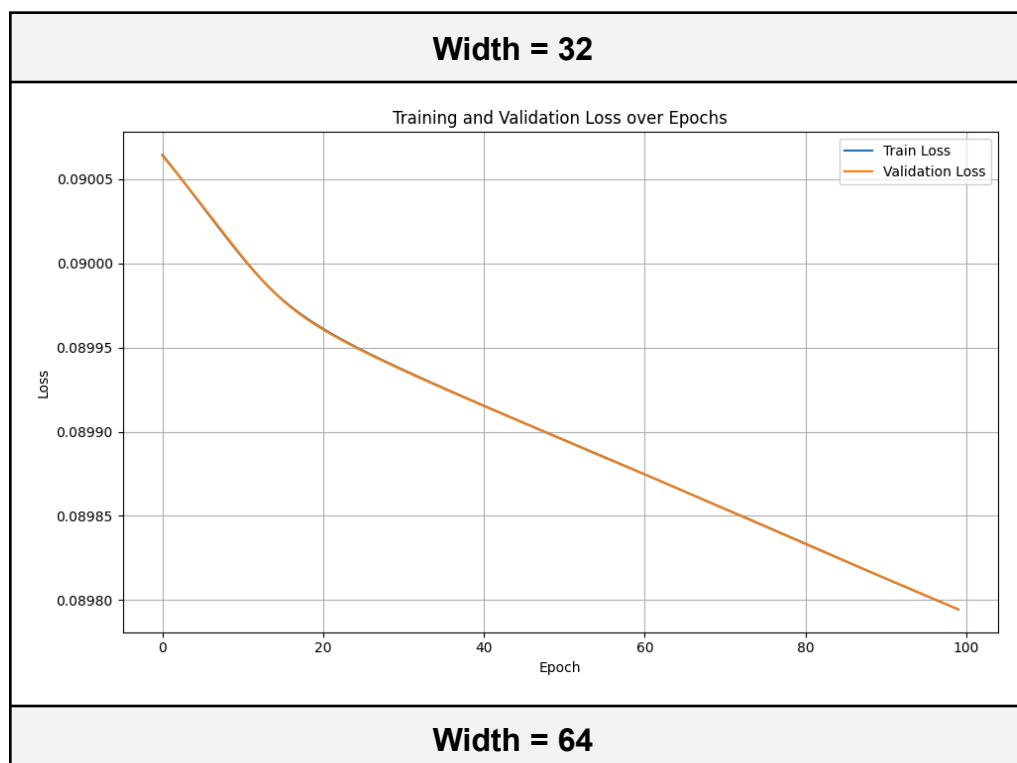
Perbandingan hasil:

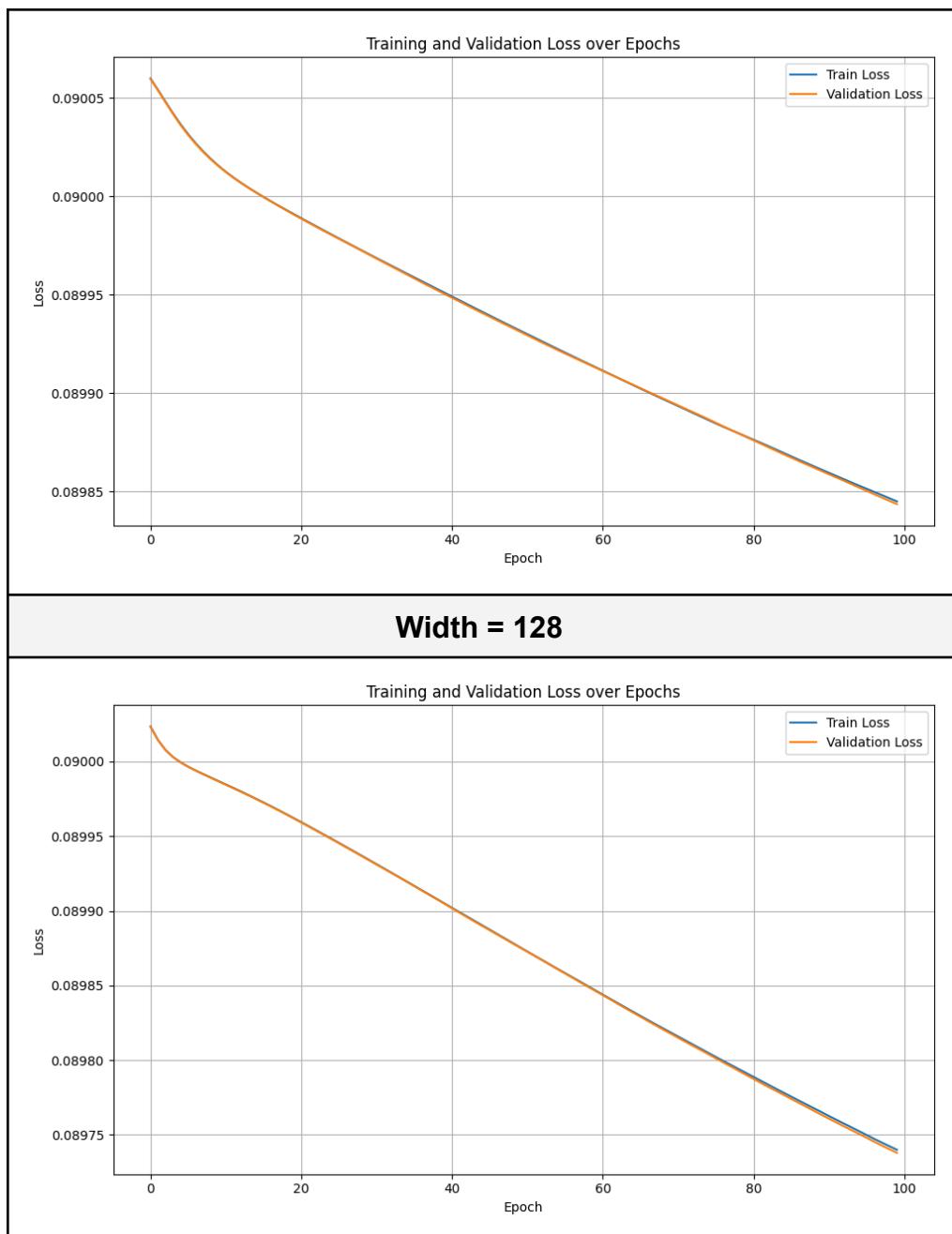
Aspek	Width = 32	Width = 64	Width = 128
Hasil prediksi (10 data pertama dari data <i>test</i> )	[1 1 1 1 1 1 1 1 1 1]	[6 6 6 6 6 6 6 6 6 6]	[9 9 9 9 9 9 9 9 9 9]
Target (10 data pertama)	[2 9 0 0 4 3 2 9 0 4]		

dari data test)			
Akurasi (seluruh 5000 data test)	0.1126	0.0982	0.0994

Berdasarkan hasil percobaan yang dilakukan didapatkan bahwa model dengan *width* 32 menghasilkan nilai akurasi yang lebih besar daripada model dengan *width* 64 dan 128. Namun, hasil akurasi dari *width* 64 tidak lebih baik daripada model dengan *width* 128. Jadi, tidak ada hubungan yang pasti antara akurasi yang dihasilkan model dengan jumlah *width* pada model.

Loss Curve:





Berdasarkan hasil percobaan di atas, dapat terlihat bahwa model dengan *width* 32 memiliki grafik penurunan *loss* yang lebih cepat daripada model dengan *width* 64 dan 128. Semakin besar *width* pada model, penurunan *loss* yang terjadi lebih lambat. Hal tersebut dapat dikarenakan semakin banyak *width* yang digunakan maka semakin banyak juga bobot yang perlu dilakukan perubahan sehingga penurunan *loss* yang terjadi cenderung lebih lambat.

### 2.2.2. Pengaruh Fungsi Aktivasi (Basic FFNN) → dengan RMSNorm

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test. Mekanisme fungsi aktivasi yang digunakan adalah linear, relu, leaky relu, elu, sigmoid, dan tanh. Pengujian dilakukan dengan menggunakan fungsi aktivasi yang sama pada semua *layer* kecuali *layer* terakhir selalu menggunakan fungsi aktivasi Softmax.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: CCE
- *Learning Rate*: 0.01
- *Batch size*: 200
- *Epochs*: 100

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron	Initialization	use_rmsnorm
1.	128	<i>He Uniform</i>	True
2.	64	<i>He Uniform</i>	True
3.	32	<i>He Uniform</i>	True
4.	10	<i>He Uniform</i>	True

Untuk layer terakhir, selalu digunakan fungsi aktivasi **softmax**.

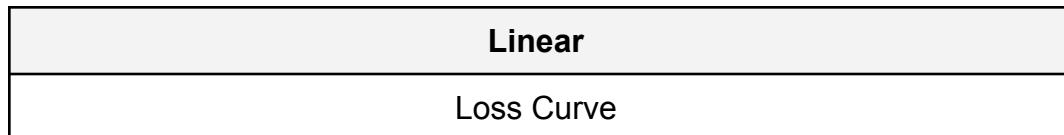
Perbandingan hasil:

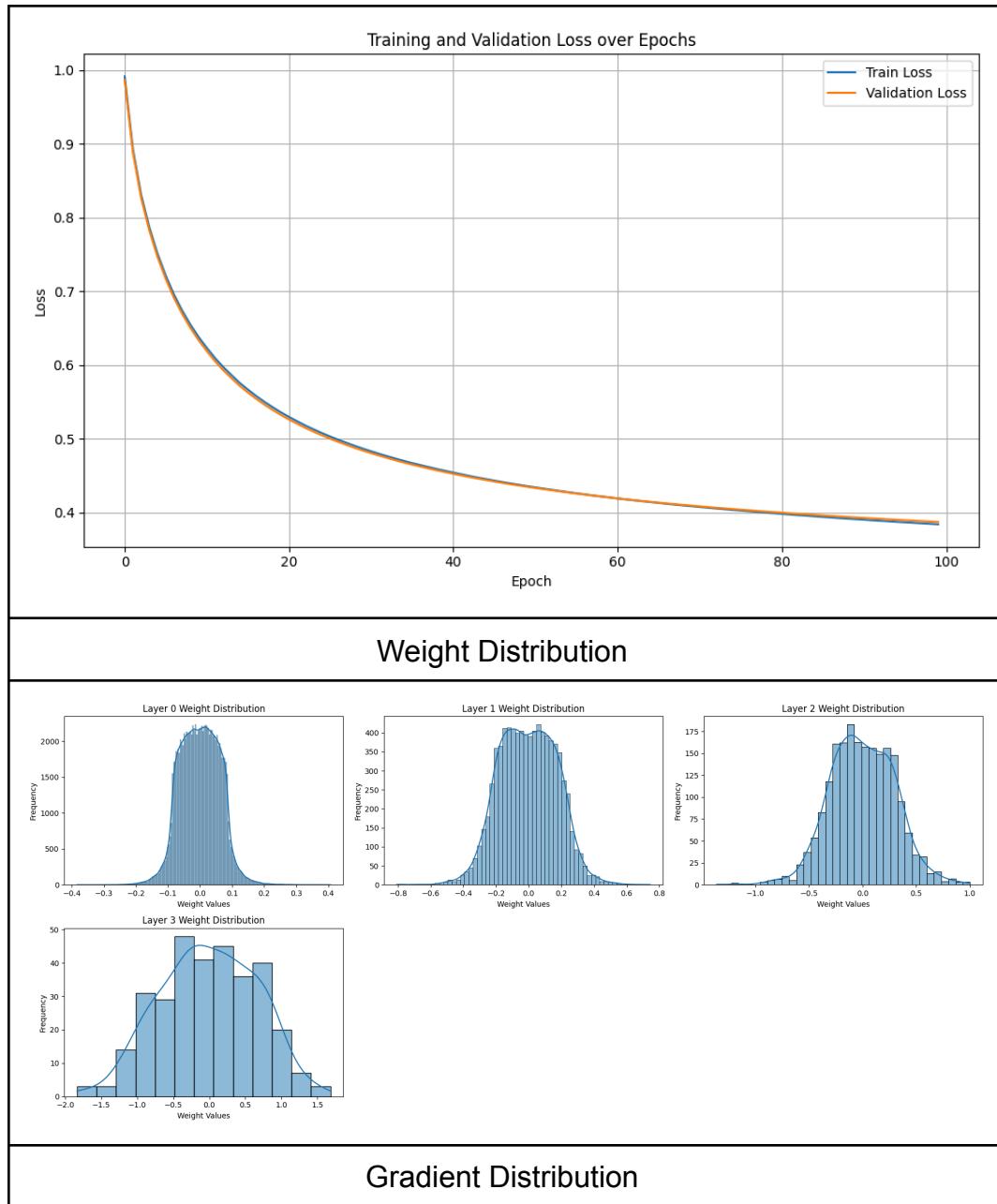
Jenis Fungsi Aktivasi	Hasil prediksi (10 data pertama dari data test)	Target (10 data pertama dari data test)	Akurasi (seluruh 5000 data test)

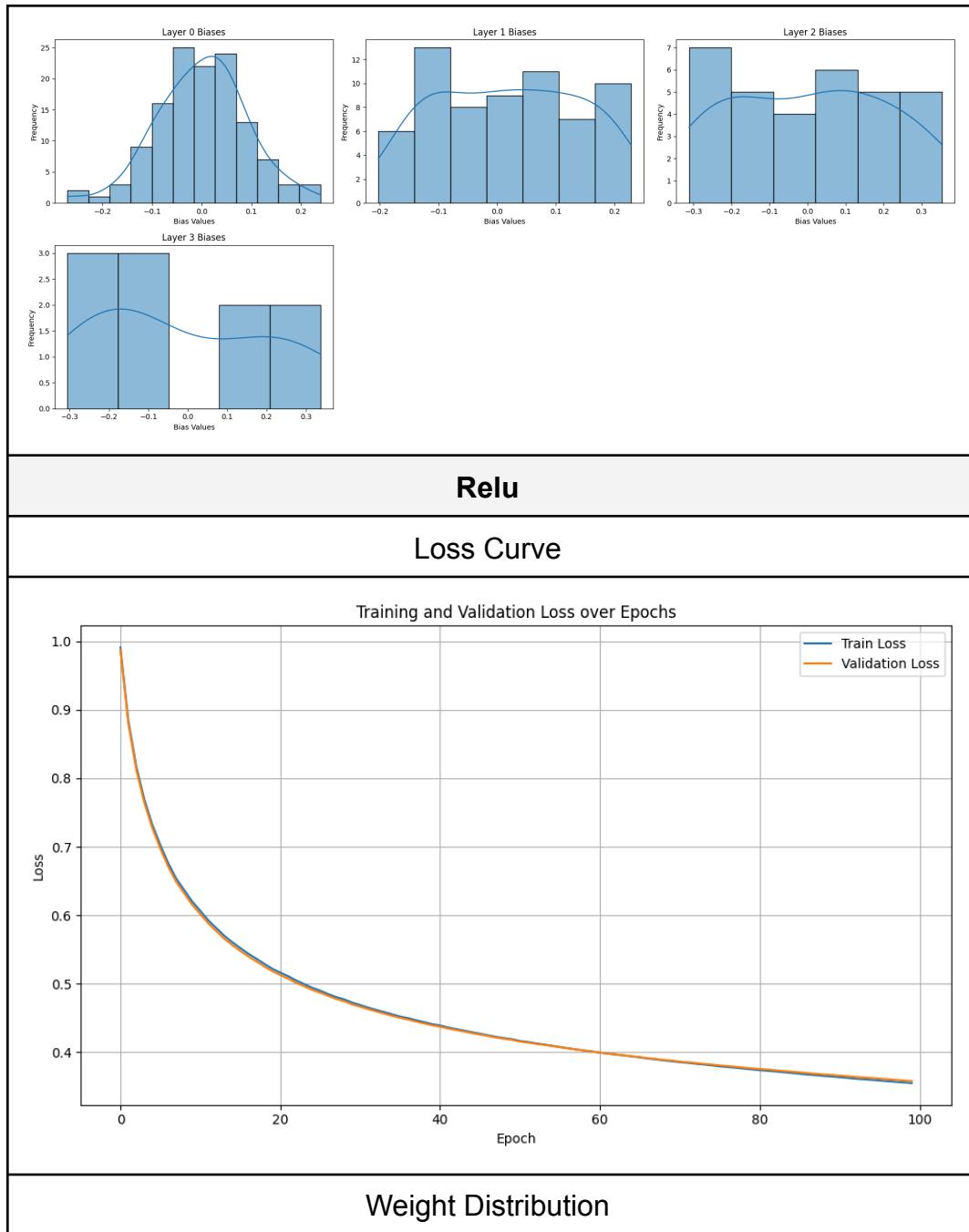
Linear	[0 9 0 0 4 3 2 7 0 4]	[2 9 0 0 4 3 2 9 0 4]	0.8948
Relu	[0 9 0 0 4 3 2 1 0 4]		0.9004
Leaky Relu	[0 9 0 0 4 3 2 1 0 4]		0.8834
Elu	[0 9 0 0 4 3 2 1 0 4]		0.8948
Sigmoid	[0 9 0 0 4 3 2 1 0 4]		0.9276
Tanh	[2 9 0 0 4 3 2 9 0 4]		0.9690
Softmax	[1 1 1 1 1 1 1 1 1 1]		0.1126

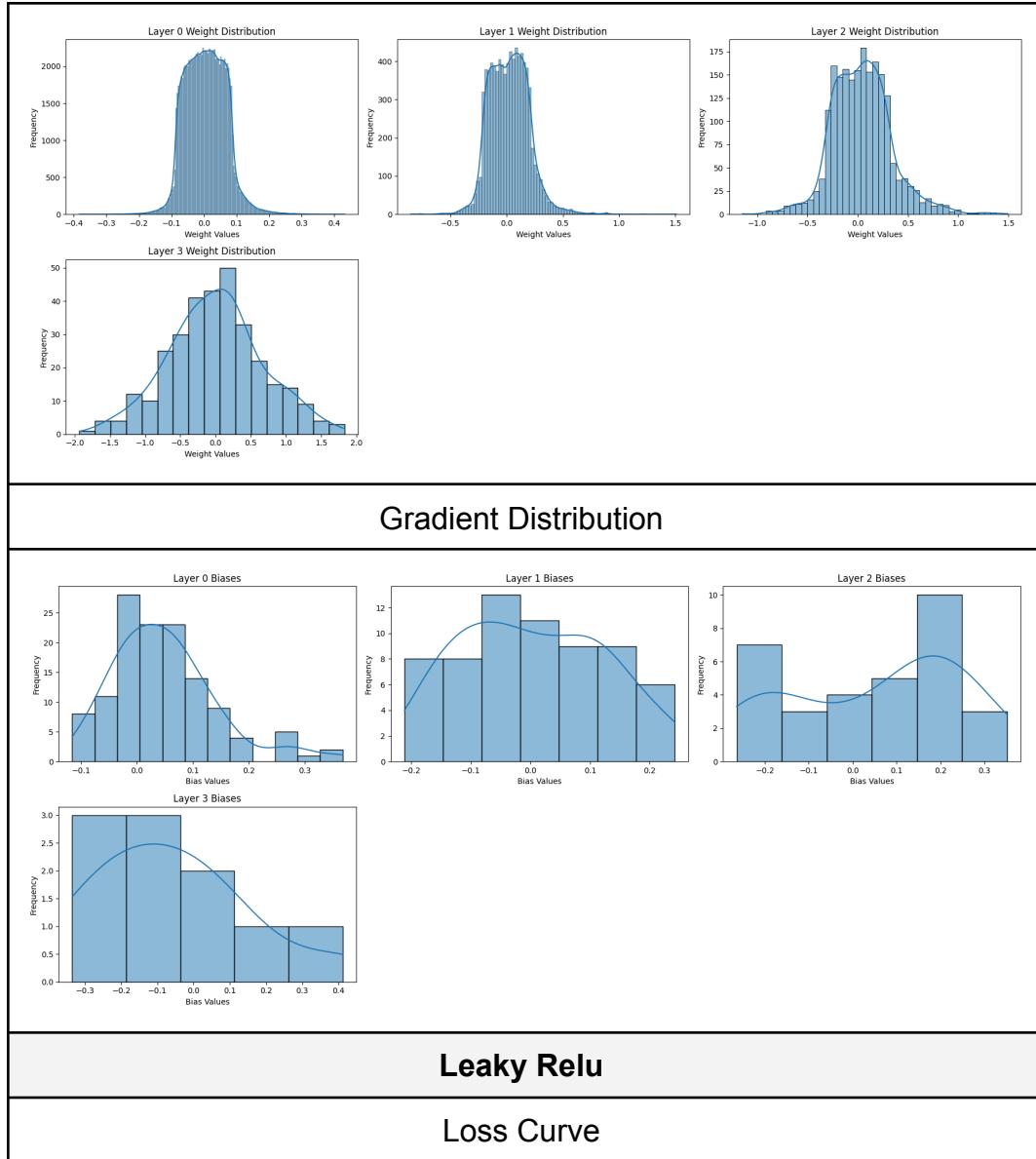
Berdasarkan hasil percobaan, *activation function* yang memiliki hasil akhir akurasi bernilai lebih besar dari 0.9 adalah Tanh (0.9690), Sigmoid (0.9276), dan Relu (0.9004). *Activation function* Softmax memiliki nilai akurasi paling kecil karena fungsi aktivasi Softmax tidak cocok dijadikan sebagai fungsi aktivasi pada *hidden layer*, Softmax lebih cocok jika digunakan pada *layer* terakhir saja.

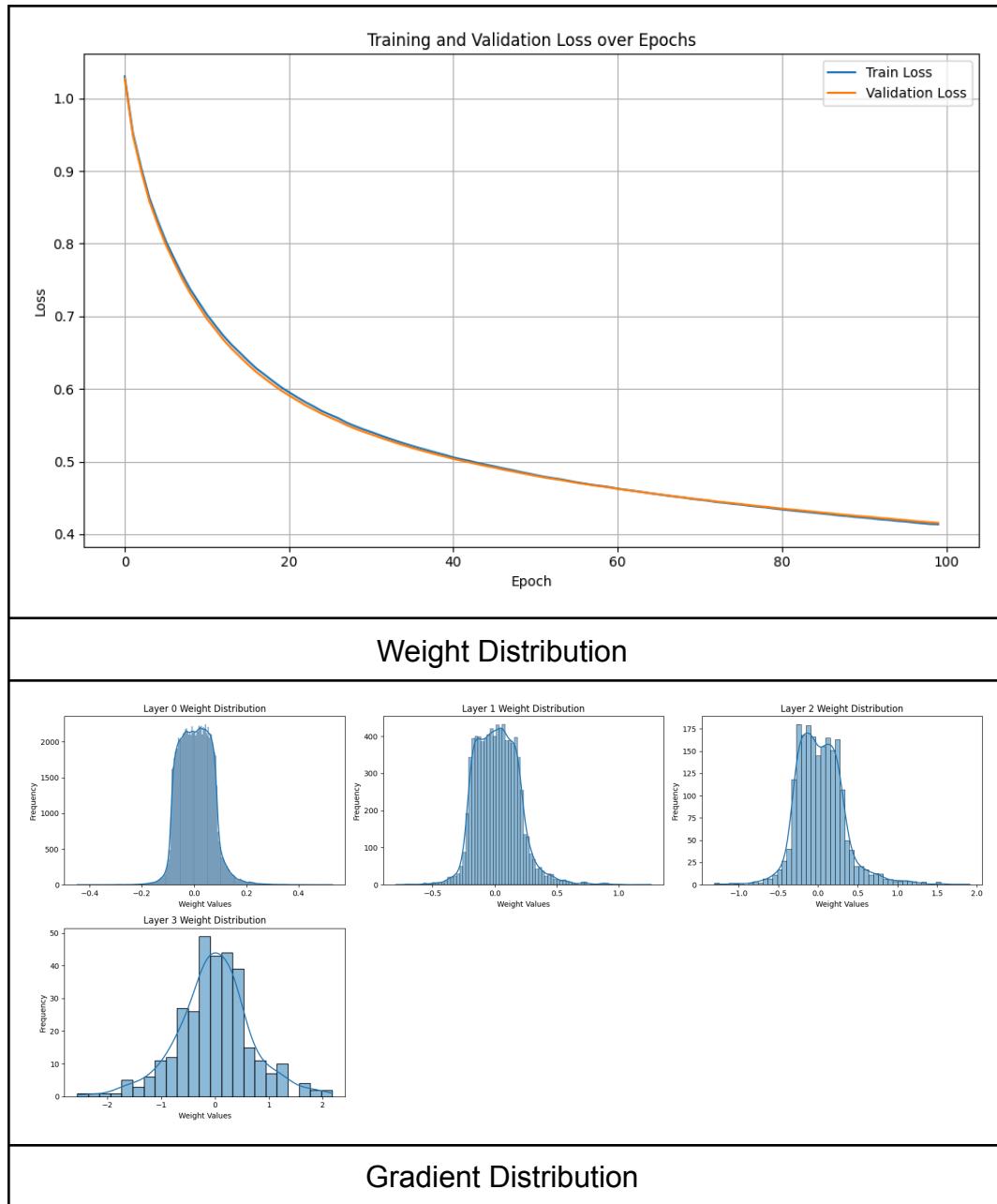
Perbandingan plot:

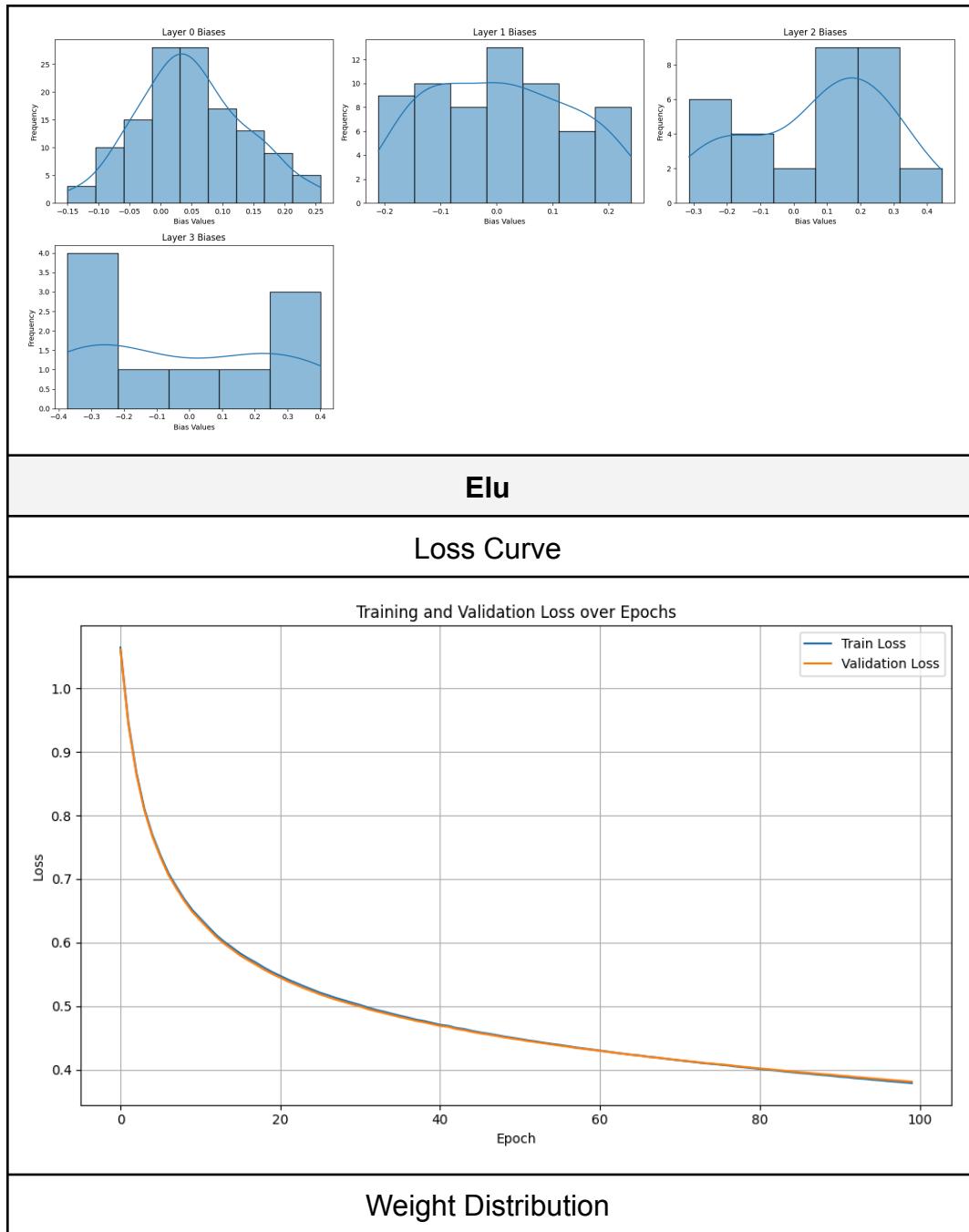


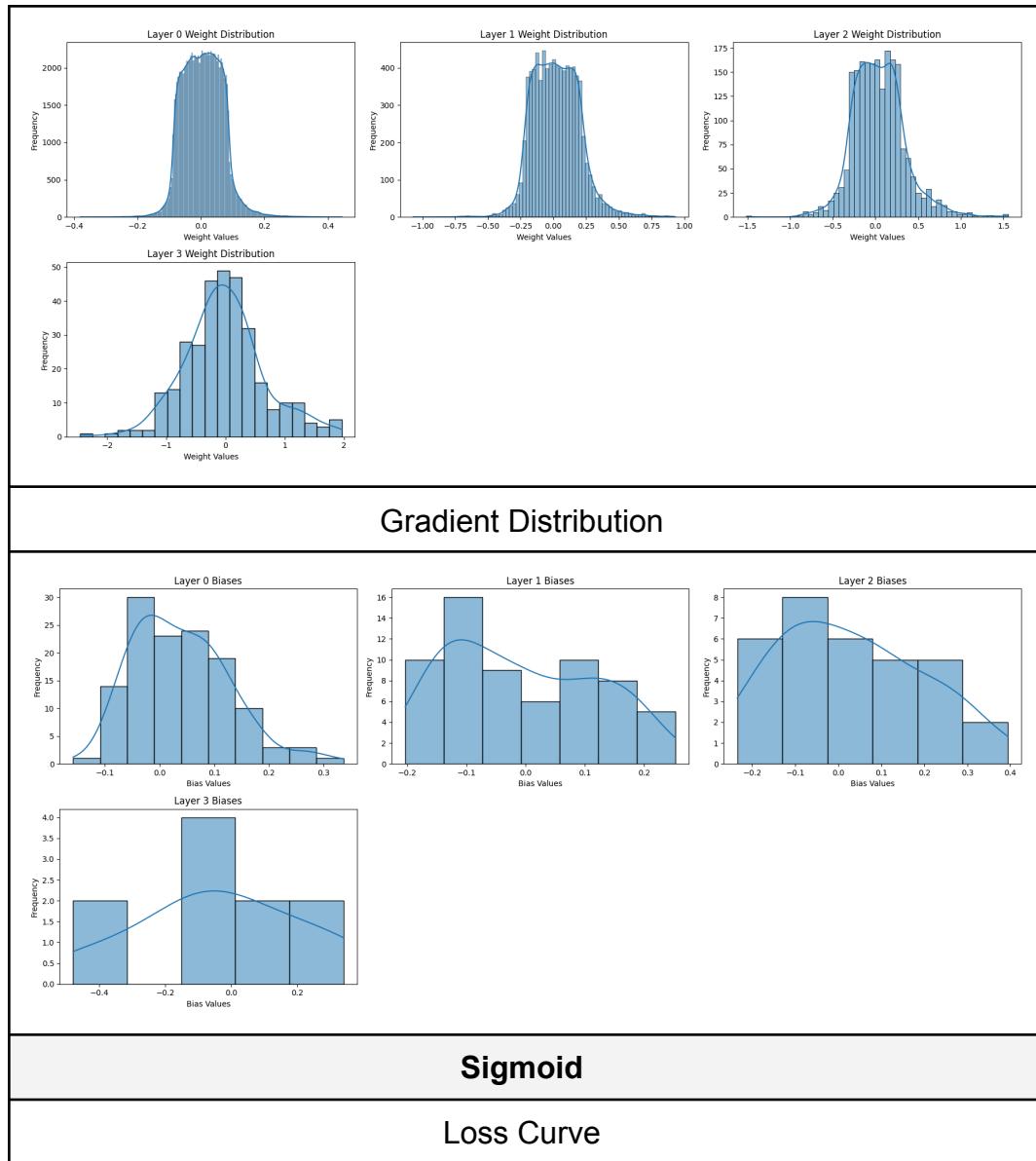


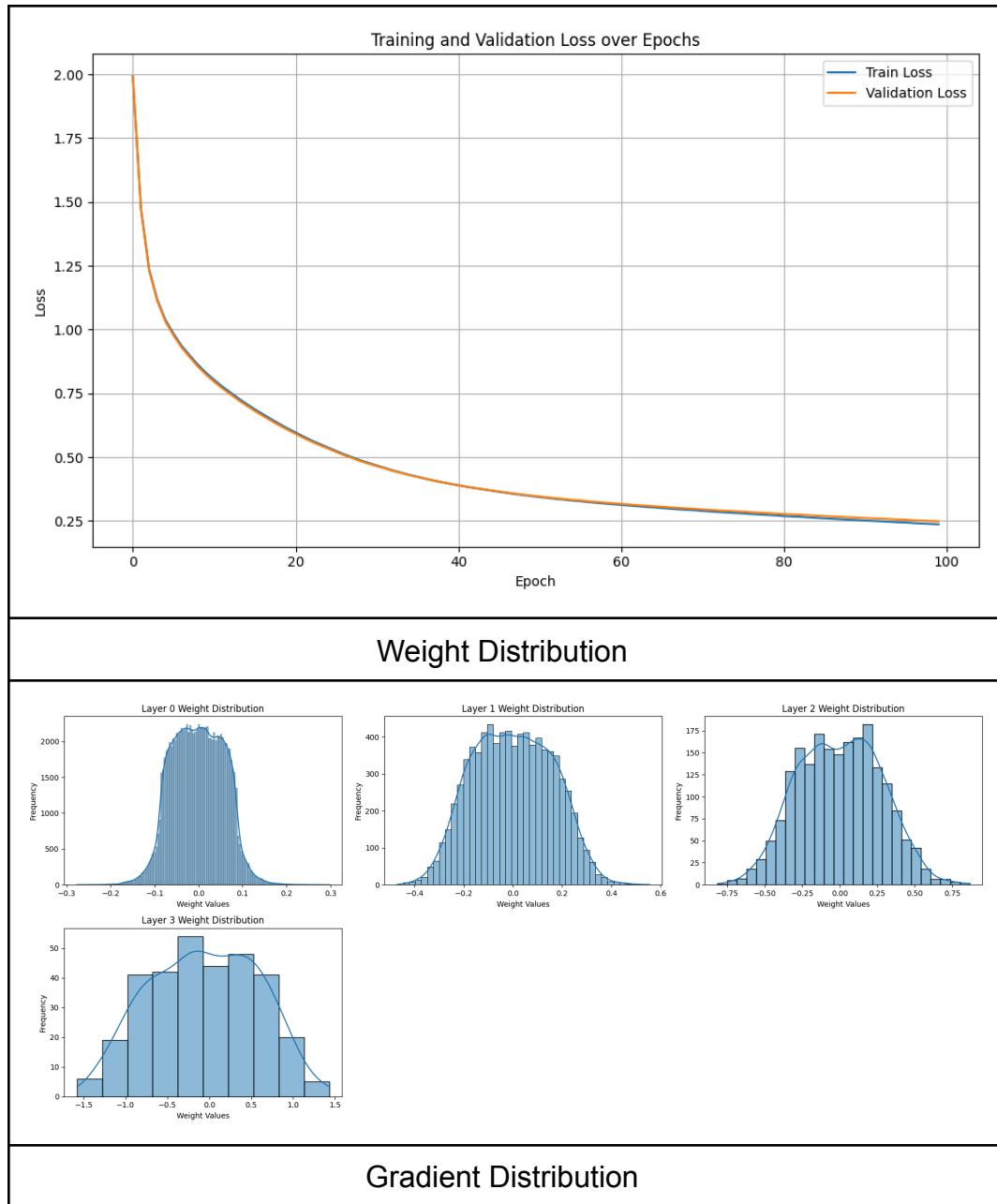


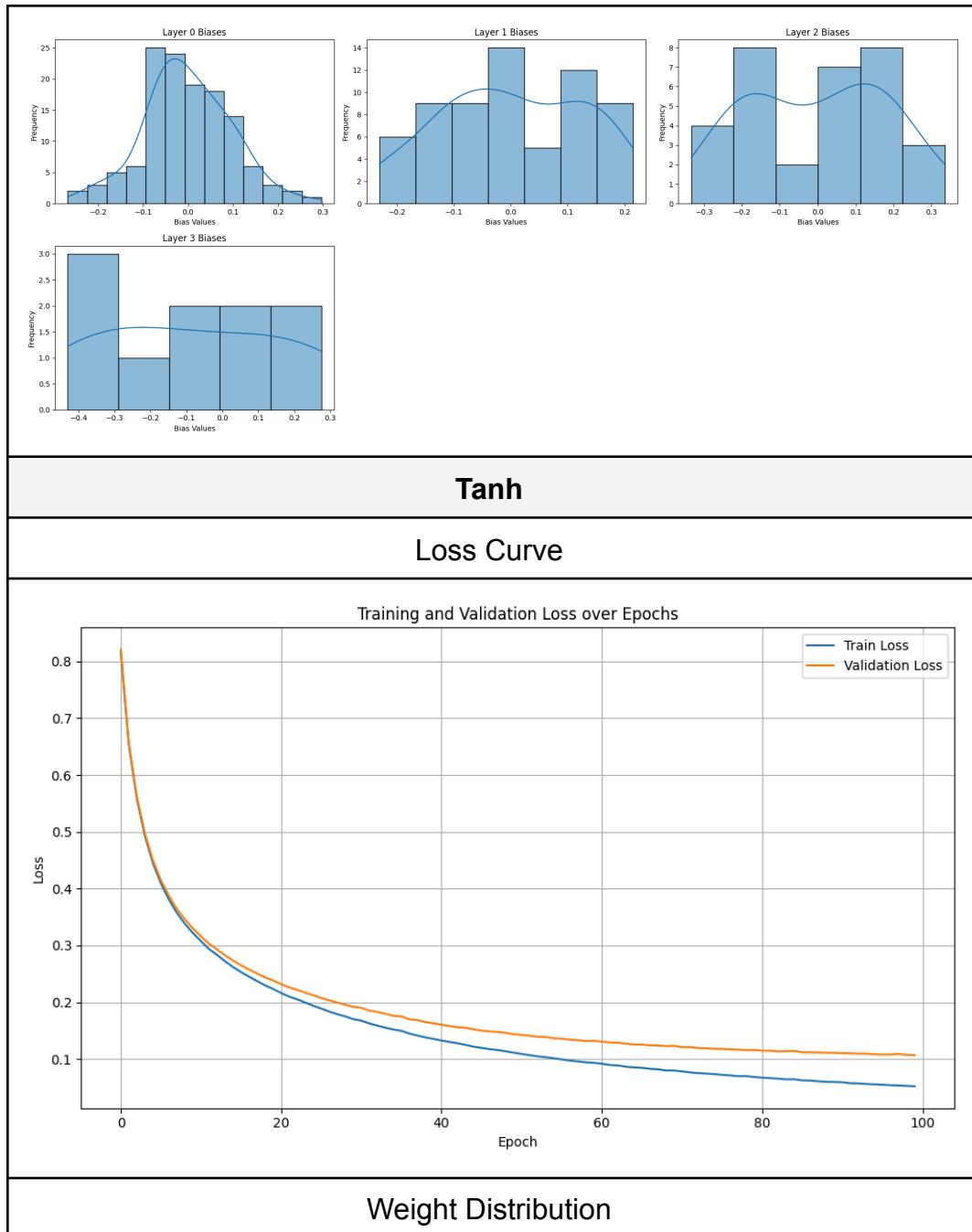


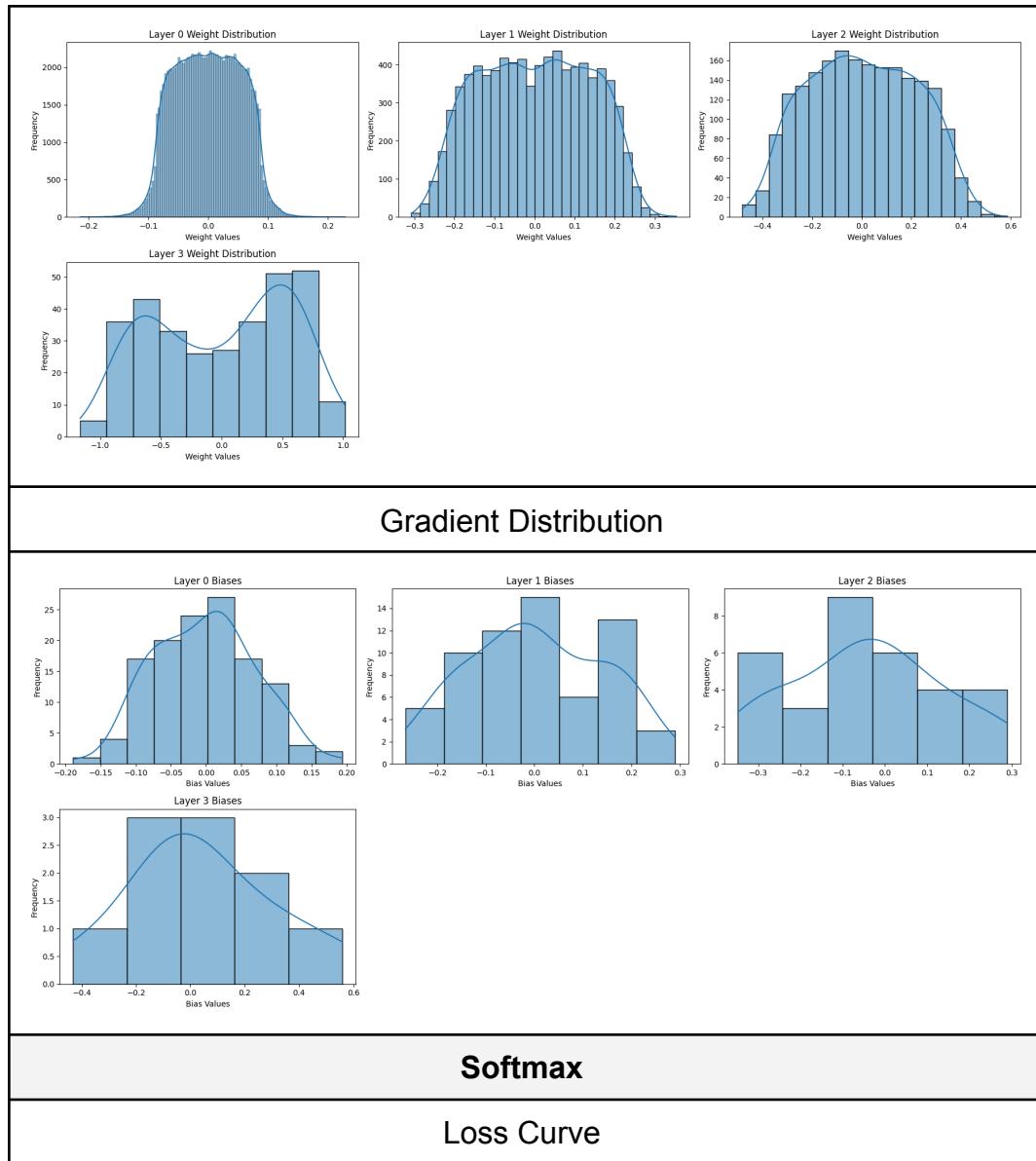














Berdasarkan hasil percobaan, hampir seluruh fungsi aktivasi menghasilkan grafik *loss* yang mengalami penurunan dengan cukup cepat pada *epoch* awal kecuali pada fungsi aktivasi Softmax. Pada grafik *loss* pada fungsi aktivasi Softmax dapat terlihat bahwa pergerakan *loss* tidak semakin berkurang pada akhirnya justru kembali seperti awal mula, hal tersebut karena fungsi aktivasi Softmax kurang tepat untuk digunakan pada *hidden layer* selain pada *layer* terakhir.

Distribusi bobot dan gradien bobot dari percobaan di atas pada umumnya terdistribusi secara normal. Namun, terdapat beberapa bobot dan gradien bobot yang *skew left* ataupun *skew right*.

### 2.2.3. Pengaruh Fungsi Aktivasi (Basic FFNN) → tanpa RMSNorm

Pada bagian ini, yang akan dibahas adalah pengaruh **RMSNorm**.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: CCE
- *Learning Rate*: 0.01
- *Batch size*: 200
- *Epochs*: 100

Arsitektur yang digunakan adalah sebagai berikut:

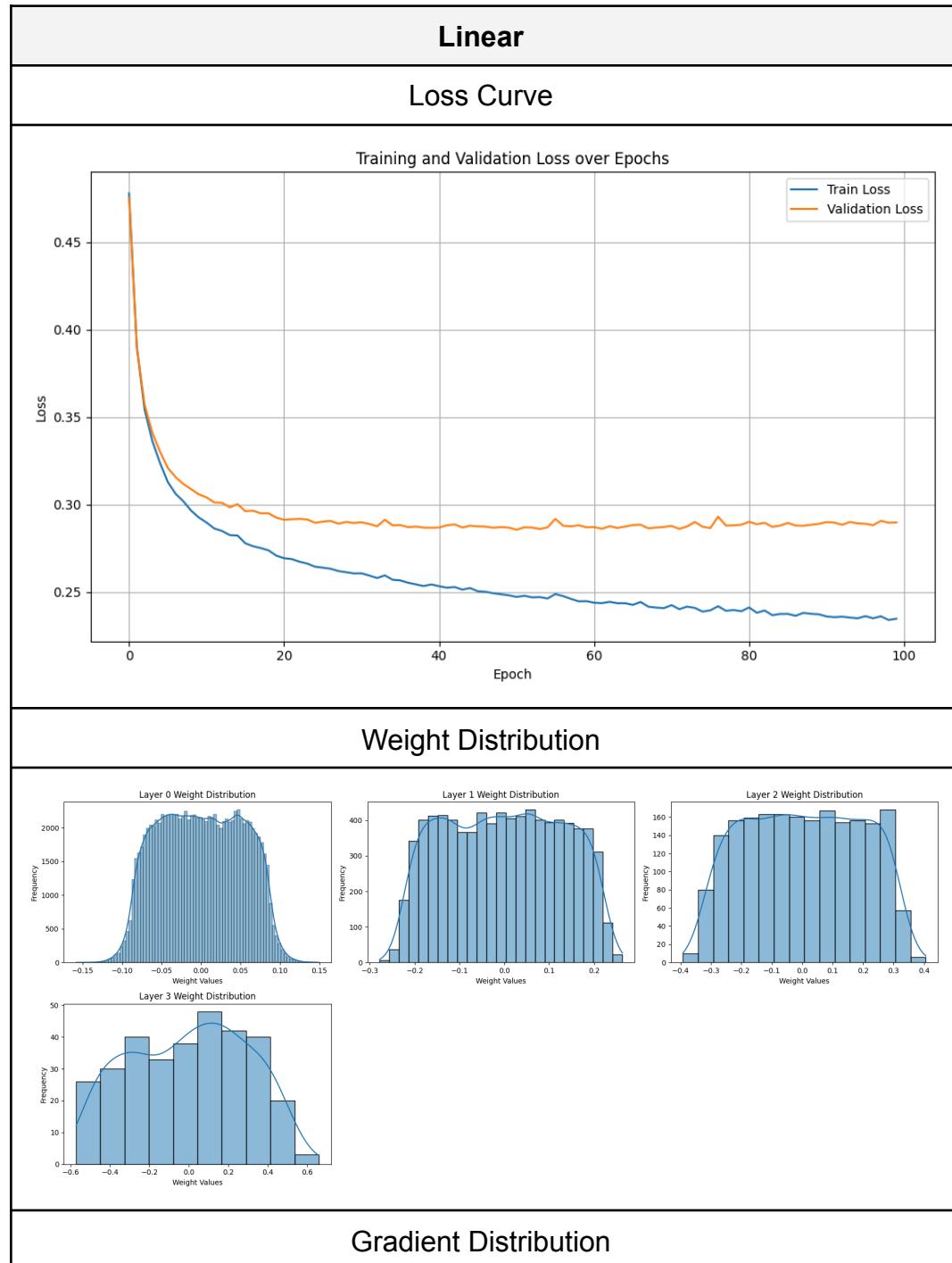
No. Layer	Jumlah Neuron	Initialization	use_rmsnorm
1.	128	<i>He Uniform</i>	<i>False</i>
2.	64	<i>He Uniform</i>	<i>False</i>
3.	32	<i>He Uniform</i>	<i>False</i>
4.	10	<i>He Uniform</i>	<i>False</i>

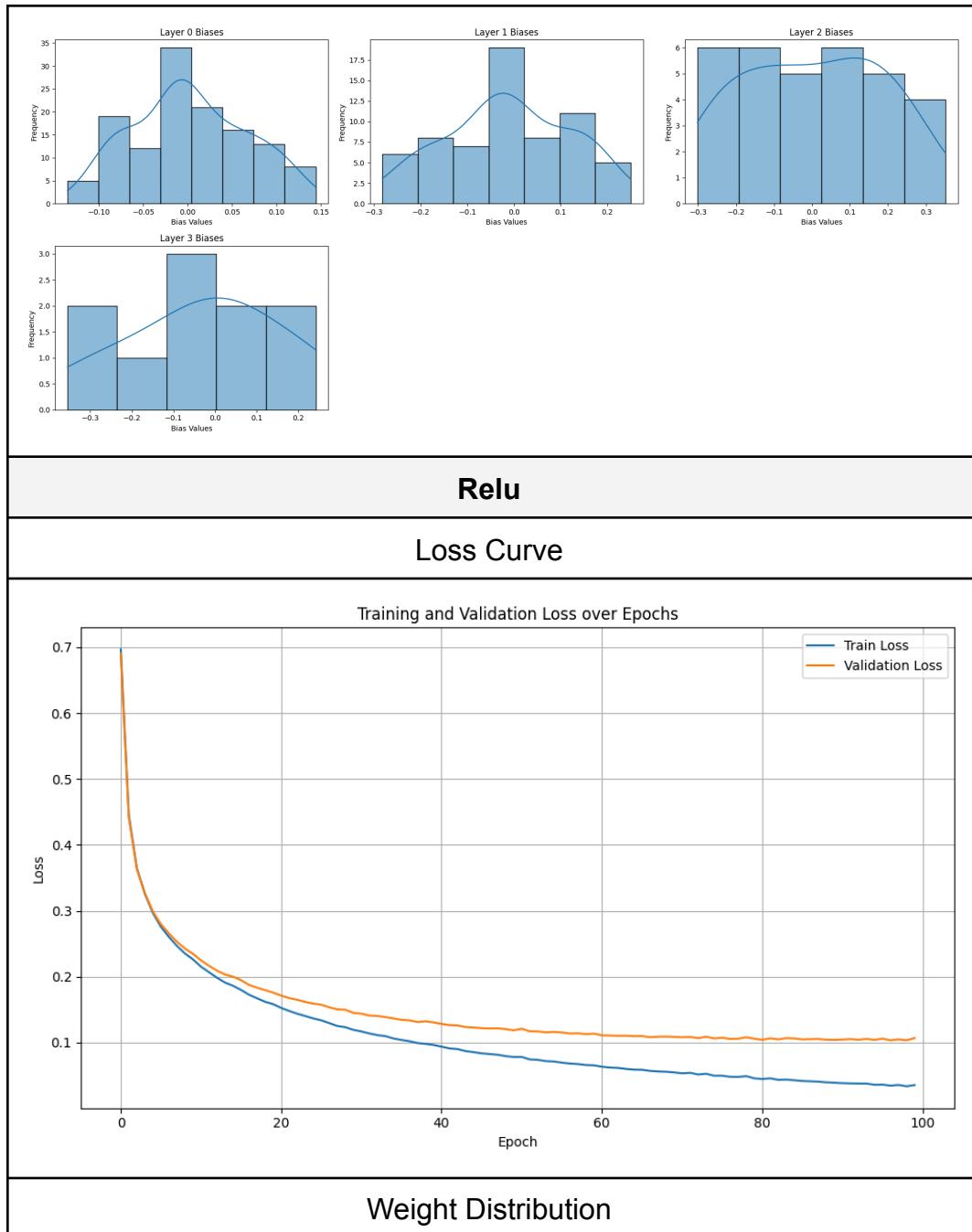
Untuk layer terakhir, selalu digunakan fungsi aktivasi **softmax**.

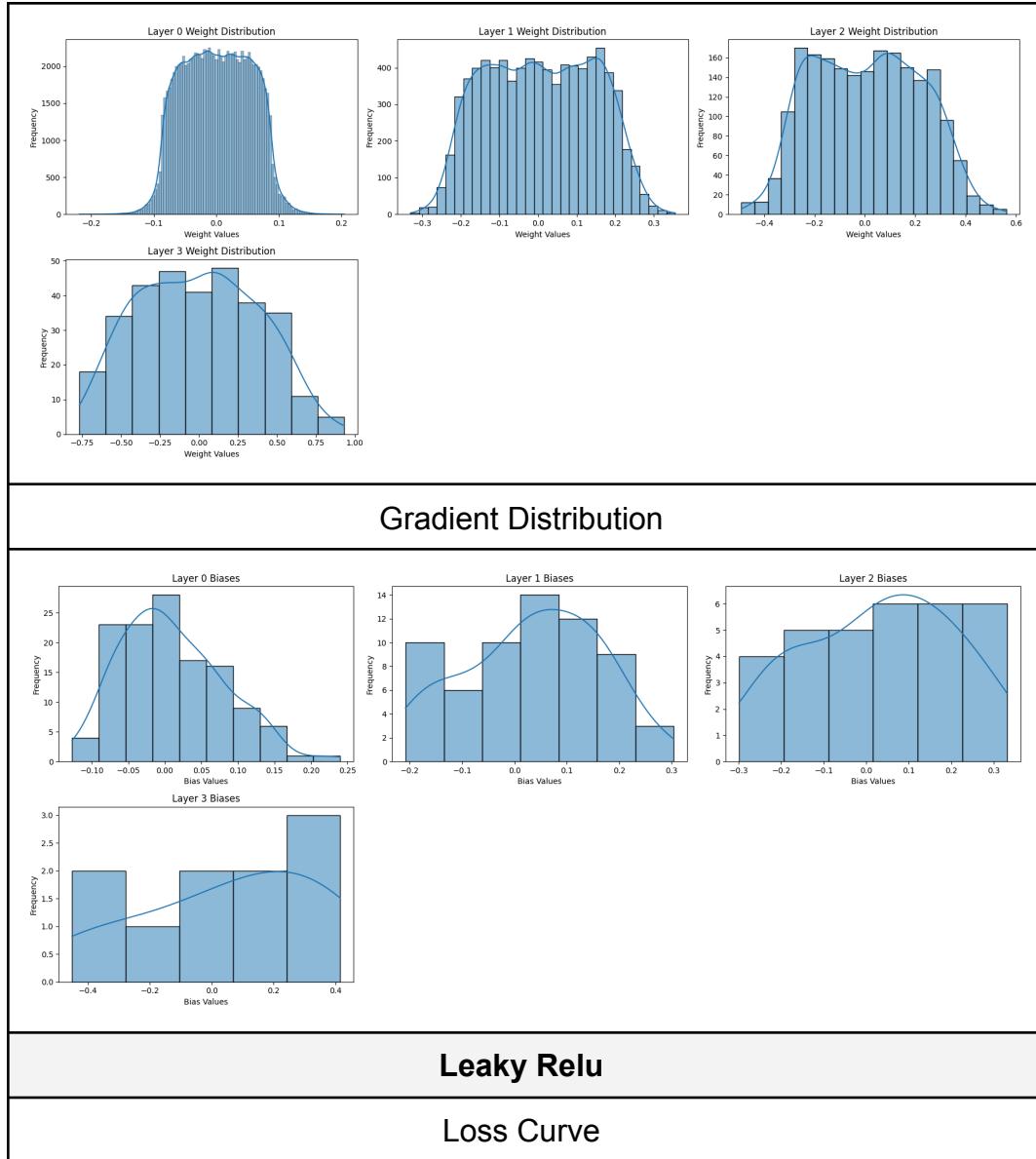
Perbandingan hasil:

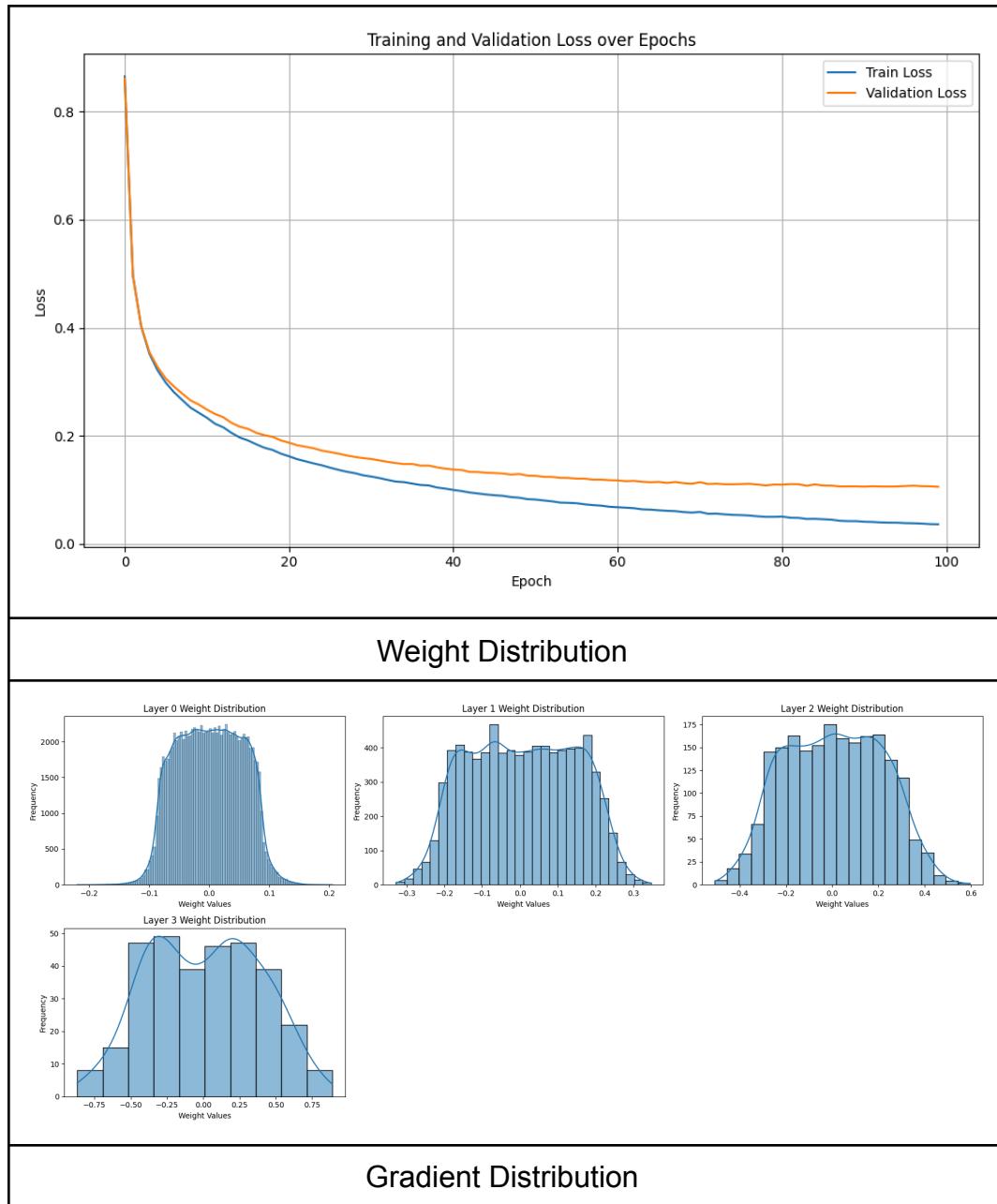
Jenis Fungsi Aktivasi	Hasil prediksi (10 data pertama dari data test)	Target (10 data pertama dari data test)	Akurasi (seluruh 5000 data test)
Linear	[2 9 0 0 4 3 2 1 0 4]	[2 9 0 0 4 3 2 9 0 4]	0.9238
Relu	[2 9 0 0 4 3 2 9 0 4]		0.9716
Leaky Relu	[2 9 0 0 4 3 2 9 0 4]		0.9706
Elu	[2 9 0 0 4 3 2 9 0 4]		0.9720
Sigmoid	[0 9 0 0 4 3 2 1 0 4]		0.8764
Tanh	[5 9 0 0 4 3 2 9 0 4]		0.9662
Softmax	[0 0 0 0 0 1 1 1 0 0]		0.2078

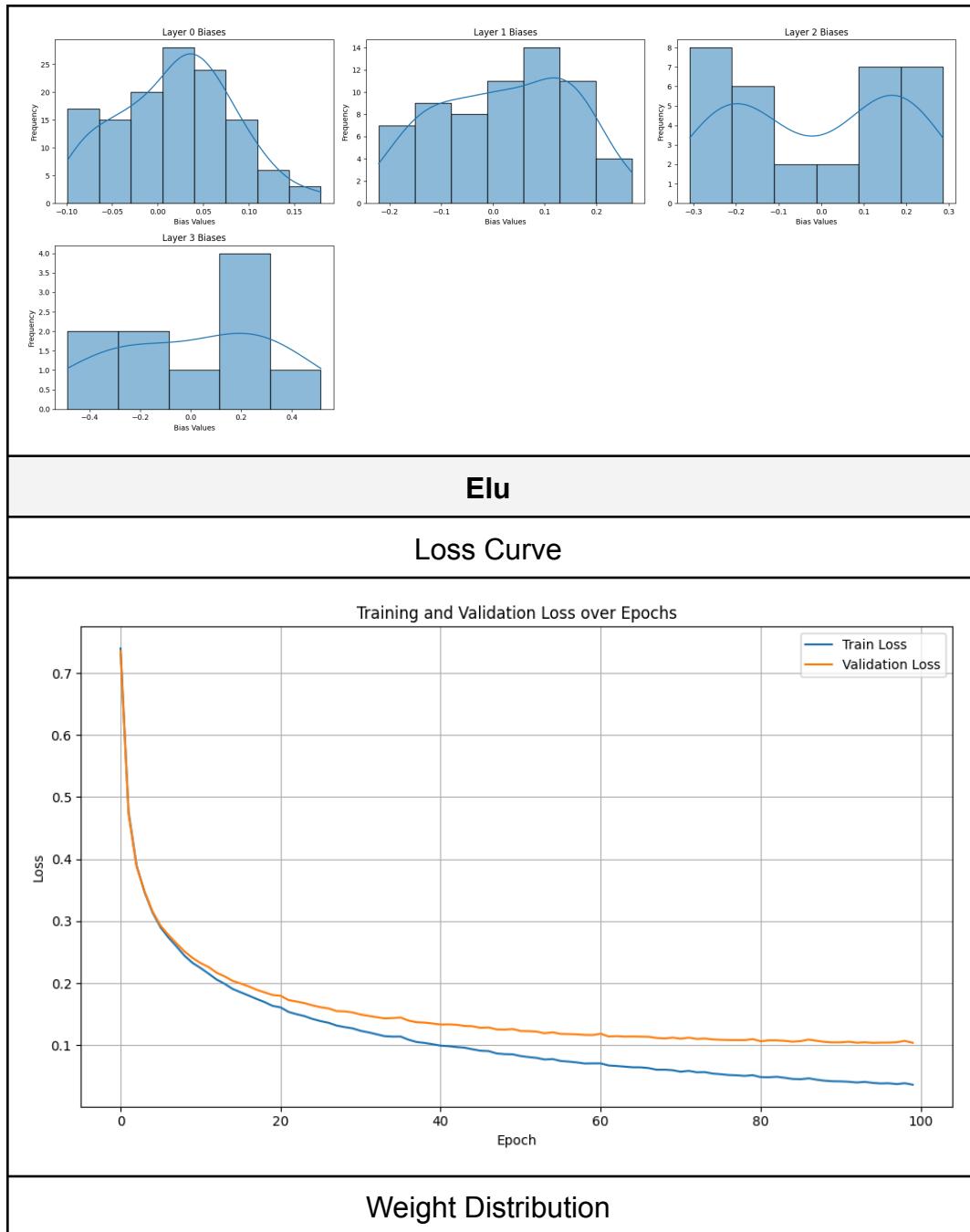
Perbandingan plot:

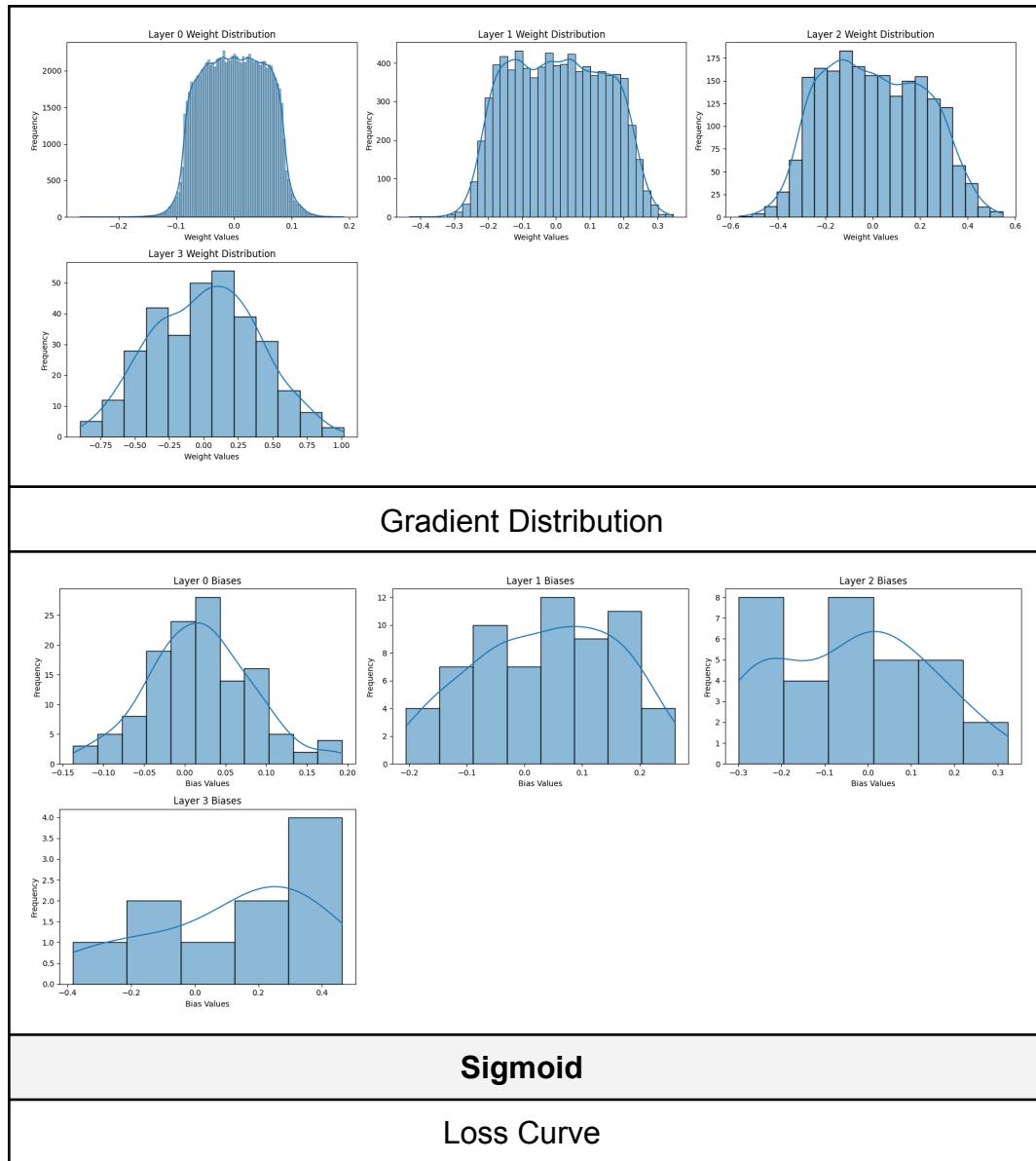


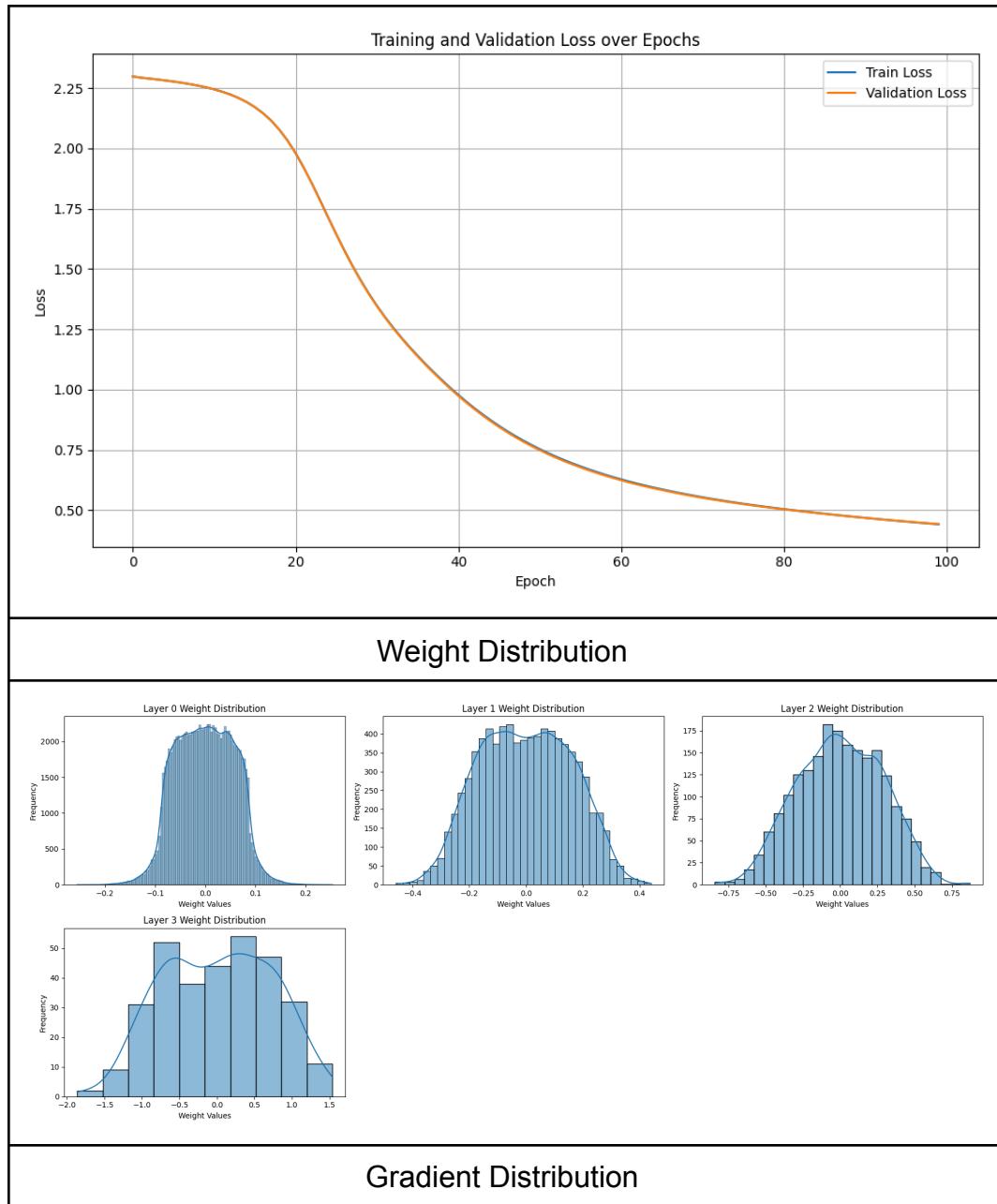


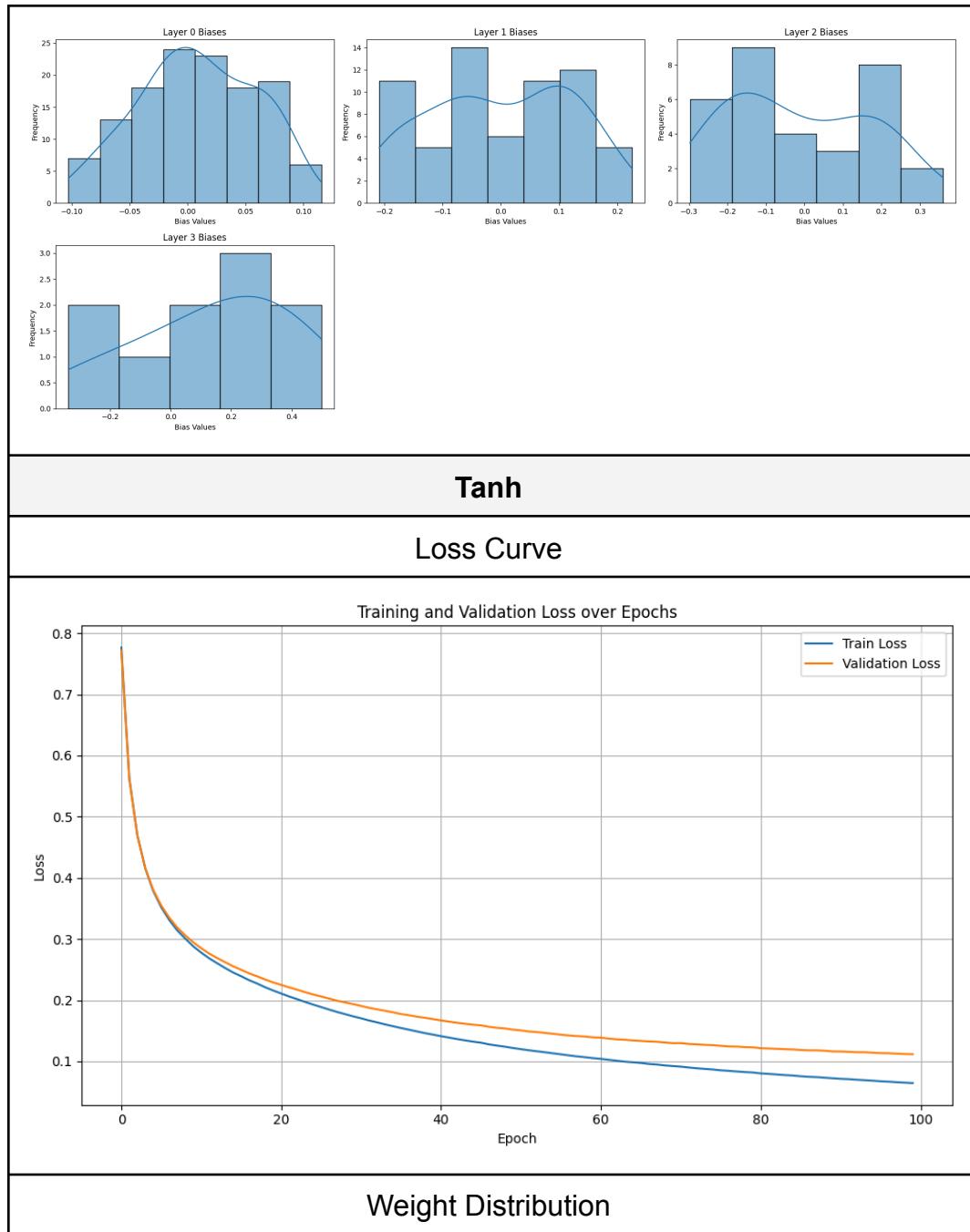


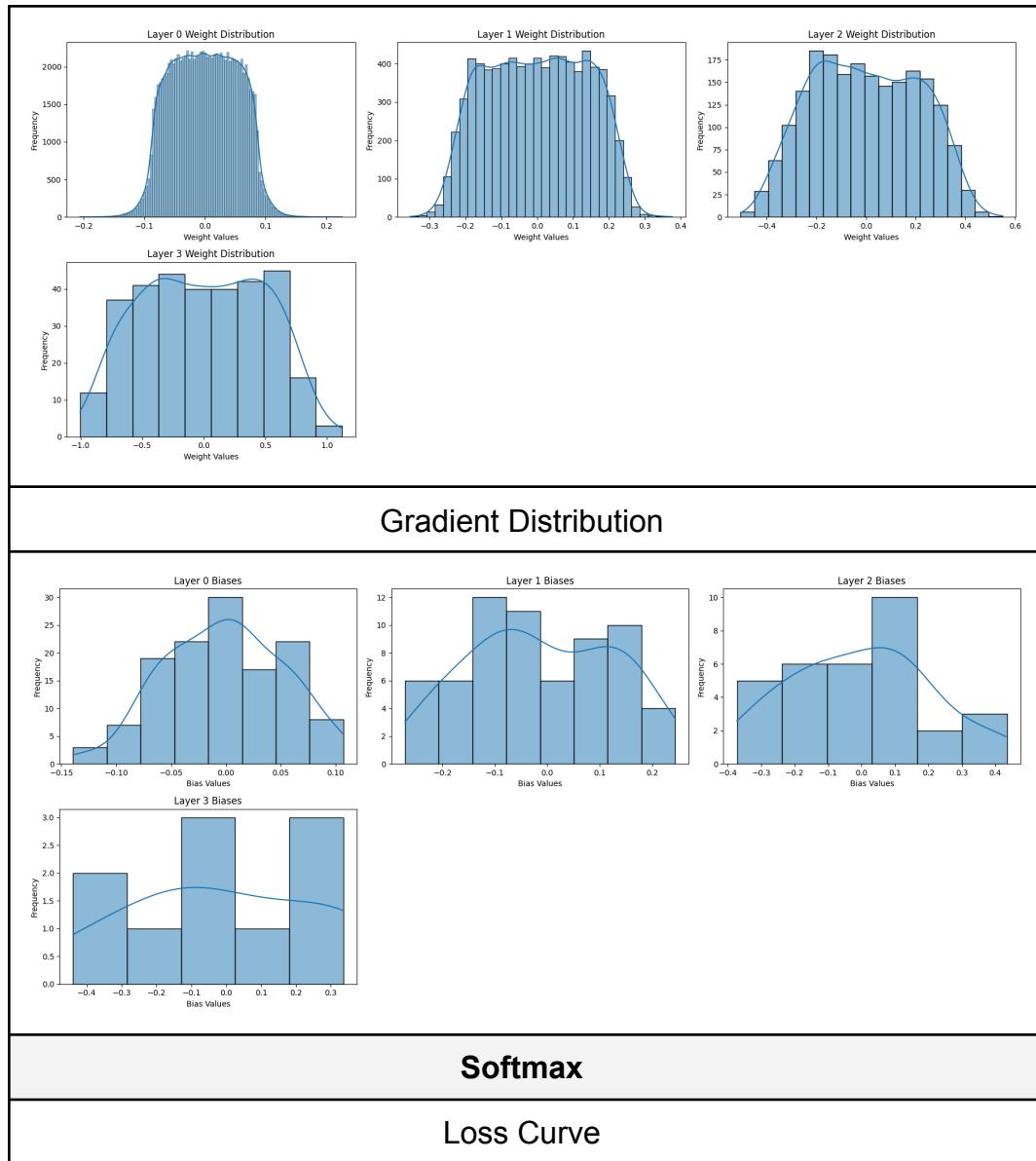


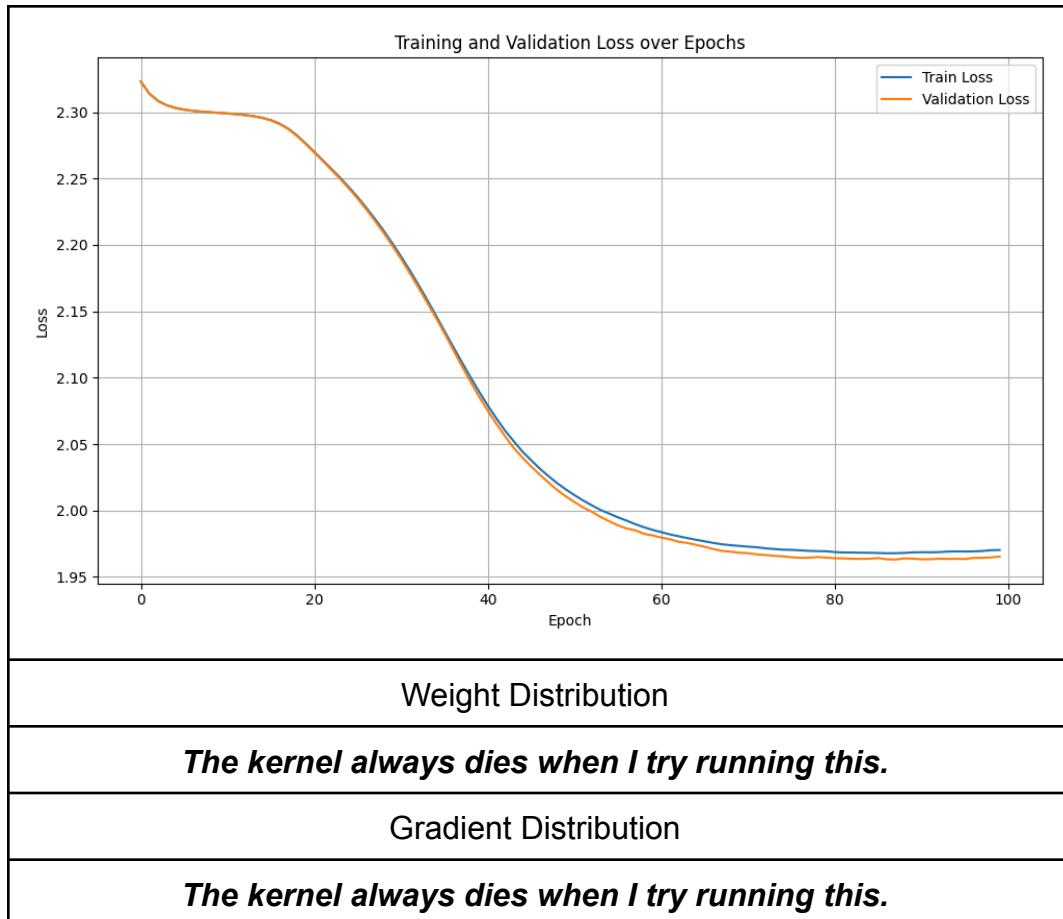












Dalam percobaan dengan RMSNorm, hasil akurasi menunjukkan bahwa fungsi aktivasi yang menghasilkan akurasi lebih dari 0.9 adalah Tanh (0.9690), Sigmoid (0.9276), dan Relu (0.9004). Akurasi yang relatif tinggi pada Tanh dan Sigmoid mengindikasikan bahwa kedua fungsi ini berfungsi dengan baik dalam pengaturan model yang menggunakan RMSNorm. Namun, Softmax sebagai fungsi aktivasi di layer tersembunyi menunjukkan hasil yang sangat buruk, dengan akurasi hanya 0.1126, yang menegaskan bahwa Softmax sebaiknya hanya digunakan pada layer terakhir. Sementara itu, pada percobaan tanpa RMSNorm, beberapa fungsi aktivasi, seperti Relu (0.9716), Leaky Relu (0.9706), dan Elu (0.9720), memberikan hasil yang lebih baik dibandingkan dengan eksperimen dengan RMSNorm, menunjukkan bahwa tanpa RMSNorm, model dapat memanfaatkan fungsi aktivasi tersebut dengan lebih optimal.

Softmax, meskipun lebih baik dari eksperimen dengan RMSNorm, masih memiliki akurasi rendah (0.2078).

Pada eksperimen dengan RMSNorm, fungsi aktivasi seperti Sigmoid dan Tanh menunjukkan hasil yang lebih baik dalam hal akurasi dibandingkan dengan fungsi aktivasi lainnya, seperti Leaky Relu dan Elu, meskipun akurasi keseluruhan pada Leaky Relu dan Relu masih cukup baik. Hal ini menunjukkan bahwa RMSNorm mungkin lebih cocok untuk beberapa fungsi aktivasi yang membutuhkan penyeimbangan dan stabilisasi lebih lanjut pada proses pelatihan. Sebaliknya, pada eksperimen tanpa RMSNorm, fungsi aktivasi seperti Relu, Leaky Relu, dan Elu memberikan hasil yang lebih baik, dengan Relu menunjukkan akurasi tertinggi di antara fungsi aktivasi lainnya. Ini mengindikasikan bahwa tanpa adanya RMSNorm, fungsi aktivasi tersebut lebih efisien dalam mencapai akurasi yang lebih tinggi pada model.

Sementara itu, tidak ada perbedaan yang sangat signifikan atas penurunan grafik loss antara eksperimen dengan RMSNorm dan tanpa RMSNorm.

#### 2.2.4. Pengaruh Learning Rate (*Basic FFNN*)

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test. Variasi learning rate yang digunakan adalah 0.001, 0.01, dan 0.1.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: MSE
- *Batch size*: 200
- *Epochs*: 100

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron	Initialization	Activation Function
-----------	---------------	----------------	---------------------

1.	128	<i>Uniform</i>	<i>Softmax</i>
2.	64	<i>Uniform</i>	<i>Softmax</i>
3.	32	<i>Uniform</i>	<i>Softmax</i>
4.	10	<i>Uniform</i>	<i>Softmax</i>

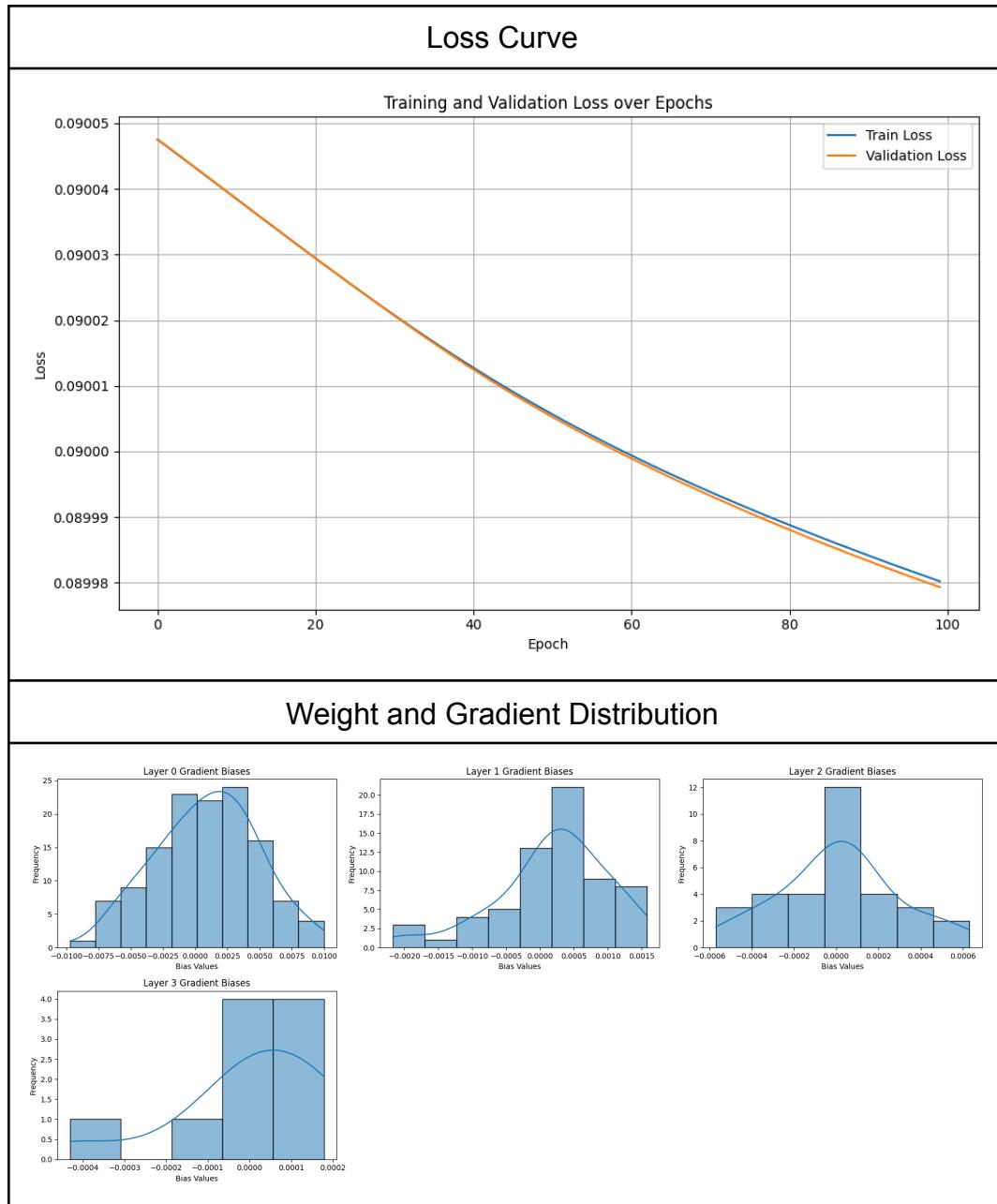
Perbandingan hasil:

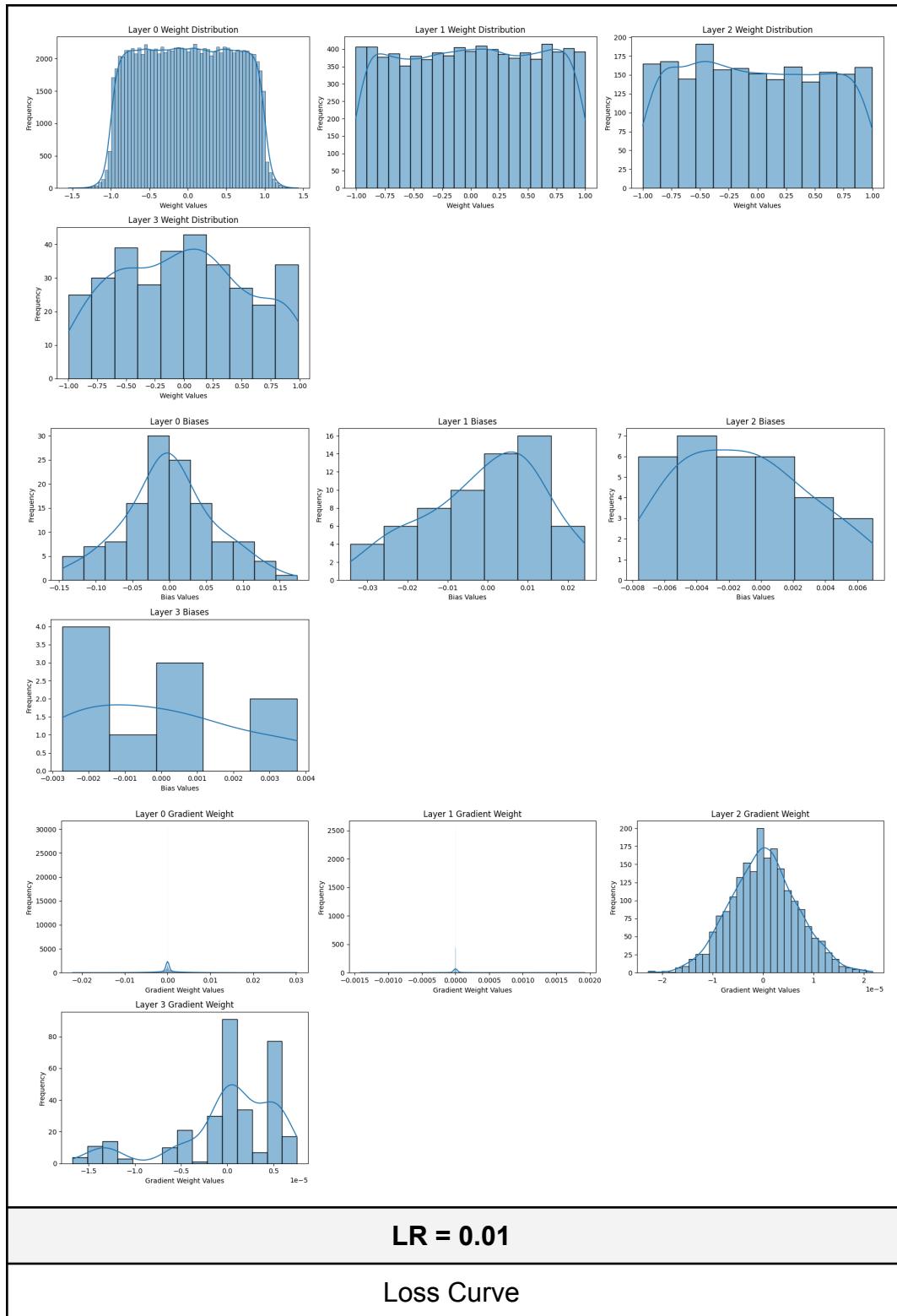
Aspek	LR = 0.001	LR = 0.01	LR = 0.1
Hasil prediksi (10 data pertama dari data test)	[8 8 8 0 8 8 8 8 8]	[2 2 2 2 2 2 2 2 2]	[0 9 0 0 9 3 3 3 0 9]
Target (10 data pertama dari data test)		[2 9 0 0 4 3 2 9 0 4]	
Akurasi (seluruh 5000 data test)	0.1548	0.0998	0.3312

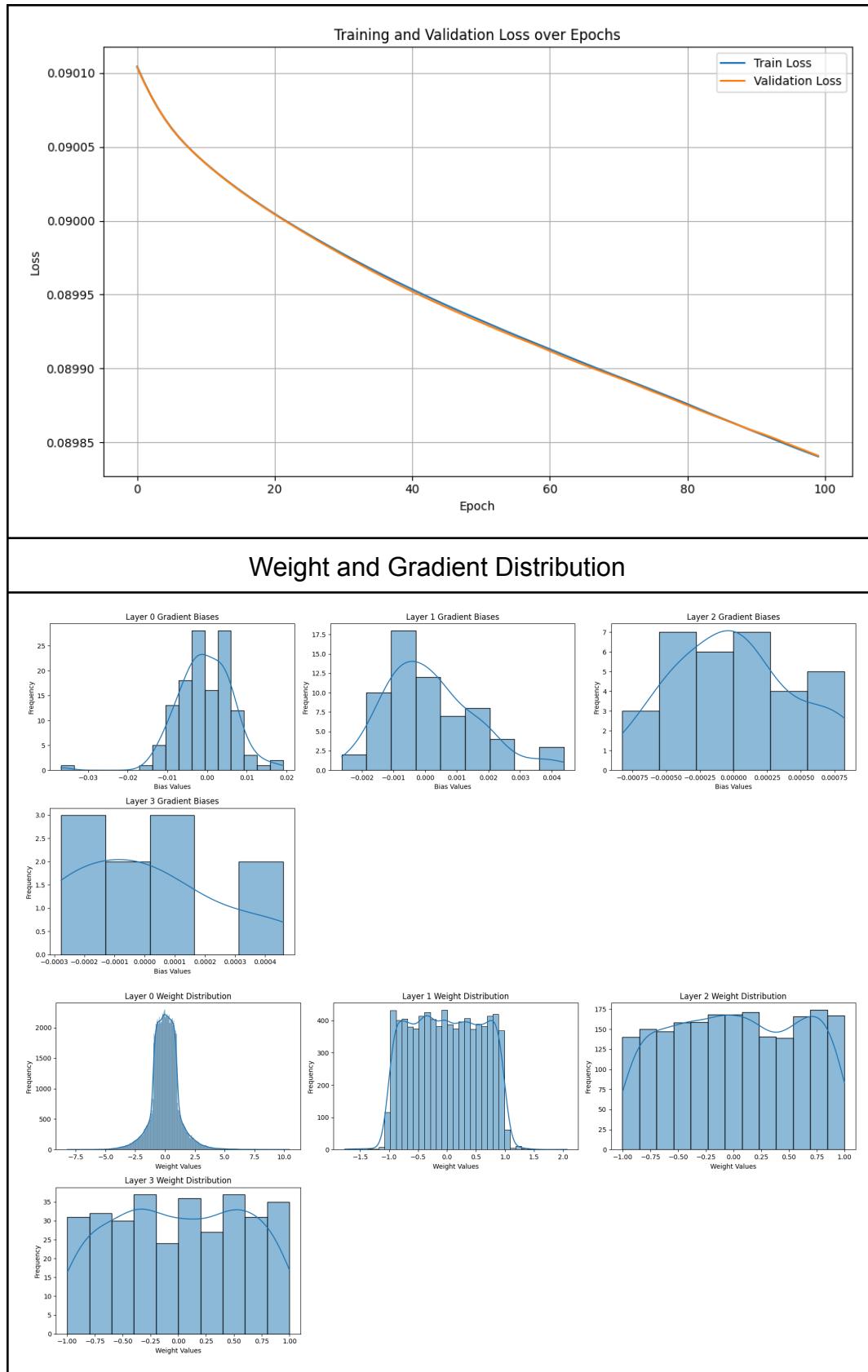
Akurasi model pada seluruh 5000 data test menunjukkan penurunan signifikan seiring dengan peningkatan learning rate. Dengan learning rate 0.001, akurasi mencapai 0.1548, yang menunjukkan bahwa model mampu mempelajari data dengan lebih hati-hati meskipun tidak sepenuhnya tepat. Ketika learning rate diatur ke 0.01, akurasi sedikit menurun menjadi 0.0998, yang menunjukkan bahwa model mulai belajar terlalu cepat, sehingga tidak dapat mengoptimalkan bobot dengan baik. Pada learning rate 0.1, akurasi meningkat sedikit menjadi 0.3312, namun hasilnya tetap rendah, yang mengindikasikan bahwa learning rate yang terlalu tinggi menyebabkan model melompat-lompat terlalu jauh dalam pencarian minimum.

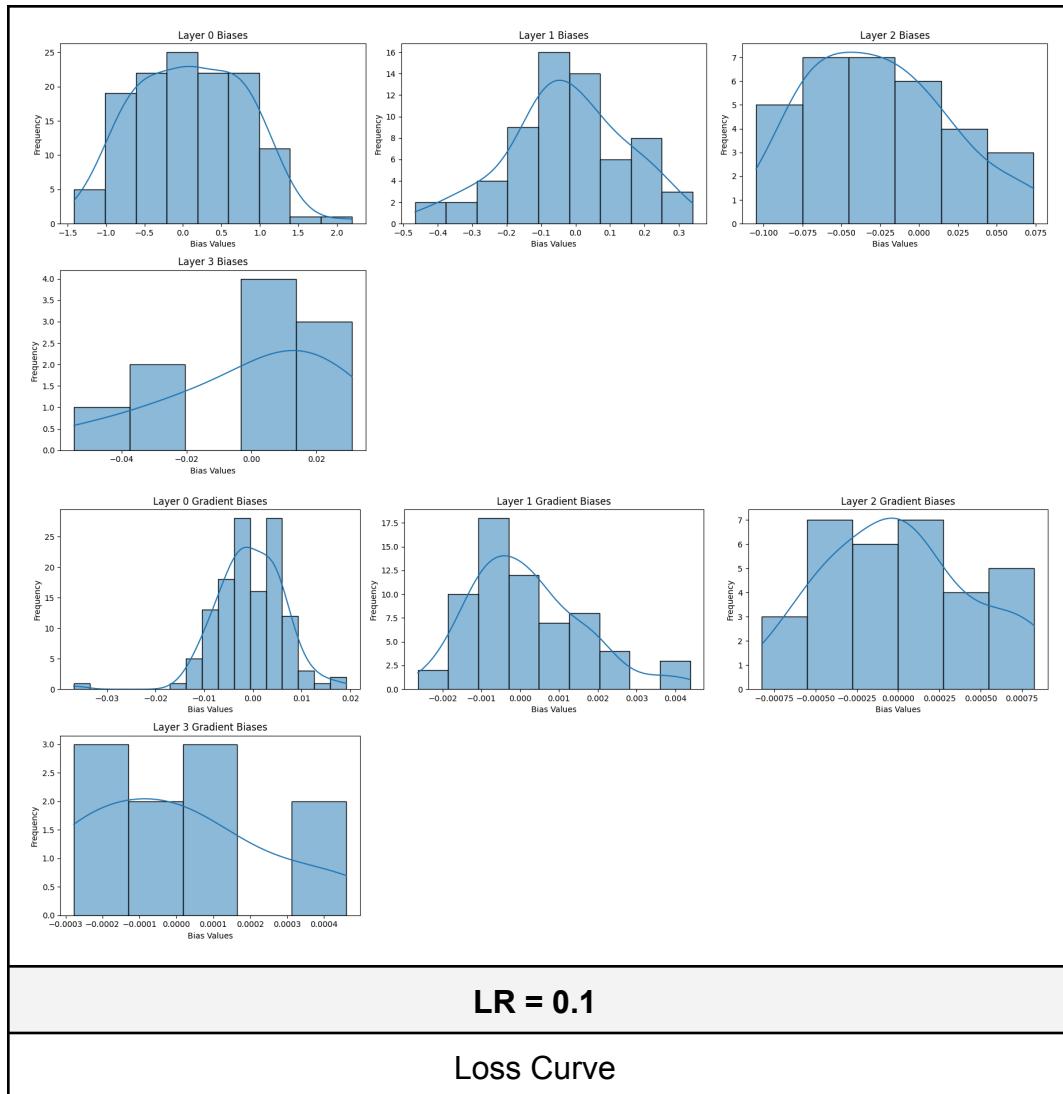
Perbandingan plot:

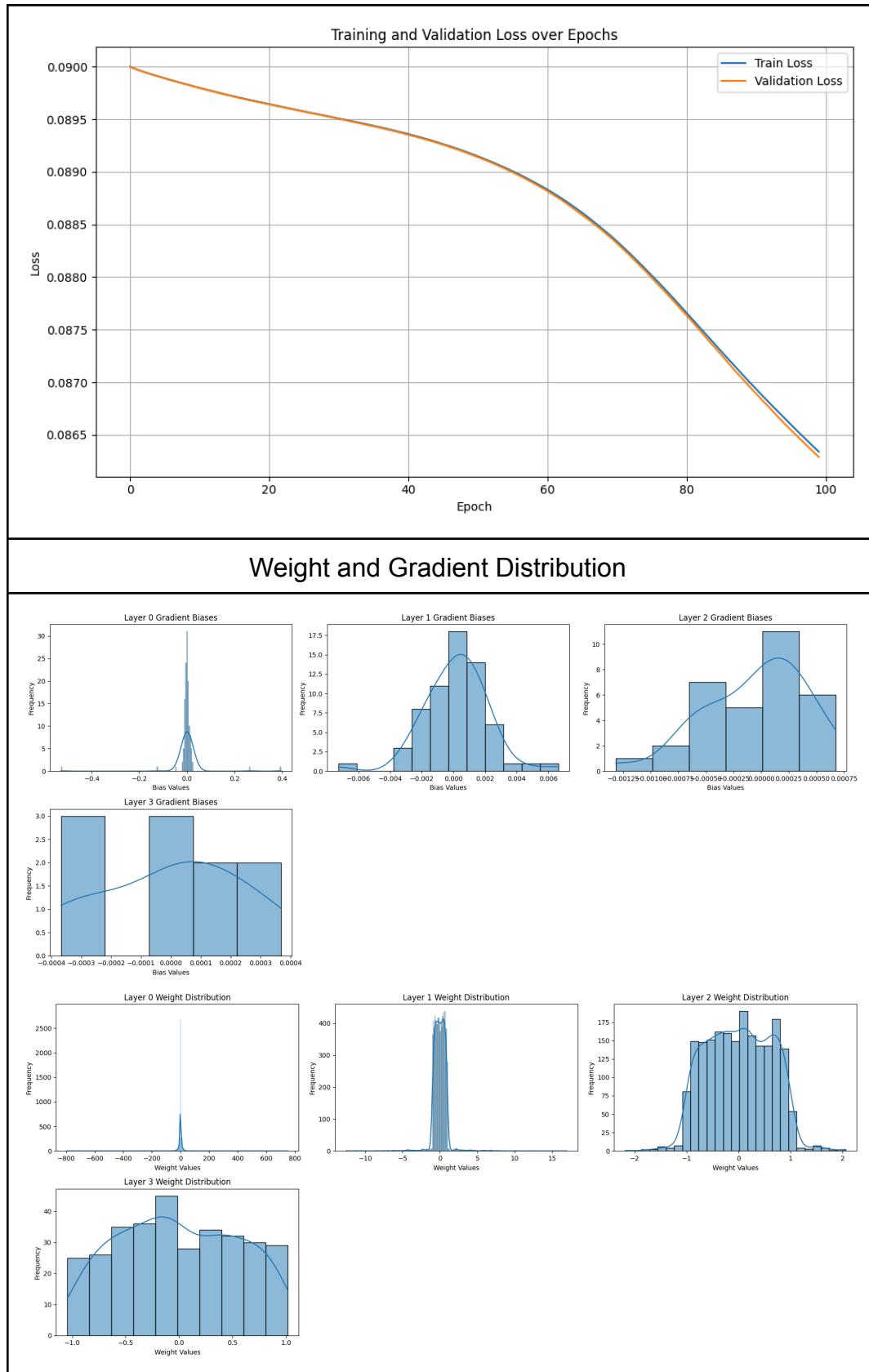
<b>LR = 0.001</b>
-------------------

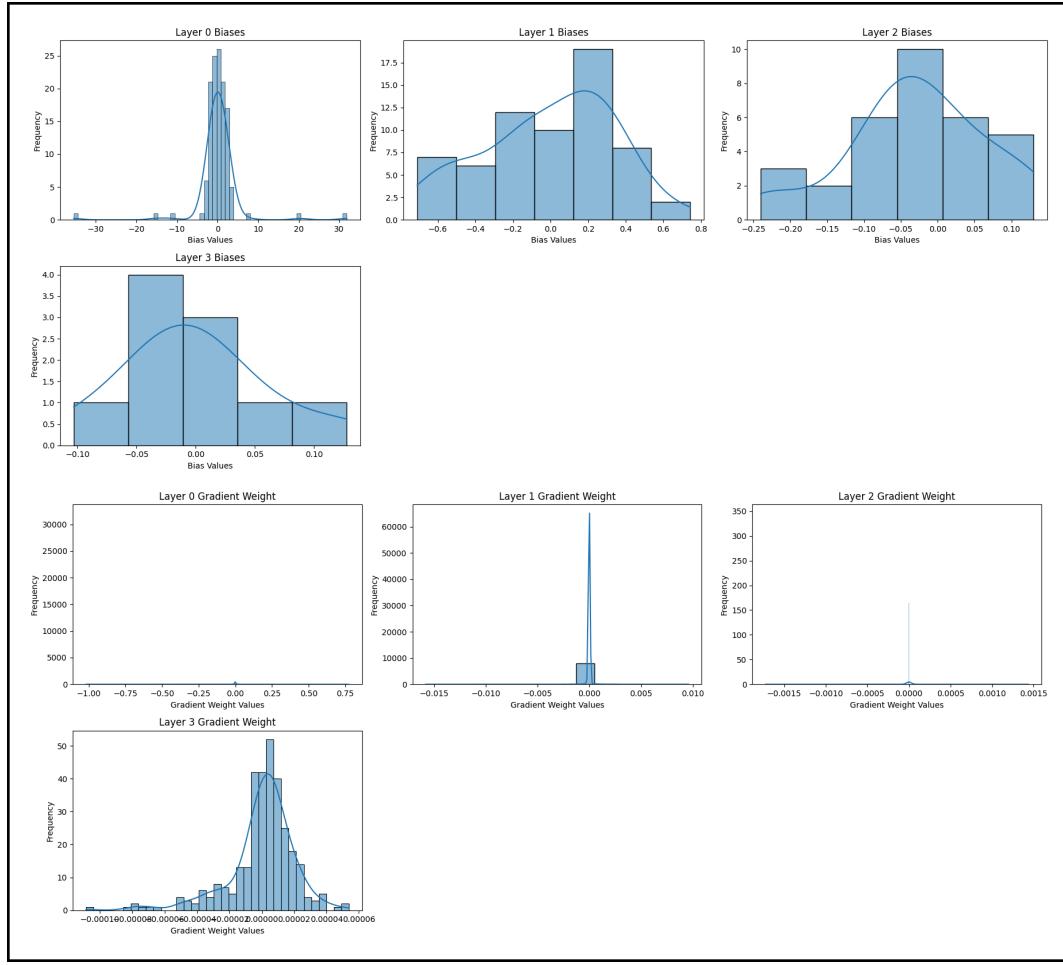












Berdasarkan hasil percobaan, grafik *loss* pada *learning rate* 0.001 cenderung mengalami penurunan yang lebih cepat daripada model yang dilatih dengan *learning rate* 0.01 dan 0.1. Tetapi seiring bertambahnya *epoch*, model yang dilatih dengan *learning rate* 0.1 mengalami penurunan *loss* yang lebih cepat daripada *learning rate* 0.001 dan 0.01.

Distribusi bobot dan gradien bobot dari percobaan di atas pada umumnya terdistribusi secara normal. Namun, terdapat beberapa bobot dan gradien bobot yang *skew left* ataupun *skew right*.

### 2.2.5. Pengaruh Inisialisasi Bobot (Basic FFNN)

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

Mekanisme inisialisasi bobot yang digunakan adalah Zero, Uniform, Normal, Xavier Uniform, Xavier Normal, He Normal, dan He Uniform.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: MSE
- *Learning Rate*: 0.01
- *Batch size*: 200
- *Epochs*: 100
- *Verbose*: 1

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron	Activation Function
1.	128	Softmax
2.	64	Softmax
3.	32	Softmax
4.	10	Softmax

Perbandingan hasil:

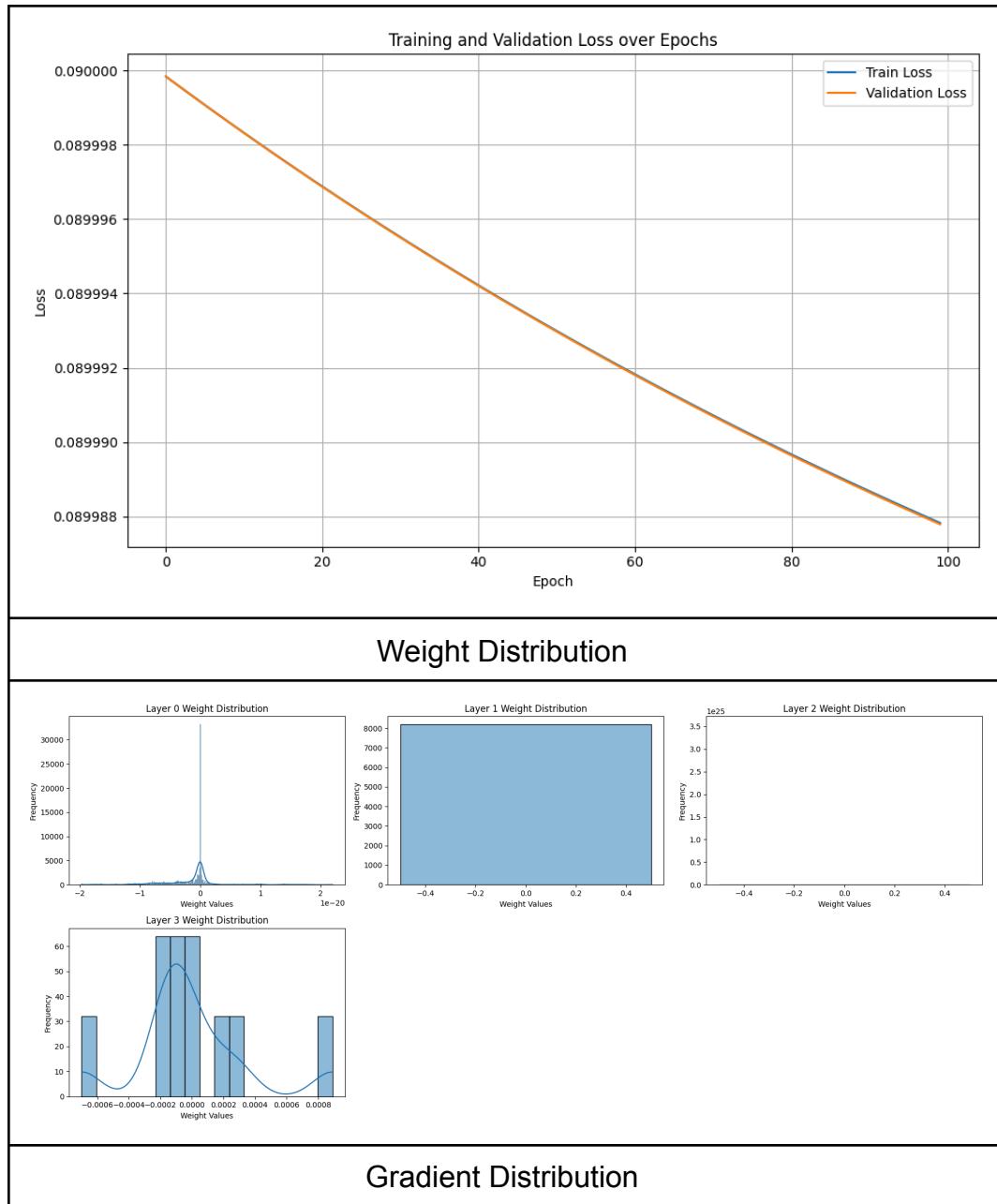
Jenis Weight Initialization	Hasil prediksi (10 data pertama dari data test)	Target (10 data pertama dari data test)	Akurasi (seluruh 5000 data test)
Zero	[1 1 1 1 1 1 1 1 1 1]		0.1126
Uniform	[3 3 3 3 3 3 3 3 3 3]	[2 9 0 0 4 3 2 9 0 4]	0.1020
Normal	[7 7 7 7 7 7 7 1 7 7]		0.2030
Xavier Uniform	[1 1 1 1 1 1 1 1 1 1]		0.1126
Xavier	[2 2 2 2 2 2 2 2 2 2]		0.0998

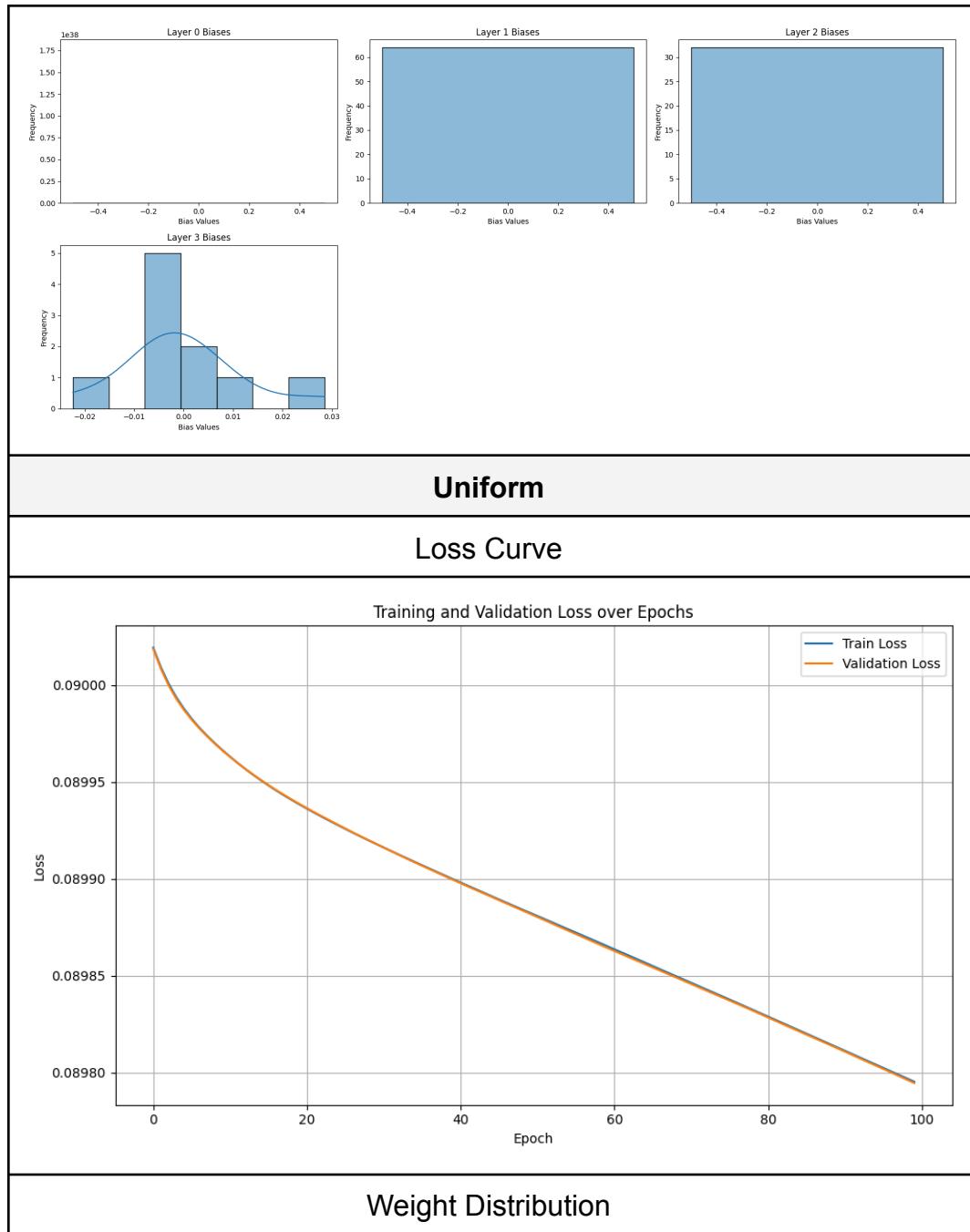
Normal	2]		
He Normal	[7 7 7 7 7 7 7 7 7]		0.1042
He Uniform	[2 2 2 2 2 2 2 2 2]		0.0998

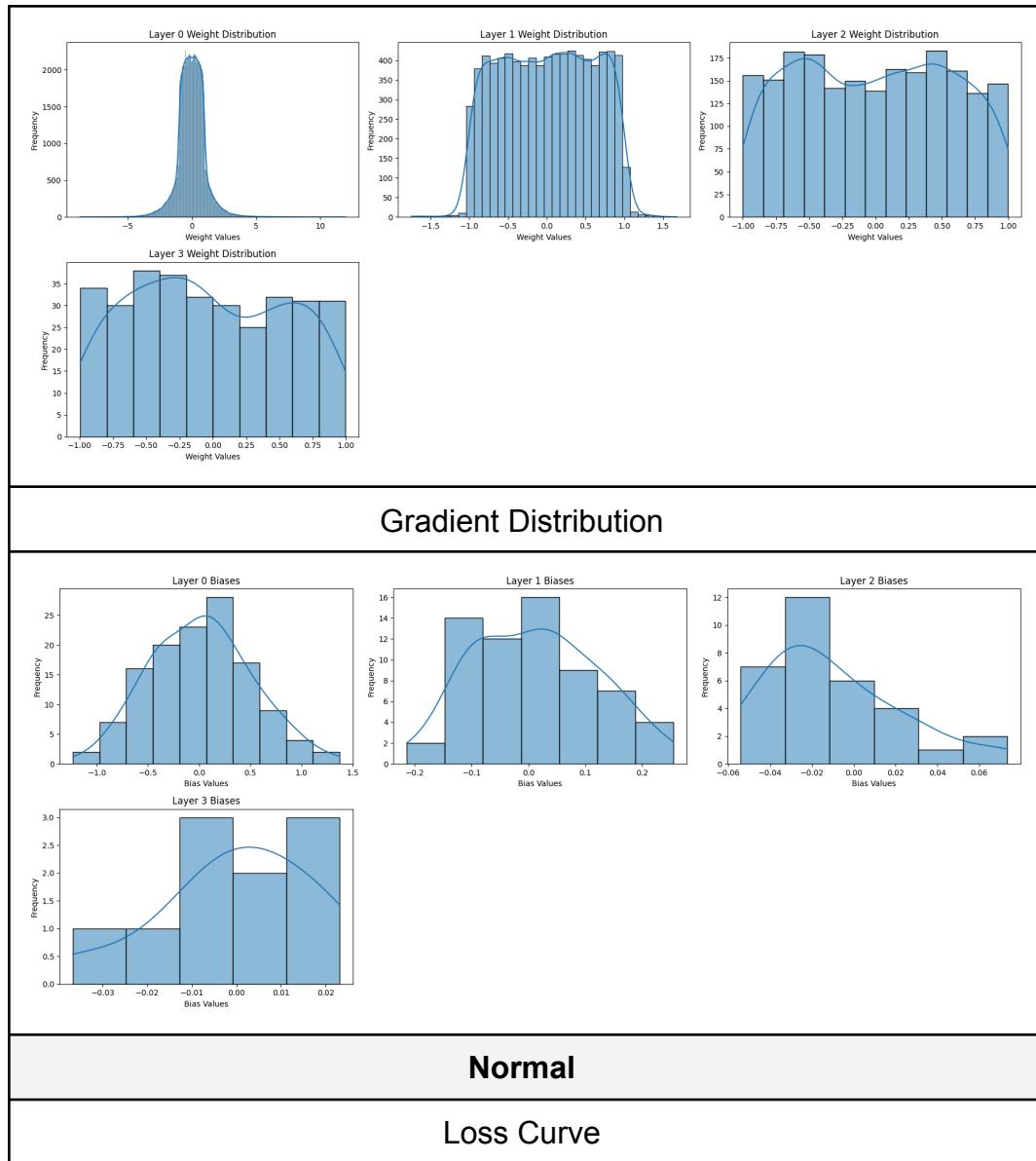
Metode inisialisasi bobot memberikan dampak signifikan terhadap hasil prediksi dan akurasi model. Dengan target yang sama, hasil prediksi dari 10 data pertama dan akurasi keseluruhan pada 5000 data test berbeda tergantung pada metode inisialisasi bobot yang digunakan. Inisialisasi dengan metode Normal memberikan akurasi tertinggi sebesar 0.2030, menunjukkan bahwa bobot awal yang dihasilkan lebih mendukung proses pembelajaran. Sebaliknya, metode seperti Xavier Normal dan He Uniform memberikan akurasi terendah, yaitu 0.0998, mendekati prediksi acak, yang menandakan model gagal belajar dengan baik. Inisialisasi bobot dengan nilai Zero, meskipun umum digunakan sebagai baseline, juga menunjukkan performa yang buruk dengan akurasi 0.1126. Hal ini berarti pemilihan metode inisialisasi bobot sangatlah penting dan berpengaruh dalam pelatihan model neural network karena dapat mempengaruhi konvergensi dan kemampuan generalisasi model.

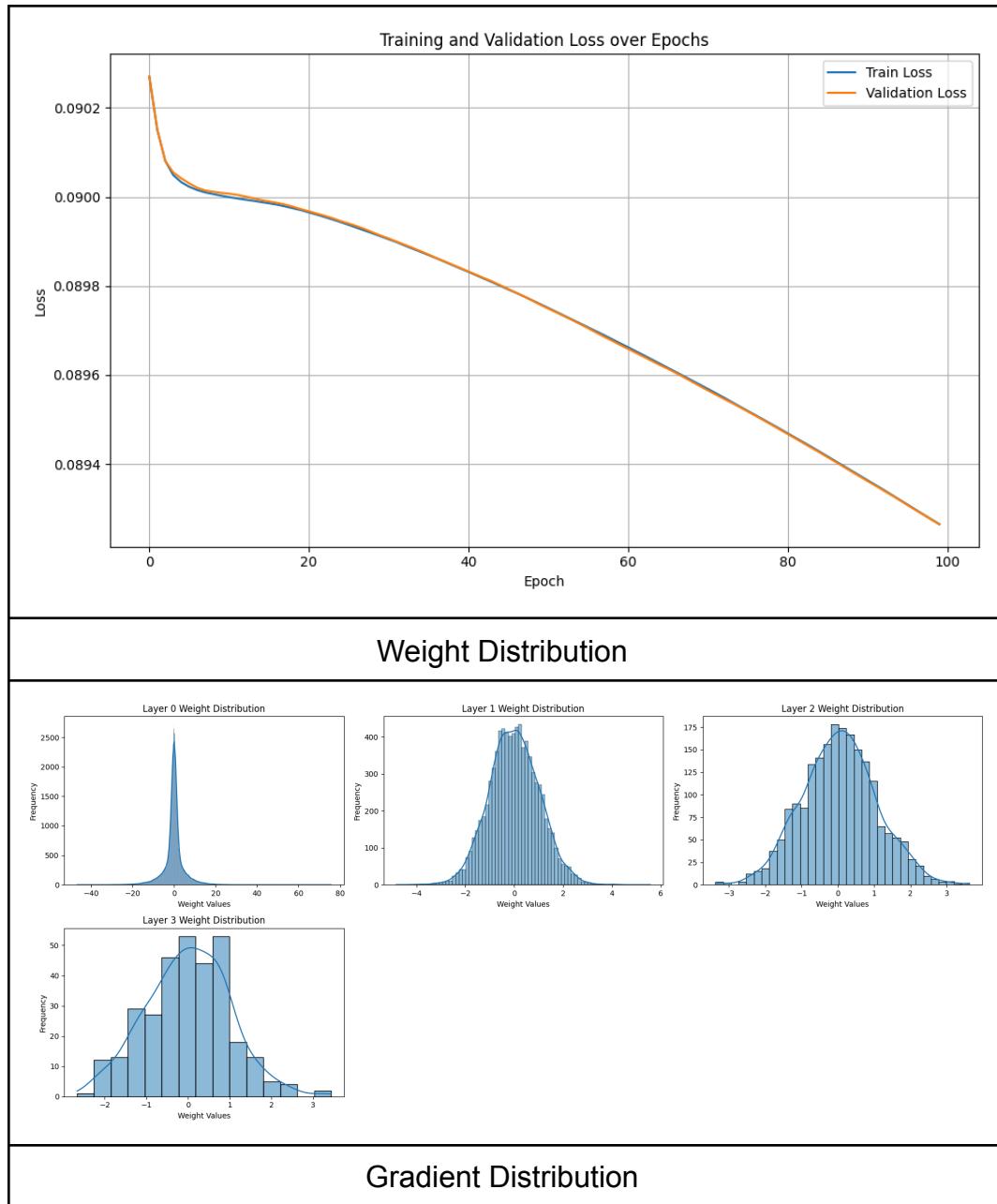
Perbandingan plot:

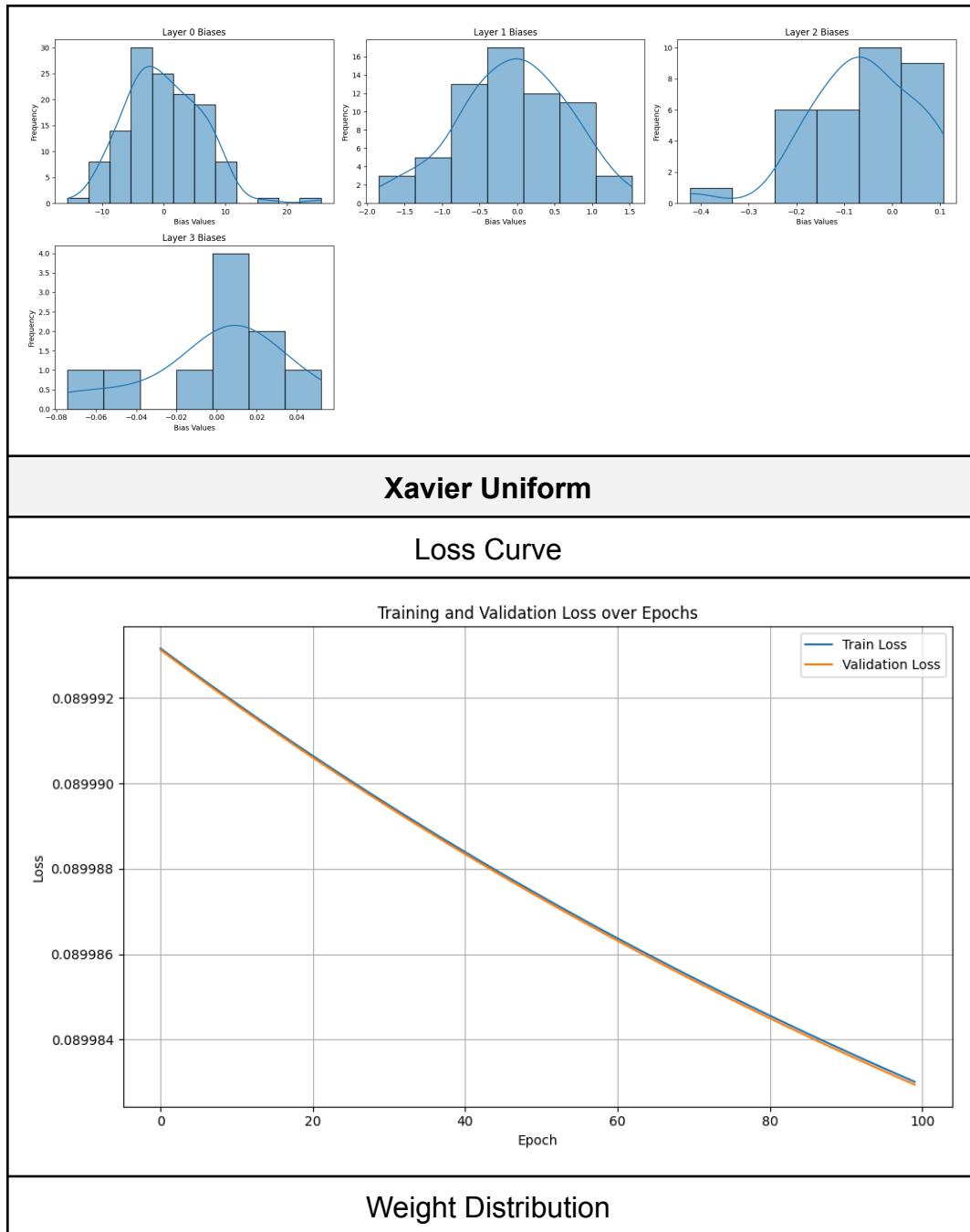
<b>Zero</b>
Loss Curve

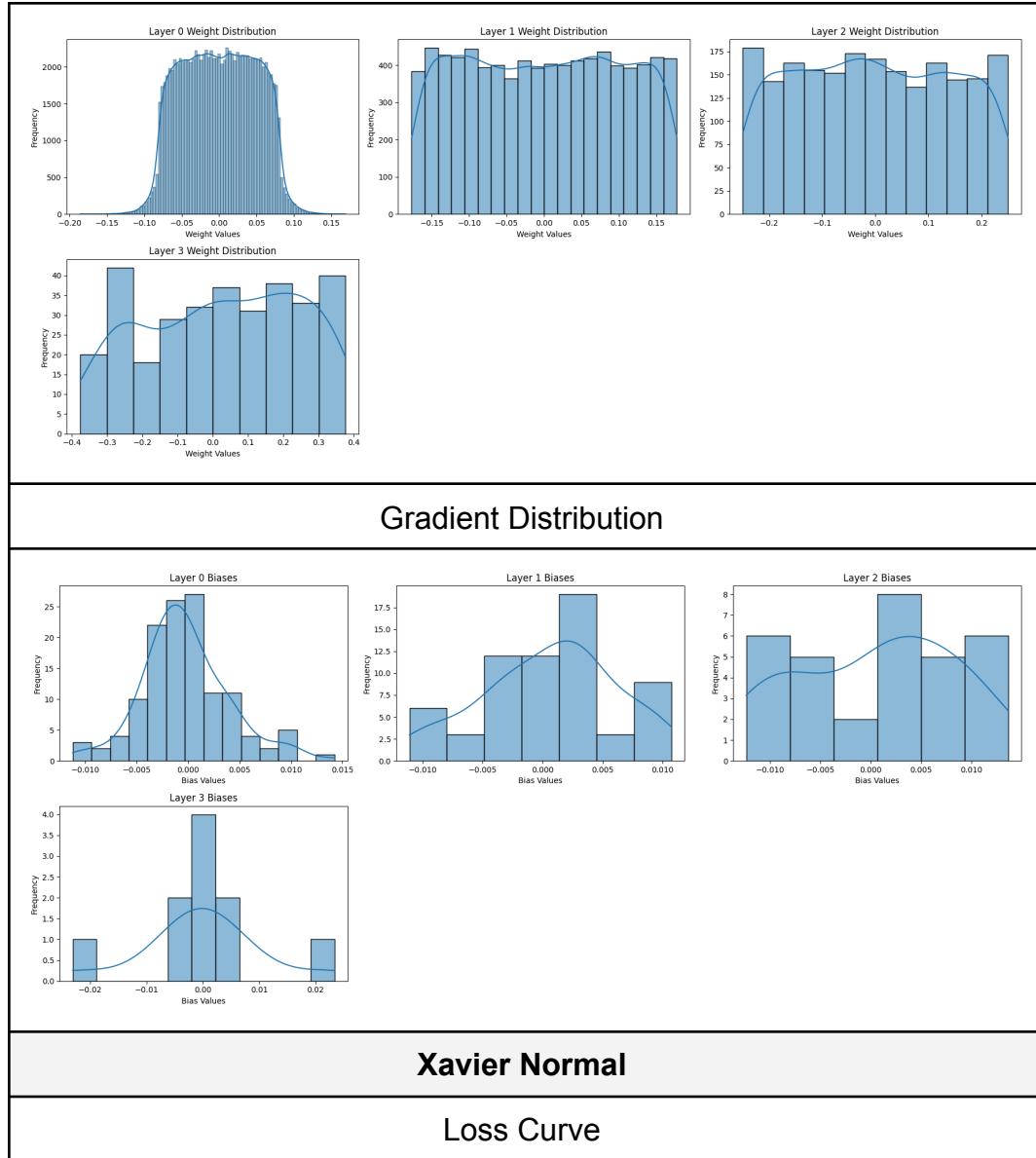


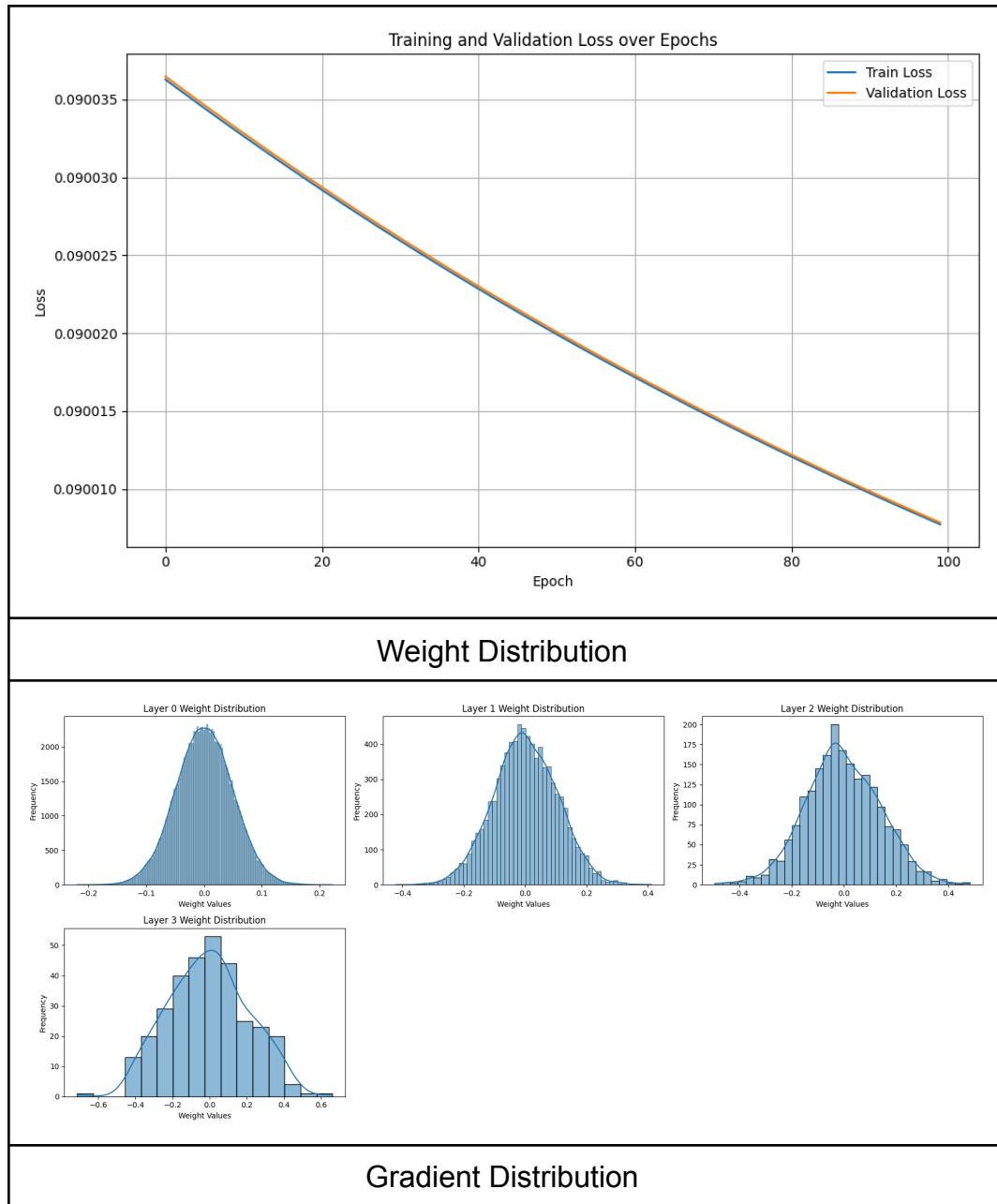


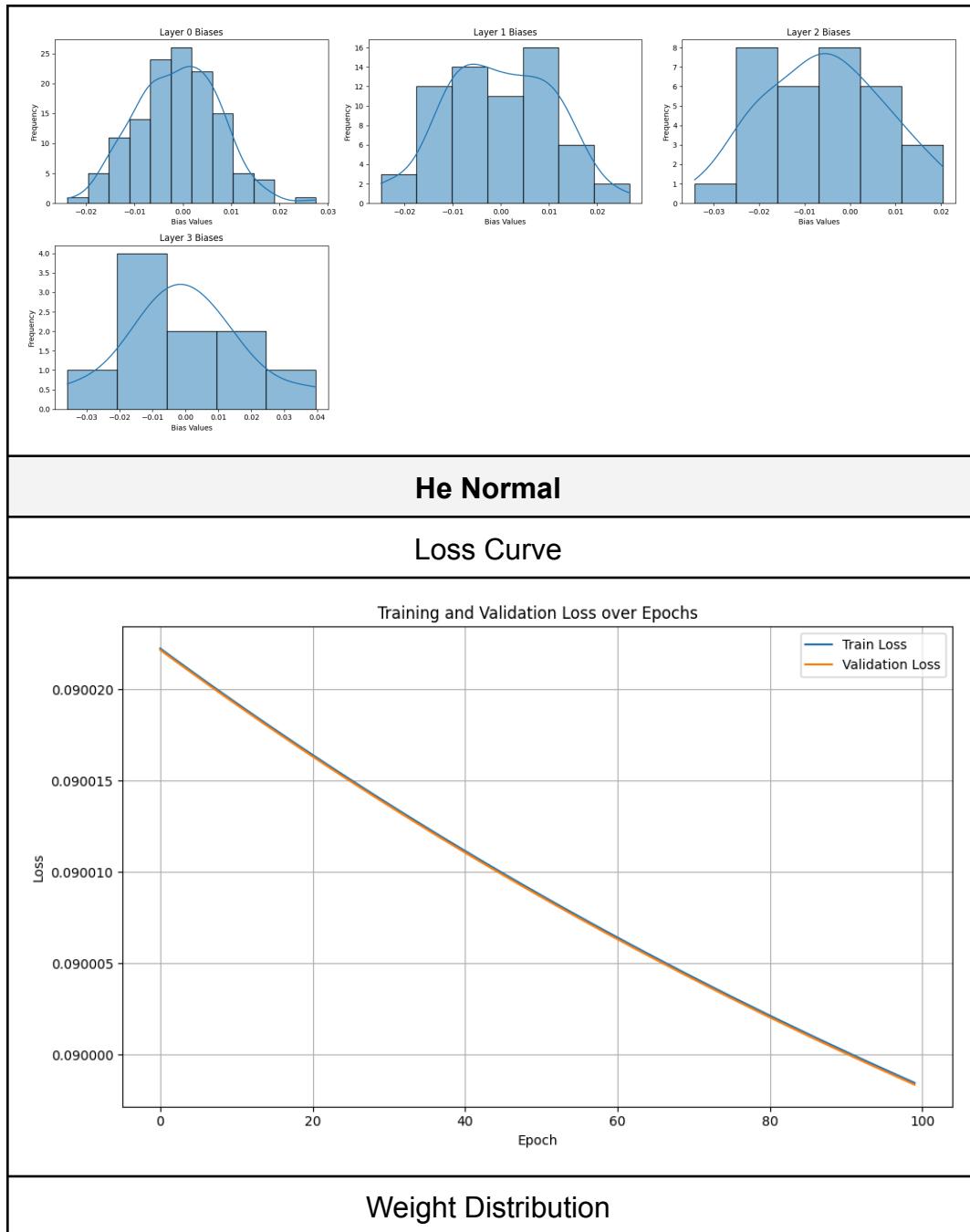


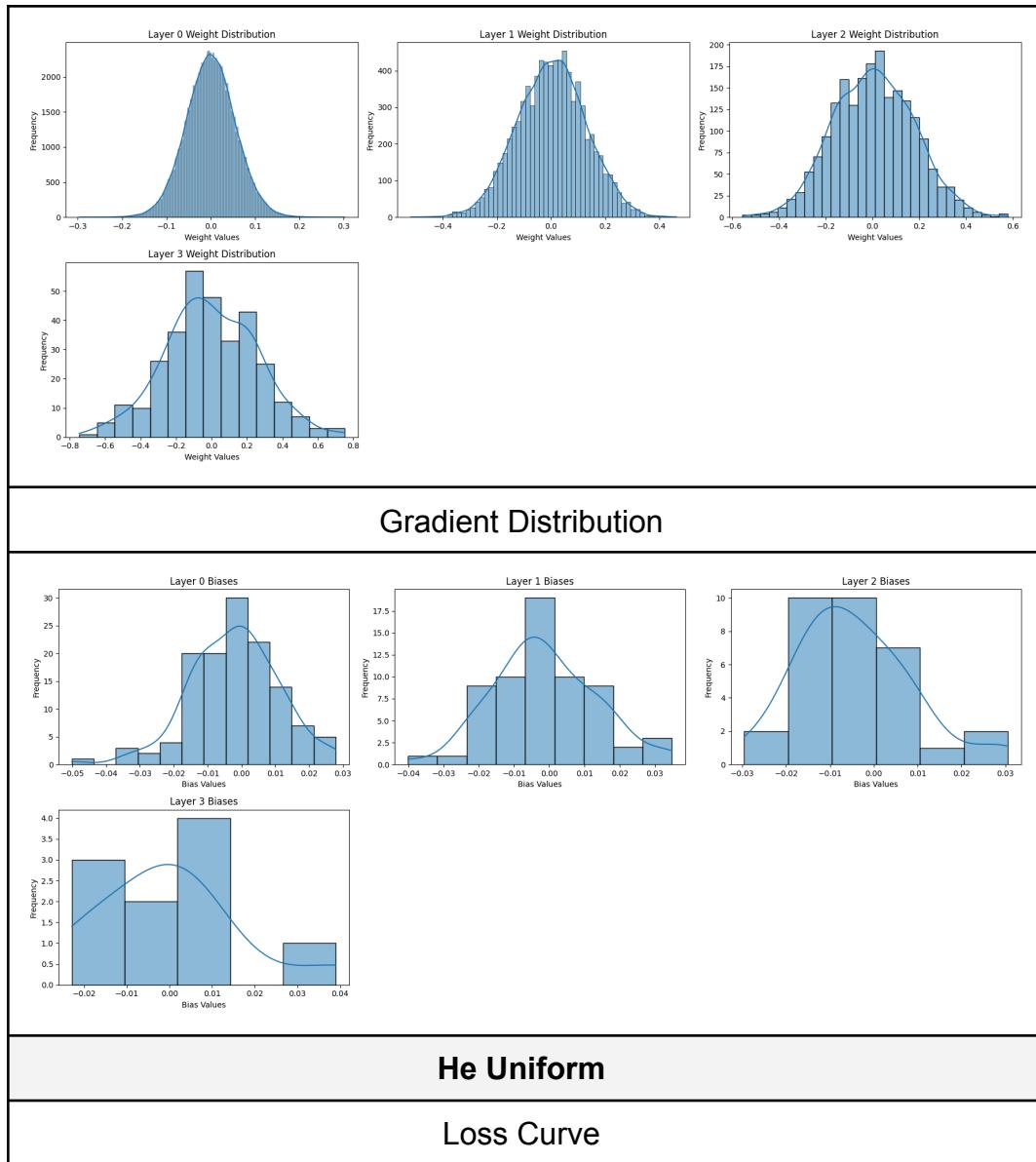


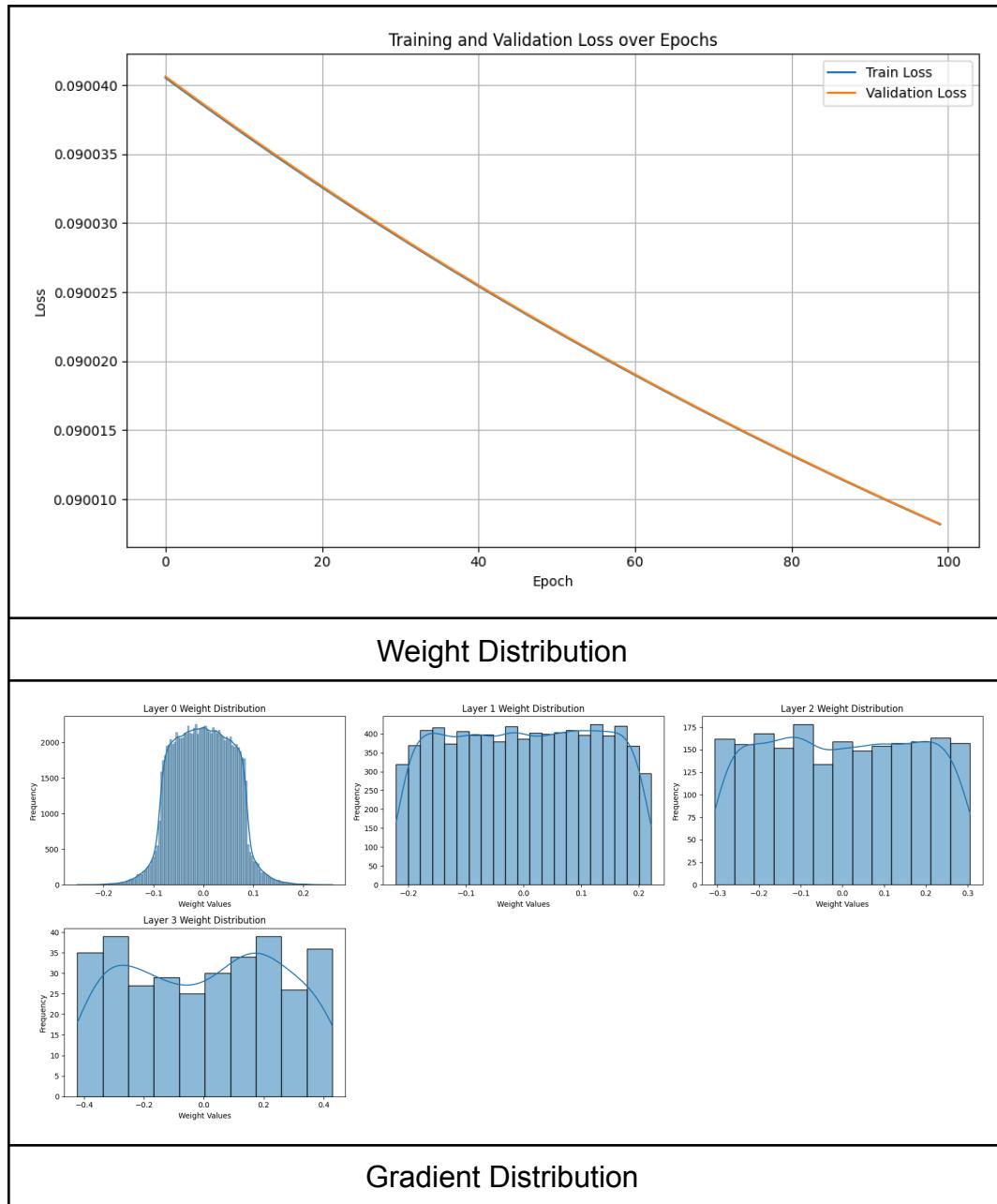


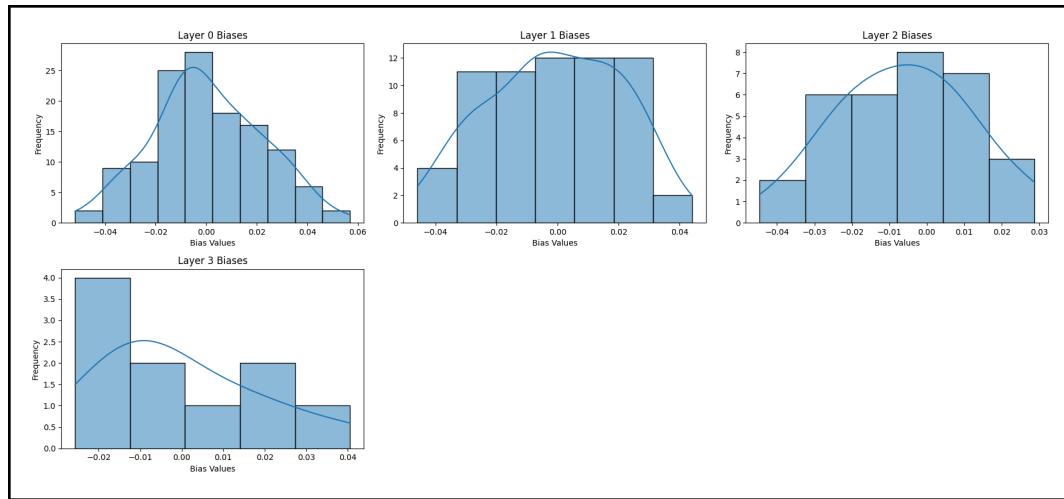












Berdasarkan hasil percobaan di atas, seluruh jenis metode inisialisasi memiliki nilai *loss* awal yang tidak jauh berbeda satu sama lain. Metode yang memiliki nilai *loss* yang paling kecil pada awal *epoch* adalah Zero dan Uniform. Sedangkan metode yang memiliki *loss* paling kecil pada akhir *epoch* adalah Normal.

Sebagian besar bobot dan gradien bobot dari seluruh metode inisialisasi terdistribusi secara normal, namun terdapat juga beberapa bobot dan gradien bobot yang memiliki distribusi *skew left* ataupun *skew right*.

### 2.2.6. Perbandingan dengan Library Sklearn

#### a) FFNN - *Scratch (Autodiff)* vs *MLPClassifier - Library Sklearn*

Percobaan dilakukan dengan menggunakan data MNIST dengan 2000 data untuk *train* dan 20000 data untuk *test*. Jumlah data *train* dan jumlah *epoch* yang digunakan tidak terlalu banyak dikarenakan keterbatasan dari implementasi *autodiff* yang akan membutuhkan waktu yang lebih lama untuk melatih lebih banyak data.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss: Categorical Cross-entropy*
- *Batch size: 20*

- *Learning rate*: 0.1
- *Epochs*: 2

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron	Initialization	Activation Function
1.	4	<i>He</i>	<i>Relu</i>
2.	3	<i>He</i>	<i>Relu</i>
3.	2	<i>He</i>	<i>Relu</i>
4.	10	<i>He</i>	<i>Relu</i>

Perbandingan hasil:

Aspek	Scratch	Library
Hasil prediksi (10 data pertama dari data <i>test</i> )	[7, 7, 7, 7, 7, 7, 7, 7, 7, 7]	[4, 6, 6, 6, 4, 6, 6, 6, 6, 6]
Target (10 data pertama dari data <i>test</i> )		[4, 4, 6, 3, 7, 2, 2, 1, 5, 2]
<i>Train Loss</i> terakhir	3.4021	1.86877649
Waktu eksekusi	3973.61 detik	8 detik
Akurasi (seluruh 20000 data <i>test</i> )	0.1	0.2

Berdasarkan hasil tersebut, *training loss* dari MLPClassifier lebih kecil daripada implementasi dari scratch. Akurasi yang dihasilkan

oleh MLPClassifier juga lebih besar daripada implementasi dari *scratch*. Hal tersebut dikarenakan MLPClassifier secara default menggunakan *adam* sebagai *solver* yang memungkinkan proses pelatihan menjadi lebih adaptif.

**b) FFNN - *Scratch (Basic)* vs *MLPClassifier* - *Library Sklearn***

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: *Categorical Cross-entropy*
- *Batch size*: 200
- *Learning rate*: 0.01
- *Epochs*: 100

Arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah Neuron	Initialization	Activation Function
1.	128	<i>He Uniform</i>	<i>Relu</i>
2.	64	<i>He Uniform</i>	<i>Relu</i>
3.	32	<i>He Uniform</i>	<i>Relu</i>
4.	10	<i>He Uniform</i>	<i>Relu</i>

Perbandingan hasil:

Aspek	Scratch	Library
Hasil prediksi (10)	[0 9 0 0 4 3 2 9 0 4]	[2 9 0 0 4 3 2 9 0 4]

data pertama dari data <i>test</i> )		
Target (10 data pertama dari data <i>test</i> )		[2 9 0 0 4 3 2 9 0 4]
Akurasi (seluruh 5000 data <i>test</i> )	0.9646	0.9704

Berdasarkan hasil tersebut, prediksi yang dihasilkan oleh MLPClassifier lebih akurat daripada implementasi dari scratch. Pada 10 data pertama dari data test, model *library* berhasil memprediksi angka dengan tepat, sementara model *scratch* memberikan beberapa prediksi yang salah. Hal ini menunjukkan bahwa model yang dibangun menggunakan library lebih stabil dan memiliki kemampuan prediksi yang lebih baik.

Selain itu, akurasi keseluruhan pada seluruh 5000 data test juga menunjukkan hasil yang lebih baik pada MLPClassifier dengan nilai 0.9704 dibandingkan dengan model *scratch* yang hanya mencapai 0.9646. Perbedaan akurasi ini disebabkan oleh optimasi internal yang dimiliki oleh MLPClassifier, seperti penggunaan algoritma optimisasi yang lebih efisien dan parameter yang lebih teratur.

Secara keseluruhan, MLPClassifier yang menggunakan *library* Sklearn memberikan hasil yang lebih optimal dan lebih mudah untuk diterapkan dibandingkan dengan implementasi FFNN dari awal (*scratch*), yang memerlukan lebih banyak pengaturan dan tuning.

#### 2.2.7. Perbandingan dengan dan tanpa Regularisasi

Untuk percobaan ini, arsitektur yang digunakan adalah sebagai berikut:

No. Layer	Jumlah	Initialization	Activation

	<b>Neuron</b>		<b>Function</b>
1.	128	<i>He Uniform</i>	<i>Relu</i>
2.	64	<i>He Uniform</i>	<i>Relu</i>
3.	32	<i>He Uniform</i>	<i>Relu</i>
4.	10	<i>He Uniform</i>	<i>Softmax</i>

#### 2.2.7.1. Tanpa Regularisasi

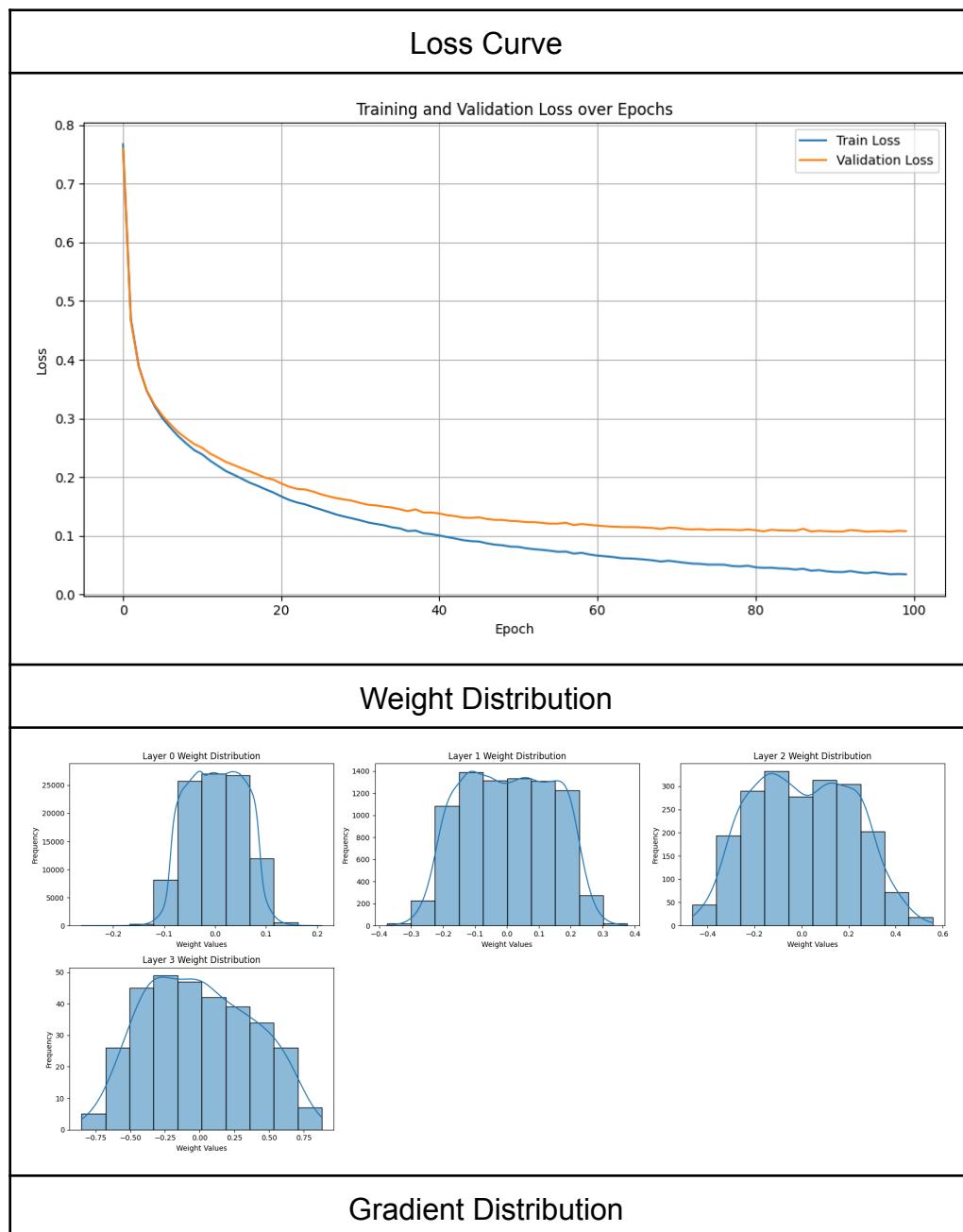
Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

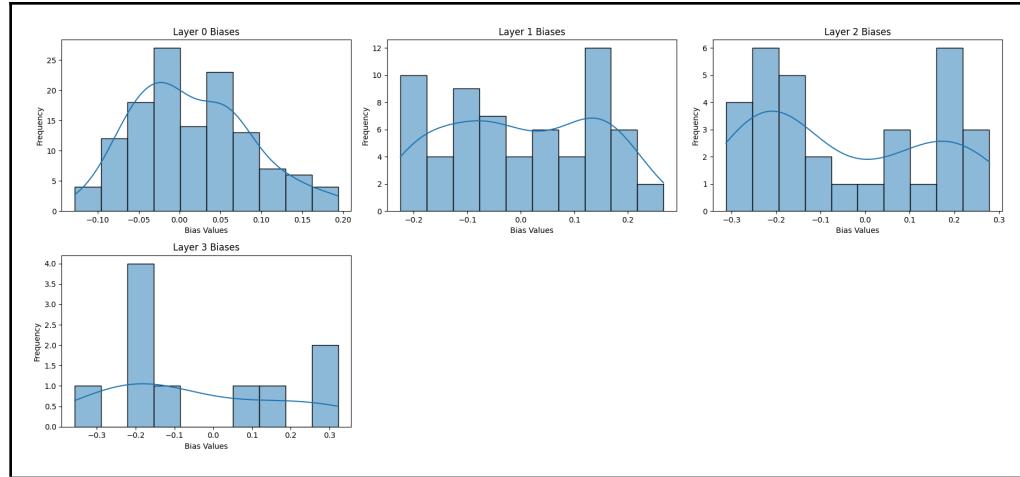
*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: *Categorical Cross-entropy*
- *Batch size*: 200
- *Learning rate*: 0.01
- *Epochs*: 100

Perbandingan hasil:

<b>Aspek</b>	<b>Hasil</b>
Hasil prediksi (10 data pertama dari data <i>test</i> )	[2 9 0 0 4 3 2 9 0 4]
Target (10 data pertama dari data test)	[2 9 0 0 4 3 2 9 0 4]
Akurasi (seluruh 5000 data <i>test</i> )	0.9696





### 2.2.7.2. Dengan Regularisasi L1

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

*Hyperparameter* yang digunakan adalah sebagai berikut:

- *Loss*: Categorical Cross-entropy
- *Batch size*: 200
- *Learning rate*: 0.01
- *Epochs*: 100
- L1: 0.1

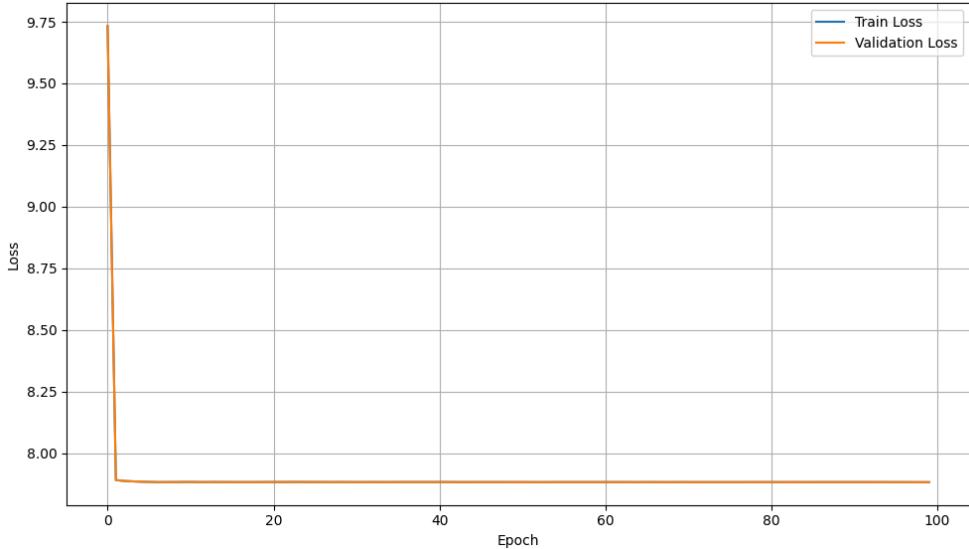
Perbandingan hasil:

Aspek	Hasil
Hasil prediksi (10 data pertama dari data <i>test</i> )	[1 1 1 1 1 1 1 1 1]
Target (10 data pertama dari data <i>test</i> )	[2 9 0 0 4 3 2 9 0 4]
Akurasi (seluruh 5000)	0.1126

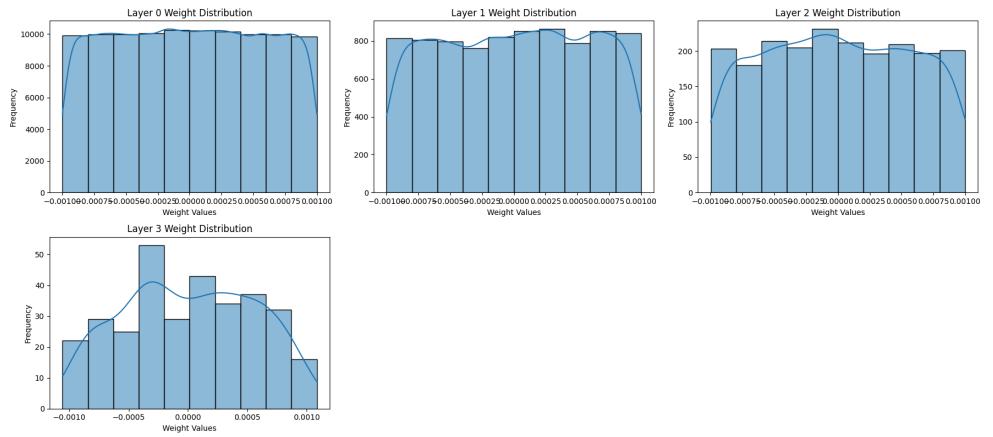
data test)

### Loss Curve

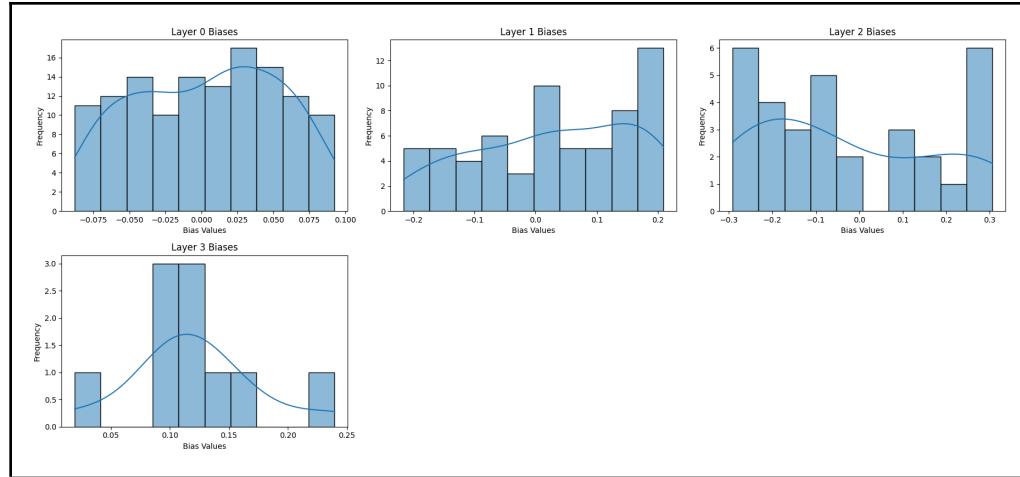
Training and Validation Loss over Epochs



### Weight Distribution



### Gradient Distribution



### 2.2.7.3. Dengan Regularisasi L2

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

Hyperparameter yang digunakan adalah sebagai berikut:

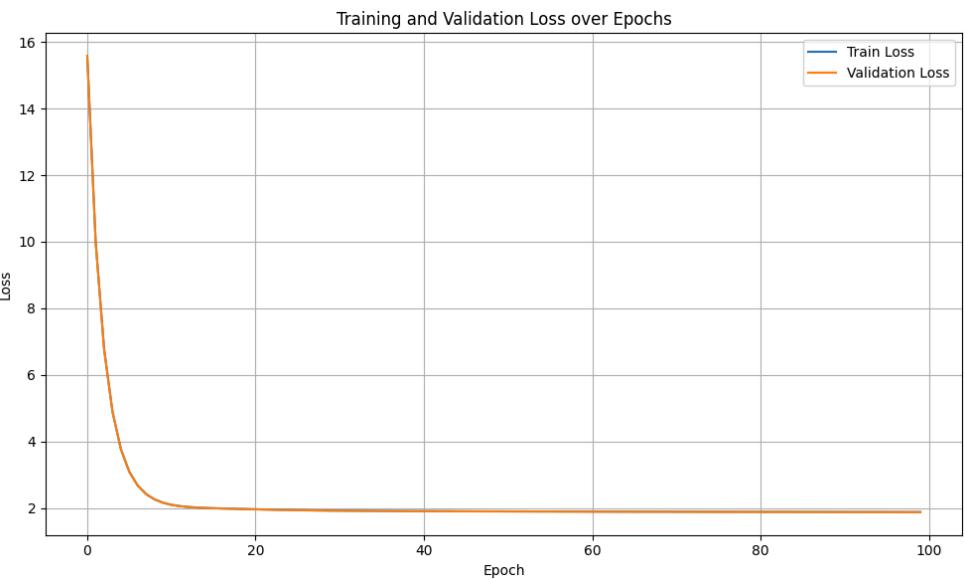
- *Loss: Categorical Cross-entropy*
- *Batch size: 200*
- *Learning rate: 0.01*
- *Epochs: 100*
- *L2: 0.1*

Perbandingan hasil:

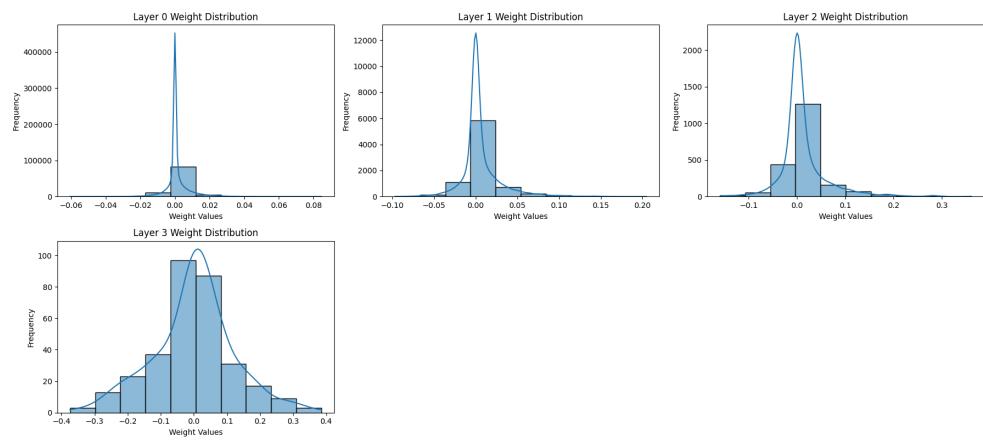
Aspek	Hasil
Hasil prediksi (10 data pertama dari data <i>test</i> )	[5 9 0 0 4 3 2 1 0 4]
Target (10 data pertama dari data <i>test</i> )	[2 9 0 0 4 3 2 9 0 4]
Akurasi (seluruh 5000)	0.7680

data test)

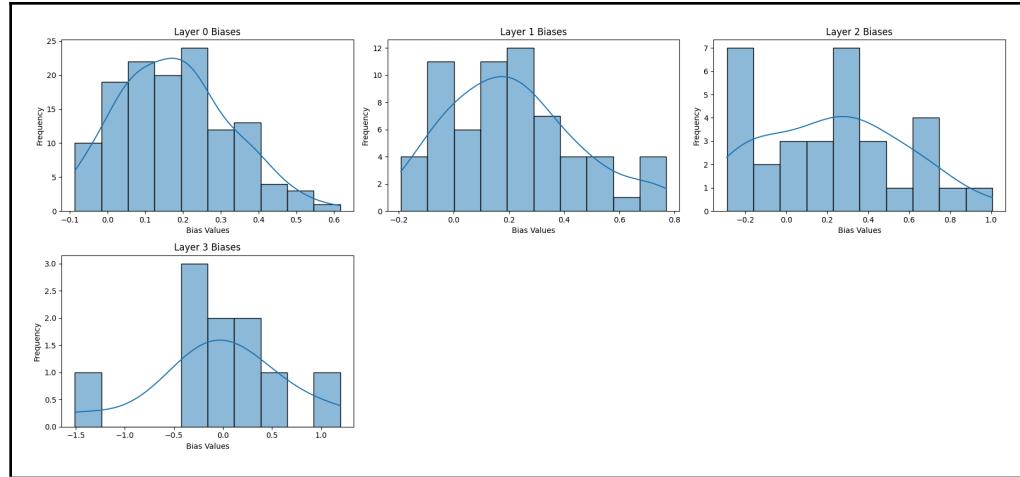
### Loss Curve



### Weight Distribution



### Gradient Distribution



#### 2.2.7.4. Dengan Regularisasi L1 dan L2

Percobaan dilakukan dengan menggunakan data MNIST dengan 52000 data untuk train, 13000 data untuk val, dan 5000 data untuk test.

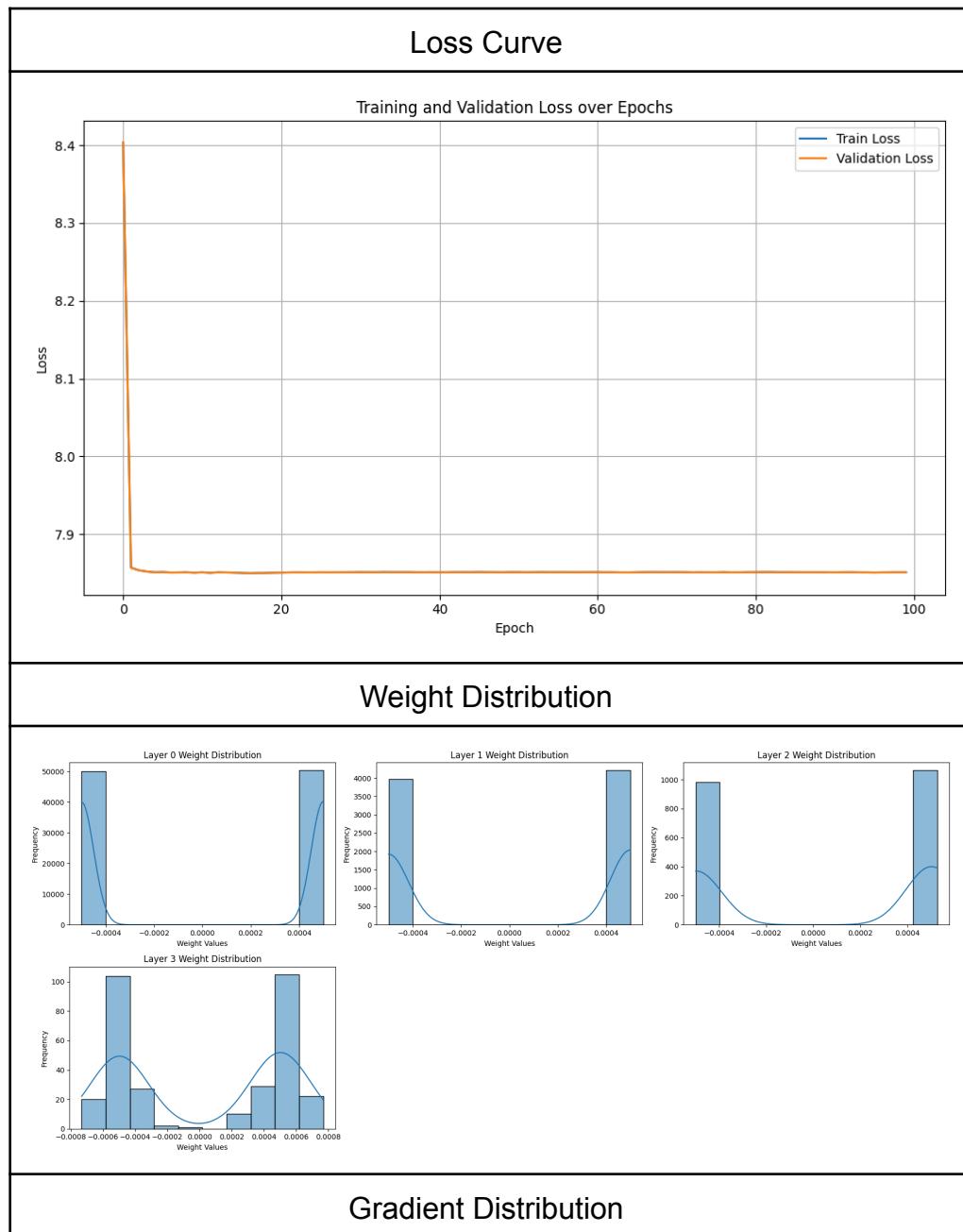
*Hyperparameter* yang digunakan adalah sebagai berikut:

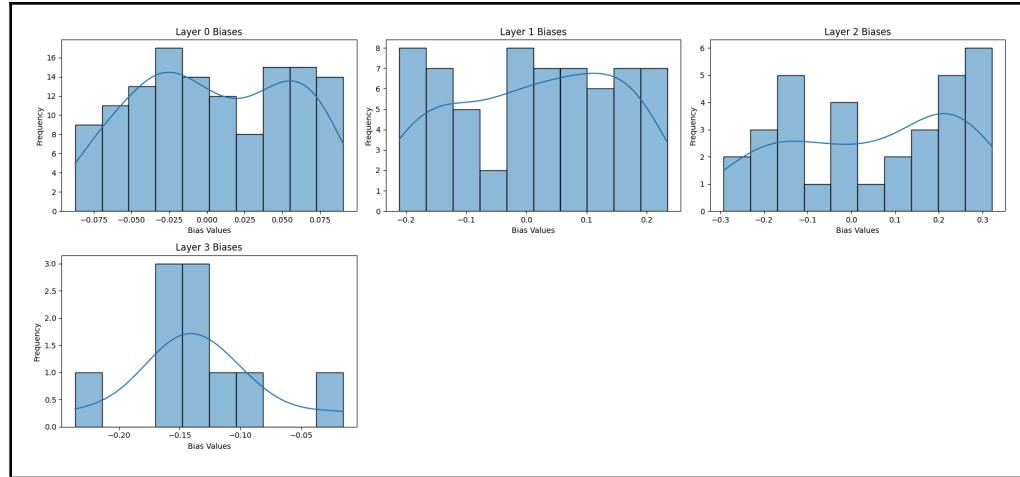
- *Loss*: Categorical Cross-entropy
- *Batch size*: 200
- *Learning rate*: 0.01
- *Epochs*: 100
- L1: 0.1
- L2: 0.1

Perbandingan hasil:

Aspek	Hasil
Hasil prediksi (10 data pertama dari data <i>test</i> )	[1 1 1 1 1 1 1 1 1 1]
Target (10 data pertama dari data <i>test</i> )	[2 9 0 0 4 3 2 9 0 4]

Akurasi (seluruh 5000 data test)	0.1126
----------------------------------	--------





Berdasarkan hasil akurasinya, model ANN tanpa regularisasi menunjukkan akurasi yang sangat tinggi pada data latih (0.9696), mengindikasikan potensi overfitting dan kemungkinan akan berkinerja buruk pada data baru.

Penerapan regularisasi L1 secara drastis menurunkan akurasi menjadi sangat rendah (0.1126), menunjukkan bahwa regularisasi ini terlalu kuat dalam menurunkan weight dan menyebabkan underfitting yang parah.

Regularisasi L2 memberikan akurasi yang lebih baik (0.7680) dibandingkan L1, menunjukkan bahwa ia berhasil mengurangi overfitting sampai batas tertentu, meskipun masih di bawah akurasi tanpa regularisasi, mengindikasikan perlunya optimasi parameter.

Kombinasi regularisasi L1 dan L2 menghasilkan akurasi rendah, menunjukkan bahwa efek sparsity yang diinduksi oleh L1 mendominasi dan penambahan L2 tidak memberikan perbaikan. Oleh karena itu, evaluasi pada data uji dan penyetelan hyperparameter regularisasi, terutama kekuatan regularisasi L1 dan L2, sangat penting untuk menemukan model yang memiliki kemampuan generalisasi yang lebih baik dan menghindari masalah overfitting atau underfitting.

# Bab III. Kesimpulan dan Saran

## 3.1. Kesimpulan

Berdasarkan hasil percobaan yang telah dilakukan, kami berhasil melakukan klasifikasi pada dataset MNIST menggunakan FFNN buatan sendiri dengan nilai akurasi tertinggi (0.9690) yang menggunakan Tanh sebagai fungsi aktivasi. Setelah dilakukan perbandingan antara model buatan FFNN sendiri (yang dengan *Auto Diff* maupun yang *basic*) dengan model MLPClassifier dari *library* Sklearn, ternyata nilai akurasi yang dihasilkan dari MLPClassifier lebih baik daripada model buatan sendiri. Namun, nilai akurasi yang dihasilkan model buatan sendiri tidak berbeda terlalu jauh dengan implementasi *library*. Hasil dari MLPClassifier bisa sedikit lebih baik karena model tersebut memiliki Adam *optimizer* sehingga model menjadi lebih baik dalam melakukan klasifikasi dan mendapatkan nilai akurasi yang sedikit lebih baik (0.9704) daripada model FFNN buatan sendiri (0.9646).

## 3.2. Saran

1. Untuk implementasi *Auto Diff* seharusnya bisa lebih dipertimbangkan lebih baik lagi mengenai performanya, terutama dalam aspek penggunaan memori.
2. Mungkin dalam hal pembentukan arsitektur model, bisa dilakukan lebih banyak lagi eksperimen untuk mengetahui seberapa dalam, seberapa lebar, ataupun fungsi aktivasi dan metode inisialisasi bobot mana yang paling bagus untuk menghasilkan hasil yang terbaik.

# Pembagian Tugas

Kegiatan	Nama (NIM)
Desain struktur kelas (VarValue, Layer, FFNN)	Steven Tjhia (13522103)
Implementasi fungsi forward() di Layer	Steven Tjhia (13522103)
Implementasi fungsi backward() dan update bobot	Denise Felicia Tiowanni (13522013)
Penanganan fungsi aktivasi (ReLU, Sigmoid, Softmax, dll)	Denise Felicia Tiowanni (13522013) Evelyn Yosiana (13522083) Steven Tjhia (13522103)
Pembuatan fungsi loss (MSE, BCE, CCE)	Denise Felicia Tiowanni (13522013) Evelyn Yosiana (13522083) Steven Tjhia (13522103)
Implementasi proses training (fit()) dan prediksi (predict())	Evelyn Yosiana (13522083)
Visualisasi arsitektur jaringan (visualize())	Evelyn Yosiana (13522083)
Visualisasi distribusi bobot dan gradien dan grafik loss per epoch	Denise Felicia Tiowanni (13522013)
Regularisasi	Evelyn Yosiana (13522083)
Pembuatan Laporan	Denise Felicia Tiowanni (13522013) Evelyn Yosiana (13522083) Steven Tjhia (13522103)

# Referensi

GeeksforGeeks. (2025, March 1). Building Artificial Neural Networks (ANN) from Scratch. GeeksforGeeks.

<https://www.geeksforgeeks.org/building-artificial-neural-networks-ann-from-scratch/>

GeeksforGeeks. (2023, October 25). Xavier initialization. GeeksforGeeks.

<https://www.geeksforgeeks.org/xavier-initialization/>

Institut Teknologi Bandung. (2024). Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin 2024/2025. Retrieved March 28, 2025, from

≡ Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin

Papers with Code - Kaiming Initialization Explained. (n.d.).

<https://paperswithcode.com/method/he-initialization>