

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



TUGAS BESAR 1

Pembelajaran Mesin Feedforward Neural Network

Dibuat oleh:

Group 12

Anggota:

13522032 - Tazkia Nizami

13522040 - Dhidit Abdi Aziz

13522074 - Muhammad Naufal Aulia

IF3270 - Pembelajaran Mesin

2025

Daftar Isi

Daftar Isi.....	1
Bab I. Deskripsi Persoalan.....	2
Bab II. Pembahasan.....	3
2.1. Penjelasan Implementasi.....	3
2.1.1. Penjelasan tentang FFNN.....	3
2.1.2. Deskripsi kelas beserta deskripsi atribut dan methodnya.....	4
2.1.3. Penjelasan forward propagation.....	13
2.1.4. Penjelasan backward propagation dan weight update.....	14
2.2. Hasil pengujian.....	15
2.2.1. Pengaruh depth dan width.....	15
2.2.2. Pengaruh fungsi aktivasi.....	21
2.2.3. Pengaruh learning rate.....	29
2.2.4. Pengaruh inisialisasi bobot.....	32
2.2.5. Pengaruh regularisasi.....	38
2.2.6. Pengaruh normalisasi RMS Norm.....	41
2.2.7. Perbandingan dengan library sklearn.....	44
Bab III. Kesimpulan dan Saran.....	46
3.1. Kesimpulan.....	46
3.2. Saran.....	46
Pembagian Tugas.....	47
Referensi.....	48

Bab I. Deskripsi Persoalan

Tugas Besar 1 mata kuliah IF3270 Pembelajaran Mesin bertujuan untuk mengimplementasikan modul Feedforward Neural Network (FFNN) dari awal menggunakan Python dengan hanya memanfaatkan library matematika seperti NumPy. Tugas ini mencakup:

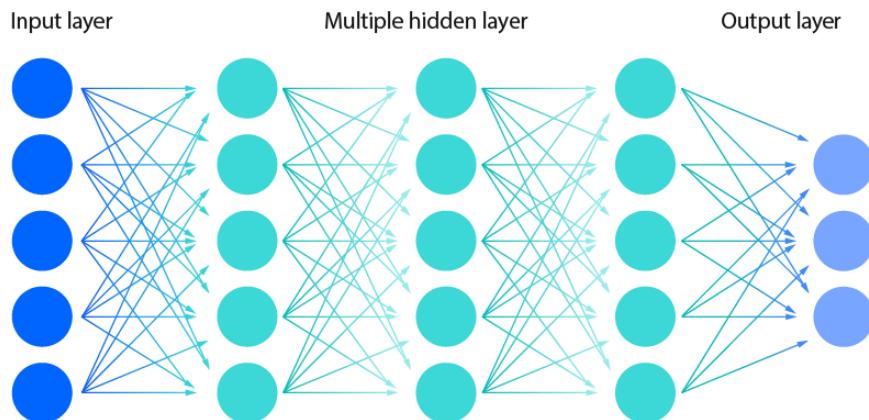
- Arsitektur FFNN : Membangun modul yang fleksibel untuk menentukan jumlah neuron di setiap lapisan (input, hidden, output) dan mendukung berbagai fungsi aktivasi (Linear, ReLU, Sigmoid, Tanh, Softmax, ELU, dan LeakyRELU).
- Fungsi Loss : Implementasi fungsi loss umum seperti Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy.
- Inisialisasi Bobot : Menyediakan metode inisialisasi bobot (Zero, Random Uniform, Random Normal) serta kemampuan menyimpan dan memuat model.
- Forward dan Backward Propagation : Mengimplementasikan algoritma forward propagation untuk batch input dan backward propagation menggunakan aturan rantai (chain rule) untuk menghitung gradien. Proses pembaruan bobot menggunakan gradient descent juga harus diimplementasikan.
- Regularisasi L1 dan L2 dan Normalisasi RMS: Mengimplementasikan regularisasi L1 dan L2 serta normalisasi RMS untuk menyeragamkan nilai bobot
- Visualisasi dan Analisis : Modul harus dapat memvisualisasikan struktur jaringan, distribusi bobot dan gradien, serta menyediakan fitur penyimpanan (save) dan pemuatan (load) model.
- Pengujian dan Evaluasi : Melakukan pengujian untuk menganalisis pengaruh hyperparameter (jumlah layer, jumlah neuron, fungsi aktivasi, learning rate, metode inisialisasi) terhadap performa model. Selain itu, hasil prediksi model FFNN yang dibangun harus dibandingkan dengan model MLP dari library scikit-learn menggunakan dataset MNIST.

Tugas ini bertujuan memberikan pemahaman praktis tentang konsep dasar FFNN, mulai dari arsitektur, fungsi aktivasi dan loss, pelatihan (forward/backward propagation, weight update), hingga evaluasi performa dan eksperimen hyperparameter.

Bab II. Pembahasan

2.1. Penjelasan Implementasi

2.1.1. Penjelasan tentang FFNN



Implementasi algoritma Feedforward Neural Network (FFNN) dalam file `model.py` terangkum dalam kelas utama FFNN yang menyediakan struktur dan fungsi esensial jaringan saraf tiruan feedforward. Kelas ini diinisialisasi dengan menerima konfigurasi arsitektur melalui `layer_sizes`, metode inisialisasi bobot, fungsi aktivasi untuk setiap layer, dan fungsi loss yang akan digunakan, kemudian membangun list layer menggunakan kelas Layer yang berada di `layer.py` serta menginisiasi dictionary untuk menyimpan riwayat loss pelatihan dan validasi.

Operasi inti jaringan saraf, yaitu forward propagation, diimplementasikan dalam metode `forward` yang meneruskan input melalui seluruh layer, sementara backward propagation untuk menghitung gradien dilaksanakan oleh metode `backward` yang memanfaatkan turunan fungsi loss dan memanggil metode `backward` pada setiap layer secara terbalik, dan pembaruan bobot berdasarkan gradien dilakukan oleh metode `update_weights`.

Proses pelatihan model diatur dalam metode `train` yang mengimplementasikan mini-batch gradient descent, melakukan shuffling data, menghitung loss per batch, menjalankan forward dan backward pass, memperbarui bobot, serta mencatat riwayat loss training dan validasi sesuai dengan tingkat verbositas yang ditentukan.

Selain itu, kelas FFNN juga menyediakan metode `predict` untuk melakukan inferensi, `save` dan `load` untuk menyimpan dan memuat model menggunakan pickle, serta serangkaian metode visualisasi seperti `plot_model` untuk menampilkan struktur jaringan dengan bobot, dan `plot_weight_distribution` serta `plot_gradient_distribution` untuk memvisualisasikan distribusi nilai bobot dan gradien pada layer yang dipilih menggunakan `matplotlib` dan `networkx`. Secara keseluruhan, kode ini menyediakan implementasi lengkap dari sebuah

model FFNN yang memenuhi spesifikasi, memungkinkan konfigurasi, pelatihan, evaluasi, dan visualisasi.

2.1.2. Deskripsi kelas beserta deskripsi atribut dan methodnya

Implementasi model FFNN ini dibagi menjadi beberapa kelas, terdapat kelas Activation, Loss, Initialization, Layer, dan kelas utamanya yakni kelas FFNN.

2.1.2.1. Kelas Activation

Merupakan kelas abstrak untuk fungsi aktivasi yang akan digunakan pada tiap layer. Setiap kelas lainnya merupakan turunan dari kelas Activation yang mengimplementasikan fungsi aktivasi berbeda.

Method:

- `__call__(x)`: Method aktivasi yang dapat dipanggil untuk diterapkan pada input.
- `derivative(x)`: Mengembalikan turunan dari fungsi aktivasi.

Kelas turunan:

1. Linear

Kelas turunan untuk fungsi aktivasi linear ($f(x) = x$)

Method:

- `__call__(x)`: Mengembalikan input langsung
- `derivative(x)`: Mengembalikan array berisi 1 dengan ukuran yang sama dengan input

2. ReLU

Kelas turunan untuk fungsi aktivasi ReLU ($f(x) = \max(0, x)$)

Method:

- `__call__(x)`: Mengembalikan 0 untuk input negatif, input asli untuk input positif
- `derivative(x)`: Mengembalikan 0 untuk input negatif, 1 untuk input positif

3. Sigmoid

Kelas turunan untuk fungsi aktivasi Sigmoid ($f(x) = 1 / (1 + e^{-x})$)

Method:

- `__call__(x)`: Menghitung sigmoid
- `derivative(x)`: Menghitung turunan sigmoid

4. Tanh

Kelas turunan untuk fungsi aktivasi Sigmoid ($f(x) = (e^x - e^{-x}) / (e^x + e^{-x})$)

Method:

- `__call__(x)`: Menghitung tanh
- `derivative(x)`: Menghitung turunan tanh

5. Softmax

Kelas turunan untuk fungsi aktivasi Sigmoid ($f(x)_i = e^{(x_i)} / \text{sum}(e^{(x_j)})$)

Method:

- `__call__(x)`: Menghitung softmax
- `derivative(x)`: Mengembalikan array berisi 1

6. Leaky ReLU - BONUS

ReLU memiliki permasalahan “Dying ReLU”, yaitu ketika x memiliki nilai negatif maka turunannya akan selalu 0 sehingga berpotensi gradien suatu neuron tidak terupdate (dead neuron). Maka dari itu, dibuatlah Leaky ReLU untuk mengatasi permasalahan tersebut yaitu mengalikan nilai x negatif dengan alpha (diinisialisasi 0.01) terlebih dahulu .

Method:

- `__call__(x)`: Menghitung Leaky ReLU dengan alfa 0.01

$$f(x) = \max(0.1x, x)$$

- `derivative(x)`: Mengembalikan array berisi 1 jika x positif, sedangkan alfa apabila negatif atau nol

7. Exponential Linear Unit (ELU) - BONUS

Sama seperti Leaky ReLU, ELU juga bertujuan untuk memperbaiki permasalahan “Dying ReLU”. Perbedaannya terletak pada nilai negatif yang dipetakan secara eksponensial alih-alih hanya nilai 0 mutlak

Method:

- `__call__(x)`: Menghitung ELU dengan alfa = 1

$$\begin{cases} x & \text{for } x \geq 0 \\ \alpha(e^x - 1) & \text{for } x < 0 \end{cases}$$

- `derivative(x)`: Mengembalikan array berisi 1 apabila x positif. Sedangkan nilai ELU + alfa.x jika kurang dari nol

2.1.2.2. Kelas Loss

Merupakan kelas abstrak untuk fungsi Loss yang akan digunakan untuk menghitung error. Setiap kelas lainnya merupakan turunan dari kelas ini yang mengimplementasikan fungsi Loss berbeda.

Method:

- `__call__(x)`: Method aktivasi yang dapat dipanggil untuk diterapkan pada input.
- `derivative(x)`: Mengembalikan nilai loss dari `y_true` dengan `y_pred`

Kelas turunan:

1. Mean Squared Error (MSE)

MSE menghitung loss dengan cara rata-rata dari selisih antara `ytrue` dan `ypred` dikuadratkan

Method:

- `__call__(x)`: Menghitung selisih antara nilai sebenarnya dan nilai prediksi, kemudian mengkuadratkannya, lalu mengambil rata-rata
- `derivative(x)`: Mengembalikan turunan MSE terhadap `ypred` sehingga didapatkan gradien yang bermanfaat untuk memperbarui bobot

2. Binary Cross Entropy (BCE)

BCE umumnya digunakan dalam klasifikasi biner. Fungsi ini digunakan ketika output berupa probabilitas (antara 0 dan 1) yang sebelumnya sudah dihasilkan dari sigmoid

Method:

- `__call__(x)`: Pertama-tama nilai dinormalisasi untuk menghindari nilai $\log(0)$ yang dapat menyebabkan eror pembagian oleh nol serta memastikan nilai y_{pred} selalu berada di rentang yang betul. Kemudian dihitung loss dengan rumus BCE yaitu

$$L_{BCE} = -\frac{1}{n} \sum_{i=1}^n (Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

- `derivative(x)`: Mengembalikan turunan BCE terhadap y_{pred} sehingga didapatkan gradien yang bermanfaat untuk memperbarui bobot

3. Categorical Cross Entropy (CCE)

CCE umumnya digunakan dalam klasifikasi multi-kelas. Fungsi ini digunakan ketika label dikodekan sudah dalam bentuk one-hot vector.

Method:

- `__call__(x)`: Menghitung kelas yang benar yang berkontribusi terhadap loss melalui fungsi sebagai berikut

$$CE = - \sum_{i=1}^{i=N} y_{true_i} \cdot \log(y_{pred_i})$$

- `derivative(x)`: Mengembalikan turunan CCE terhadap y_{pred} sehingga didapatkan gradien yang bermanfaat untuk memperbarui bobot

2.1.2.3. Kelas Layer

Merupakan kelas yang menyimpan cara kerja dari sebuah layer pada model FFNN ini. Kelas ini menyimpan beberapa atribut dan method, berikut adalah

penjelasannya.

Atribut:

- Input_size: Menyimpan informasi banyaknya fitur input yang diterima dari layer sebelumnya. Untuk layer pertama akan sesuai jumlah fitur dataset.
- output_size: menyimpan banyak output / neuron yang dihasilkan pada layer ini.
- activation: menyimpan objek fungsi aktivasi yang akan diterapkan pada output layer ini. Atribut ini akan berisi list dari objek Fungsi Aktivasi seperti ReLU, Sigmoid, dan lain-lain.
- weight_bias_initializer: menyimpan objek *initializer* yang digunakan untuk menginisialisasi bobot dan bias layer. Jika tidak diberikan saat pembuatan objek Layer, maka akan menggunakan *initializer* He.
- weights: menyimpan matriks numpy yang memiliki bobot dari koneksi antara neuron-neuron di layer sebelumnya dengan neuron-neuron di layer ini. Dimensinya akan (input_size, output_size).
 - Setiap baris akan merepresentasikan bobot dari semua input yang terhubung ke satu neuron di layer ini. Sehingga setiap kolom merepresentasikan bobot dari satu input yang terhubung ke semua neuron di layer ini.
- biases: menyimpan vektor baris numpy yang memiliki nilai bias untuk setiap neuron di layer ini. Dimensinya adalah (1, output_size). Bias ditambahkan ke hasil perkalian input dengan bobot sebelum fungsi aktivasi diterapkan.
- weights_grad: Ini adalah matriks numpy dengan dimensi yang sama dengan weights. Atribut ini digunakan untuk menyimpan nilai gradien (turunan) dari loss function terhadap bobot layer ini selama proses backpropagation. Nilai ini akan digunakan untuk memperbarui bobot.
- biases_grad: sama seperti weights_grad, namun untuk memperbarui bias.

- input: menyimpan input yang diterima oleh layer selama proses forward pass. Input ini disimpan untuk digunakan kembali selama proses backward pass dalam perhitungan gradien.
- output_before_activation: menyimpan output dari layer setelah perkalian dengan bobot dan penambahan bias, tetapi sebelum fungsi aktivasi diterapkan. Nilai ini diperlukan untuk menghitung turunan fungsi aktivasi selama backward pass.
- output: menyimpan output akhir dari layer setelah fungsi aktivasi diterapkan. Output ini akan menjadi input untuk layer berikutnya (jika ada) atau output dari seluruh jaringan.

Method:

- `__init__(input_size, output_size, activation, initializer)`: konstruktor kelas, dipanggil ketika sebuah Layer diinisialisasi.
- `forward(x)`: Melakukan forward pass melalui layer. x pada parameter adalah input yang akan diteruskan.
- `backward(grad_output)`: Melakukan backpropagation pada layer ini, grad_output adalah gradient dari layer selanjutnya yang digunakan dalam proses backpropagation di layer ini.
- `update_weights(learning_rate)`: memperbarui bobot dan bias layer menggunakan algoritma gradient descent.

2.1.2.4. Kelas Initialization

Merupakan kelas abstrak dasar untuk inisialisasi bobot pada lapisan neural network. Setiap kelas turunan akan mengimplementasikan metode inisialisasi bobot yang berbeda-beda.

Method:

- `__call__(shape)`: Method abstrak untuk menginisialisasi bobot dengan bentuk (shape) tertentu.

Kelas turunan:

1. ZeroInitialization

Kelas turunan yang menginisialisasi yang mengisi semua bobot dengan nilai nol.

Method:

- `__call__(shape)`: Mengembalikan array numpy dengan ukuran shape yang diisi dengan nilai 0.

2. UniformInitialization

Kelas turunan untuk inisialisasi yang mengisi bobot dengan bilangan acak yang terdistribusi secara seragam (uniform).

Atribut:

- `low`: Batas bawah rentang bilangan acak
- `high`: Batas atas rentang bilangan acak
- `seed`: Seed untuk reproducibility pembangkitan bilangan acak

Method:

- `__init__(low, high, seed)`: Inisialisasi parameter distribusi uniform
- `__call__(shape)`: Mengembalikan array numpy dengan nilai acak terdistribusi seragam dalam rentang [low, high]

3. NormalInitialization

Kelas turunan untuk inisialisasi yang mengisi bobot dengan bilangan acak yang terdistribusi secara normal.

Atribut:

- `mean`: Rata-rata distribusi normal
- `variance`: Varians distribusi normal
- `seed`: Seed untuk reproducibility pembangkitan bilangan acak

Method:

- `__init__(mean, variance, seed)`: Inisialisasi parameter distribusi normal
- `__call__(shape)`: Mengembalikan array numpy dengan nilai acak terdistribusi normal dengan mean dan standar deviasi tertentu

4. XavierInitialization - BONUS

Kelas turunan untuk inisialisasi bobot yang mempertimbangkan jumlah neuron pada lapisan input dan output untuk menjaga varians tetap stabil.

Atribut:

- distribution: Jenis distribusi yang digunakan ('uniform' atau 'normal', default 'uniform')
- seed: Seed untuk reproducibility pembangkitan bilangan acak

Method:

- `__init__(distribution, seed)`: Inisialisasi parameter Xavier initialization
- `__call__(shape)`: Mengembalikan array numpy dengan inisialisasi bobot menggunakan metode Xavier, dengan pilihan distribusi uniform atau normal

5. HeInitialization - BONUS

Kelas turunan untuk inisialisasi bobot yang mempertimbangkan jumlah neuron pada lapisan input.

Atribut:

- distribution: Jenis distribusi yang digunakan ('uniform' atau 'normal', default 'normal')
- seed: Seed untuk reproducibility pembangkitan bilangan acak

Method:

- `__init__(distribution, seed)`: Inisialisasi parameter He initialization
- `__call__(shape)`: Mengembalikan array numpy dengan inisialisasi bobot menggunakan metode He, dengan pilihan distribusi uniform atau normal

2.1.2.5. Kelas Model (FFNN)

Kelas ini berfungsi sebagai kelas utama untuk menjalankan program. Kelas-kelas yang sebelumnya didefinisikan di atas akan di-import dan dipergunakan pada kelas FFNN ini

Method:

- `__init__`: Menginisialisasi model dengan ukuran layer, metode inisialisasi bobot, fungsi aktivasi, dan fungsi loss

- forward(x): Melakukan forward pass pada input x melalui semua layer untuk menghasilkan output prediksi
- backward(x, y): Melakukan backward pass untuk menghitung gradien loss terhadap bobot. X merupakan data masukan sedangkan y adalah label target
- update_weights(learning_rate): Memperbarui bobot setiap layer berdasarkan hasil dari backward pass
- train(x_train, y_train, x_y_val=None, batch_size=32, learning_rate=0.01, epochs=10, verbose=1): Melatih model dengan data pelatihan dalam beberapa epoch
- predict(x): Melakukan prediksi untuk data input x
- save(filename): Menyimpan model ke dalam file menggunakan pickle
- load(filename) Melakukan loading model dari file
- plot_model(): Membuat visualisasi arsitektur jaringan dengan bantuan library networkx dan matplotlib
- plot_weight_distribution(layers=None): Menampilkan grafik histogram distribusi bobot dari layer tertentu
- plot_gradient_distribution(layers=None): Menampilkan grafik histogram distribusi gradien dari layer tertentu

2.1.2.6. Kelas RMS Norm - BONUS

Merupakan kelas untuk normalisasi RMS yang digunakan pada layer-layer di FFNN

Atribut:

- eps: Untuk mencegah pembagian dengan nol ketika menghitung sqrt
- gamma: Mengatur skala output setelah normalisasi
- grad_gamma: Gradien dari gamma yang dihitung saat backward pass
- input: Menyimpan input saat forward pass, dapat digunakan ulang saat backward
- norm: Nilai RMS dari input yang dihitung saat forward, digunakan ulang saat backward

Method:

- forward(x): Menerapkan RMS Norm ke input x
- backward(grad_output): Hitung gradien terhadap input dan gamma menggunakan chain rule
- updateweights(lr): Melakukan pembaruan bobot gamma berdasarkan gradiennya dengan learning rate lr

2.1.3. Penjelasan forward propagation

Metode forward pada kelas Layer (di 'layer.py') bertanggung jawab untuk melakukan forward pass melalui layer tersebut. Proses ini menerima input x dari layer sebelumnya (atau input data jika ini adalah layer pertama / input layer). Langkah-langkahnya adalah sebagai berikut:

1. Input x disimpan dalam atribut 'self.input'. Input ini disimpan karena nilai input ini akan digunakan kembali selama proses backward propagation untuk menghitung gradien bobot.
2. Perhitungan Output Sebelum Aktivasi, merupakan operasi linear dalam layer. Input x dikalikan dengan matriks bobot layer ('self.weights') menggunakan perkalian dot product ('np.dot'). Kemudian, vektor bias layer ('self.biases') ditambahkan ke hasil perkalian tersebut. Hasil dari operasi ini adalah output layer sebelum fungsi aktivasi diterapkan.
3. Fungsi aktivasi yang telah ditentukan saat inisialisasi layer (ada pada 'self.activation') diterapkan pada hasil perhitungan output sebelumnya. Hasil dari penerapan fungsi aktivasi ini adalah output akhir dari layer, yang kemudian akan menjadi input untuk layer berikutnya dalam jaringan.

2.1.4. Penjelasan backward propagation dan weight update

Metode backward pada kelas Layer bertanggung jawab untuk melakukan backward propagation melalui layer, yang bertujuan untuk menghitung gradien bobot dan bias terhadap fungsi loss. Metode ini menerima `grad_output`, yang merupakan gradien loss dari layer berikutnya (atau gradien loss terhadap output layer jika ini adalah layer terakhir). Langkah kerjanya adalah sebagai berikut:

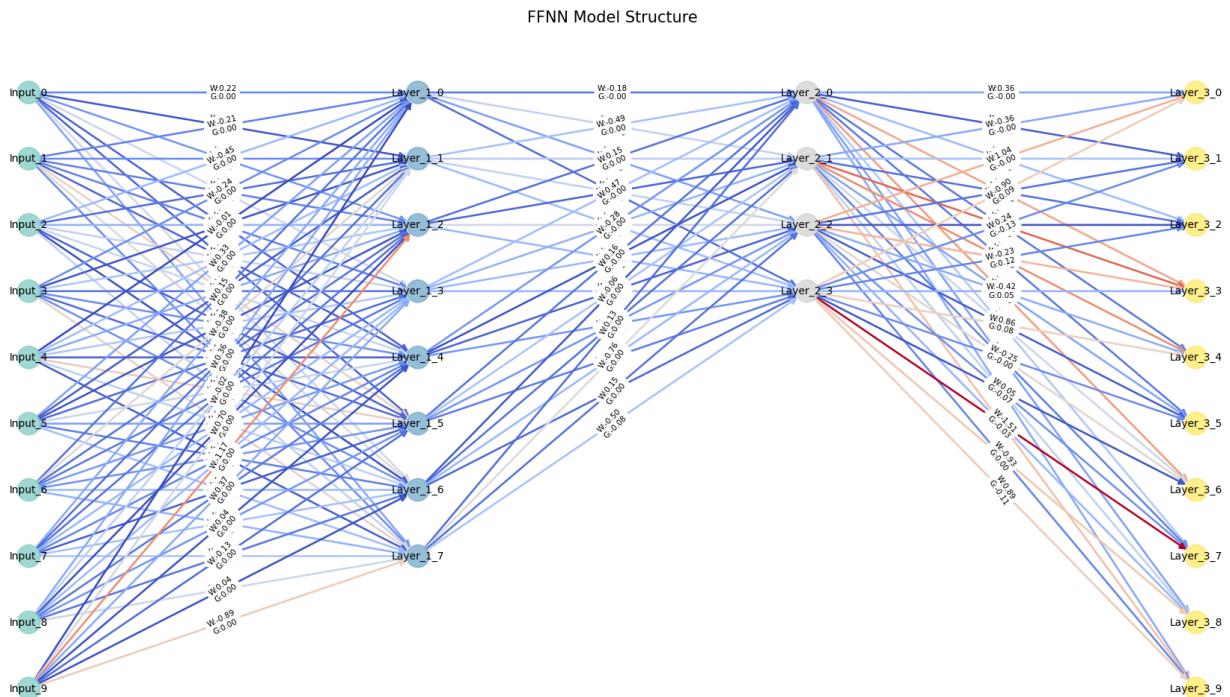
1. Perhitungan Gradien Input dilakukan dengan 2 kasus, yaitu ketika sebuah layer menggunakan Fungsi Aktivasi Softmax dan selain itu.
 - a. Terdapat penanganan khusus untuk fungsi aktivasi Softmax. Ini karena turunan dari fungsi Softmax sudah dihitung secara eksplisit dalam implementasi fungsi loss. Dalam kasus ini, karena fungsi aktivasi Softmax hanya diterapkan pada layer terakhir, maka gradien inputnya (`grad_input`) bisa langsung ditentukan untuk layer terakhir.
 - b. Untuk layer lainnya, ketika fungsi aktivasi bukan Softmax, gradien input untuk layer ini (`grad_input`) dihitung menggunakan aturan rantai (chain rule). Gradien dari layer berikutnya (`grad_output`) dikalikan dengan turunan pertama dari fungsi aktivasi layer saat ini, yang dievaluasi pada output sebelum aktivasi (`self.output_before_activation`). Turunan fungsi aktivasi diperoleh dengan memanggil metode derivative pada objek fungsi aktivasi.
2. Gradien loss terhadap bobot layer (`'self.weights_grad'`) dihitung dengan melakukan perkalian dot product antara transpose dari input layer saat forward pass (`'self.input.T'`) dan gradien input layer saat backward pass (`'grad_input'`).
3. Gradien loss terhadap bias layer (`'self.biases_grad'`) dihitung dengan menjumlahkan gradien input (`'grad_input'`)
4. Gradien yang akan diteruskan ke layer sebelumnya (`'grad_prev'`) dihitung dengan melakukan perkalian dot product antara gradien input layer saat ini (`'grad_input'`) dan transpose dari bobot layer saat ini

(`self.weights.T`). Gradien ini merepresentasikan seberapa besar perubahan pada input layer ini akan mempengaruhi loss.

Hasil dari perhitungan gradien tersebut kemudian diteruskan ke layer sebelumnya dan langkah-langkah tadi dilakukan kembali pada layer sebelumnya.

2.2. Hasil pengujian

Pada pemaparan hasil pengujian ini, hasil akhir yang dilampirkan hanya berupa hasil akurasi prediksi. Untuk hasil tampilan graf model berupa struktur jaringan beserta bobot dan gradien bobot tiap neuron kira-kira akan terlihat seperti berikut (contoh layer sederhana).

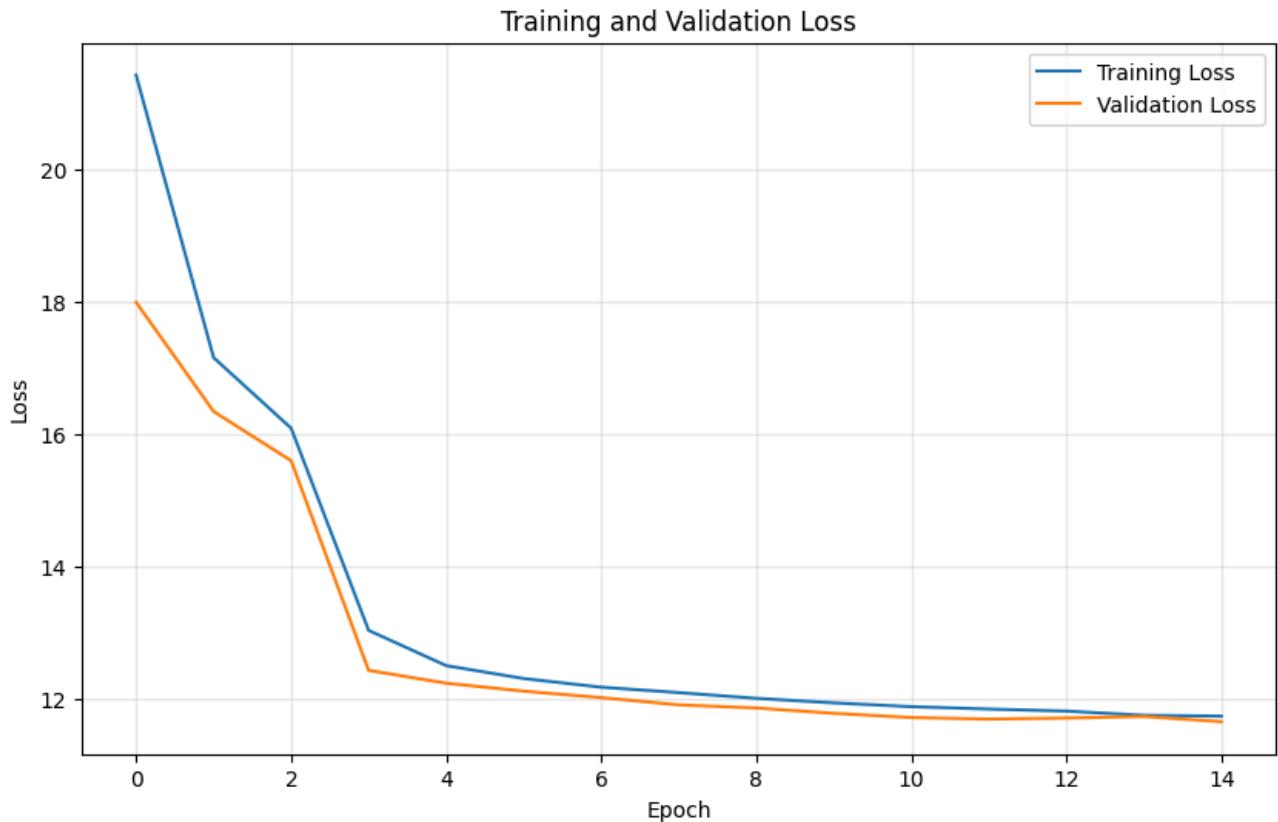


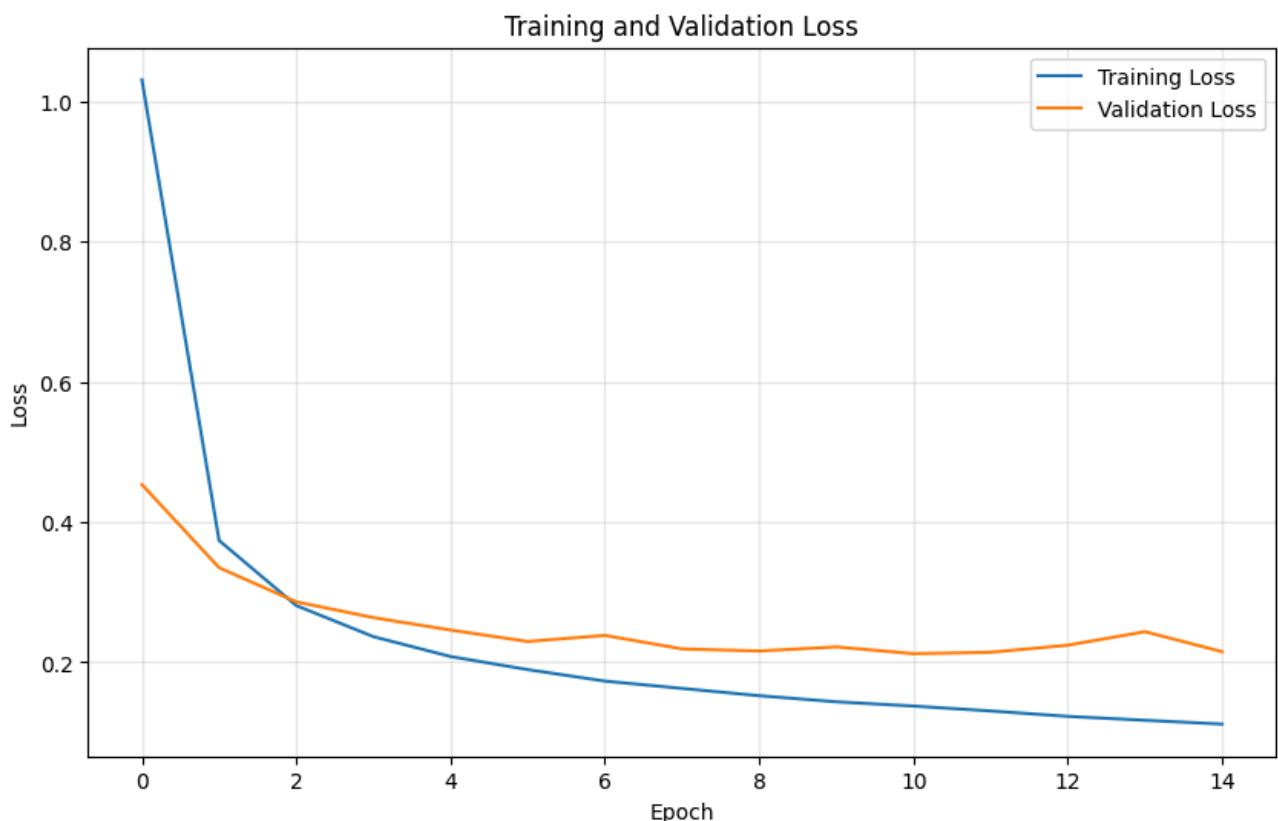
2.2.1. Pengaruh depth dan width

a. Pengaruh Depth

Asumsi, input layer tidak termasuk ke dalam depth (depth dimulai dari hidden layer pertama), dan width = 20. Pengujian dilakukan dengan menggunakan Fungsi Aktivasi ReLU di setiap layernya kecuali layer terakhir yang menggunakan Fungsi Aktivasi Softmax. Banyaknya Epoch adalah 15, Batch Size = 32, dan learning rate = 0.01. Untuk metode inisialisasi bobot yang digunakan adalah He dan Xavier secara bergantian tiap layernya dengan seed = 42.

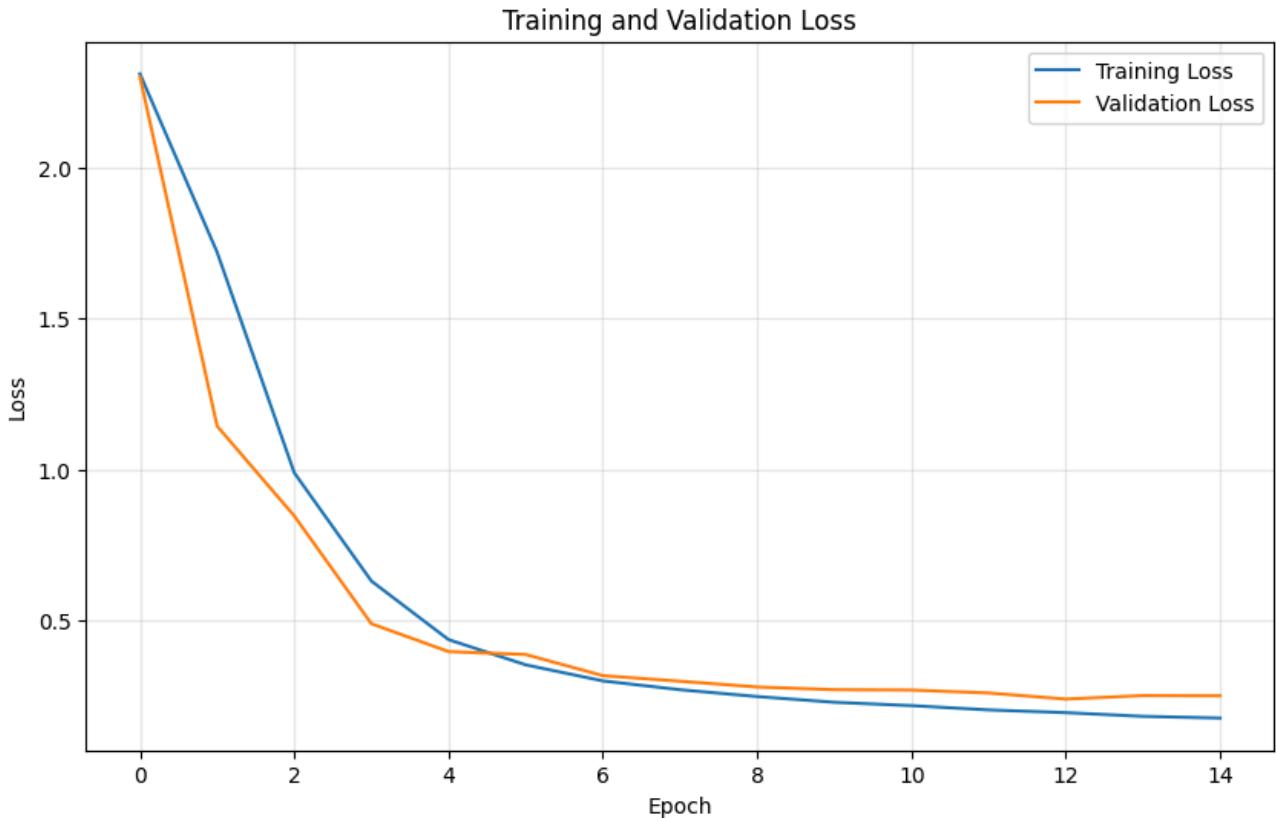
i. Depth = 5





Test accuracy: 0.9435

iii. Depth = 20

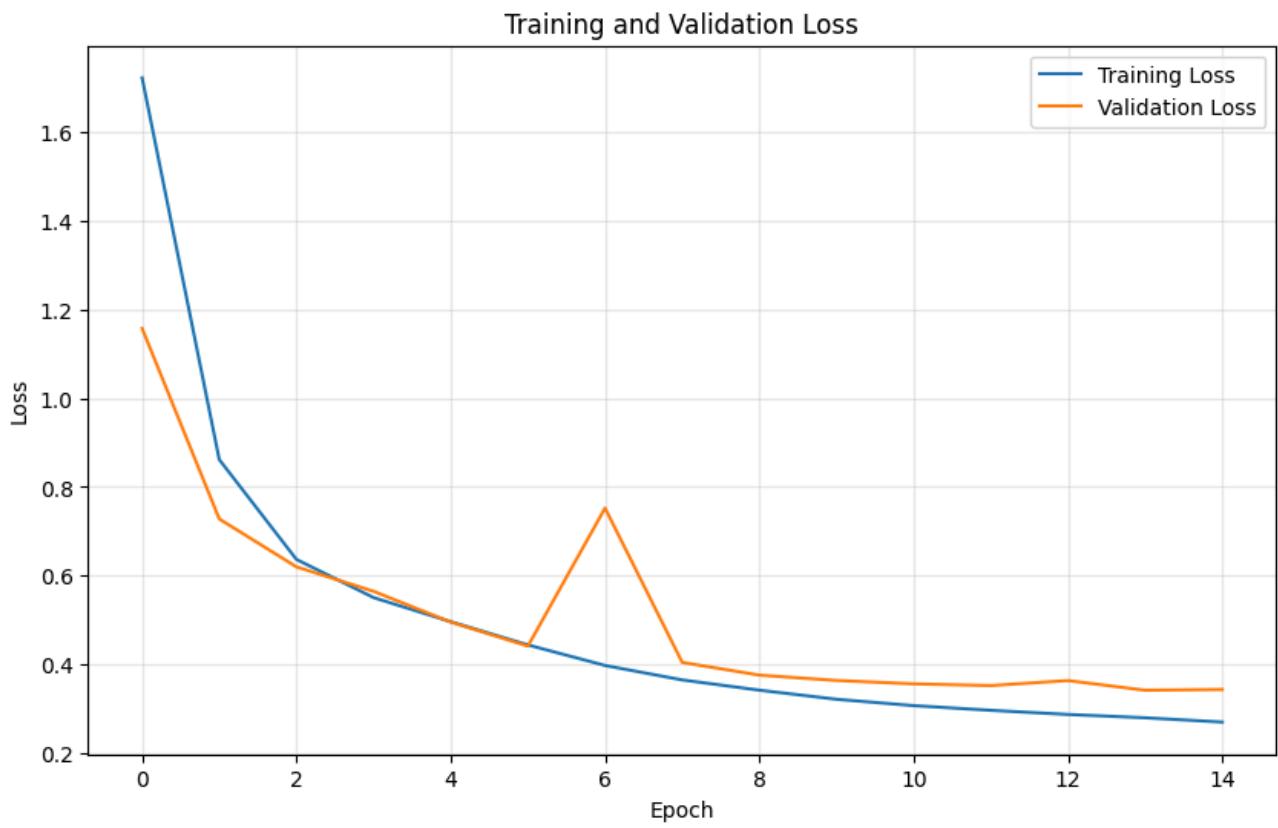


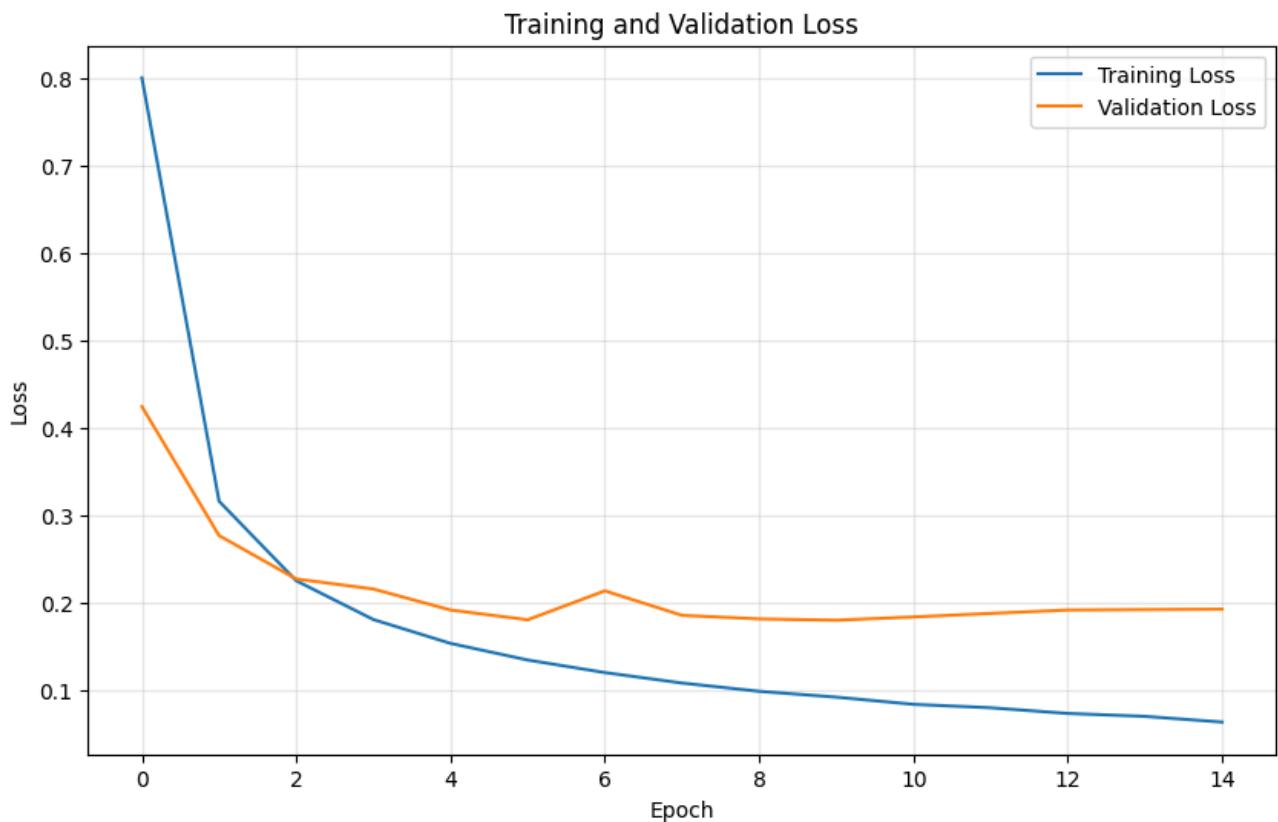
Berdasarkan ketiga hasil train dengan Depth beragam tersebut, dapat disimpulkan bahwa ukuran depth yang lebih kecil (depth = 5) memiliki akurasi yang lebih buruk daripada depth yang lebih besar (depth = 10 atau 20). Perbedaan akurasi antara depth = 10 dan depth = 20 sendiri sangat sedikit, dengan akurasi yang lebih tinggi dimiliki oleh model yang ditrain dengan depth = 10. Diperlukan beberapa pengujian lainnya untuk menemukan Depth yang tepat dengan akurasi tertinggi.

b. Pengaruh Width

Pada pengujian pengaruh width, depth diatur ke angka 10. Pengujian dilakukan dengan menggunakan Fungsi Aktivasi ReLu di setiap layernya kecuali layer terakhir yang menggunakan Fungsi Aktivasi Softmax. Banyaknya Epoch adalah 15, Batch Size = 32, dan learning rate = 0.01. Untuk metode inisialisasi bobot yang digunakan adalah He dan Xavier secara bergantian tiap layernya dengan seed = 42.

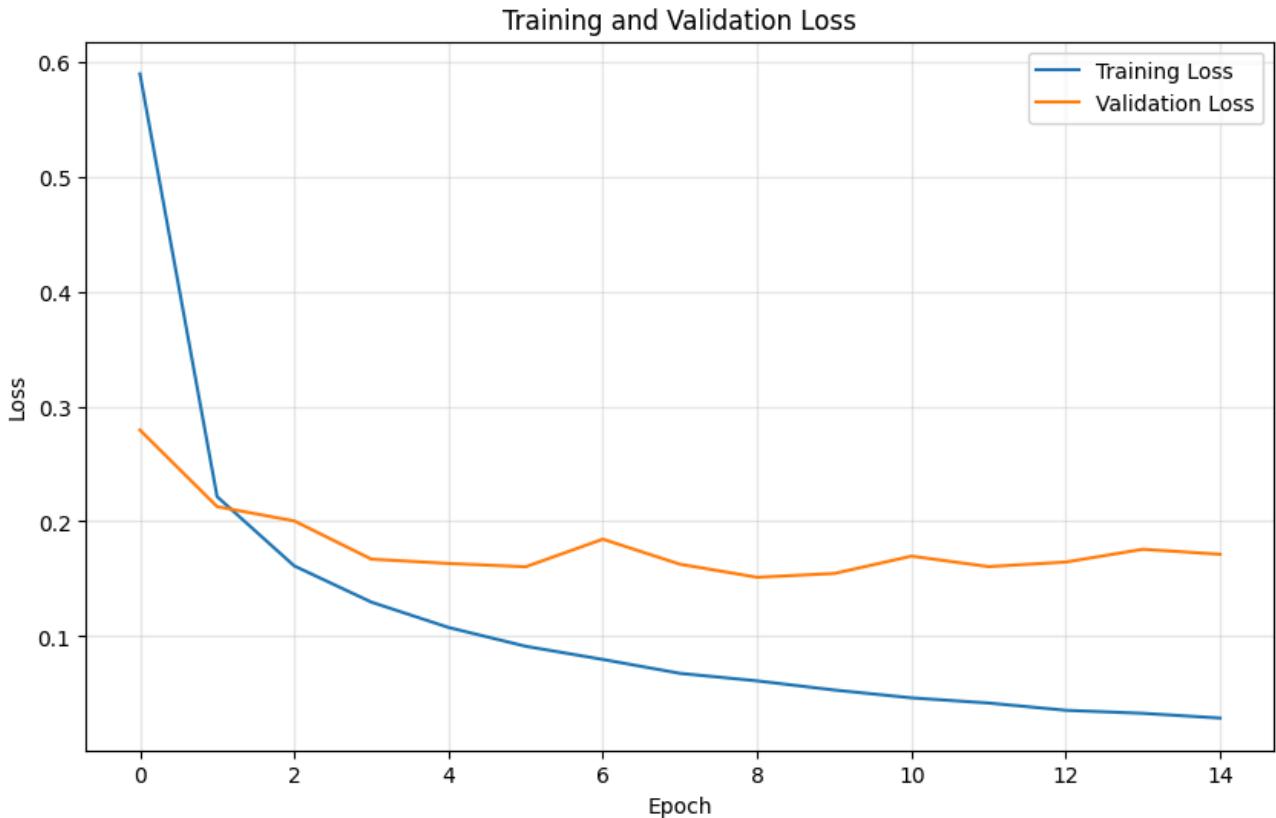
i. Width = 10





Test accuracy: 0.9538

iii. Width = 64

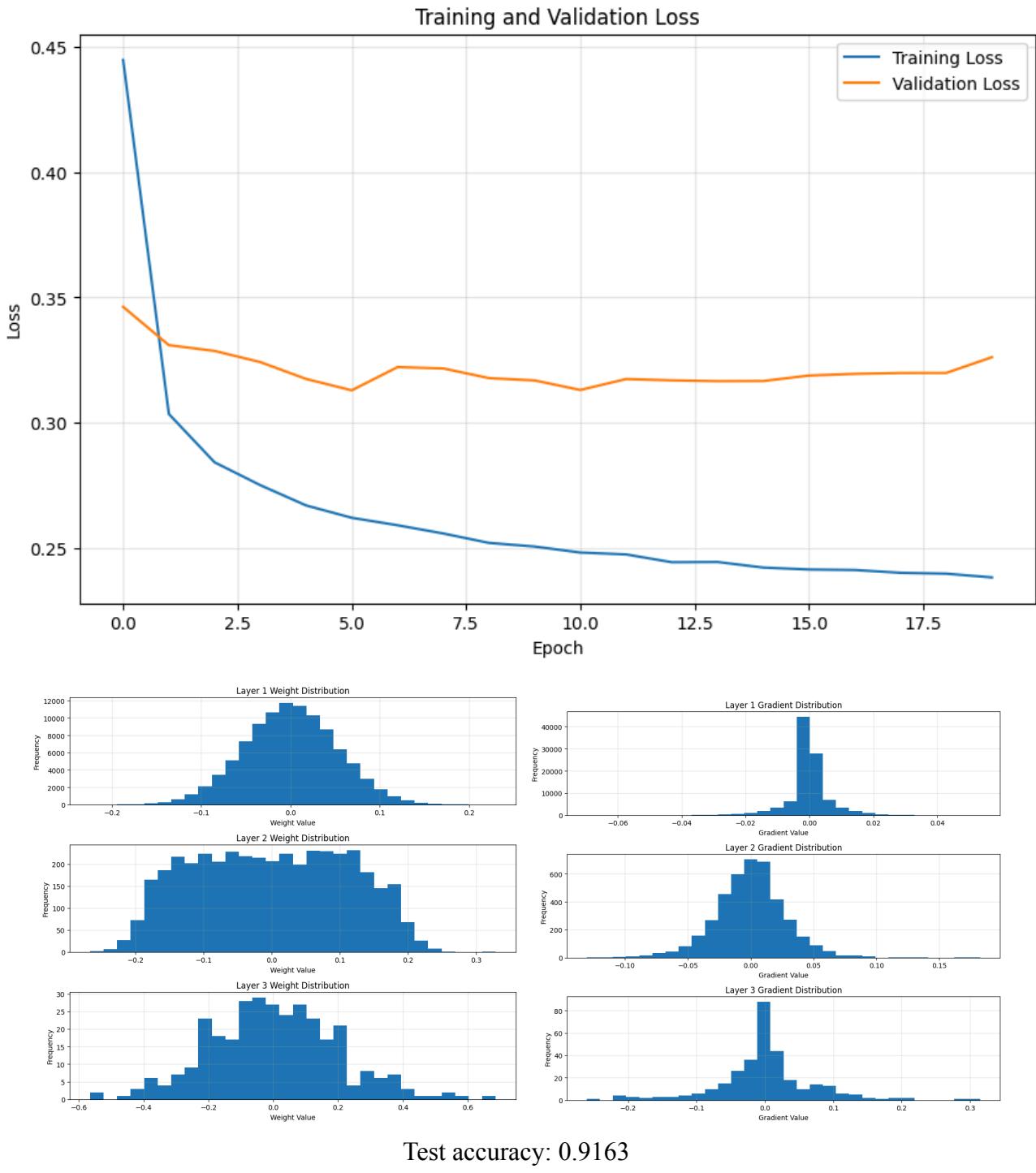


Dari hasil perbandingan tersebut, dapat disimpulkan bahwa model dengan width yang lebih besar (dalam hal ini width = 64) memiliki akurasi yang lebih baik daripada model dengan width yang lebih kecil (width = 10 dan 32). Maka kedepannya dapat diprediksi bahwa model dengan width yang lebih besar akan memiliki akurasi yang lebih baik.

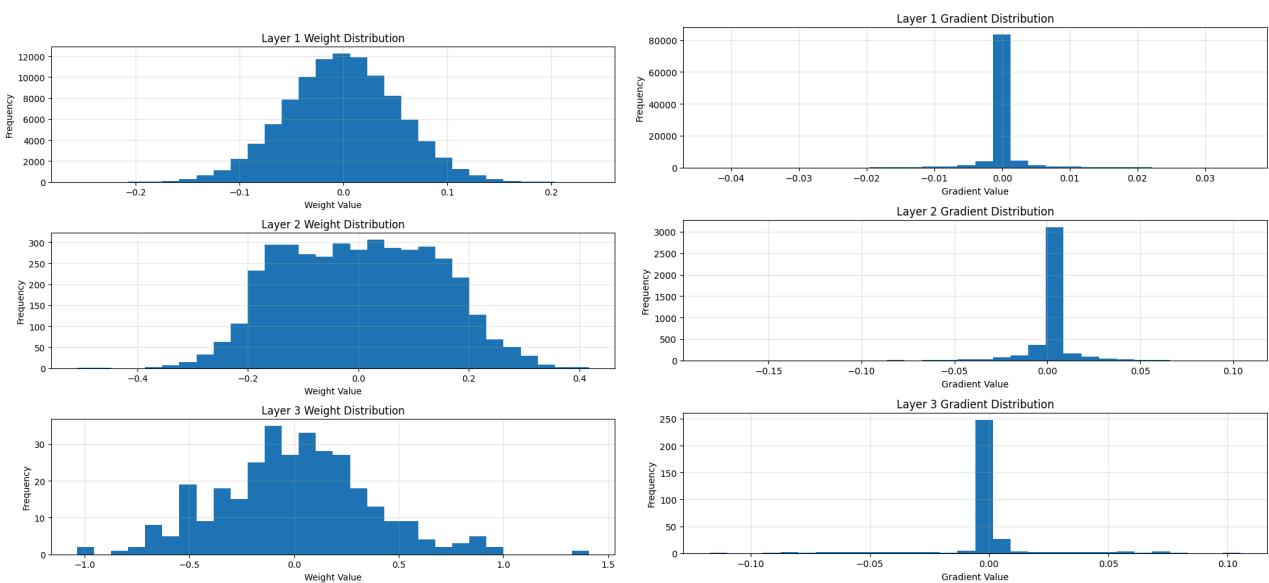
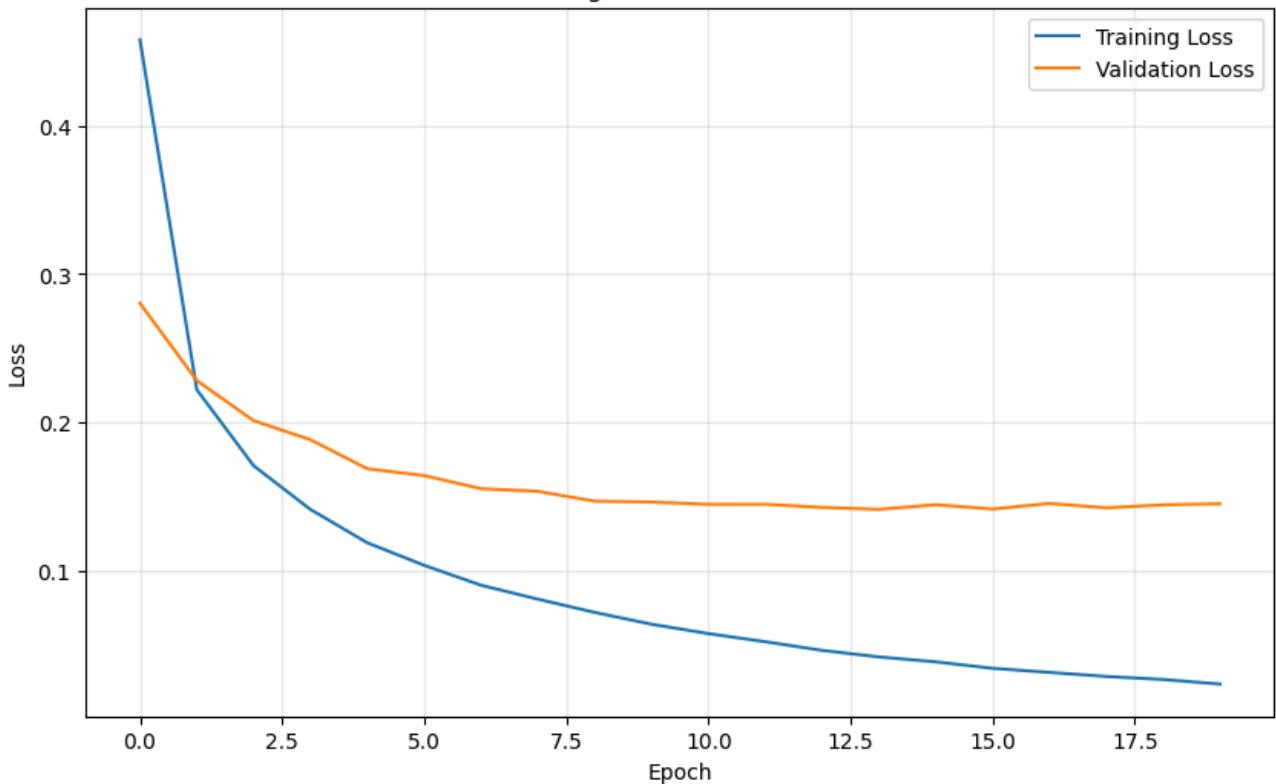
2.2.2. Pengaruh fungsi aktivasi

Pada pengujian Fungsi Aktivasi, pengujian dilakukan dengan menggunakan 3 layer dengan neuron 128, 32, dan 10. Banyak epoch adalah 20 dan learning rate 0.01. Fungsi aktivasi yang digunakan adalah sesuai dengan fungsi aktivasi yang sedang diuji pada bagian tersebut kecuali layer terakhir (selalu menggunakan Softmax), dengan initialization menggunakan He dan Xavier secara bergantian.

- a. Linear

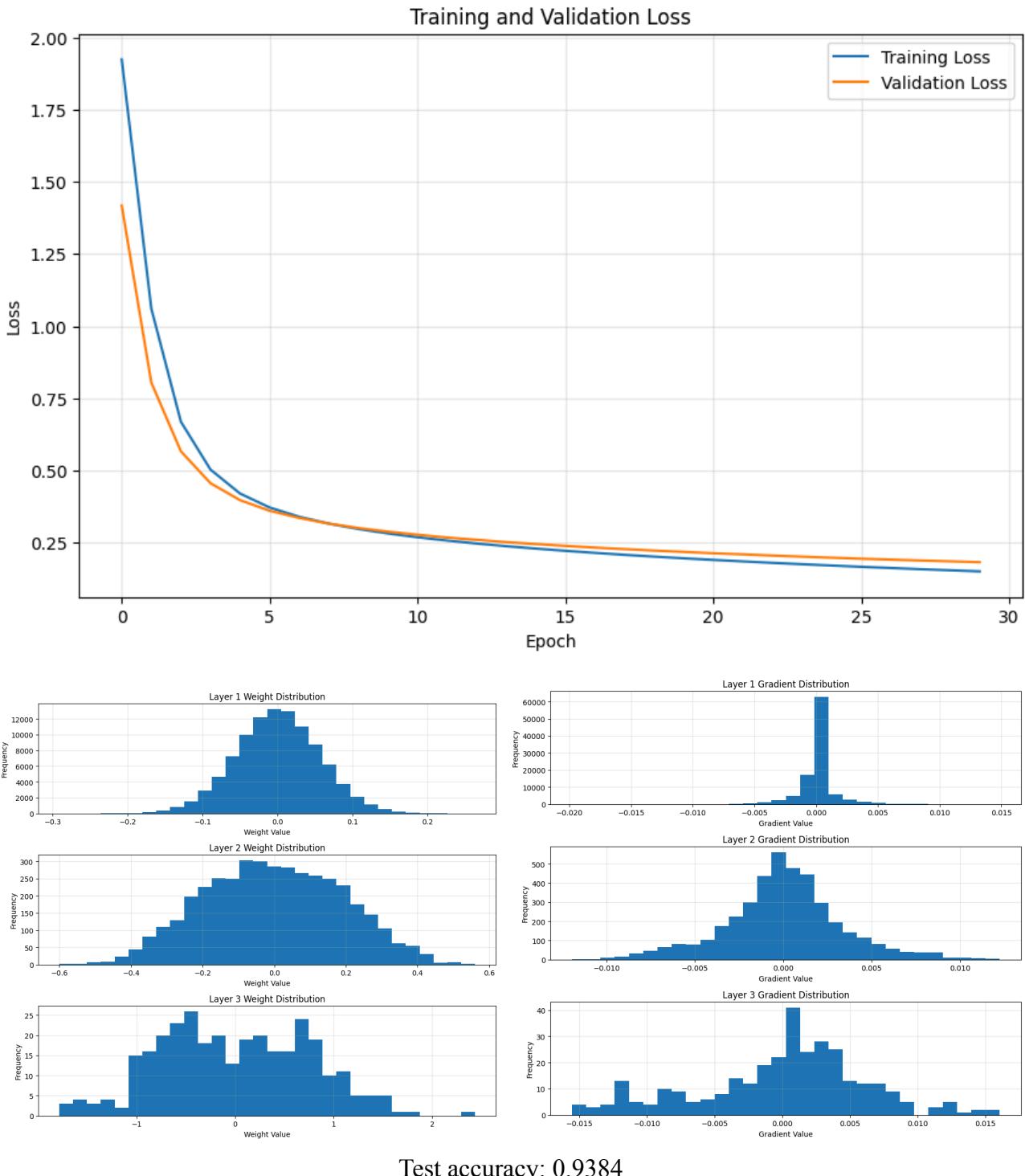


Training and Validation Loss



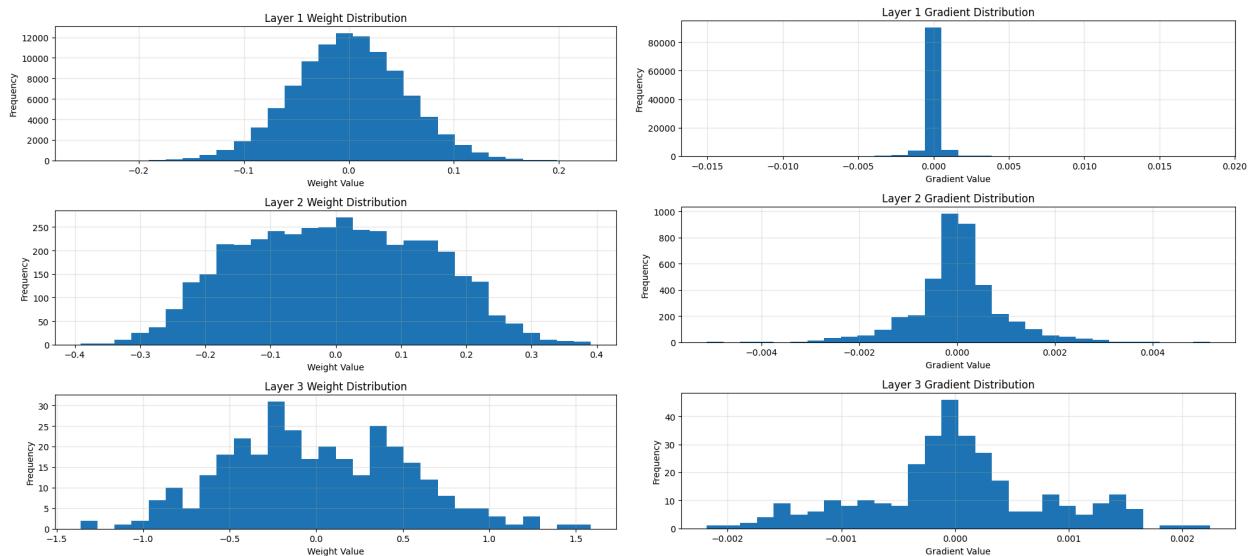
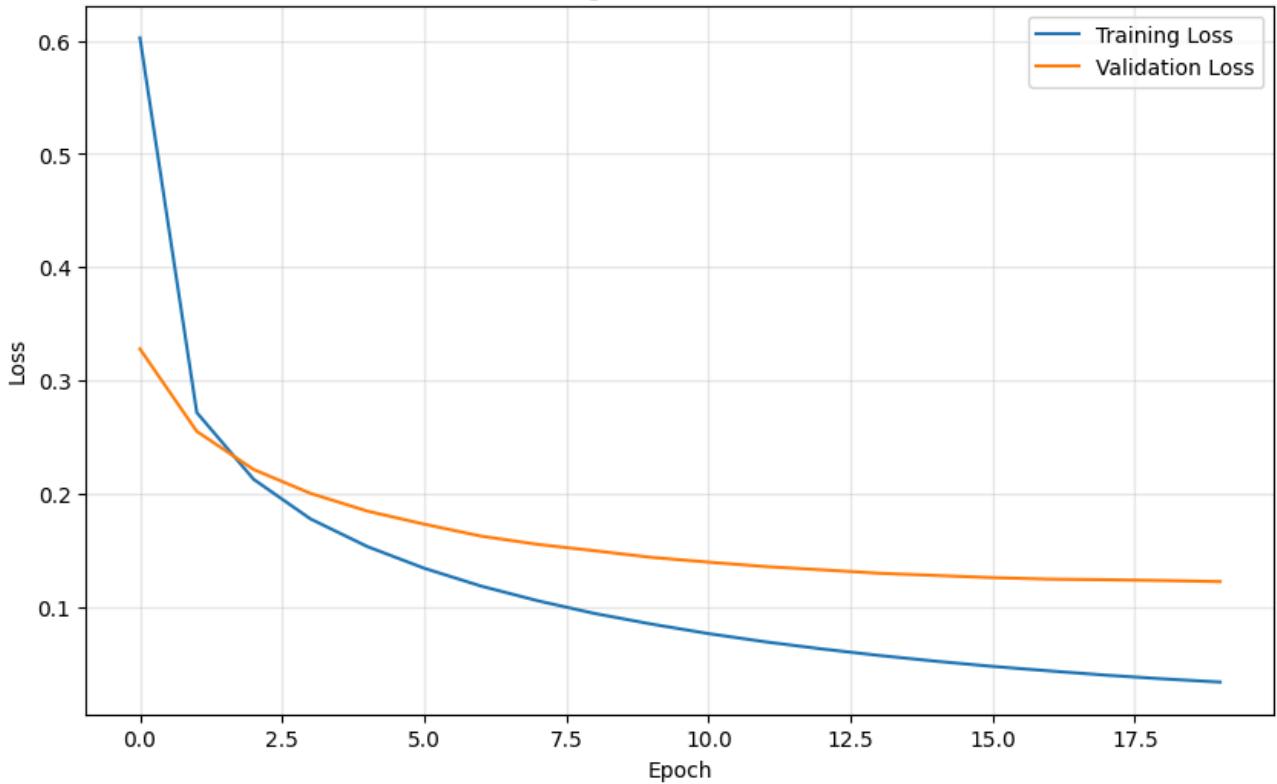
Test accuracy: 0.9651

c. Sigmoid



d. Tanh

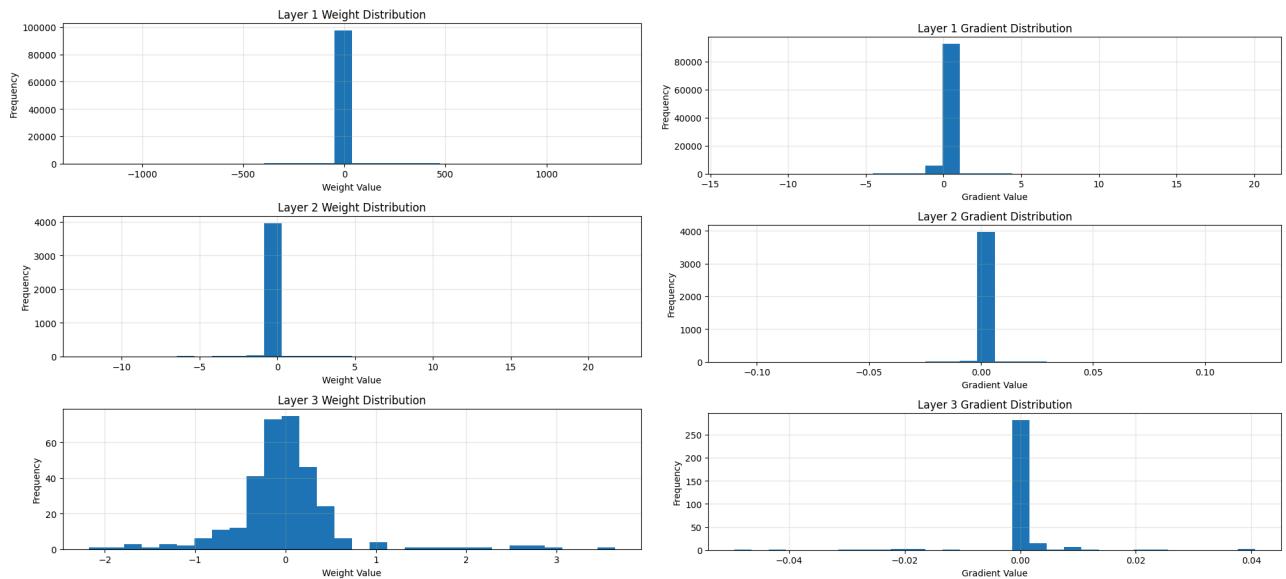
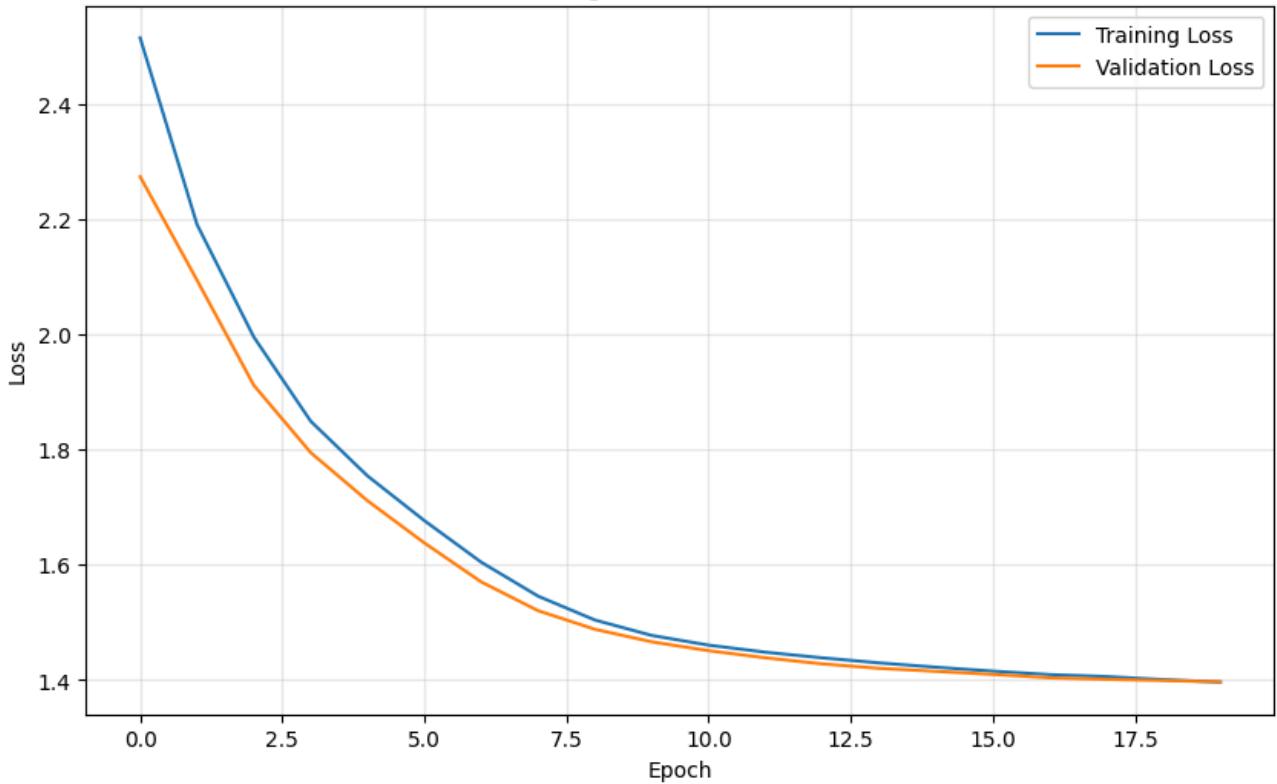
Training and Validation Loss



Test accuracy: 0.9656

e. Softmax

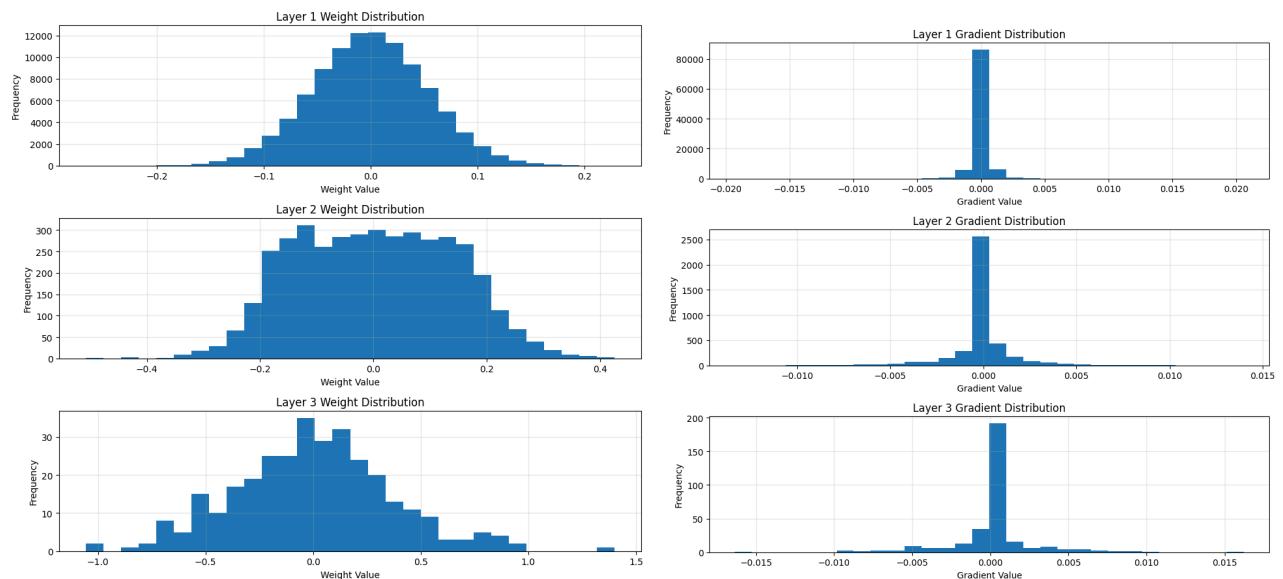
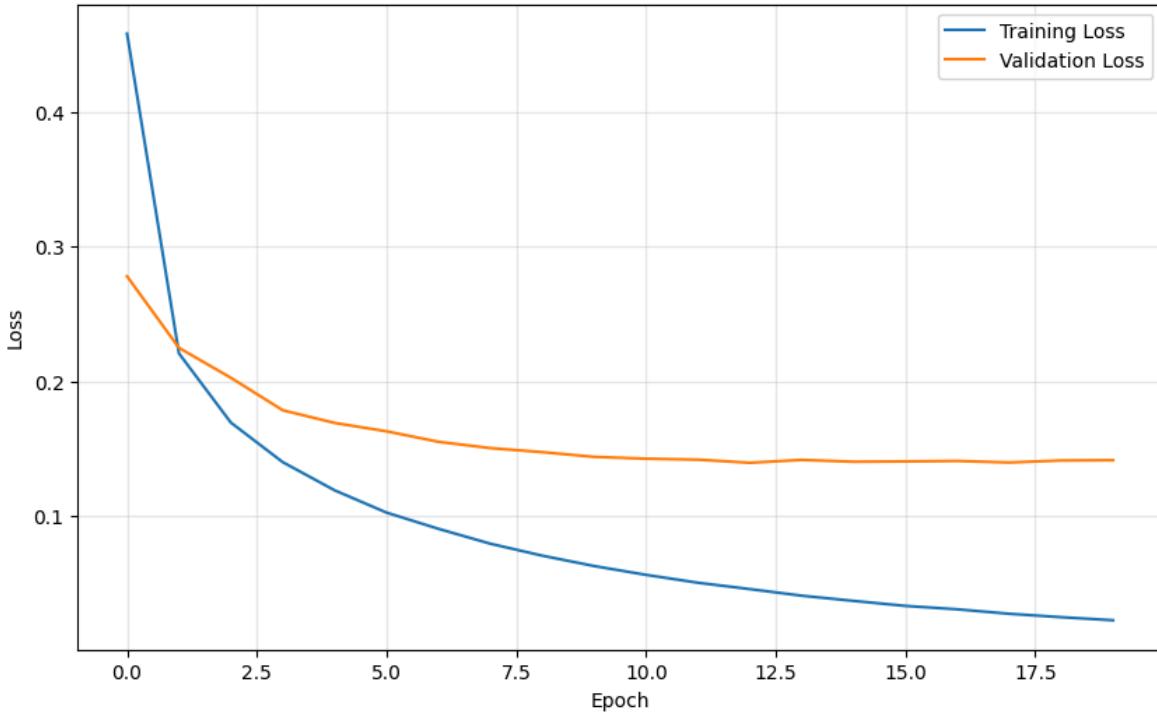
Training and Validation Loss



Test accuracy: 0.4554

f. LeakyReLU

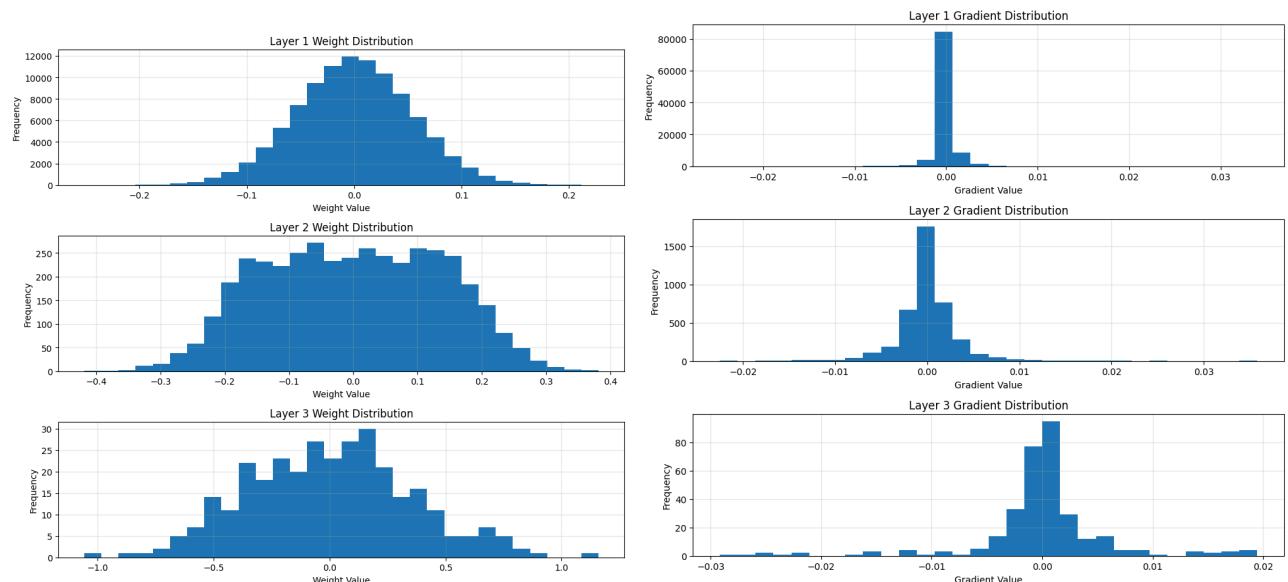
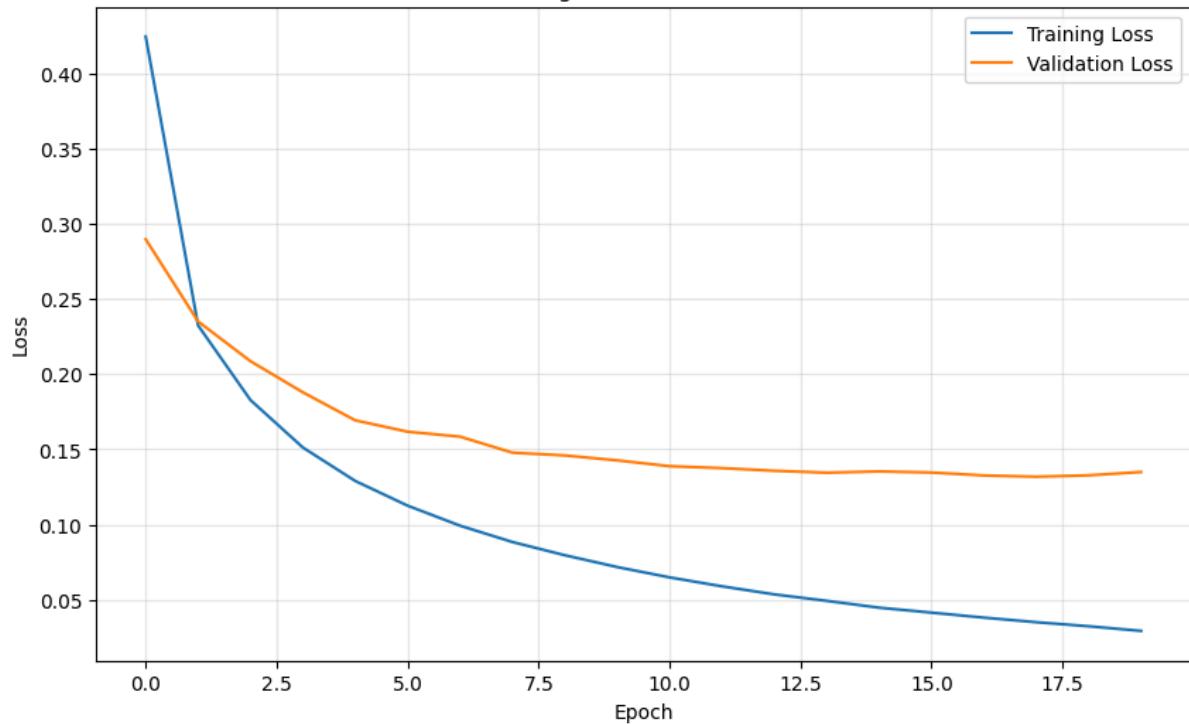
Training and Validation Loss



Test accuracy: 0.9662

g. ELU

Training and Validation Loss



Test accuracy: 0.9666

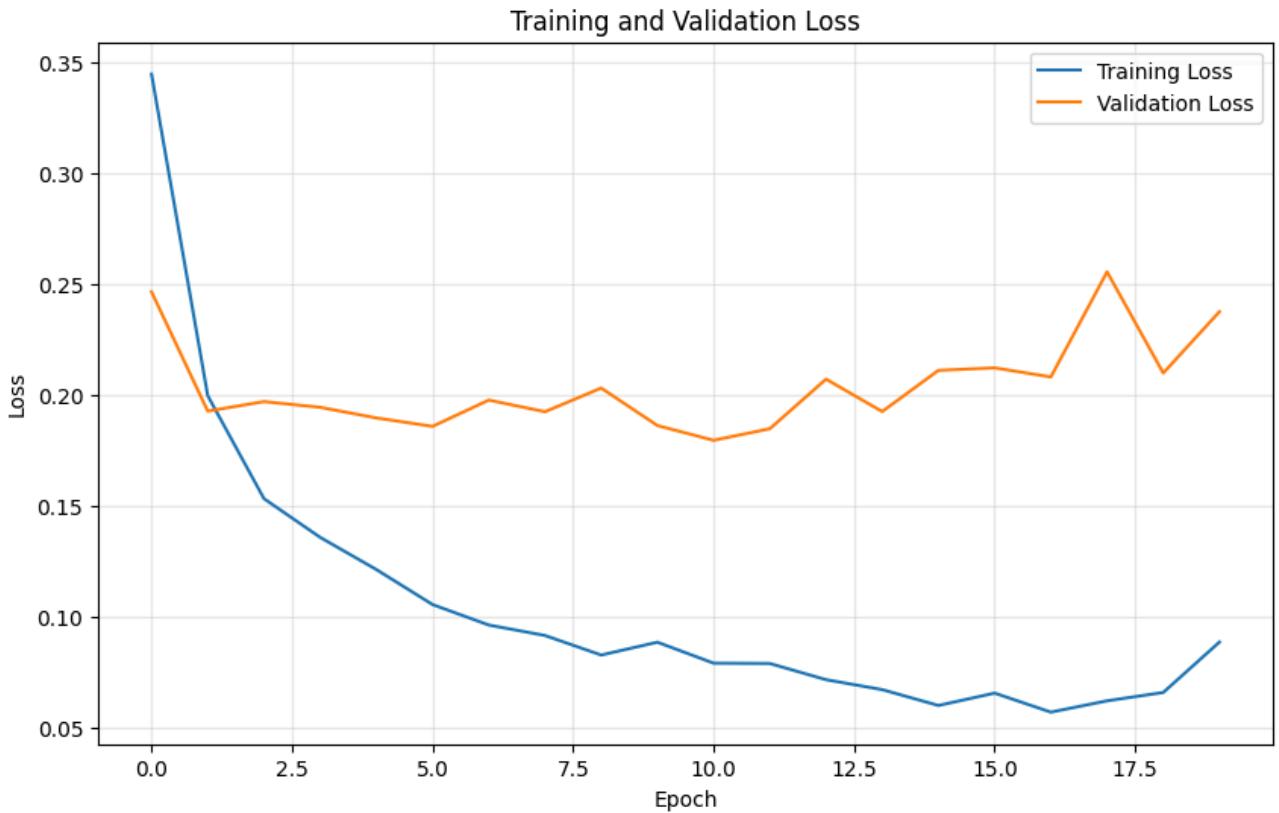
Dari hasil perbandingan tersebut, dapat dilihat bahwa beberapa fungsi dapat diterapkan dan menghasilkan prediksi yang baik (diatas 95%) seperti ELU, LeakyReLU, Tanh dan ReLU. Kemudian diikuti oleh Linear, Sigmoid, dan Softmax. Karena itu, sangat disarankan untuk menggunakan ELU atau LeakyReLU sebagai fungsi aktivasinya.

2.2.3. Pengaruh learning rate

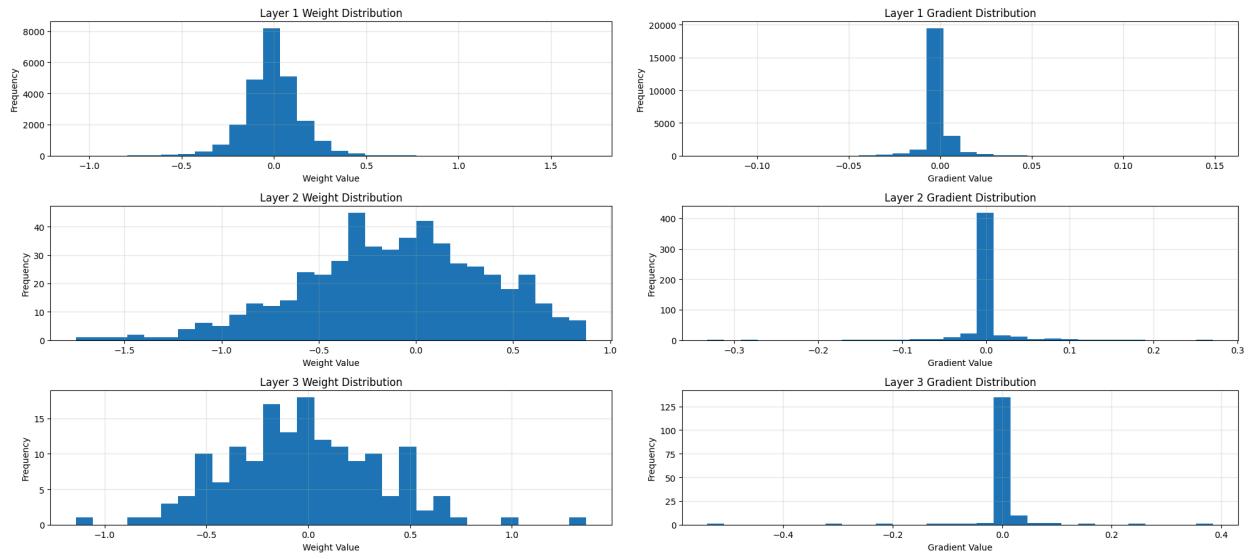
Pengujian dilakukan dengan menggunakan Fungsi Aktivasi ReLU di setiap layernya kecuali layer terakhir yang menggunakan Fungsi Aktivasi Softmax. Banyaknya Epoch adalah 20, Batch Size = 32, dan Fungsi Loss menggunakan Categorical Cross Entropy. Untuk metode inisialisasi bobot yang digunakan adalah He dan Xavier secara bergantian tiap layernya dengan seed = 42. Variasi learning rate yang digunakan adalah 0.1, 0.01, dan 0.001.

a. Learning rate = 0.1

- Hasil akhir: nilai akurasi 0.9522
- Grafik Loss:

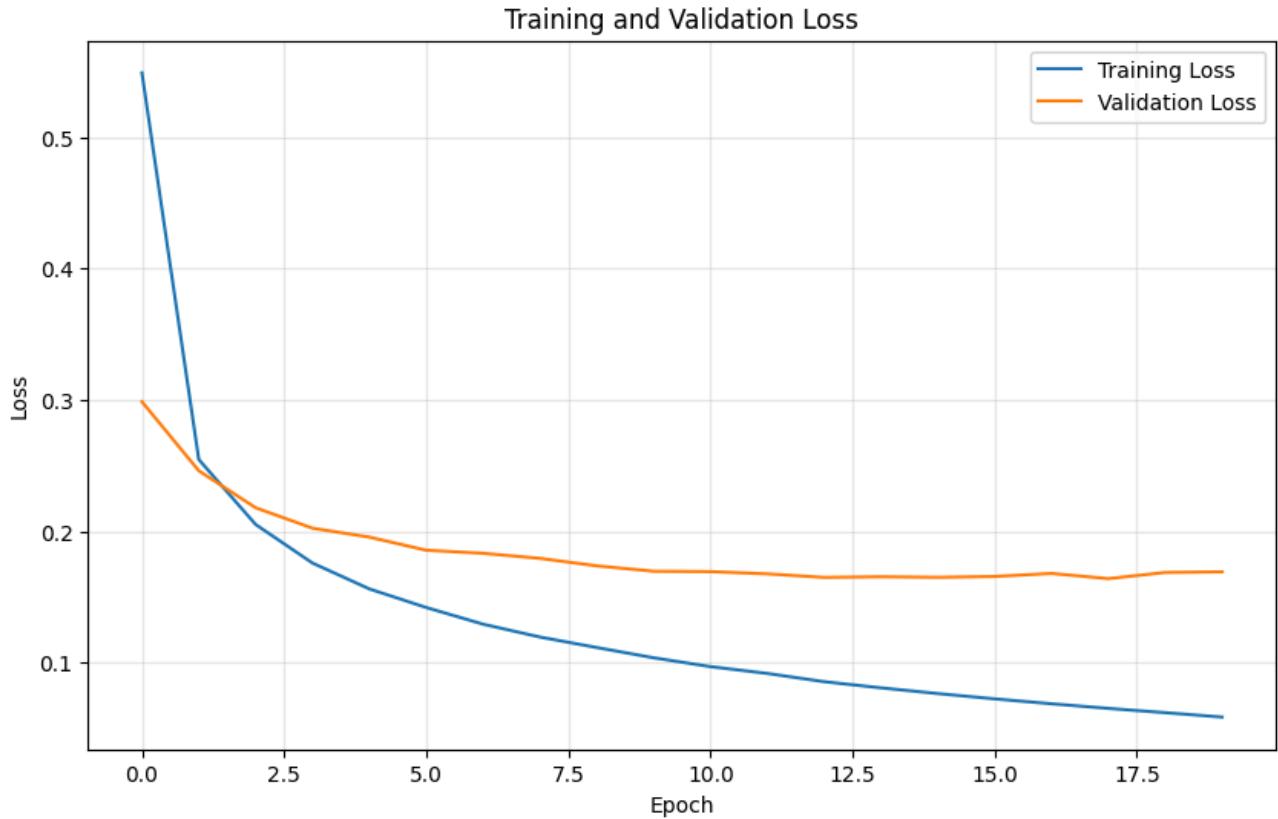


- Distribusi bobot dan gradien bobot:

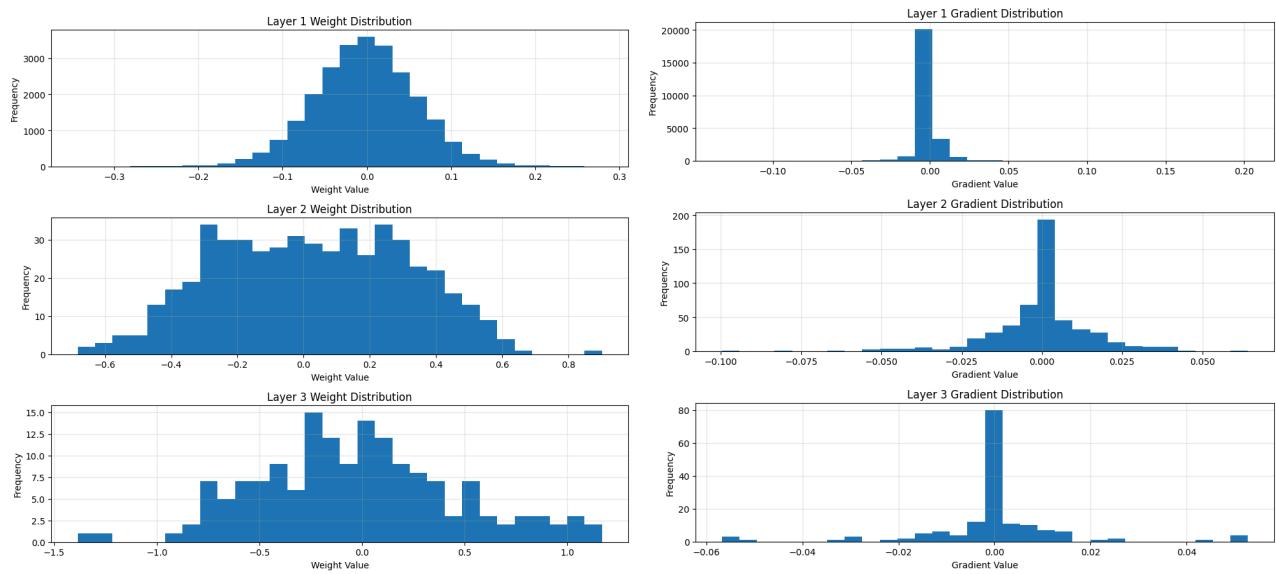


b. Learning rate = 0.01

- Hasil akhir: nilai akurasi 0.9556
- Grafik Loss:

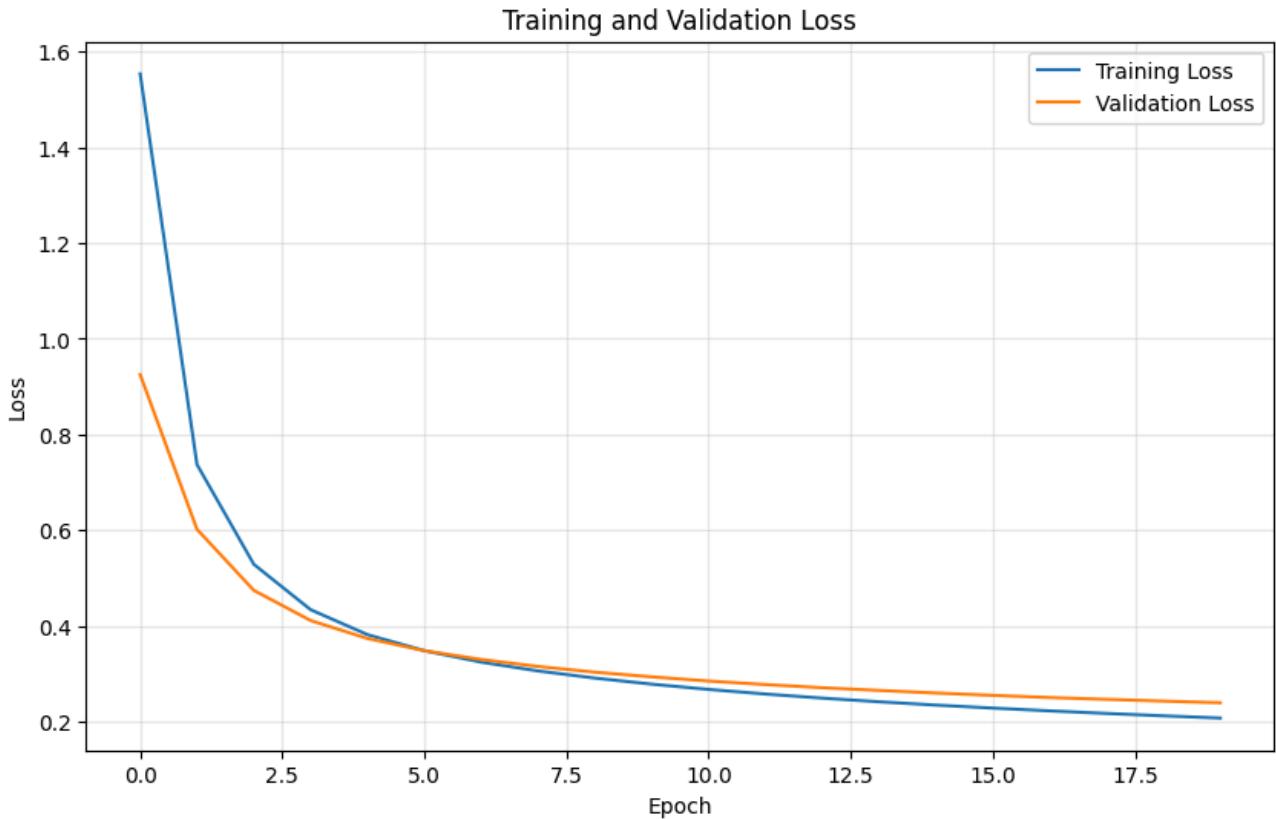


- Distribusi bobot dan gradien bobot:

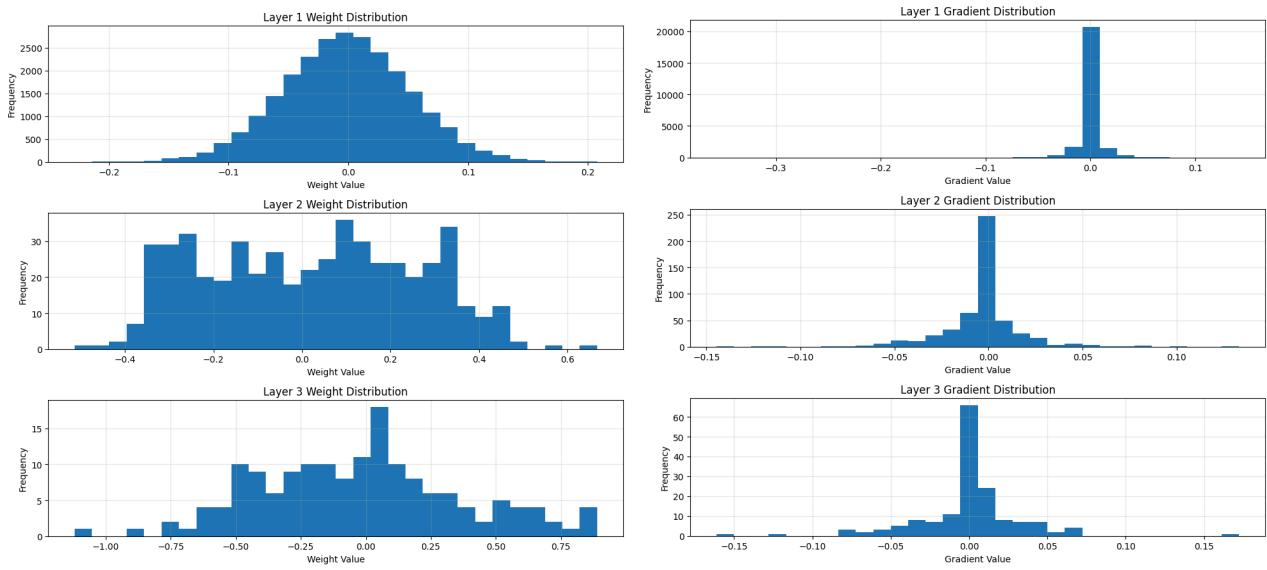


c. Learning rate = 0.001

- Hasil akhir: nilai akurasi 0.9317
- Grafik Loss:



- Distribusi bobot dan gradien bobot:



Berdasarkan hasil pengujian variasi learning rate di atas, terlihat bahwa model memiliki performa yang sangat baik pada semua learning rate dengan akurasi tes paling tinggi untuk learning rate 0.01 (0.9556), disusul oleh learning rate 0.1 (0.9522), dan terakhir learning rate 0.001 (0.9317). Pada learning rate 0.1, loss turun paling cepat, dengan penurunan tajam di awal epoch. Training loss hampir mendekati nol, sementara validation loss sedikit lebih tinggi. Semakin kecil learning rate, penurunan loss lebih lambat namun lebih landai mendekati nol, training dan validation loss turun lebih bertahap dan semakin dekat jarak antarkeduanya. Untuk distribusi bobot, semua learning rate berbentuk normal yang simetris pada Layer 1 dengan pusat di 0, namun pada Layer 2 dan 3 distribusi semakin tidak seragam dengan beberapa variasi. Untuk distribusi gradien, semua learning rate punya gradien terkonsentrasi sangat dekat dengan nol, namun learning rate yang lebih besar menunjukkan gradien dengan rentang yang lebih lebar.

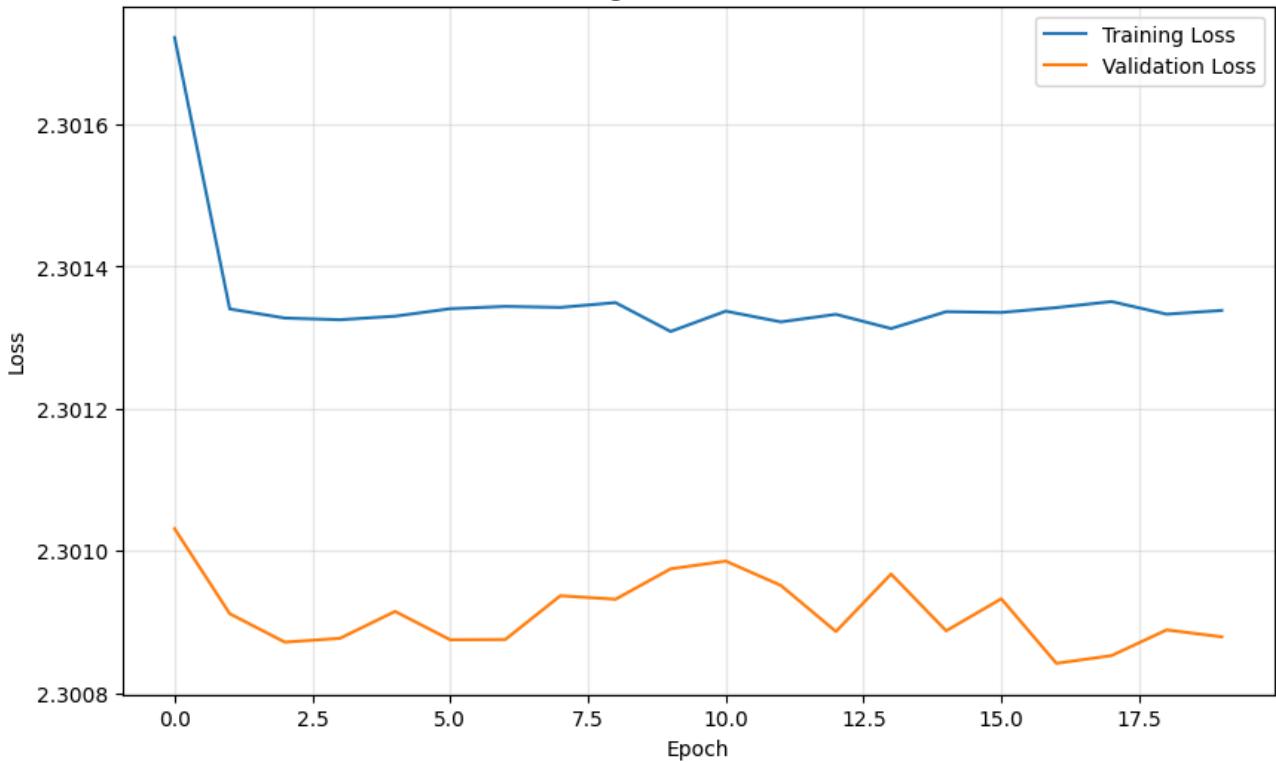
2.2.4. Pengaruh inisialisasi bobot

Pengujian dilakukan dengan menggunakan Fungsi Aktivasi ReLU di setiap layernya kecuali layer terakhir yang menggunakan Fungsi Aktivasi Softmax. Banyaknya Epoch adalah 20, Batch Size = 32, Fungsi Loss menggunakan Categorical Cross Entropy, dan learning rate 0.01. Untuk inisialisasi bobot dibuat sama pada setiap layernya, berikut variasinya dengan seed = 42.

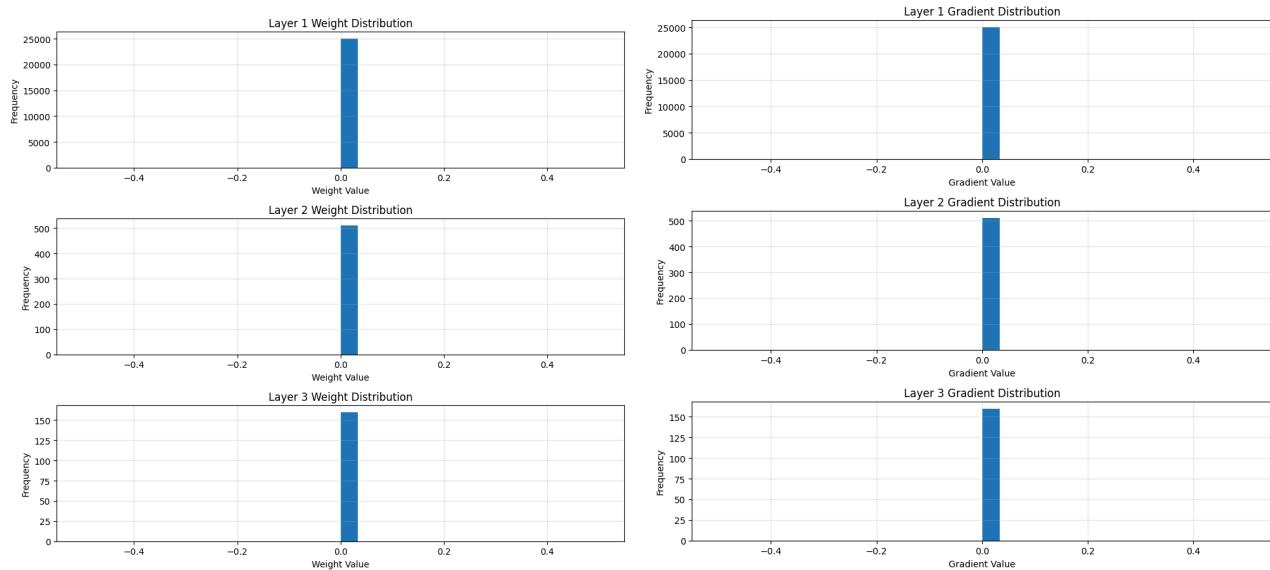
a. Zero Initialization

- Hasil akhir: nilai akurasi 0.1143
- Grafik Loss:

Training and Validation Loss

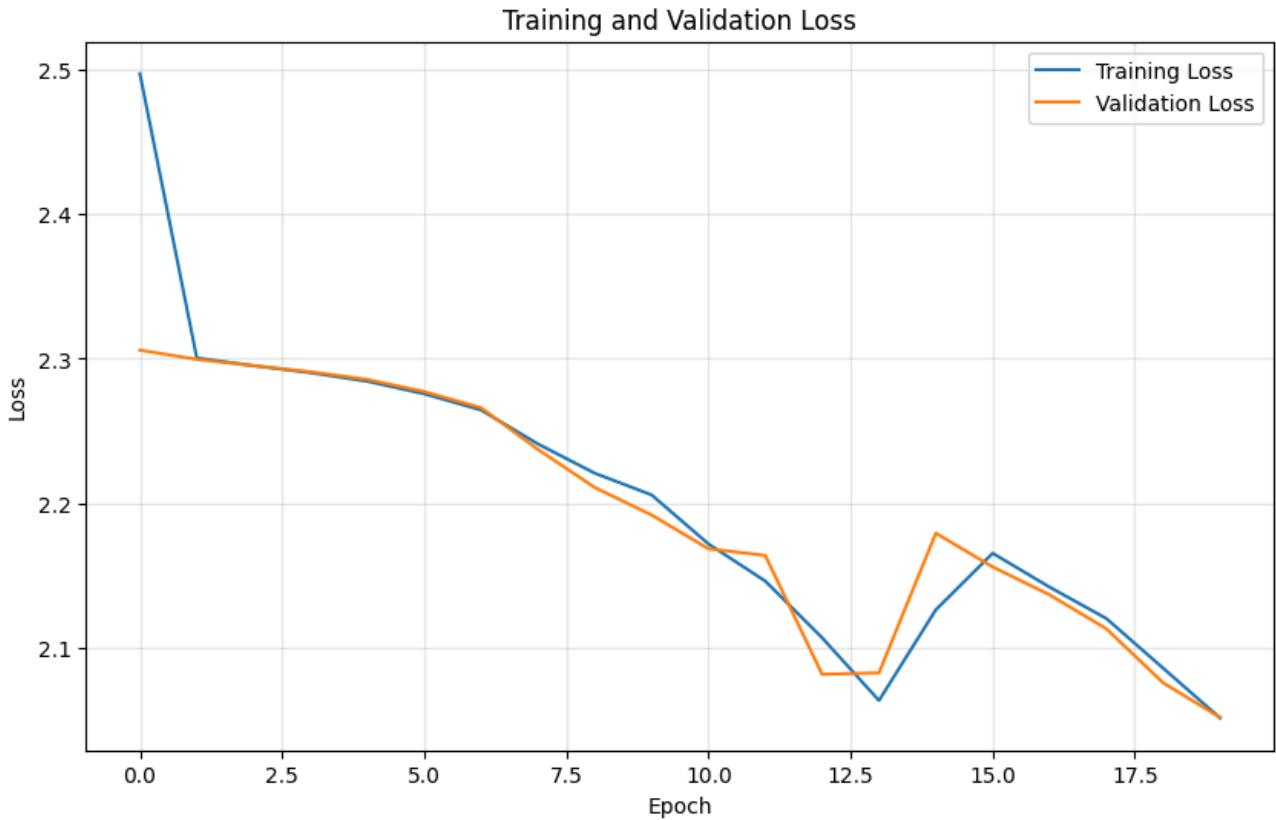


- Distribusi bobot dan gradien bobot:

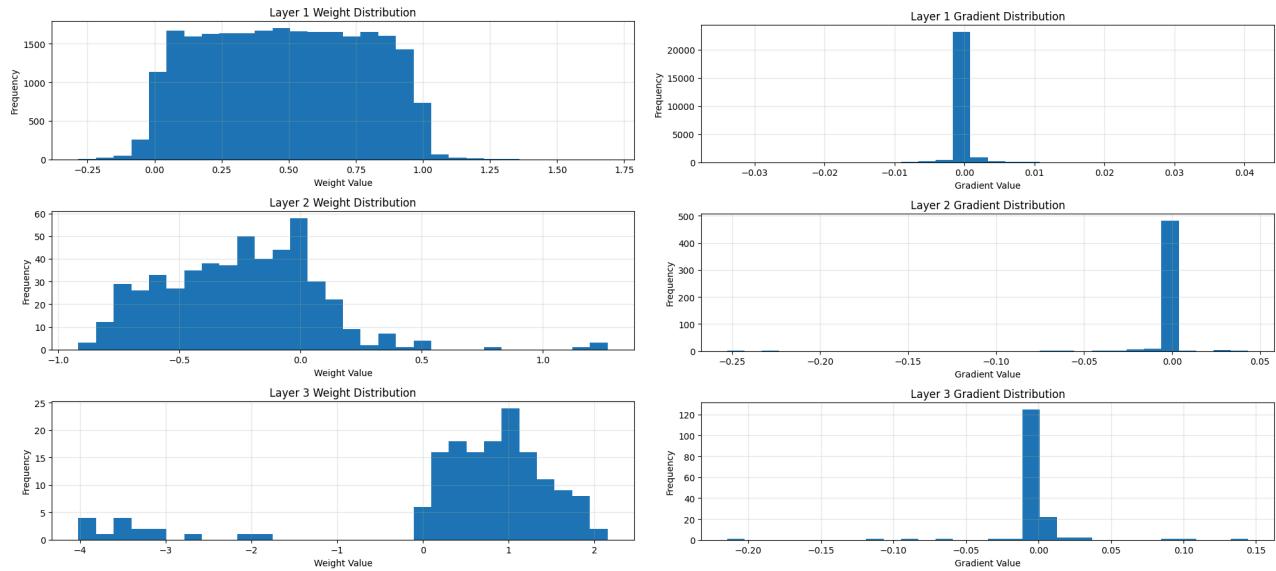


- b. Uniform Initialization

- Hasil akhir: nilai akurasi 0.2394
- Grafik Loss:



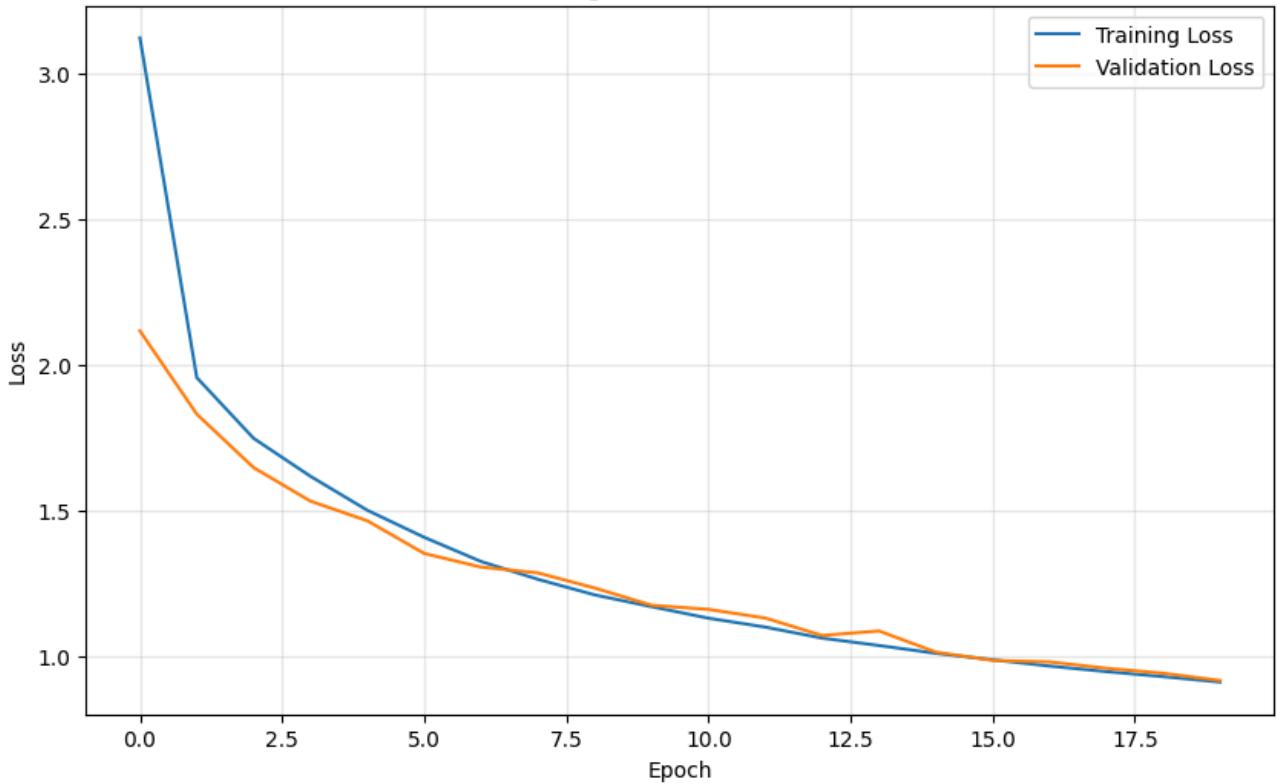
- Distribusi bobot dan gradien bobot:



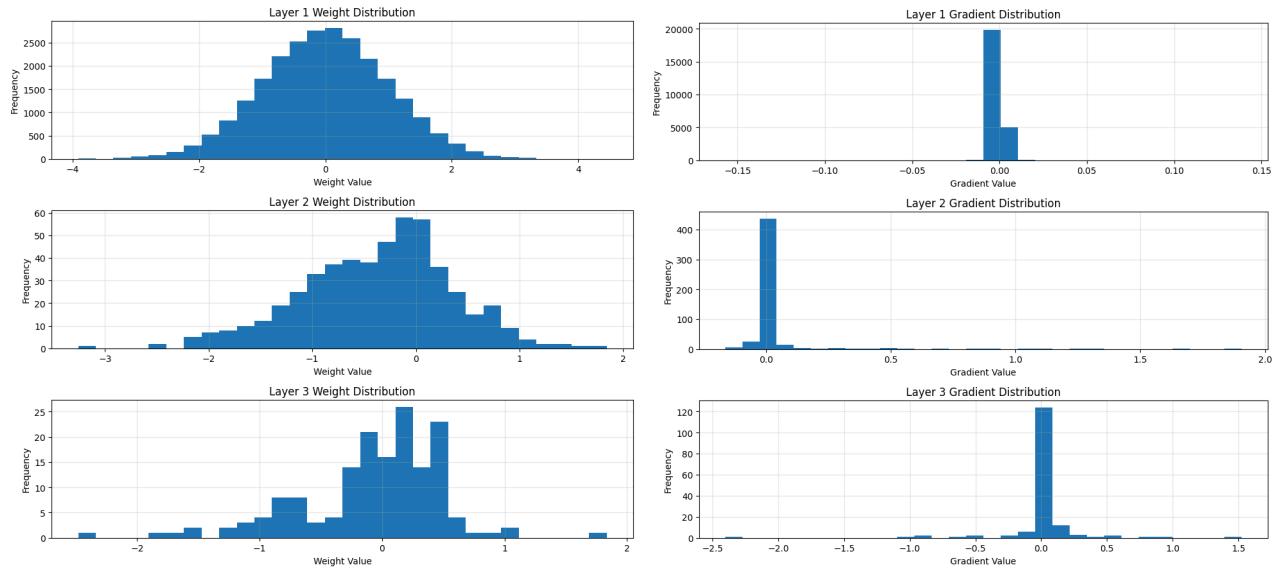
c. Normal Initialization

- Hasil akhir: nilai akurasi 0.6710
- Grafik Loss:

Training and Validation Loss



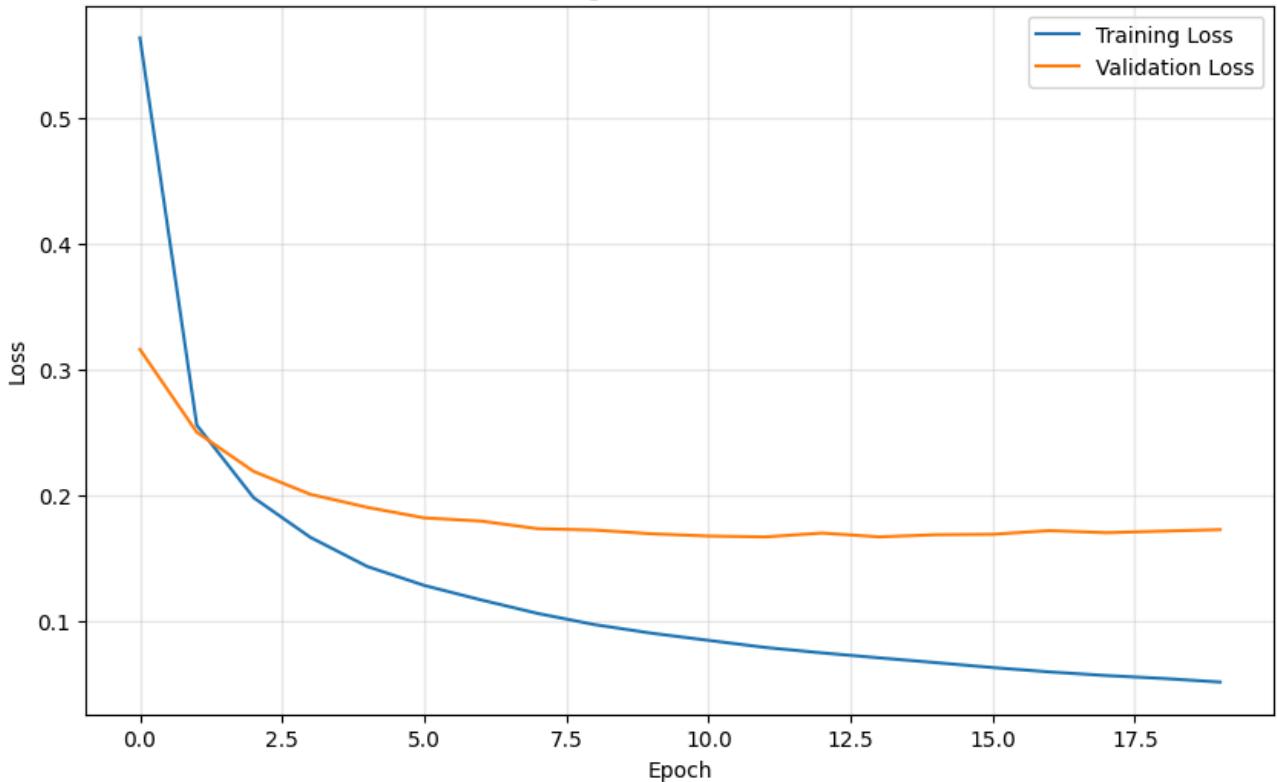
- Distribusi bobot dan gradien bobot:



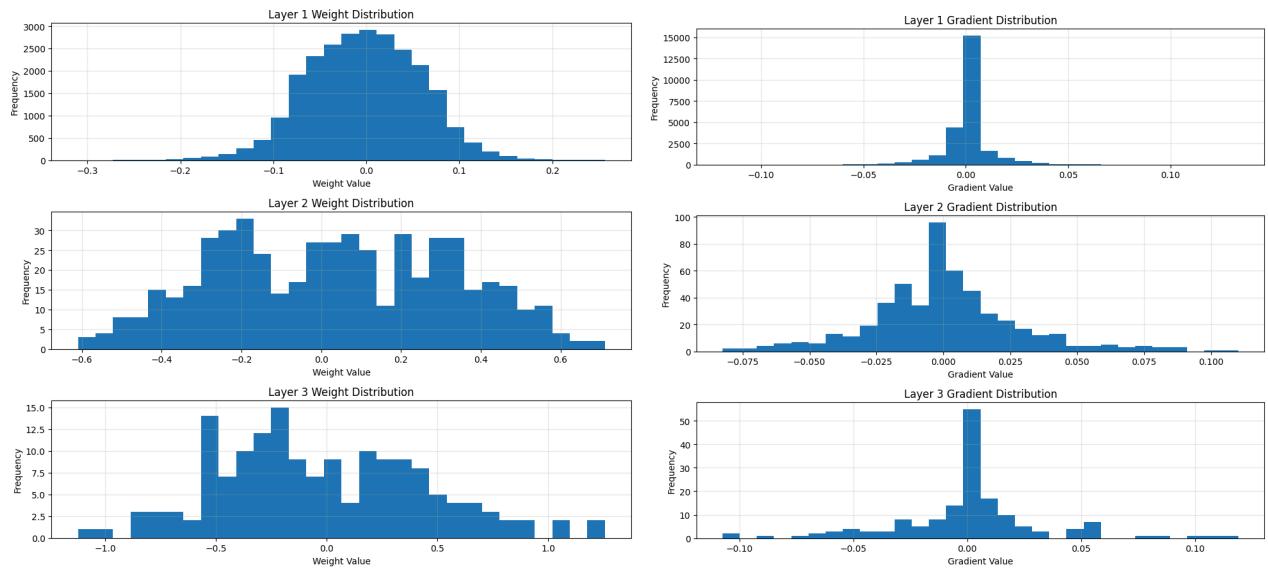
d. Xavier Initialization

- Hasil akhir: nilai akurasi 0.9591
- Grafik Loss:

Training and Validation Loss



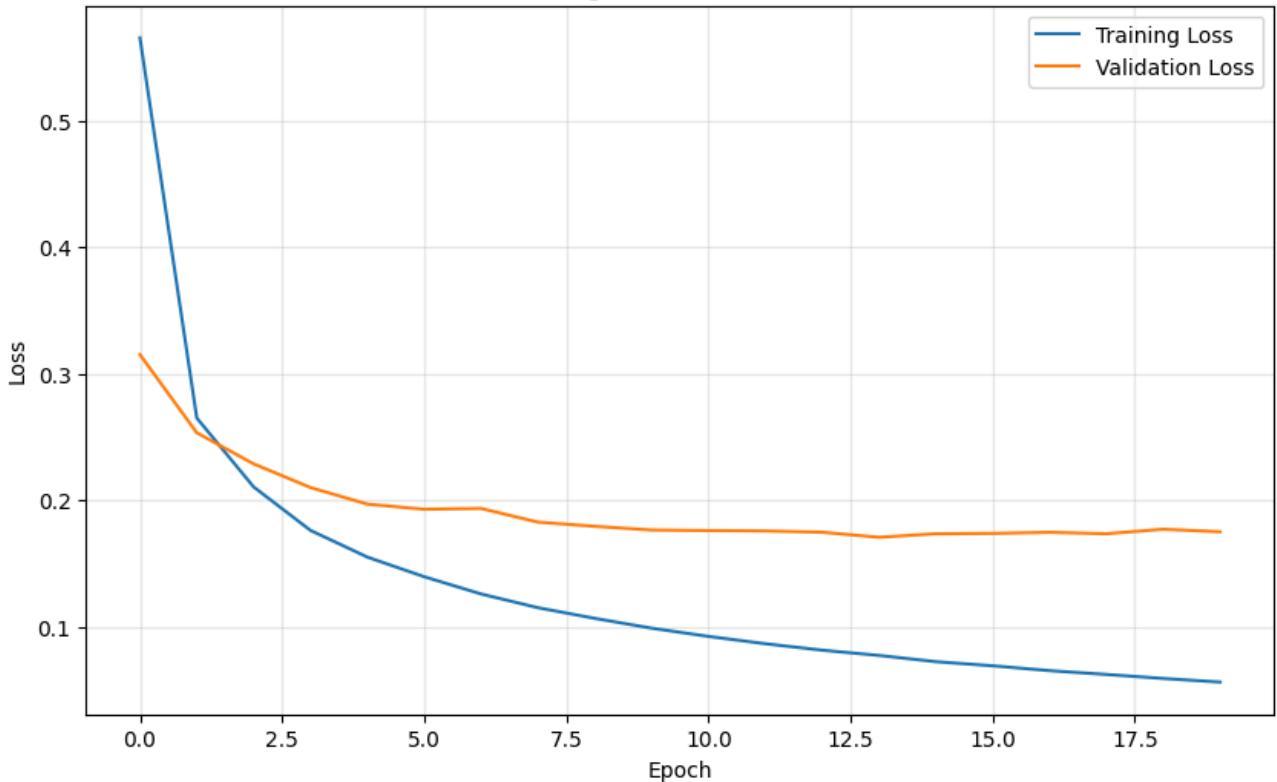
- Distribusi bobot dan gradien bobot:



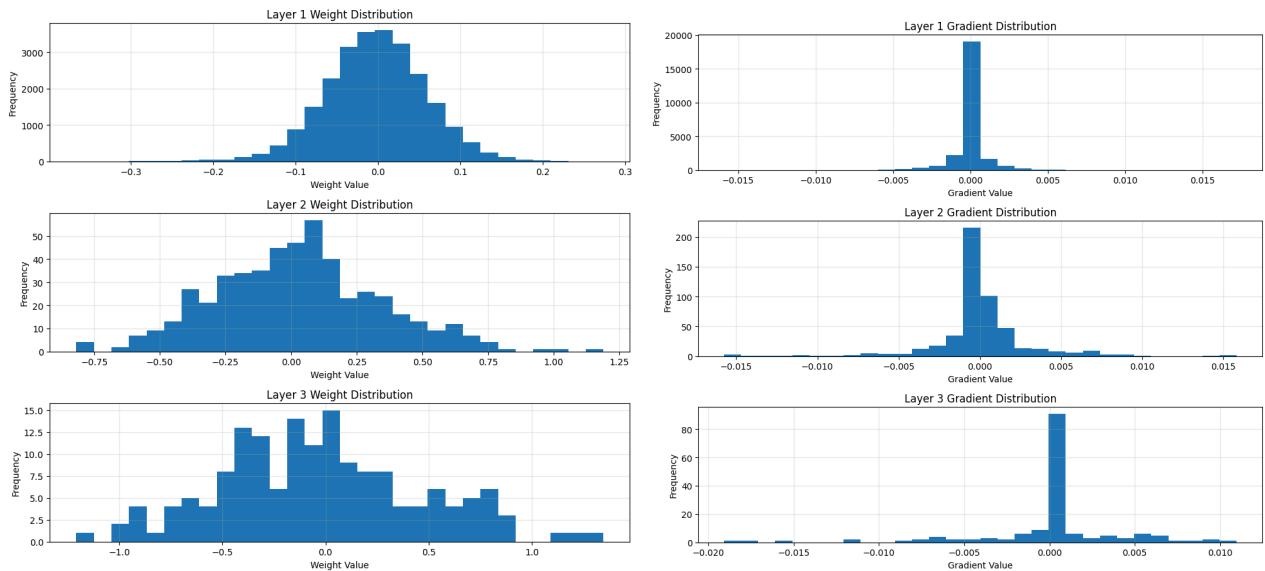
e. He Initialization

- Hasil akhir: nilai akurasi 0.9574
- Grafik Loss:

Training and Validation Loss



- Distribusi bobot dan gradien bobot:



Berdasarkan hasil pengujian variasi inisialisasi bobot di atas, terlihat bahwa berbeda metode inisialisasi bobot, berbeda pula hasil yang diberikan. Zero Initialization memiliki performa yang paling buruk dengan akurasi hanya 0.1143. Uniform Initialization meningkat dengan akurasi 0.2394, Normal mencapai 0.6710,

dan Xavier juga He meningkat signifikan sehingga memperoleh hasil terbaik yakni 0.9591. Untuk grafik loss, pada Zero terlihat tidak mengalami penurunan dan tetap tinggi sepanjang epoch. Sementara Uniform dan Normal, loss mulai turun dengan training lossnya mendekati validation loss. Xavier dan He menunjukkan penurunan loss yang sangat cepat di awal dan mencapai jarak paling kecil antara training dan validation loss. Untuk distribusi bobot, pada Zero, distribusinya sangat tidak merata dengan mayoritas di titik nol. Pada Uniform, distribusi lebih menyebar namun masih kurang merata. Sedangkan Normal, Xavier dan He, distribusinya terbilang sangat baik dengan sebaran normal yang ideal. Untuk distribusi gradien, pada Zero, gradiennya hanya terkumpul di sekitar nol (tidak ada pembelajaran). Pada Uniform, gradien mulai tersebar namun terbatas. Pada Normal sudah lebih tersebar, begitu juga pada Xavier dan He. Dari hal-hal tersebut, diketahui bahwa semakin terarah inisialisasi awal bobot, semakin baik juga hasil akhirnya.

2.2.5. Pengaruh regularisasi

Nilai bobot yang terlalu beragam berpotensi menghilangkan makna dari fitur-fitur yang sebenarnya penting atau juga dapat menyebabkan overfitting. Untuk mencegah dua hal tersebut, kami mengimplementasikan regularisasi dengan L1 dan L2. Rumus dasar L1 dan L2 adalah sebagai berikut:

L1 Regularization:

$$\underset{\beta, \lambda}{\operatorname{argmin}} \text{ loss} + \lambda \sum_{j=1}^p |\beta_j|$$

L2 Regularization:

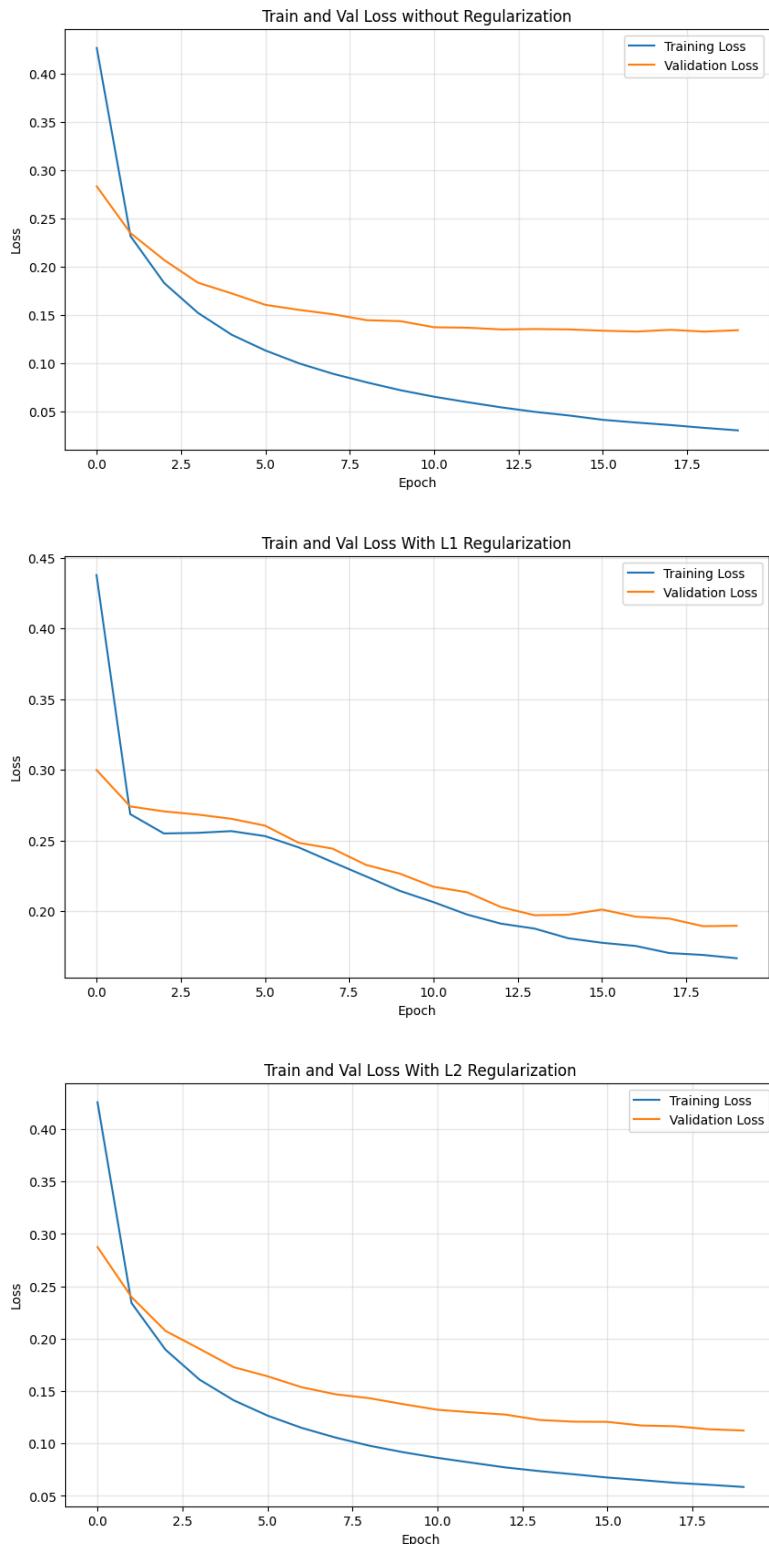
$$\underset{\beta, \lambda}{\operatorname{argmin}} \text{ loss} + \lambda \sum_{j=1}^p \beta_j^2$$

Ketika diimplementasikan di fungsi update bobot (gradient descent) maka diambil turunan pertamanya terlebih dahulu kemudian ditambahkan seperti ini

```
# Rumus Regularisasi L1
self.weights_grad += l1_lambda * np.sign(self.weights)
```

```
# Rumus Regularisasi L2
self.weights_grad += l2_lambda * 2 * self.weights
```

Berikut ini perbandingan antara tanpa regularisasi, dengan regularisasi L1, dan dengan regularisasi L2:



Dari tiga grafik loss di atas, dapat dilihat bahwa model dengan regularisasi L2 memiliki loss yang cenderung lebih rendah dan landai. Hal ini tentu disebabkan

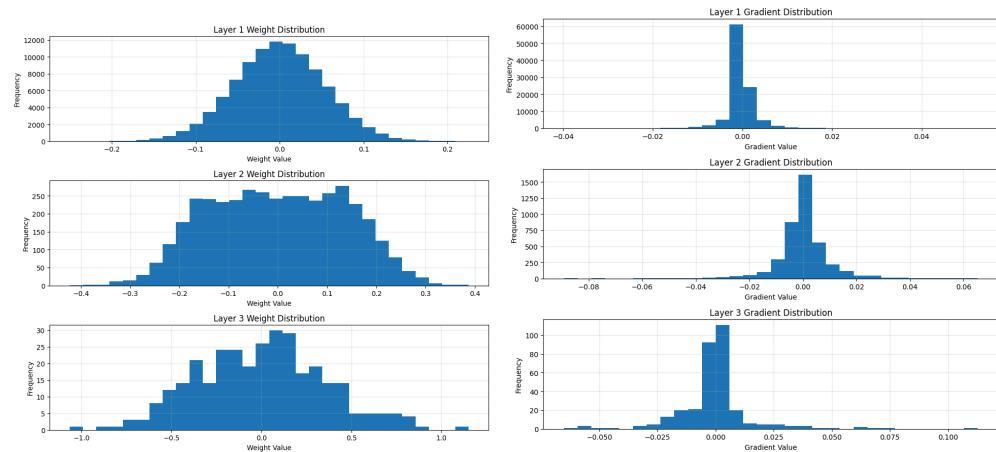
“penalty” yang didapatkan dari L2. Akan tetapi, pada L1, nilai loss justru lebih tinggi.

Di samping itu, didapatkan juga data akurasi sebagai berikut

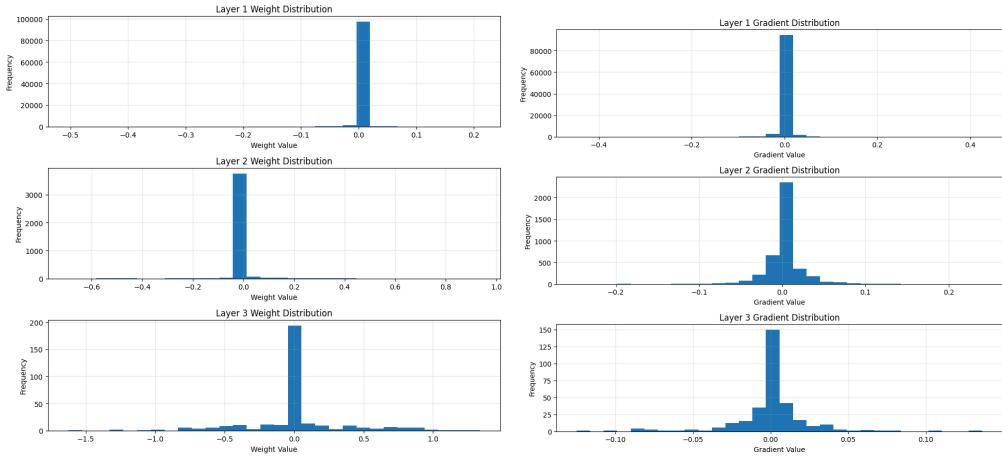
- Akurasi tanpa regularisasi: 0.9679
- Akurasi dengan regularisasi L1: 0.9481
- Akurasi dengan regularisasi L2: 0.9685

Dari sini dapat dilihat bahwa akurasi dengan L2 cenderung lebih naik. Akurasi pada L1 menjadi lebih rendah dapat dimungkinkan sebab penalti yang terlalu ekstrem secara tidak sengaja malah menghilangkan makna suatu neuron yang penting

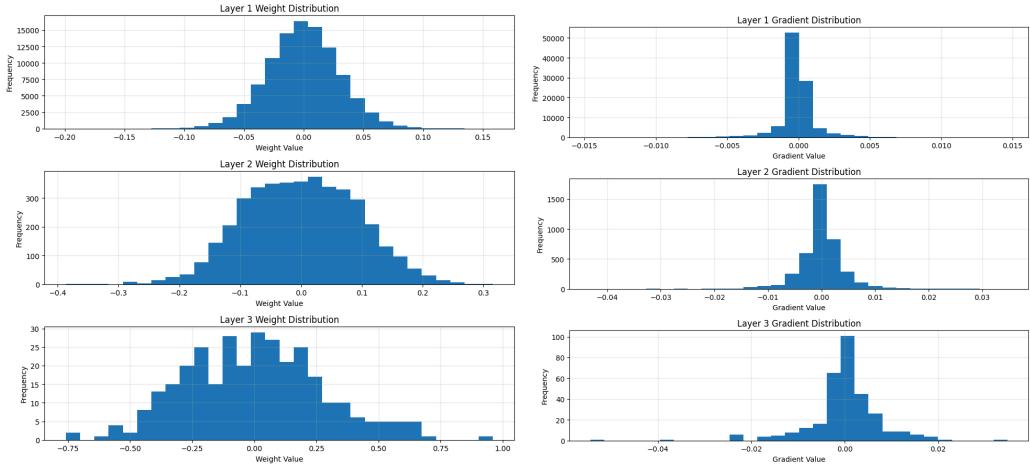
Distribusi bobot dan gradient tanpa regularisasi:



Distribusi bobot dan gradient dengan L1:



Distribusi bobot dan gradient dengan L2



Dari tiga histogram di atas, dapat dilihat bahwa distribusi bobot dan gradient dengan regularisasi L2 lebih membentuk distribusi normal dibandingkan dengan tanpa regularisasi. Perubahan dari tanpa regularisasi ke regularisasi L2 terlihat tidak begitu signifikan sebab pinalti yang diberikan L2 memang tidak begitu ekstrem. Sedangkan, distribusi bobot pada L1 cenderung ekstrem menonjolkan beberapa neuron saja. Hal ini dapat dipahami sebab L1 memang lebih cocok digunakan untuk feature selection.

2.2.6. Pengaruh normalisasi RMS Norm

Nilai aktivasi yang terlalu besar atau terlalu kecil dapat menyebabkan proses pelatihan menjadi tidak stabil atau sulit memahami pola dalam data. Untuk mengatasi hal tersebut, kami mencoba berekspresimen dengan

mengimplementasikan normalisasi RMS (Root Mean Square Normalization). Metode ini kami terapkan pada layer pertama dan kedua persis sebelum diterapkan fungsi aktivasi.

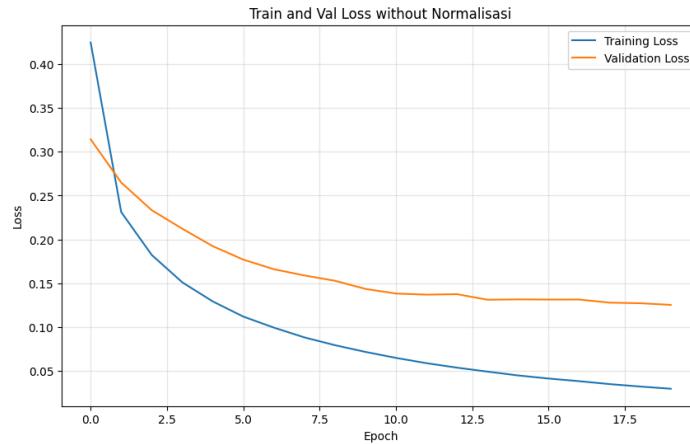
RMS adalah proses normalisasi tanpa menghitung nilai rata-rata (hanya menggunakan Root Mean Square (RMS) saja). RMS dari suatu input x didefinisikan sebagai berikut:

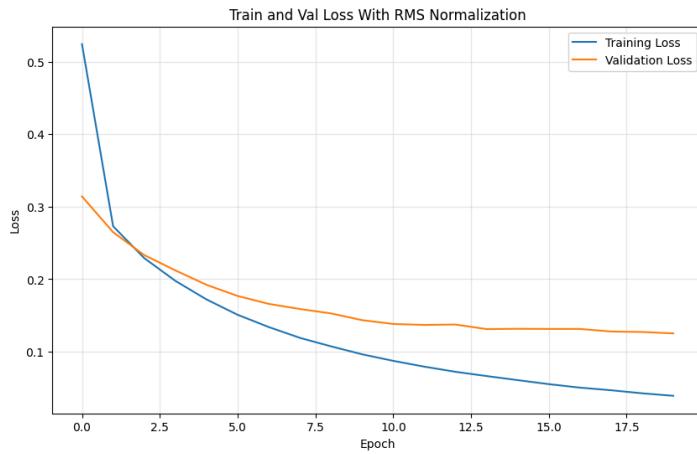
$$\hat{x} = \frac{x}{RMS(x) + \epsilon}$$

Dimana $RMS(x)$ merupakan nilai akar dari rata-rata kuadrat setiap fitur (root mean square).

$$RMS(x) = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2}$$

Berikut ini perbandingan antara tanpa normalisasi RMS Norm, dengan menggunakan RMS Norm:



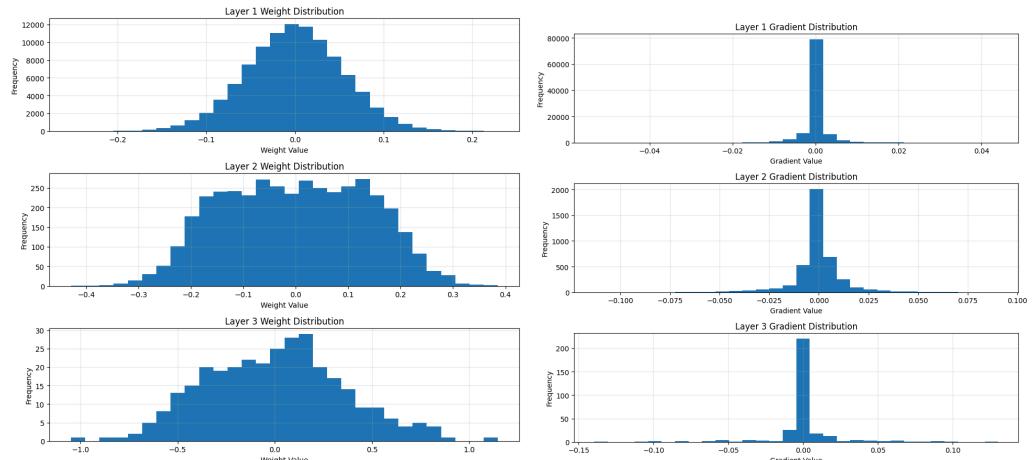


Dari tiga grafik loss di atas, dapat dilihat bahwa model dengan RMS Normalization justru memiliki loss yang lebih tinggi. Di samping itu, dapat dilihat data akurasi sebagai berikut bahwa akurasi dengan tanpa normalisasi justru lebih tinggi.

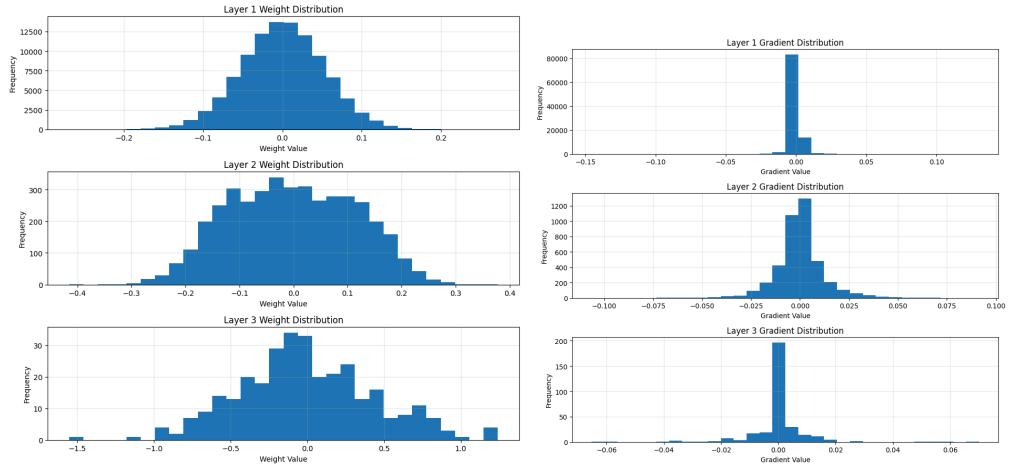
- Akurasi tanpa normalisasi: 0.9688
- Akurasi dengan normalisasi RMS Norm: 0.9636

Hipotesis yang kami ajukan terkait analisis data di atas yaitu model yang kami buat cenderung sederhana dengan jumlah layer yang sedikit sehingga pengaruh normalisasi RMS tidak begitu terlihat.

Selanjutnya, berikut ini distribusi bobot dan gradient tanpa normalisasi:



Distribusi bobot dan gradient dengan RMS Norm:



Dari grafik-grafik di atas, dapat dilihat bahwa distribusi bobot pada layer 1 tanpa normalisasi justru memiliki range nilai yang stabil dibandingkan dengan RMS Norm. Akan tetapi, pengaruh normalisasi RMS baru terasa pada layer 2 dimana distribusi bobot menjadi lebih merata dibandingkan tanpa normalisasi.

2.2.7. Perbandingan dengan library sklearn

Pada perbandingan ini, baik model buatan kami maupun dari library sklearn sama-sama melakukan pelatihan dengan **dua buah hidden layer**, yakni berukuran 32 neuron dan 16 neuron. Hyperparameter yang digunakan diantaranya pada fungsi aktivasi, menggunakan **ReLU** untuk hidden layer dan **Softmax** untuk output layer. Ukuran **batch sebesar 32**, learning rate yang digunakan sebesar **0.01**, dan dengan epoch dibatasi hanya **21 epoch**. Untuk inisialisasi bobot menggunakan **Xavier** dengan seed = 42 karena diketahui sklearn juga menggunakannya, sedangkan fungsi loss menggunakan **Categorical Cross Entropy**.

Berikut adalah perbandingan hasil akhir prediksi berupa akurasi dari kedua model.

- Model sendiri:

FFNN accuracy: 0.9593

- Sklearn:

sklearn accuracy: 0.9391

Dari hasil tersebut, didapat akurasi antara model FFNN kami tidak jauh berbeda dengan dari library sklearn. Keduanya berada di sekitar angka 0.9 dengan hasil yang didapat oleh model kami sekitar 2% lebih tinggi dibanding library sklearn.

Bab III. Kesimpulan dan Saran

3.1. Kesimpulan

Berdasarkan pengujian yang telah dilakukan untuk mengoptimalkan model FFNN tersebut, dapat ditarik beberapa kesimpulan mengenai pengaruh berbagai hyperparameter terhadap performa model.

- Penambahan jumlah hidden layer (depth) secara signifikan meningkatkan akurasi model. Hal ini mengindikasikan bahwa kedalaman yang cukup memungkinkan model untuk mempelajari representasi data yang kompleks.
- Peningkatan jumlah neuron pada setiap hidden layer (width) juga membantu meningkatkan akurasi model. Hal ini menunjukkan bahwa width yang lebih besar memberikan kapasitas representasional yang lebih besar kepada model dan meningkatkan performa dari model FFNN yang dilatih.
- Pemilihan fungsi aktivasi memiliki dampak signifikan terhadap performa model. Fungsi aktivasi non-linear seperti ELU dan LeakyReLU menunjukkan performa terbaik dengan akurasi di atas 96%, diikuti oleh Tanh dan ReLU.
- Metode inisialisasi bobot juga memengaruhi kemampuan model untuk belajar. Inisialisasi dengan nilai nol dan nilai acak yang seragam menghasilkan performa yang lebih buruk daripada inisialisasi yang lebih terarah seperti dengan He dan Xavier.

Jika dibandingkan dengan model yang dilatih dengan library sklearn, maka model kami memiliki performa yang sedikit lebih baik daripada model yang dilatih dengan sklearn. Hal ini berarti model yang kami kembangkan memiliki kemampuan untuk memprediksi yang lebih baik.

3.2. Saran

Berdasarkan hasil pengujian ini, berikut adalah beberapa saran yang dapat dipertimbangkan untuk pengembangan model selanjutnya:

1. Melakukan pengujian lebih lanjut di sekitar nilai 10 (misalnya 12) untuk menemukan depth yang benar-benar optimal. Sekaligus menguji Depth yang lebih tinggi lagi untuk melihat apakah model akan memiliki performa lebih baik atau tidak.
2. Karena terlihat tren bahwa width yang lebih besar memiliki akurasi yang lebih baik, maka patut dicoba nilai width yang lebih besar dari 64 (misalnya 128 atau 256).
3. Meskipun learning rate 0.01 memberikan hasil terbaik, mencoba nilai-nilai di sekitarnya (misalnya 0.005 atau 0.02) mungkin dapat menghasilkan peningkatan akurasi.
4. Selain akurasi, metrik lain untuk melakukan evaluasi model juga patut dipertimbangkan untuk mengevaluasi model, seperti precision, recall, F1-score, dan confusion matrix.

Dengan mempertimbangkan saran-saran di atas, diharapkan pengembangan model FFNN selanjutnya dapat mencapai performa yang lebih optimal.

Pembagian Tugas

Kegiatan	Nama (NIM)
Implementasi fungsi aktivasi	Muhammad Naufal Aulia (13522074) Dhidit Abdi Aziz (13522040)
Implementasi fungsi loss	Muhammad Naufal Aulia (13522074)
Implementasi layer neural network	Tazkia Nizami (13522032)
Implementasi metode inisialisasi	Muhammad Naufal Aulia (13522074)
Implementasi dan Pengujian Regularisasi	Dhidit Abdi Aziz (13522040)
Implementasi dan Pengujian Normalisasi RMS	Dhidit Abdi Aziz (13522040)
Visualisasi informasi model	Tazkia Nizami (13522032) Muhammad Naufal Aulia (13522074)
Integrasi model FFNN	Muhammad Naufal Aulia (13522074)
Implementasi notebook pengujian	Tazkia Nizami (13522032) Muhammad Naufal Aulia (13522074)
Pengujian model	Tazkia Nizami (13522032) Dhidit Abdi Aziz (13522040) Muhammad Naufal Aulia (13522074)
Pengerjaan laporan	Tazkia Nizami (13522032) Dhidit Abdi Aziz (13522040) Muhammad Naufal Aulia (13522074)

Referensi

Medium. L1 and L2 Regularization (Part 1): A Complete Guide. Diakses dari <https://medium.com/@alejandro.itoaramendia/l1-and-l2-regularization-part-1-a-complete-guide-51cf45bb4ade>

Medium. Mastering LLama – Layer Normalization RMSNorm. Diakses dari [https://medium.com/@hugmanskj/mastering-llama-rmsnorm-%ec%9e%90%ec%97%94\]-701f8281ddbb](https://medium.com/@hugmanskj/mastering-llama-rmsnorm-%ec%9e%90%ec%97%94]-701f8281ddbb)

Medium. Derivative of the Softmax Function and the Categorical Cross-Entropy Loss. Diakses dari <https://medium.com/data-science/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1>

Scikit-Learn. MLPClassifier. Diakses dari https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Scikit-Learn. Sparse Logistic Regression on MNIST. Diakses dari https://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_mnist.html

Stack Overflow. What Weight Initialization Does the MLPClassifier in Sklearn Use? Diakses dari <https://stackoverflow.com/questions/69226750/what-weight-initialization-does-the-mlp-classifier-in-sklearn-use>