

# **Tugas Besar 1**

## **Feedforward Neural Network**

**IF3270 Pembelajaran Mesin**



Elbert Chailes 13522045

Farel Winalda 13522047

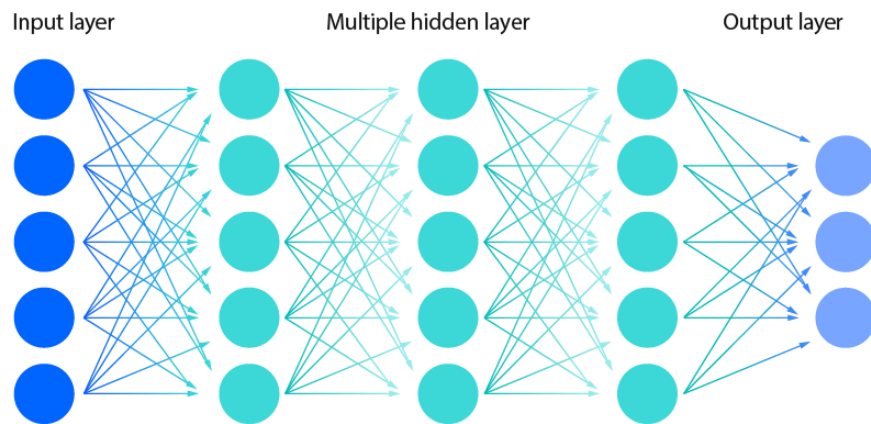
Derwin Rustanly 13522115

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

# Daftar Isi

Daftar Isi.....	2
BAB 1: Deskripsi Persoalan.....	3
BAB 2: Pembahasan.....	5
2.1. Penjelasan Implementasi.....	5
2.1.1. Deskripsi Kelas.....	5
2.1.2. Penjelasan Forward Propagation.....	9
2.1.3. Penjelasan Backward Propagation.....	10
2.2. Hasil Pengujian.....	11
2.2.1. Pengaruh Depth dan Width.....	11
2.2.2. Pengaruh fungsi aktivasi.....	14
2.2.3. Pengaruh learning rate.....	17
2.2.4. Pengaruh inisialisasi bobot.....	19
2.2.5. Pengaruh regularisasi.....	22
2.2.6. Pengaruh normalisasi RMSNorm.....	26
2.2.7. Perbandingan dengan library sklearn.....	29
BAB 3: Kesimpulan dan Saran.....	30
3.1. Kesimpulan.....	30
3.2. Saran.....	31
BAB 4: Pembagian Tugas.....	32
Referensi.....	33

# BAB 1: Deskripsi Persoalan



*Gambar 1.1.* Feedforward Neural Network (FFNN)

Tugas besar 1 pada mata kuliah IF3270 Pembelajaran Mesin bertujuan untuk mendapatkan wawasan tentang cara mengimplementasikan Feedforward Neural Network (FFNN). Modul FFNN yang diimplementasikan memenuhi ketentuan sebagai berikut.

- FFNN yang diimplementasikan dapat **menerima jumlah neuron dari tiap layer** (termasuk input layer dan output layer).
- FFNN yang diimplementasikan dapat menerima fungsi **aktivasi dari tiap layer**.
- FFNN yang diimplementasikan dapat **menerima fungsi loss** dari model tersebut.
- Terdapat mekanisme untuk **inisialisasi bobot** tiap neuron (termasuk bias).
- Instance model yang diinisialisasikan harus bisa **menyimpan bobot** tiap neuron (termasuk bias) dan **menyimpan gradien bobot** tiap neuron (termasuk bias).
- Instance model memiliki method untuk **menampilkan model** berupa **struktur jaringan** beserta **bobot** dan **gradien bobot** tiap neuron dalam bentuk graf.
- Instance model memiliki method untuk **menampilkan distribusi bobot** dan **menampilkan distribusi gradien bobot** dari tiap layer.
- Instance model memiliki method untuk **save** dan **load**.
- Model memiliki implementasi **forward propagation** dan **backward propagation**.
- Model memiliki implementasi **weight update** dengan menggunakan **gradient descent** untuk memperbarui bobot berdasarkan gradien yang telah dihitung.

- Implementasi untuk pelatihan model harus dapat menerima parameter **Batch size**, **Learning rate**, **Jumlah epoch**, dan **Verbose**
- Melakukan **pengujian** terhadap implementasi FFNN dengan ketentuan:
  - Analisis pengaruh beberapa hyperparameter : pengaruh **depth (banyak layer)** dan **width (banyak neuron per layer)**, pengaruh **fungsi aktivasi** hidden layer, pengaruh **learning rate**, pengaruh **inisialisasi bobot**
  - Analisis perbandingan hasil prediksi dengan [library sklearn MLP](#)
  - Menggunakan dataset [mnist\\_784](#) untuk menguji model

## BAB 2: Pembahasan

### 2.1. Penjelasan Implementasi

#### 2.1.1. Deskripsi Kelas

Pada source code, terdapat dua kelas utama, yaitu kelas Layer dan kelas Neural Network. Secara garis besar, kelas Layer mewakili hidden dan output layer yang terdapat di ANN, sementara kelas Neural Network terdiri dari array of layer.

Kelas Layer memiliki atribut yang dijelaskan pada **Tabel 2.1.1.** dan method yang dijelaskan pada **Tabel 2.1.2.**

*Tabel 2.1.1.* Atribut Kelas *Layer*

Atribut	Deskripsi
input_size	Jumlah fitur input yang masuk ke layer.
output_size	Jumlah neuron (output) pada layer.
activation_name	Nama fungsi aktivasi yang digunakan (dikonversi ke huruf kecil).
activation	Fungsi aktivasi yang diterapkan pada output (misal: ReLU, sigmoid, tanh, dll.).
activation_deriv	Fungsi turunan dari fungsi aktivasi, digunakan untuk perhitungan backpropagation.
W	Matriks bobot (weights) yang menghubungkan input ke neuron pada layer.
b	Vektor bias yang ditambahkan ke hasil perkalian bobot dengan input.

use_rmsnorm	Boolean untuk menentukan apakah RMS normalization digunakan pada layer.
epsilon	Nilai kecil untuk menghindari pembagian dengan nol saat perhitungan RMS normalization.
g	Parameter skala yang digunakan dalam RMS normalization (jika diaktifkan).
dg	Gradient dari parameter g, dihitung selama backpropagation (jika RMS normalization digunakan).
dW	Gradient untuk bobot W, dihitung saat proses backward.
db	Gradient untuk bias b, dihitung saat proses backward.
X	Input yang disimpan dari forward pass untuk digunakan saat backpropagation.
z_raw	Hasil perkalian linear ( $X * W + b$ ) sebelum normalisasi (jika RMS normalization diaktifkan).
rms	Nilai Root Mean Square dari z_raw, digunakan dalam normalisasi (jika diaktifkan).
z	Nilai pre-aktivasi yang akan diberikan ke fungsi aktivasi (hasil normalisasi atau z_raw).
a	Output akhir layer setelah diterapkan fungsi aktivasi.

**Tabel 2.1.2.** Method Kelas *Layer*

Method	Deskripsi
init	Konstruktor yang menginisialisasi layer dengan menentukan ukuran input/output, memilih fungsi aktivasi, inisialisasi bobot

	dan bias, serta mengatur penggunaan RMS normalization jika diperlukan.
forward	Melakukan forward propagation: menghitung nilai linear ( $z_{raw}$ ), melakukan normalisasi (jika diaktifkan), menerapkan fungsi aktivasi, dan menghasilkan output ( $a$ ).
backward	Melakukan backward propagation: menghitung gradient ( $dW$ , $db$ , dan $dg$ jika RMS normalization digunakan), mengupdate parameter ( $W$ , $b$ , dan $g$ ), serta mengembalikan error propagasi ke layer sebelumnya.

Kelas `NeuralNetwork` memiliki atribut yang dijelaskan pada **Tabel 2.1.3.** dan method yang dijelaskan pada **Tabel 2.1.4.**

***Tabel 2.1.3.*** Atribut Kelas *NeuralNetwork*

Atribut	Deskripsi Singkat
layers	List berisi objek-objek Layer yang merepresentasikan setiap layer (input, hidden, output) pada jaringan.
loss_name	Nama fungsi loss yang digunakan (misalnya "mse", "binary_crossentropy", atau "categorical_crossentropy").
loss	Fungsi loss yang digunakan untuk menghitung error antara output jaringan dan target sebenarnya.
loss_deriv	Turunan dari fungsi loss, digunakan untuk perhitungan gradien saat backpropagation.
regularization	Jenis regularisasi yang diterapkan (misalnya "L1" atau "L2") atau None jika tidak digunakan.

reg_lambda	Nilai hyperparameter yang mengatur kekuatan regularisasi.
use_rmsnorm	Boolean yang menunjukkan apakah RMS normalization diaktifkan di setiap layer.
epsilon	Nilai kecil untuk menghindari pembagian dengan nol dalam perhitungan RMS normalization.

**Tabel 2.1.4.** Method Kelas *NeuralNetwork*

Method	Deskripsi Singkat
init	Konstruktor yang menginisialisasi jaringan dengan ukuran input, konfigurasi layer, fungsi loss, metode inisialisasi bobot, regularisasi, dan opsi RMS normalization.
forward	Melakukan forward propagation melalui setiap layer untuk menghasilkan output prediksi jaringan.
backward	Melakukan backward propagation dengan menghitung gradien loss dan mengupdate parameter (bobot, bias, dan parameter RMS normalization jika ada) tiap layer.
train	Melatih jaringan dengan menggunakan data training dan validasi melalui mini-batch gradient descent, serta mencatat riwayat loss per epoch.
predict	Menghasilkan prediksi output untuk input tertentu dengan melakukan forward propagation.
save	Menyimpan model jaringan ke file menggunakan modul pickle.
load	(Static Method) Memuat model yang telah disimpan dari file, mengembalikan objek NeuralNetwork.



display_model	Menampilkan struktur model jaringan, termasuk informasi layer, ukuran, fungsi aktivasi, bobot, bias, dan gradien (jika tersedia).
visualize_network	Memvisualisasikan jaringan secara grafis dengan menggambar neuron dan koneksi antar layer beserta representasi bobot dan gradien.
plot_weight_distribution	Menampilkan histogram distribusi bobot dan bias untuk layer-layer yang dipilih.
plot_gradient_distribution	Menampilkan histogram distribusi gradien ( $dW$ dan $db$ ) untuk layer-layer yang dipilih.
plot_training_loss	Menggambarkan grafik training loss dan validation loss per epoch dari proses pelatihan.

### 2.1.2. Penjelasan Forward Propagation

Pada kelas Neural Network, tahap forward propagation dimulai dengan pemanggilan method forward yang menerima input data. Input tersebut secara berurutan diproses melalui setiap objek layer yang terdapat dalam array layers. Proses ini dilakukan dengan memanggil method forward pada masing-masing layer, sehingga output dari layer sebelumnya menjadi input bagi layer berikutnya. Dengan cara ini, data bergerak secara linier dari input layer melalui hidden layer hingga mencapai output layer, menghasilkan prediksi akhir dari jaringan.

Pada kelas Layer, method forward menerima input dari jaringan dan melakukan transformasi linear dengan cara menghitung perkalian dot antara input ( $X$ ) dengan bobot ( $W$ ) dan kemudian menambahkan bias ( $b$ ) sehingga diperoleh nilai  $z_{\text{raw}}$ . Jika RMS normalization diaktifkan, nilai  $z_{\text{raw}}$  dinormalisasi menggunakan perhitungan root mean square ( $\text{rms}$ ) yang kemudian dikalikan dengan parameter skala  $g$ . Setelah langkah normalisasi (jika diterapkan), nilai tersebut kemudian dilewatkan ke fungsi aktivasi yang sesuai (misalnya, ReLU, sigmoid, tanh, dsb.) untuk menghasilkan output aktivasi ( $a$ ), yang merupakan hasil akhir dari proses forward pada layer tersebut.

Hasil dari setiap proses forward pada tiap layer (output aktivasi  $a$ ) disimpan dan diteruskan ke layer berikutnya. Proses ini memastikan bahwa setiap layer dapat memproses informasi secara independen namun tetap saling terhubung, sehingga secara keseluruhan jaringan dapat memetakan input ke output yang diinginkan. Output akhir dari forward propagation pada Neural Network merupakan prediksi yang dapat digunakan untuk menghitung loss, dan selanjutnya dioptimasi melalui proses backpropagation.

### **2.1.3. Penjelasan Backward Propagation**

Pada kelas Neural Network, proses backward propagation dimulai setelah forward propagation menghasilkan prediksi output. Metode backward di kelas ini pertama-tama menghitung selisih error antara output yang dihasilkan dengan target menggunakan turunan fungsi loss, menghasilkan variabel delta. Delta inilah yang menggambarkan seberapa besar kesalahan pada setiap output dan menjadi dasar untuk menghitung gradien parameter di setiap layer. Proses propagasi error ini dilakukan secara terbalik, dimulai dari output layer hingga ke input layer, dengan memanggil metode backward pada setiap layer secara terbalik.

Pada kelas Layer, metode backward menerima delta yang sudah diperbarui dari layer selanjutnya dan melakukan penyesuaian berdasarkan fungsi aktivasi yang digunakan. Jika layer menggunakan fungsi aktivasi softmax, perhitungan turunannya dilakukan dengan fungsi khusus yang mengakomodasi sifat softmax. Sedangkan untuk fungsi aktivasi lainnya, delta dikalikan dengan turunan fungsi aktivasi yang telah dihitung dari nilai pre-aktivasi ( $z$ ). Jika RMS normalization diaktifkan, perhitungan delta juga mencakup koreksi dari normalisasi tersebut, sehingga diperoleh `effective_delta` yang digunakan dalam perhitungan gradien.

Selanjutnya, layer menghitung gradien bobot ( $dW$ ) dan bias ( $db$ ) dengan menerapkan aturan rantai (chain rule) menggunakan `effective_delta` dan input yang disimpan dari forward propagation. Gradien tersebut kemudian ditambahkan dengan kontribusi regularisasi jika regularisasi L1 atau L2 diaktifkan. Setelah itu, parameter-parameter seperti bobot dan bias diperbarui dengan menggunakan metode gradient descent, dimana perubahan masing-masing parameter dikurangi dengan hasil

perkalian learning rate dan gradiennya. Hasil delta yang telah dihitung untuk layer tersebut kemudian dihitung kembali dan dikembalikan ke layer sebelumnya, sehingga secara keseluruhan proses backward propagation dapat menyesuaikan parameter di seluruh jaringan untuk meminimalkan loss.

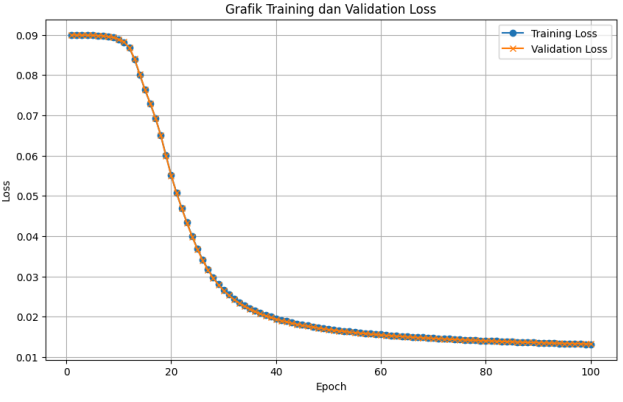
## 2.2. Hasil Pengujian

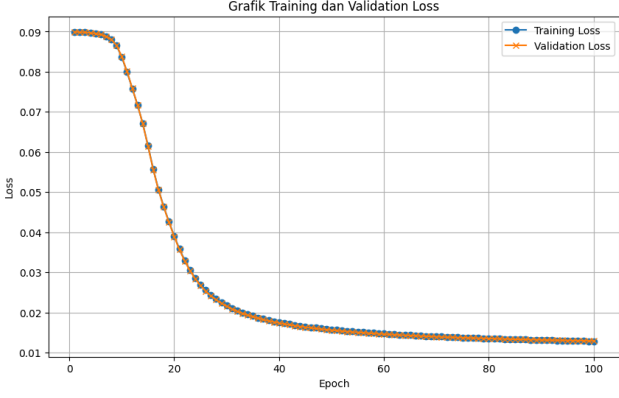
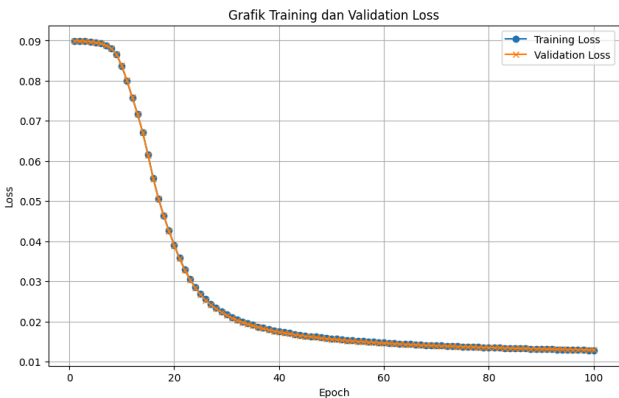
### 2.2.1. Pengaruh Depth dan Width

Sebagai konteks bahwa depth disini berarti banyak layer, dan width berarti banyak neuron yang digunakan untuk setiap layer-nya. Dalam eksperimen **Bagian 2.2.1.**, dilakukan variasi terhadap depth dan width seperti berikut.

- a. Variasi width dengan depth tetap sedalam 2 layer
  - i. 16 neuron untuk layer 1 (Linear) dan 10 neuron untuk layer 2 (Softmax)
  - ii. 64 neuron untuk layer 1 (Linear) dan 10 neuron untuk layer 2 (Softmax)
  - iii. 128 neuron untuk layer 1 (Linear) dan 10 neuron untuk layer 2 (Softmax)

**Tabel 2.2.1.1.** Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai Akurasi	Grafik Training Loss
i	0.91414285714 28571	 <p>The figure is a line graph titled 'Grafik Training dan Validation Loss'. The x-axis is labeled 'Epoch' and ranges from 0 to 100. The y-axis is labeled 'Loss' and ranges from 0.01 to 0.09. There are two data series: 'Training Loss' represented by a blue line with circular markers, and 'Validation Loss' represented by an orange line with cross markers. Both lines start at a loss of approximately 0.09 at epoch 0. They remain relatively flat until around epoch 10, then drop sharply. By epoch 40, the loss is around 0.02, and it continues to decrease slowly, reaching approximately 0.015 by epoch 100. The training and validation losses are very close to each other throughout the entire process.</p>

ii	0.917	
iii	0.9175	
<p><b>Analisis</b></p> <p>Pada konfigurasi dengan satu hidden layer dan aktivasi linear, peningkatan jumlah neuron dari 16 ke 64 dan 128 memberikan ruang representasi yang lebih besar, sehingga model memiliki kapasitas lebih tinggi untuk memproyeksikan data input ke dalam ruang fitur yang lebih mendetail. Dengan hanya 16 neuron, model terbatas dalam menangkap variasi fitur penting, sehingga sulit membedakan kelas-kelas secara optimal. Sebaliknya, dengan 64 atau 128 neuron, meskipun transformasi yang dilakukan masih linear, model dapat membuat pemetaan yang lebih kaya dan kompleks sehingga batas keputusan untuk klasifikasi dapat ditemukan dengan lebih baik.</p>		

b. Variasi depth dengan width tetap sebanyak 64 neuron untuk layer 1 (Linear)

i. 1 layer 64 neuron Linear dan 10 neuron untuk layer Softmax

- ii. 2 layer 64 neuron Linear dan 10 neuron untuk layer Softmax
- iii. 3 layer 64 neuron Linear dan 10 neuron untuk layer Softmax

**Tabel 2.2.1.2.** Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai Akurasi	Grafik Training Loss
i	0.917	
ii	0.31007142857 142855	
iii	0.11428	

### Analisis

Dalam konfigurasi tersebut, setiap layer hidden menggunakan aktivasi linear sehingga penambahan layer tidak menambah kapasitas representasional jaringan; secara matematis, komposisi beberapa transformasi linear setara dengan satu transformasi linear tunggal. Dengan demikian, meskipun jumlah parameter bertambah seiring bertambahnya layer, model tidak mampu menangkap hubungan non-linear yang lebih kompleks, sementara kompleksitas optimisasi dan risiko munculnya masalah gradien meningkat, sehingga cenderung menurunkan performansi dari model.

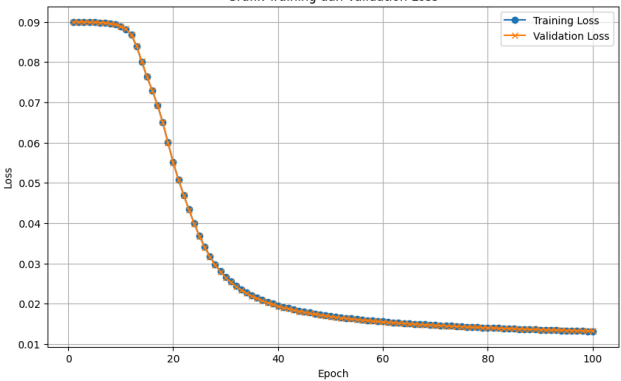
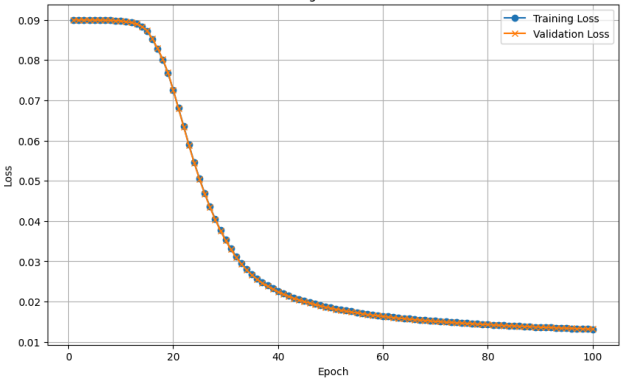
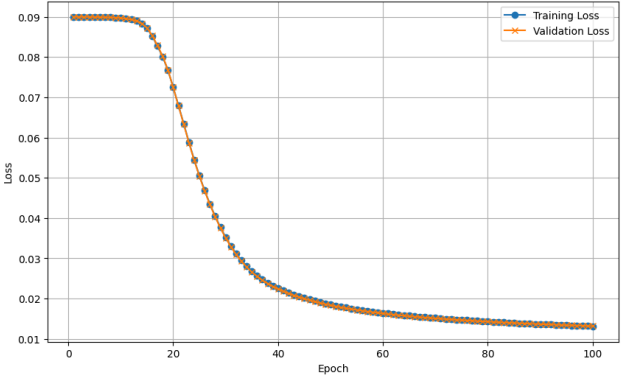
#### 2.2.2. Pengaruh fungsi aktivasi

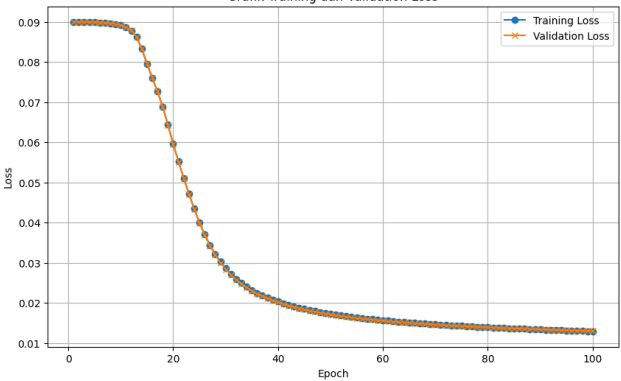
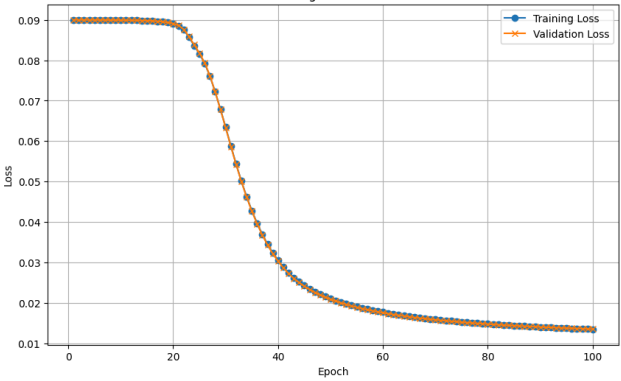
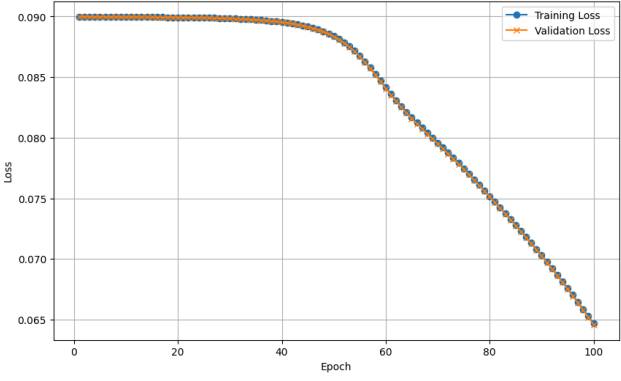
- Penggunaan fungsi aktivasi linear
- Penggunaan fungsi aktivasi ReLU
- Penggunaan fungsi aktivasi Leaky ReLU
- Penggunaan fungsi aktivasi eLU
- Penggunaan fungsi aktivasi swish
- Penggunaan fungsi aktivasi Sigmoid
- Penggunaan fungsi aktivasi tanh

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah dengan menggunakan inisialisasi bobot *random uniform*, penggunaan *learning rate* sebesar 0.01 dan menggunakan dua layer, yaitu layer 1 dengan fungsi aktivasi yang disesuaikan sebanyak 16 neuron dan layer 2 (softmax dengan 10 neuron). Hasil eksperimen dapat dilihat pada **Tabel 2.2.2**

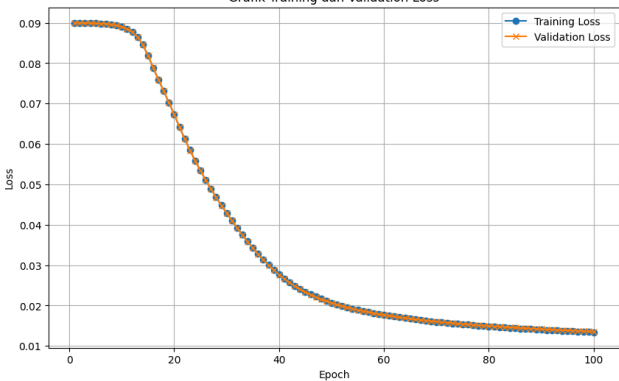
**Tabel 2.2.2.** Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai Akurasi	Grafik Training Loss
-------	---------------	----------------------

a	0.91414285714 28571	<p>Gratik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>
b	0.91642857142 85715	<p>Gratik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>
c	0.91657142857 14286	<p>Gratik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>

d	0.91664285714 28571	<p>Grafik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>
e	0.9135	<p>Grafik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>
f	0.50571428571 42857	<p>Grafik Training dan Validation Loss</p>  <p>Loss</p> <p>Epoch</p> <p>Training Loss</p> <p>Validation Loss</p>



g	0.91628571428 57143	<p style="text-align: center;">Grafik Training dan Validation Loss</p>  <p>The graph displays two data series: Training Loss and Validation Loss. The x-axis represents the number of epochs from 0 to 100, and the y-axis represents the loss value from 0.01 to 0.09. Both curves show a sharp decline from epoch 0 to 40, followed by a gradual decrease towards epoch 100. The Training Loss is consistently slightly lower than the Validation Loss throughout the training process.</p>
<p><b>Analisis</b></p> <p>Dari hasil training yang didapatkan, penggunaan fungsi aktivasi linear, ReLU, Leaky ReLU, eLU, swish dan tanh menghasilkan akurasi yang hampir seragam, yakni sekitar <math>\geq 91\%</math>, dengan grafik loss yang menunjukkan konvergensi stabil dan cepat. Sedangkan fungsi aktivasi sigmoid, yang hanya mencapai akurasi sekitar 50.57% dengan grafik loss yang cenderung menunjukkan kesulitan dalam proses optimisasi. Hal ini terjadi karena sigmoid cenderung mengalami masalah saturasi, di mana outputnya mendekati 0 atau 1 sehingga gradiennya mendekati nol (<i>vanishing gradient</i>) yang menghambat proses learning, sehingga optimisasi menjadi lambat dan performa menurun secara signifikan.</p>		

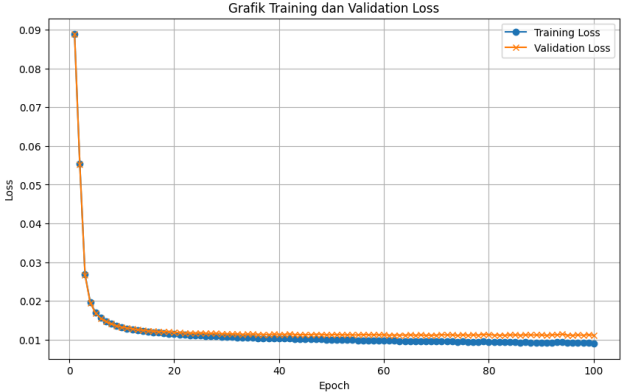
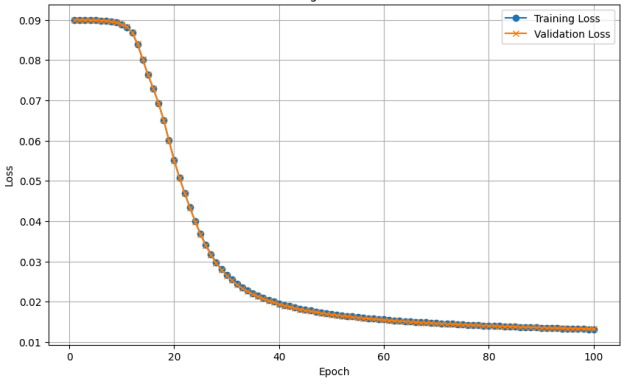
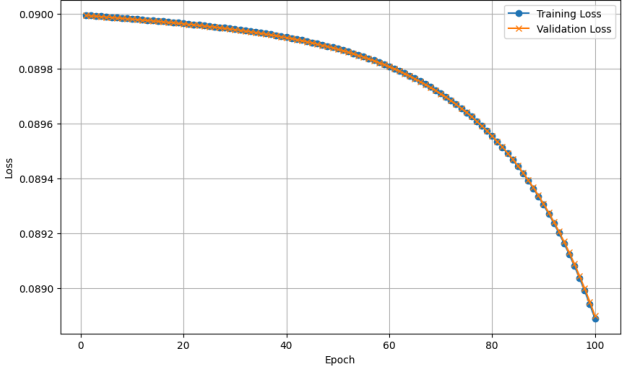
### 2.2.3. Pengaruh learning rate

- Penggunaan learning rate sebesar 0,1
- Penggunaan learning rate sebesar 0,01
- Penggunaan learning rate sebesar 0,001

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah dengan menggunakan inisialisasi bobot *random uniform*, penggunaan *learning rate* yang disesuaikan dan menggunakan dua layer, yaitu layer 1 (linear

dengan 16 neuron) dan layer 2 (softmax dengan 10 neuron). Hasil eksperimen dapat dilihat pada **Tabel 2.2.3**.

**Tabel 2.2.3.** Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai Akurasi	Grafik Training Loss
a	0.93014285714 28572	
b	0.91414285714 28571	
c	0.27357142857 14286	

### **Analisis**

Dari hasil training yang didapatkan, dengan learning rate 0,1 model mencapai akurasi tertinggi sebesar 93,01%, menunjukkan bahwa pembaruan bobot berjalan cepat dan efektif dalam menyesuaikan parameter. Sementara itu, learning rate 0,01 menghasilkan akurasi sekitar 91,41%, yang meskipun stabil, menunjukkan bahwa laju pembelajaran sedikit lebih lambat. Di sisi lain, learning rate 0,001 menyebabkan akurasi turun drastis ke 27,36% karena gradien yang sangat kecil membuat pembaruan bobot tidak signifikan, sehingga optimisasi tidak berjalan dengan baik.

*Note: dipilih 3 learning rate dari 6 untuk dianalisis*

*Learning Rate 0.001: Accuracy = 0.2735714285714286*

*Learning Rate 0.005: Accuracy = 0.8929285714285714*

*Learning Rate 0.01: Accuracy = 0.9141428571428571*

*Learning Rate 0.05: Accuracy = 0.9285*

*Learning Rate 0.1: Accuracy = 0.9301428571428572*

*Learning Rate 0.5: Accuracy = 0.9268571428571428*

*Learning rate dipilih dari yang terbesar, terkecil, dan menengah*

#### **2.2.4. Pengaruh inisialisasi bobot**

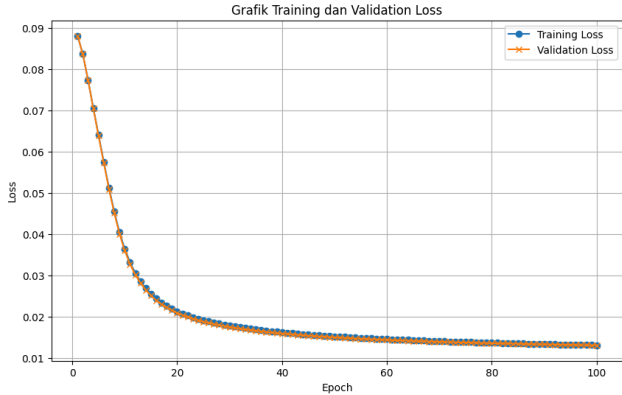
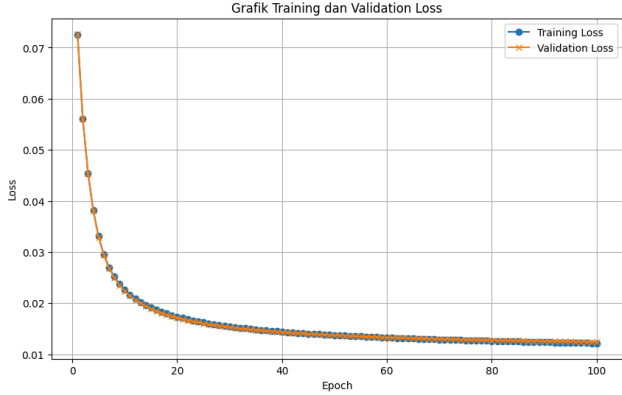
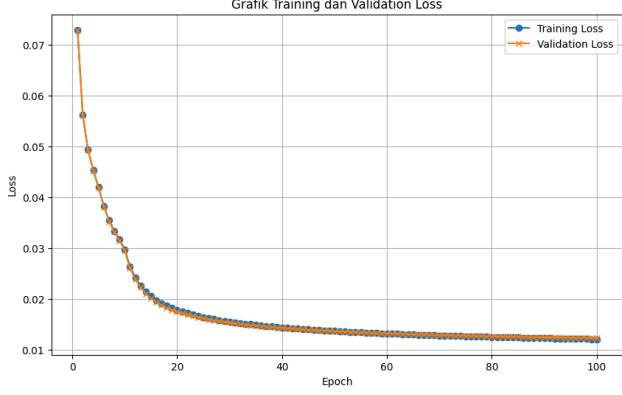
- a. Penggunaan inisialisasi bobot Zero
- b. Penggunaan inisialisasi bobot Random uniform
- c. Penggunaan inisialisasi bobot Random normal
- d. Penggunaan inisialisasi bobot Xavier
- e. Penggunaan inisialisasi bobot HE

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah dengan menggunakan inisialisasi bobot yang disesuaikan, penggunaan

*learning rate 0.01* dan menggunakan dua layer, yaitu layer 1 (linear dengan 16 neuron) dan layer 2 (softmax dengan 10 neuron). Hasil eksperimen dapat dilihat pada **Tabel 2.2.4**.

**Tabel 2.2.4.** Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai Akurasi	Grafik Training Loss
a	0.11428571428571428	
b	0.9141428571428571	

c	0.91342857142 85715	
d	0.919	
e	0.91914285714 28572	
<p><b>Analisis</b></p> <p>Dari hasil training tersebut, inisiasi Zero akurasi memiliki akurasi sangat rendah (11,43%) ini disebabkan karena model gagal memecah simetri antar neuron dan grafik loss tidak menunjukkan penurunan yang berarti yang menyebabkan akurasi rendah. Sementara itu, metode Random Uniform dan</p>		

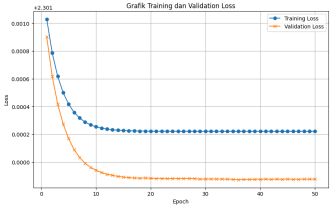
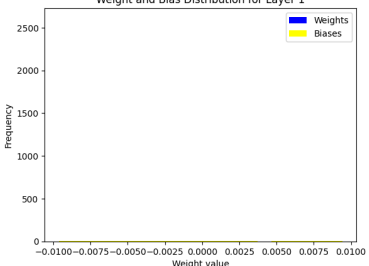
Random Normal memungkinkan bobot diinisialisasi secara acak yang di mana menghasilkan akurasi yang cukup tinggi. Lalu, metode Xavier dan HE menjaga variansi aktivasi di seluruh layer, yang membantu konvergensi menjadi lebih konsisten dan memberikan performa yang baik pula.

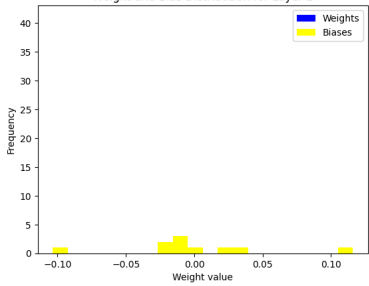
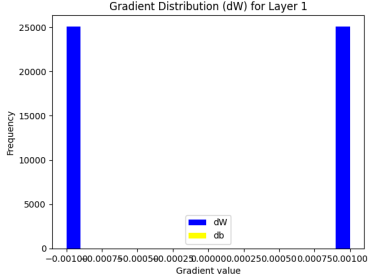
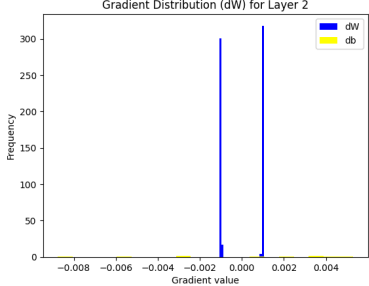
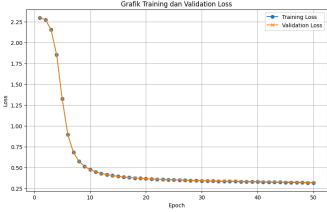
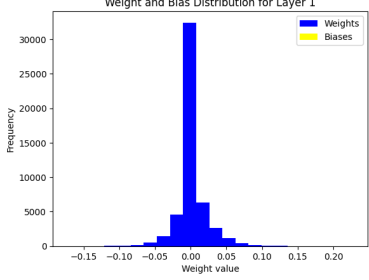
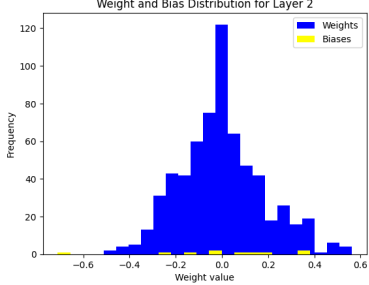
### 2.2.5. Pengaruh regularisasi

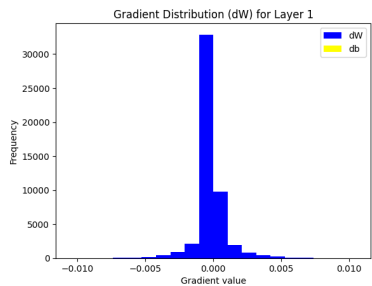
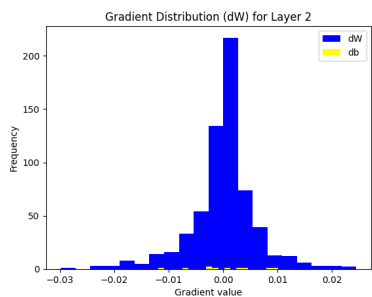
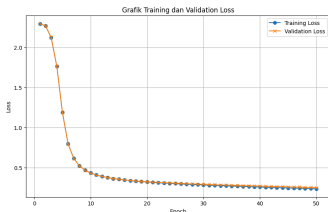
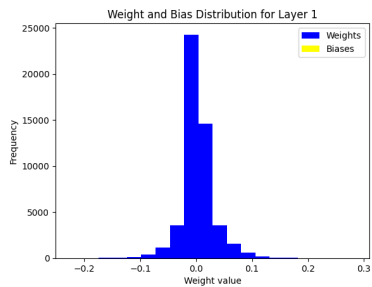
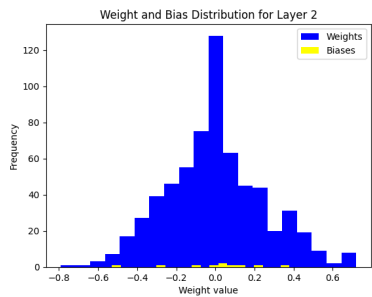
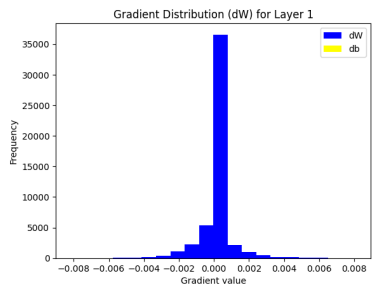
- Eksperimen model dengan regularisasi L1
- Eksperimen model dengan regularisasi L2
- Eksperimen model tanpa regularisasi

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah dengan menggunakan *loss function* berupa *categorical cross entropy*, inisialisasi bobot dengan *random uniform*, dan menggunakan dua layer, yaitu layer 1 (relu dengan 64 neuron) dan layer 2 (softmax dengan 10 neuron). Hasil eksperimen dapat dilihat pada **Tabel 2.2.5.1**.

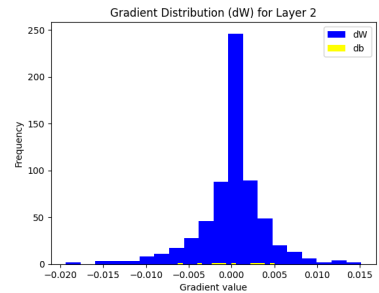
**Tabel 2.2.5.1.** Perbandingan hasil akurasi dan grafik loss pelatihan, distribusi bobot dan gradien bobot pada setiap eksperimen

No	Nilai Akurasi	Grafik Training Loss	Distribusi bobot dan gradien bobot
a	0.11428571 428571428		

			<div><div>Weight and Bias Distribution for Layer 2</div><div>Gradient Distribution (dW) for Layer 1</div><div>Gradient Distribution (dW) for Layer 2</div></div>
b	0.92628571 42857143	<div><div>Grafik Training dan Validation Loss</div></div>	<div><div>Weight and Bias Distribution for Layer 1</div><div>Weight and Bias Distribution for Layer 2</div></div>

			<div><div>Gradient Distribution (dW) for Layer 1</div><div>Gradient Distribution (dW) for Layer 2</div></div>
C	0.93542857 14285714	<div><div>Grafik Training dan Validation Loss</div></div> <div><div>Weight and Bias Distribution for Layer 1</div><div>Weight and Bias Distribution for Layer 2</div><div>Gradient Distribution (dW) for Layer 1</div></div>	





### Analisis

Dari hasil akurasi yang didapatkan ketika prediksi, urutan terbaik hingga terburuk adalah model tanpa regularisasi > model Reg L2 > model Reg L1. Berdasarkan grafik training loss pula, model regularisasi L1 menunjukkan potensi terjadinya overfitting, ditandai dengan lossnya yang cukup tinggi >2. Sedangkan, model Reg L2 dan tanpa regularisasi, mempunyai penurunan loss training yang lebih *smooth* dan *convergence*.

Model tanpa regularisasi menunjukkan performa terbaik karena bobotnya dapat sepenuhnya menyesuaikan diri dengan data tanpa batasan tambahan, sehingga model dapat menangkap kompleksitas fitur dengan lebih optimal. L2 regularization menambahkan penalti yang mendorong bobot untuk tetap kecil namun tidak menghilangkan mereka secara total, sehingga menjaga distribusi bobot yang merata dan menghasilkan proses training yang lebih smooth serta konvergensi yang stabil, meskipun sedikit mengorbankan performa dibandingkan model tanpa regularisasi. Sedangkan L1 regularization, dengan kecenderungan menghasilkan sparsity (bobot menjadi nol), dapat secara agresif mengurangi jumlah parameter yang aktif sehingga menghambat kemampuan model dalam menangkap pola yang relevan, sehingga menyebabkan training loss yang tinggi (di atas 2) dan indikasi overfitting karena hanya menyisakan bobot yang berkontribusi paling dominan di data latih, yang pada akhirnya menghasilkan akurasi yang lebih buruk pada data validasi.

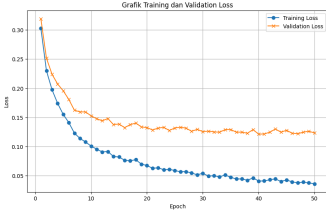
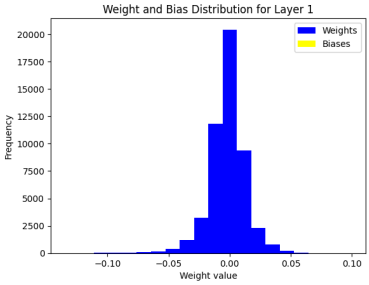
Regularisasi L1 yang naif menyebabkan banyak bobot menjadi nol, terlihat dari distribusi bobot yang menunjukkan sparsitas tinggi atau banyak nilai nol. Sebaliknya, L2 dan tanpa regularisasi memiliki distribusi bobot dan bias yang relatif mirip, karena L2 memberikan penalti yang lebih hati-hati sehingga mendorong bobot tetap kecil tanpa memaksanya menjadi nol.

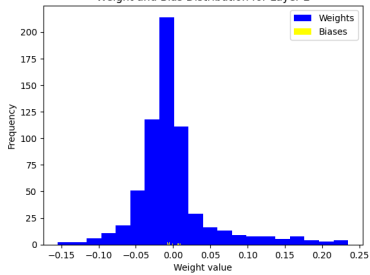
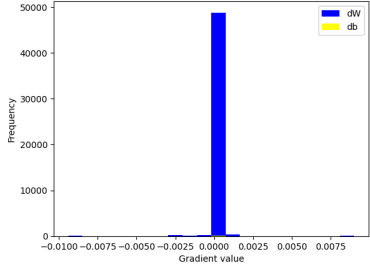
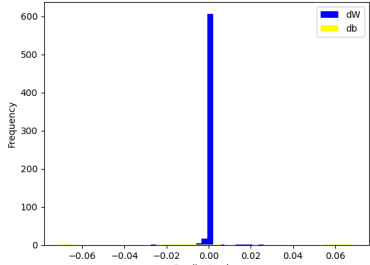
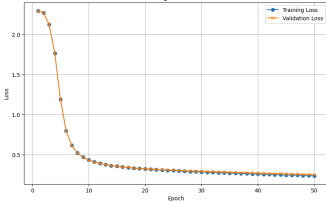
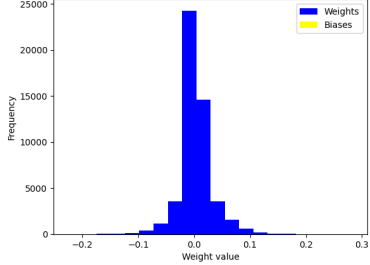
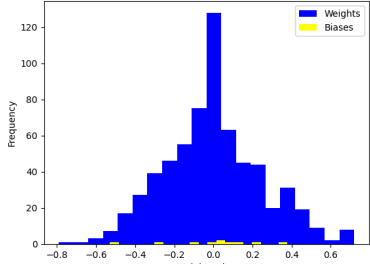
## 2.2.6. Pengaruh normalisasi RMSNorm

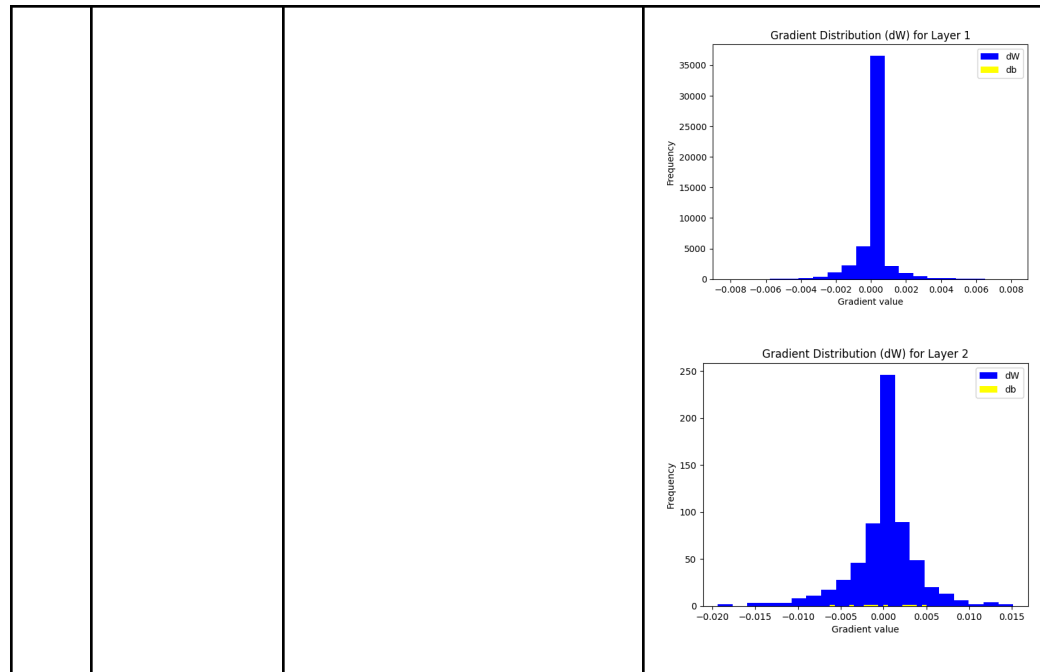
- Eksperimen model dengan normalisasi
- Eksperimen model tanpa normalisasi

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah dengan menggunakan *loss function* berupa *categorical cross entropy*, inisialisasi bobot dengan *random uniform*, dan menggunakan dua layer, yaitu layer 1 (relu dengan 64 neuron) dan layer 2 (softmax dengan 10 neuron). Hasil eksperimen dengan variasi normalisasi dapat dilihat pada **Tabel 2.2.6.1**.

**Tabel 2.2.6.1.** Perbandingan hasil akurasi dan grafik loss pelatihan, distribusi bobot dan gradien bobot pada setiap eksperimen

No	Nilai Akurasi	Grafik Training Loss	Distribusi bobot dan gradien bobot
a	0.96878571 42857143		

			<div><div>Weight and Bias Distribution for Layer 2</div><div>Gradient Distribution (dW) for Layer 1</div><div>Gradient Distribution (dW) for Layer 2</div></div>
b	0.93542857 14285714	<div>Grafik Training dan Validation Loss</div> 	<div>Weight and Bias Distribution for Layer 1</div>  <div>Weight and Bias Distribution for Layer 2</div> 



### Analisis

Berdasarkan perbandingan gambar training loss antara kedua model, model dengan normalisasi menunjukkan loss yang cenderung lebih cepat untuk konvergen dibandingkan tanpa normalisasi. Model tanpa normalisasi mengalami penurunan validation loss konsisten ketika berada di epoch > 10, sehingga menunjukkan tanda-tanda overfitting.

Selain itu, untuk distribusi bias dan bobot, model dengan normalisasi cenderung mempunyai distribusi yang lebih *tight* dan *centered*, sedangkan tanpa normalisasi menunjukkan distribusi yang lebih menyebar. Ini menunjukkan bahwa model tanpa normalisasi mempunyai mekanisme learning yang lebih tak terkontrol karena penyebaran atau variansi yang tinggi.

Berdasarkan nilai akurasi terhadap test data juga menunjukkan bahwa model dengan normalisasi mempunyai akurasi prediksi yang lebih baik dibandingkan dengan model tanpa normalisasi. Dengan begitu, pada kasus ini, model dengan normalisasi lebih direkomendasikan.

### **2.2.7. Perbandingan dengan library sklearn**

Sebagai informasi, konfigurasi model yang digunakan untuk eksperimen ini adalah inisialisasi bobot dengan *xavier*, penggunaan *learning rate* sebesar 0.01 dan menggunakan tiga layer, yaitu layer 1 (relu dengan 128 neuron), layer 2 (relu dengan 64 neuron) dan layer 3 (softmax dengan 10 neuron). Hasil akurasi yang didapatkan dengan menggunakan model yang dibuat yaitu sebesar 0.9504, sedangkan hasil akurasi yang didapatkan dengan menggunakan MLPClassifier dengan konfigurasi serupa yaitu sebesar 0.9704. Hasil yang lebih tinggi pada sklearn MLPClassifier disebabkan oleh perbedaan implementasi optimisasi. Meskipun kedua model menggunakan learning rate 0,01 dan arsitektur yang serupa, MLPClassifier memiliki fungsi-fungsi yang sudah dioptimasi dari sklearn, salah satunya Adam-optimizer, yang mengimplementasikan fitur tambahan seperti adaptive learning-rate, momentum decay, dan weight decay ter-tune) sehingga model dapat konvergen jauh lebih cepat dan mencapai akurasi lebih tinggi. Sementara itu, model yang diimplementasikan dari awal menggunakan gradient descent dasar tanpa optimasi lanjutan.

## BAB 3: Kesimpulan dan Saran

### 3.1. Kesimpulan

Berdasarkan analisis hasil pengujian pada Bagian 2.2, peningkatan jumlah neuron pada hidden layer (*width*) dengan fungsi aktivasi linear memberikan ruang representasi yang lebih besar dan kemampuan model untuk menangkap variasi fitur yang lebih mendetail. Namun, meskipun transformasi yang dilakukan linear, penambahan kedalaman layer (*depth*) tidak meningkatkan performa secara signifikan, melainkan cenderung mengurangi akurasi karena komposisi linear tetap ekuivalen dengan satu transformasi linear. Pemilihan fungsi aktivasi yang tepat juga sangat berpengaruh, dimana aktivasi seperti ReLU, Leaky ReLU, eLU, swish, dan tanh menghasilkan akurasi tinggi ( $\geq 91\%$ ) dengan konvergensi loss yang stabil, sementara fungsi sigmoid mengalami masalah saturasi gradien yang mendekati 0 (*vanishing gradient*) yang mengakibatkan optimisasi gagal.

Eksperimen terhadap learning rate menunjukkan bahwa laju pembelajaran yang lebih tinggi (0,1) mempercepat penyesuaian bobot dengan akurasi mencapai 93,01%, sedangkan learning rate terlalu rendah (0,001) membuat pembaruan bobot tidak signifikan dan akurasi turun drastis. Selain itu, inisialisasi bobot memainkan peran penting dalam menghindari permasalahan bobot simetris. Inisiasi Zero menghasilkan akurasi sangat rendah karena kegagalan memecah simetri antar neuron, sedangkan metode Random Uniform, Random Normal, Xavier, dan HE menjaga variansi aktivasi yang mendukung konvergensi dan performa yang lebih baik.

Dalam hal regularisasi, model tanpa regularisasi mencapai performa terbaik karena bobot dapat sepenuhnya menyesuaikan dengan data, meskipun penggunaan L2 regularization memberikan proses training yang lebih smooth dengan konvergensi yang stabil. Penggunaan normalisasi juga terbukti meningkatkan akurasi prediksi pada data uji dibandingkan model tanpa normalisasi. Model MLPClassifier dari sklearn lebih menunjukkan keunggulan optimasi tambahan seperti Adam-optimizer yang menggunakan learning rate adaptif dan momentum, sehingga mencapai akurasi lebih tinggi meskipun arsitektur dasarnya serupa dengan model yang diimplementasikan *from scratch*.

### 3.2. Saran

Dalam penelitian selanjutnya yang berhubungan dengan pelatihan, penelitian seharusnya melakukan lebih banyak eksperimen sehingga mendapatkan hasil model akhir yang lebih baik pula. Mitigasi risiko atas waktu pelatihan yang cukup memakan waktu, dapat dilakukan pelatihan dari awal *timeline* penelitian sehingga semakin banyak eksperimen yang dapat dieksplorasi.

Selain itu, penelitian selanjutnya juga bisa melakukan utilisasi komputasi paralel, salah satunya dengan paralelisasi GPU, sehingga waktu pelatihan dapat berkurang, dan spesifikasi komputer yang digunakan peneliti lebih dapat di utilisasi secara optimal.

Terakhir, kode juga bisa dibuat menjadi lebih bersih dan mengikuti praktik terbaik dari konsep *object oriented programming*. Untuk sekarang, kode-kode yang dibuat masih dirasa masih belum cukup modular. Dengan peningkatan modularitas ini, diharapkan pengerjaan yang dapat meminimalkan konflik yang terjadi pada *git system* dan meningkatkan keterbacaan dari kode yang telah dibuat oleh pengembang.

## BAB 4: Pembagian Tugas

*Tabel 4.1.* Pembagian tugas masing-masing anggota kelompok

Nama	NIM	Deskripsi Tugas
Elbert Chailes	13522045	Pembuatan model FFNN, inisialisasi bobot, regularisasi, normalisasi, laporan
Farel Winalda	13522047	Pembuatan model FFNN, inisialisasi bobot, regularisasi, normalisasi, laporan
Derwin Rustanly	13522115	Pembuatan model FFNN, inisialisasi bobot, regularisasi, normalisasi, laporan



## Referensi

PPT IF3270 Pembelajaran Mesin Minggu ke-2 Perceptron - [\*IF3270 Mgg02 Perceptron.pptx\*](#)

PPT IF3270 Pembelajaran Mesin Minggu ke-3 FFNN - [\*IF3270 Mgg03 FFNN.pptx\*](#)

PPT IF3270 Pembelajaran Mesin Minggu ke-3 Review Perceptron dan FFNN - [\*Review Perceptron dan FFNN\*](#)