

Laporan Tugas Besar 1
IF3270 Pembelajaran Mesin
Feedforward Neural Network
Semester II Tahun 2024/2025

Disusun Oleh:

Shafiq Irvansyah 13522003

Muhammad Al Thariq Fairuz 13522027

Randy Verdian 13522067



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.....	4
2.1. Implementasi.....	4
2.1.1. Deskripsi Class dan Method.....	4
2.1.2. Forward Propagation.....	8
2.1.3. Backward Propagation dan Weight Update.....	8
2.2. Hasil Pengujian.....	9
2.2.1. Pengaruh depth dan width.....	9
2.2.2. Pengaruh fungsi aktivasi.....	14
2.2.3. Pengaruh learning rate.....	15
2.2.4. Pengaruh inisialisasi bobot.....	17
2.2.5. Pengaruh regularisasi.....	18
2.2.6. Pengaruh RMSNorm.....	19
2.2.7. Perbandingan dengan library sklearn.....	20
Bab 3	
Kesimpulan dan Saran.....	22
Bab 4	
Pembagian Tugas.....	23
Referensi.....	24

Bab 1

Deskripsi Persoalan

Tugas Besar 1 IF3270 Pembelajaran Mesin bertujuan untuk memberikan wawasan kepada mahasiswa tentang cara mengimplementasikan Feedforward Neural Network (FFNN) dari awal (from *scratch*). Persoalan utama dalam tugas ini adalah mengembangkan modul FFNN yang dapat digunakan untuk membuat, melatih, dan menguji model neural network tanpa menggunakan *framework deep learning* yang sudah ada.

Modul FFNN yang dikembangkan harus memiliki kemampuan untuk menerima konfigurasi arsitektur jaringan berupa jumlah neuron pada tiap layer dan menerima konfigurasi fungsi aktivasi untuk tiap layer (Linear, ReLU, Sigmoid, Hyperbolic Tangent, dan Softmax). Selain itu, modul ini harus dapat menerima fungsi loss yang akan digunakan (MSE, Binary Cross-Entropy, dan Categorical Cross-Entropy) dan melakukan inisialisasi bobot dengan berbagai metode (Zero, Uniform, dan Normal).

Implementasi modul juga harus mampu menyimpan bobot dan gradien bobot tiap neuron termasuk bias, serta memvisualisasikan model dalam bentuk graf beserta bobot dan gradien bobotnya. Fitur penting lainnya adalah kemampuan untuk memvisualisasikan distribusi bobot dan gradien bobot dari layer-layer tertentu, serta menyimpan dan memuat model (save dan load).

Dari segi fungsionalitas, modul harus dapat melakukan *forward propagation* dengan kemampuan menerima input batch, *backward propagation* untuk menghitung gradien menggunakan chain rule, dan melakukan pembaruan bobot (weight update) menggunakan gradient descent. Modul ini akan diuji menggunakan dataset MNIST untuk mengevaluasi pengaruh berbagai hyperparameter seperti kedalaman jaringan (*depth*), lebar jaringan (*width*), fungsi aktivasi, learning rate, dan metode inisialisasi bobot.

Hasil dari model yang dikembangkan juga akan dibandingkan dengan implementasi MLP dari library sklearn. Selain itu, terdapat beberapa fitur bonus yang dapat diimplementasikan, seperti automatic differentiation, fungsi aktivasi tambahan, metode inisialisasi bobot Xavier dan He, metode regularisasi L1 dan L2, serta metode normalisasi RMSNorm.

Tugas ini bertujuan untuk memberikan pemahaman mendalam tentang bagaimana neural network bekerja dari aspek paling fundamental, termasuk mekanisme propagasi maju (*forward propagation*), propagasi mundur (*backward propagation*), dan pembaruan bobot (*weight update*) melalui implementasi langsung tanpa bergantung pada *library deep learning* yang sudah ada.

Bab 2

Pembahasan

2.1. Implementasi

2.1.1. Deskripsi *Class* dan *Method*

1. Class Activations

Method	Deskripsi
<code>linear(x)</code>	Fungsi aktivasi linier yang mengembalikan nilai input tanpa perubahan.
<code>linear_derivative(x)</code>	Turunan fungsi aktivasi linier, selalu bernilai 1 untuk semua input.
<code>relu(x)</code>	Fungsi aktivasi ReLU (Rectified Linear Unit) yang mengembalikan nilai maksimum antara 0 dan x.
<code>relu_derivative(x)</code>	Turunan fungsi ReLU, bernilai 1 jika $x > 0$ dan 0 jika $x \leq 0$.
<code>sigmoid(x)</code>	Fungsi aktivasi sigmoid yang memetakan input ke rentang (0,1), dengan clipping untuk menghindari overflow.
<code>sigmoid_derivative(x)</code>	Turunan fungsi sigmoid dengan formula $s*(1-s)$, dimana s adalah nilai sigmoid.
<code>tanh(x)</code>	Fungsi aktivasi hyperbolic tangent yang memetakan input ke rentang (-1,1).
<code>tanh_derivative(x)</code>	Turunan fungsi tanh dengan formula $1-\tanh^2(x)$.
<code>softmax(x, axis=-1)</code>	Fungsi aktivasi softmax yang mengubah vektor input menjadi distribusi probabilitas (jumlah = 1).
<code>softmax_derivative(x, axis=-1)</code>	Method untuk mengimplementasikan turunan fungsi softmax dengan asumsi softmax menggunakan loss categorical cross entropy.
<code>leaky_relu(x)</code>	Fungsi aktivasi untuk leaky relu, bernilai X jika $X > 0$ dan $X * 0.01$ otherwise.

<code>leaky_relu_derivative(x)</code>	Turunan dari fungsi aktivasi leaky relu, bernilai 1 jika $X > 0$ dan 0.01 <i>otherwise</i> .
<code>exponential_relu(x)</code>	Fungsi aktivasi untuk leaky relu, bernilai X jika $X > 0$ dan $e^x - 1$ <i>otherwise</i> .
<code>exponential_relu_derivative(x)</code>	Turunan fungsi aktivasi untuk exponential relu, bernilai 1 jika $X > 0$ dan e^x <i>otherwise</i> .

2. Class LinearLayer

Method	Deskripsi
<code>forward(self, x)</code>	Metode untuk melakukan operasi forward pass. Menyimpan input untuk digunakan di backward pass.
<code>backward(self, grad_output)</code>	Metode untuk melakukan backward pass, menghitung gradien untuk W , b , dan input.

3. Class Regularization

Method	Deskripsi
<code>l1_regularization(model, lambda_val=0.01)</code>	Metode statis untuk menerapkan regularisasi L1 (Lasso) pada model. Menambahkan $\lambda * w $ ke fungsi loss dan memperbarui gradien bobot dengan $\lambda * \text{sign}(w)$.
<code>l2_regularization(model, lambda_val=0.01)</code>	Metode statis untuk menerapkan regularisasi L2 (Ridge) pada model. Menambahkan $\lambda * w ^2 / 2$ ke fungsi loss dan memperbarui gradien bobot dengan $\lambda * w$.

4. Class ActivationLayer

Method	Deskripsi
<code>forward(self, x)</code>	Melakukan proses forward pass dengan menerapkan fungsi aktivasi pada input x .
<code>backward(self, grad_output)</code>	Melakukan proses backward pass; menghitung gradien berdasarkan turunan

	fungsi aktivasi atau mengembalikan gradien langsung untuk softmax.
--	--

5. Class RMSNorm

Method	Deskripsi
<code>forward(self, x)</code>	Melakukan forward pass dengan menghitung Root Mean Square (RMS), normalisasi input, lalu mengalikan dengan parameter scale.
<code>backward(self, grad_output)</code>	Menghitung backward pass; menghitung gradien terhadap parameter scale dan input, serta menangani pengaruh dari RMS normalization.

6. Class Layer

Method	Deskripsi
<code>forward(self, x)</code>	Method abstrak untuk forward pass; harus diimplementasikan di subclass.
<code>backward(self, grad_output)</code>	Method abstrak untuk backward pass; harus diimplementasikan di subclass.
<code>get_params(self)</code>	Mengembalikan dictionary parameter layer.
<code>get_grads(self)</code>	Mengembalikan dictionary gradien parameter layer.

7. Class Losses

Method	Deskripsi
<code>mse(y_pred, y_true)</code>	Menghitung Mean Squared Error (MSE) antara prediksi dan target.
<code>mse_derivative(y_pred, y_true)</code>	Menghitung turunan (gradien) dari MSE terhadap prediksi.
<code>binary_cross_entropy(y_pred, y_true)</code>	Menghitung Binary Cross-Entropy Loss antara prediksi dan target.
<code>binary_cross_entropy_derivativ</code>	Menghitung turunan (gradien) dari Binary

<code>e(y_pred, y_true)</code>	Cross-Entropy Loss terhadap prediksi.
<code>categorical_crossentropy(y_pred, y_true)</code>	Menghitung Categorical Cross-Entropy Loss antara prediksi dan target (one-hot encoded).
<code>categorical_crossentropy_derivative(y_pred, y_true)</code>	Placeholder untuk menghitung turunan (gradien) dari Categorical Cross-Entropy Loss.

8. Class Initializers

Method	Deskripsi
<code>zero_init(shape)</code>	Menginisialisasi array dengan nilai nol sesuai bentuk (shape).
<code>uniform_init(shape, low, high, seed)</code>	Menginisialisasi array dengan distribusi uniform antara low dan high.
<code>normal_init(shape, mean, var, seed)</code>	Menginisialisasi array dengan distribusi normal dengan mean dan varians tertentu.
<code>xavier_init(shape, seed)</code>	Menginisialisasi array dengan Xavier/Glorot initialization; cocok untuk aktivasi sigmoid/tanh.
<code>he_init(shape, seed)</code>	Menginisialisasi array dengan He initialization; cocok untuk aktivasi ReLU.

9. Class FFNN

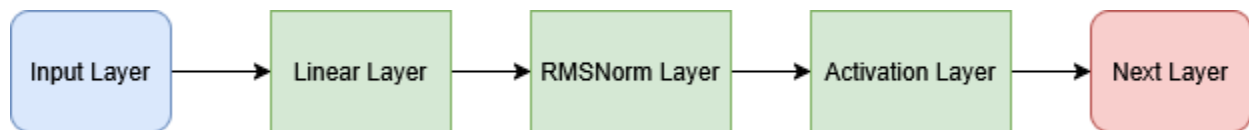
Method	Deskripsi
<code>forward(x)</code>	Melakukan forward pass input x melewati semua layer, mengembalikan output dan semua aktivasi layer.
<code>backward(y_pred, y_true, activations)</code>	Melakukan backward pass untuk menghitung gradien menggunakan backpropagation.
<code>get_params()</code>	Mengembalikan semua parameter trainable dari semua layer.
<code>get_gradients()</code>	Mengembalikan semua gradien parameter dari semua layer.

<code>train_step(x_batch, y_batch, learning_rate, reg_type, lambda_val)</code>	Melakukan 1 langkah training: forward pass, hitung loss, backward pass, update weights, dan reset gradien.
<code>train(x_train, y_train, batch_size, learning_rate, epochs, x_val, y_val, verbose, reg_type, lambda_val)</code>	Melatih model dengan data training (mini-batch SGD), menghitung loss, dan opsional validasi.
<code>predict(x)</code>	Melakukan prediksi output untuk input x.
<code>visualize_model(x)</code>	Menampilkan grafik arsitektur model FFNN yang dibangun

2.1.2. Forward Propagation

Forward propagation adalah proses di mana input data dilewatkan melalui jaringan saraf dari layer input menuju layer output. Dalam metode `forward()` pada kelas FFNN:

1. Input data x dimasukkan ke dalam jaringan.
2. Data dilewatkan melalui setiap layer secara berurutan:
 - Linear Layer: Melakukan transformasi linier dengan perkalian matriks ($W * x$) dan penambahan bias
 - RMSNorm Layer (opsional): Normalisasi input dengan root mean square
 - Activation Layer: Menerapkan fungsi aktivasi non-linear seperti ReLU, sigmoid, atau tanh

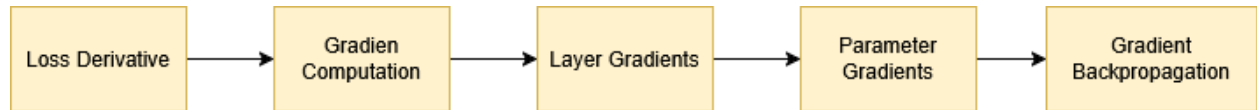


Gambar 1. Forward Propagation FFNN

2.1.3. Backward Propagation dan Weight Update

Backward propagation adalah proses menghitung gradien untuk setiap parameter dalam jaringan. Metode `backward()` melakukan hal ini:

1. Mulai dengan menghitung gradien loss menggunakan loss derivative
2. Backpropagate gradien melalui layer-layer secara terbalik
3. Setiap layer menghitung:
 - Gradien terhadap parameternya sendiri
 - Gradien yang akan diteruskan ke layer sebelumnya



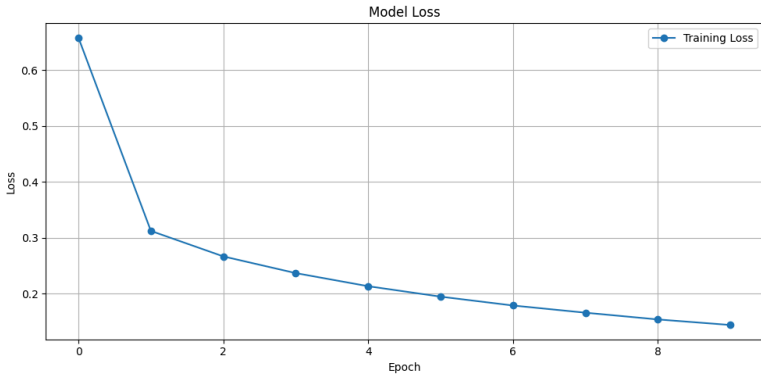
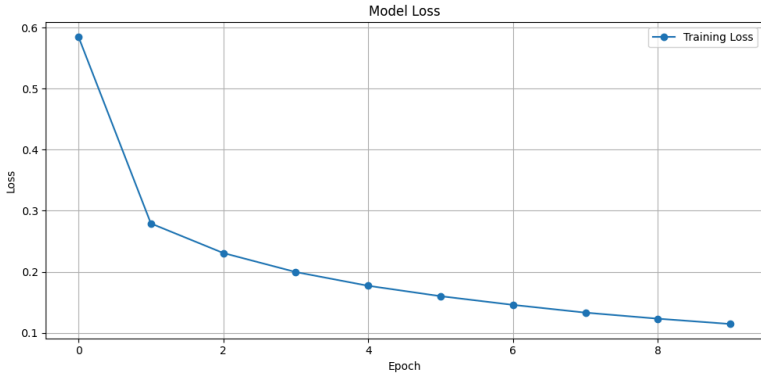
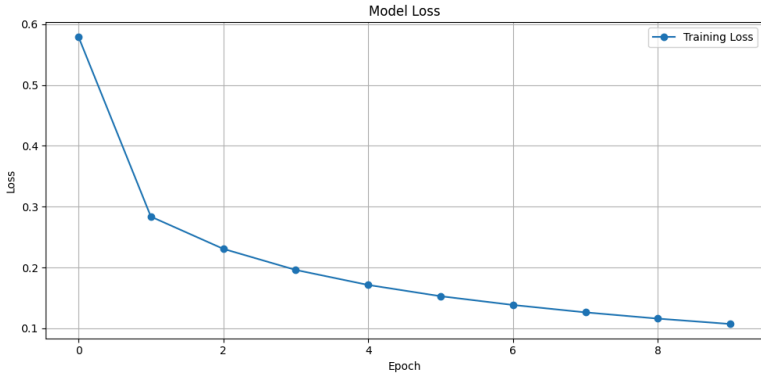
Gambar 2. *Backward Propagation* FFNN

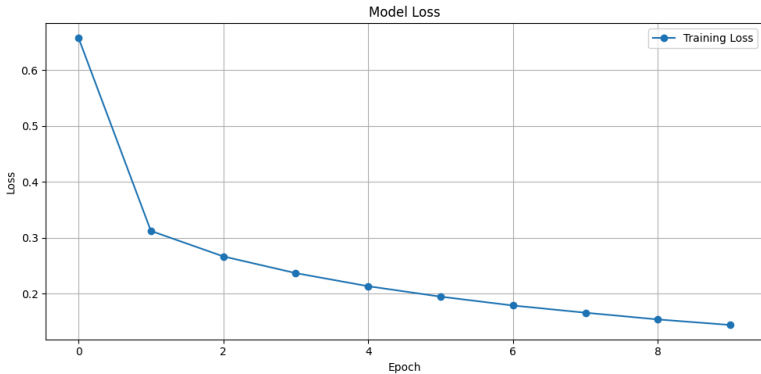
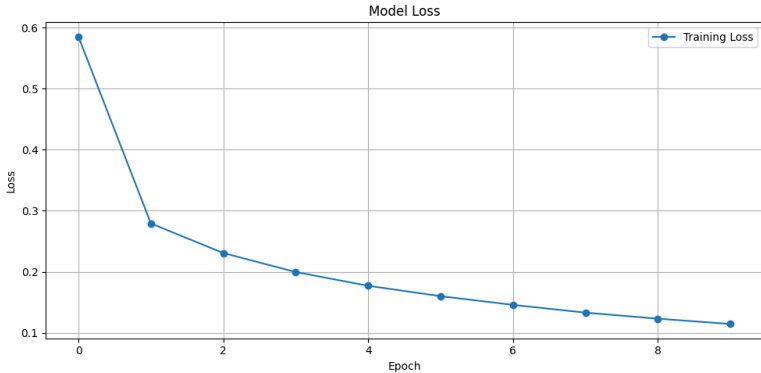
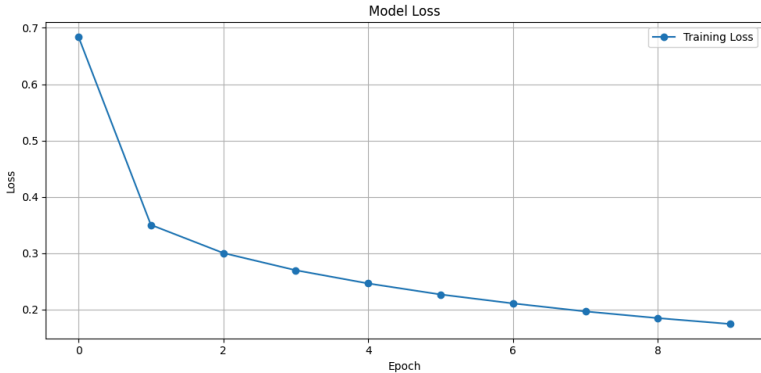
Weight update dilakukan dalam metode `train_step()`. Setelah backward propagation, gradien untuk setiap parameter tersedia. Dengan menggunakan metode gradient descent, setiap parameter dikurangi dengan $\text{learning rate} \times \text{gradiennya}$. Kemudian, gradien direset untuk iterasi selanjutnya.

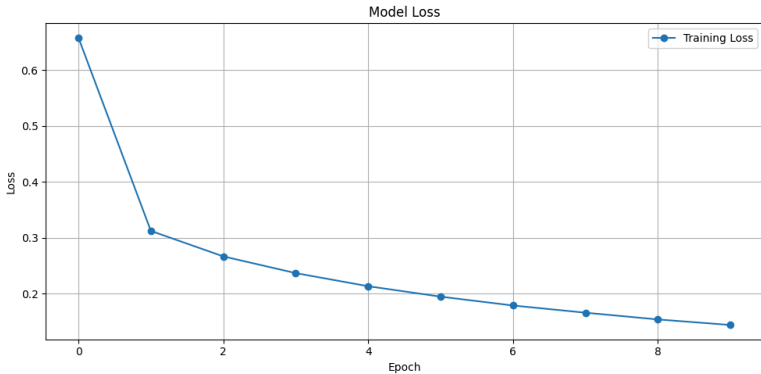
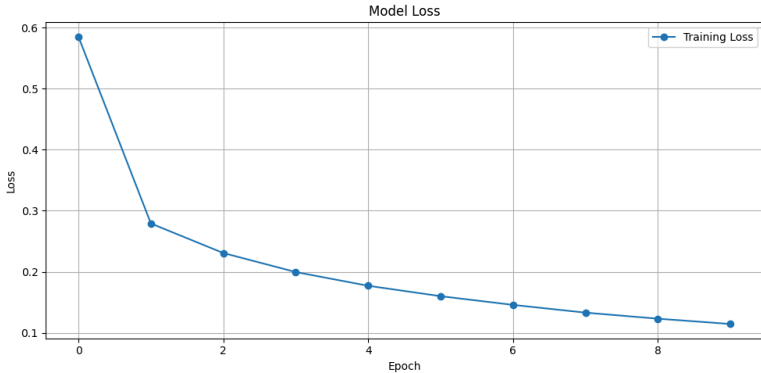
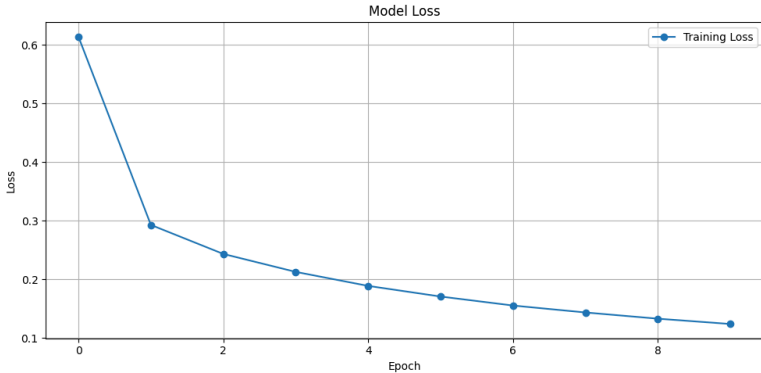
2.2. Hasil Pengujian

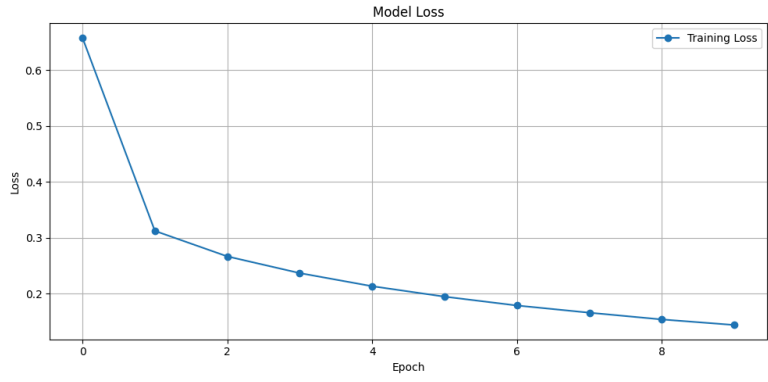
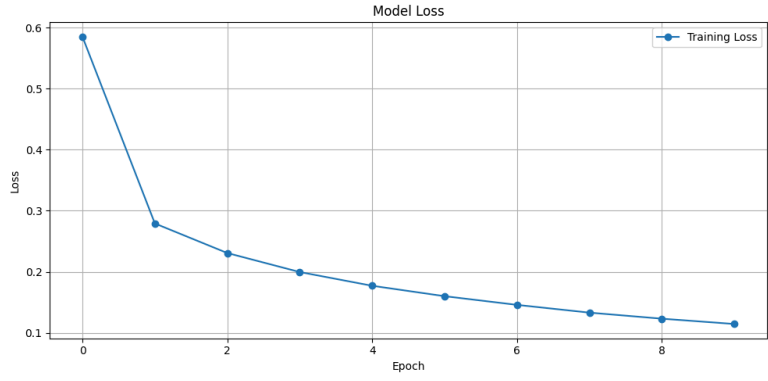
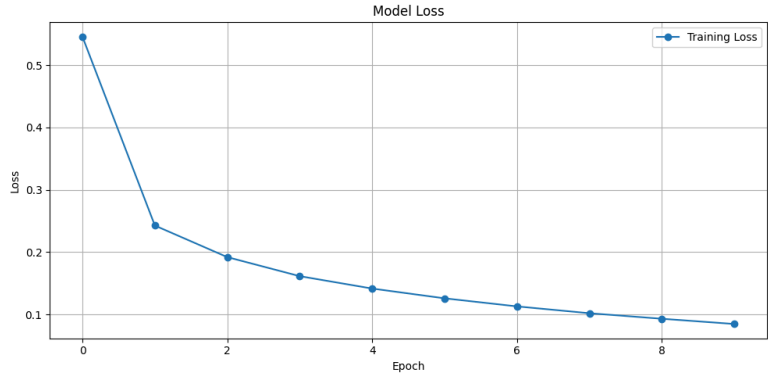
2.2.1. Pengaruh *depth* dan *width*

Variabel Kontrol	
Input Size	784
Fungsi Aktivasi	ReLU
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

Jumlah hidden layer	Jumlah neuron	Akurasi	Grafik Loss																						
2	32	0.952	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.65</td></tr><tr><td>1</td><td>0.31</td></tr><tr><td>2</td><td>0.27</td></tr><tr><td>3</td><td>0.24</td></tr><tr><td>4</td><td>0.21</td></tr><tr><td>5</td><td>0.19</td></tr><tr><td>6</td><td>0.18</td></tr><tr><td>7</td><td>0.17</td></tr><tr><td>8</td><td>0.16</td></tr><tr><td>9</td><td>0.15</td></tr></table>	Epoch	Training Loss	0	0.65	1	0.31	2	0.27	3	0.24	4	0.21	5	0.19	6	0.18	7	0.17	8	0.16	9	0.15
Epoch	Training Loss																								
0	0.65																								
1	0.31																								
2	0.27																								
3	0.24																								
4	0.21																								
5	0.19																								
6	0.18																								
7	0.17																								
8	0.16																								
9	0.15																								
2	64	0.959	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.58</td></tr><tr><td>1</td><td>0.28</td></tr><tr><td>2</td><td>0.23</td></tr><tr><td>3</td><td>0.20</td></tr><tr><td>4</td><td>0.18</td></tr><tr><td>5</td><td>0.16</td></tr><tr><td>6</td><td>0.15</td></tr><tr><td>7</td><td>0.14</td></tr><tr><td>8</td><td>0.13</td></tr><tr><td>9</td><td>0.12</td></tr></table>	Epoch	Training Loss	0	0.58	1	0.28	2	0.23	3	0.20	4	0.18	5	0.16	6	0.15	7	0.14	8	0.13	9	0.12
Epoch	Training Loss																								
0	0.58																								
1	0.28																								
2	0.23																								
3	0.20																								
4	0.18																								
5	0.16																								
6	0.15																								
7	0.14																								
8	0.13																								
9	0.12																								
2	128	0.962	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.58</td></tr><tr><td>1</td><td>0.28</td></tr><tr><td>2</td><td>0.23</td></tr><tr><td>3</td><td>0.20</td></tr><tr><td>4</td><td>0.18</td></tr><tr><td>5</td><td>0.16</td></tr><tr><td>6</td><td>0.14</td></tr><tr><td>7</td><td>0.13</td></tr><tr><td>8</td><td>0.12</td></tr><tr><td>9</td><td>0.11</td></tr></table>	Epoch	Training Loss	0	0.58	1	0.28	2	0.23	3	0.20	4	0.18	5	0.16	6	0.14	7	0.13	8	0.12	9	0.11
Epoch	Training Loss																								
0	0.58																								
1	0.28																								
2	0.23																								
3	0.20																								
4	0.18																								
5	0.16																								
6	0.14																								
7	0.13																								
8	0.12																								
9	0.11																								

Jumlah hidden layer	Jumlah neuron	Akurasi	Grafik Loss																						
2	32	0.952	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Loss</th></tr><tr><td>0</td><td>0.65</td></tr><tr><td>1</td><td>0.31</td></tr><tr><td>2</td><td>0.27</td></tr><tr><td>3</td><td>0.24</td></tr><tr><td>4</td><td>0.21</td></tr><tr><td>5</td><td>0.19</td></tr><tr><td>6</td><td>0.18</td></tr><tr><td>7</td><td>0.17</td></tr><tr><td>8</td><td>0.16</td></tr><tr><td>9</td><td>0.15</td></tr></table>	Epoch	Loss	0	0.65	1	0.31	2	0.27	3	0.24	4	0.21	5	0.19	6	0.18	7	0.17	8	0.16	9	0.15
Epoch	Loss																								
0	0.65																								
1	0.31																								
2	0.27																								
3	0.24																								
4	0.21																								
5	0.19																								
6	0.18																								
7	0.17																								
8	0.16																								
9	0.15																								
2	64	0.959	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Loss</th></tr><tr><td>0</td><td>0.58</td></tr><tr><td>1</td><td>0.28</td></tr><tr><td>2</td><td>0.23</td></tr><tr><td>3</td><td>0.20</td></tr><tr><td>4</td><td>0.18</td></tr><tr><td>5</td><td>0.16</td></tr><tr><td>6</td><td>0.15</td></tr><tr><td>7</td><td>0.14</td></tr><tr><td>8</td><td>0.13</td></tr><tr><td>9</td><td>0.12</td></tr></table>	Epoch	Loss	0	0.58	1	0.28	2	0.23	3	0.20	4	0.18	5	0.16	6	0.15	7	0.14	8	0.13	9	0.12
Epoch	Loss																								
0	0.58																								
1	0.28																								
2	0.23																								
3	0.20																								
4	0.18																								
5	0.16																								
6	0.15																								
7	0.14																								
8	0.13																								
9	0.12																								
1	64	0.949	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Loss</th></tr><tr><td>0</td><td>0.68</td></tr><tr><td>1</td><td>0.35</td></tr><tr><td>2</td><td>0.30</td></tr><tr><td>3</td><td>0.27</td></tr><tr><td>4</td><td>0.25</td></tr><tr><td>5</td><td>0.23</td></tr><tr><td>6</td><td>0.21</td></tr><tr><td>7</td><td>0.20</td></tr><tr><td>8</td><td>0.19</td></tr><tr><td>9</td><td>0.18</td></tr></table>	Epoch	Loss	0	0.68	1	0.35	2	0.30	3	0.27	4	0.25	5	0.23	6	0.21	7	0.20	8	0.19	9	0.18
Epoch	Loss																								
0	0.68																								
1	0.35																								
2	0.30																								
3	0.27																								
4	0.25																								
5	0.23																								
6	0.21																								
7	0.20																								
8	0.19																								
9	0.18																								

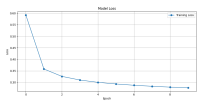
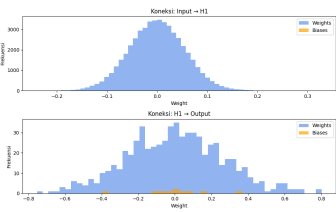
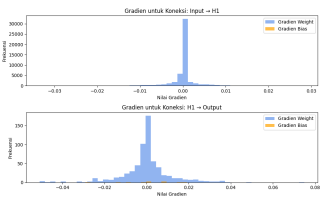
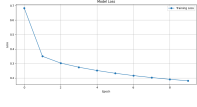
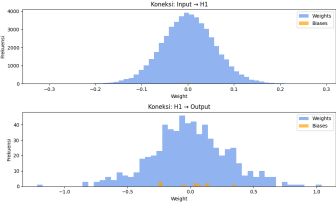
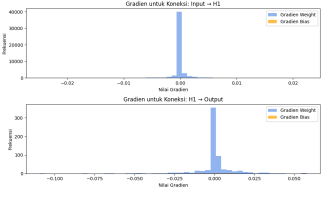
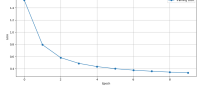
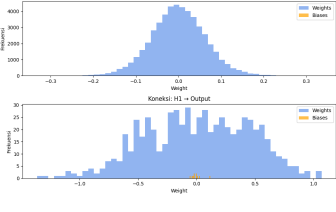
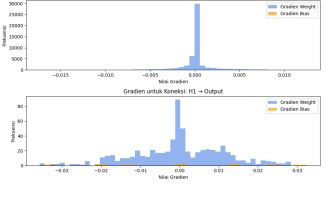
Jumlah hidden layer	Jumlah neuron	Akurasi	Grafik Loss																						
2	32	0.952	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.65</td></tr><tr><td>1</td><td>0.31</td></tr><tr><td>2</td><td>0.27</td></tr><tr><td>3</td><td>0.24</td></tr><tr><td>4</td><td>0.21</td></tr><tr><td>5</td><td>0.19</td></tr><tr><td>6</td><td>0.18</td></tr><tr><td>7</td><td>0.17</td></tr><tr><td>8</td><td>0.16</td></tr><tr><td>9</td><td>0.15</td></tr></table>	Epoch	Training Loss	0	0.65	1	0.31	2	0.27	3	0.24	4	0.21	5	0.19	6	0.18	7	0.17	8	0.16	9	0.15
Epoch	Training Loss																								
0	0.65																								
1	0.31																								
2	0.27																								
3	0.24																								
4	0.21																								
5	0.19																								
6	0.18																								
7	0.17																								
8	0.16																								
9	0.15																								
2	64	0.959	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.58</td></tr><tr><td>1</td><td>0.28</td></tr><tr><td>2</td><td>0.23</td></tr><tr><td>3</td><td>0.20</td></tr><tr><td>4</td><td>0.18</td></tr><tr><td>5</td><td>0.16</td></tr><tr><td>6</td><td>0.15</td></tr><tr><td>7</td><td>0.14</td></tr><tr><td>8</td><td>0.13</td></tr><tr><td>9</td><td>0.12</td></tr></table>	Epoch	Training Loss	0	0.58	1	0.28	2	0.23	3	0.20	4	0.18	5	0.16	6	0.15	7	0.14	8	0.13	9	0.12
Epoch	Training Loss																								
0	0.58																								
1	0.28																								
2	0.23																								
3	0.20																								
4	0.18																								
5	0.16																								
6	0.15																								
7	0.14																								
8	0.13																								
9	0.12																								
2	64	0.956	 <p>Model Loss</p> <p>Training Loss</p> <table><tr><th>Epoch</th><th>Training Loss</th></tr><tr><td>0</td><td>0.62</td></tr><tr><td>1</td><td>0.29</td></tr><tr><td>2</td><td>0.25</td></tr><tr><td>3</td><td>0.21</td></tr><tr><td>4</td><td>0.19</td></tr><tr><td>5</td><td>0.17</td></tr><tr><td>6</td><td>0.16</td></tr><tr><td>7</td><td>0.15</td></tr><tr><td>8</td><td>0.14</td></tr><tr><td>9</td><td>0.13</td></tr></table>	Epoch	Training Loss	0	0.62	1	0.29	2	0.25	3	0.21	4	0.19	5	0.17	6	0.16	7	0.15	8	0.14	9	0.13
Epoch	Training Loss																								
0	0.62																								
1	0.29																								
2	0.25																								
3	0.21																								
4	0.19																								
5	0.17																								
6	0.16																								
7	0.15																								
8	0.14																								
9	0.13																								

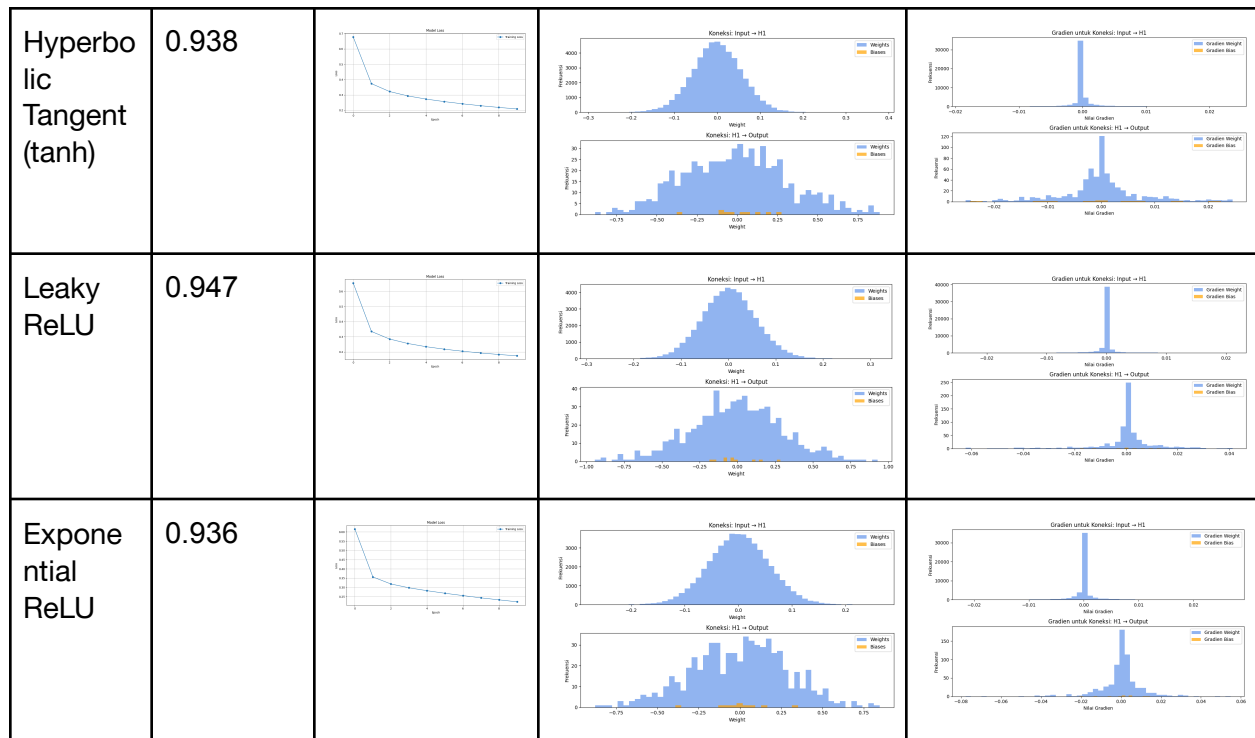
Jumlah hidden layer	Jumlah neuron	Akurasi	Grafik Loss																						
2	32	0.952	 <table><caption>Training Loss Data for 2 hidden layers, 32 neurons</caption><thead><tr><th>Epoch</th><th>Training Loss</th></tr></thead><tbody><tr><td>0</td><td>0.65</td></tr><tr><td>1</td><td>0.32</td></tr><tr><td>2</td><td>0.27</td></tr><tr><td>3</td><td>0.24</td></tr><tr><td>4</td><td>0.22</td></tr><tr><td>5</td><td>0.20</td></tr><tr><td>6</td><td>0.18</td></tr><tr><td>7</td><td>0.17</td></tr><tr><td>8</td><td>0.16</td></tr><tr><td>9</td><td>0.15</td></tr></tbody></table>	Epoch	Training Loss	0	0.65	1	0.32	2	0.27	3	0.24	4	0.22	5	0.20	6	0.18	7	0.17	8	0.16	9	0.15
Epoch	Training Loss																								
0	0.65																								
1	0.32																								
2	0.27																								
3	0.24																								
4	0.22																								
5	0.20																								
6	0.18																								
7	0.17																								
8	0.16																								
9	0.15																								
2	64	0.959	 <table><caption>Training Loss Data for 2 hidden layers, 64 neurons</caption><thead><tr><th>Epoch</th><th>Training Loss</th></tr></thead><tbody><tr><td>0</td><td>0.58</td></tr><tr><td>1</td><td>0.28</td></tr><tr><td>2</td><td>0.23</td></tr><tr><td>3</td><td>0.20</td></tr><tr><td>4</td><td>0.18</td></tr><tr><td>5</td><td>0.16</td></tr><tr><td>6</td><td>0.15</td></tr><tr><td>7</td><td>0.14</td></tr><tr><td>8</td><td>0.13</td></tr><tr><td>9</td><td>0.12</td></tr></tbody></table>	Epoch	Training Loss	0	0.58	1	0.28	2	0.23	3	0.20	4	0.18	5	0.16	6	0.15	7	0.14	8	0.13	9	0.12
Epoch	Training Loss																								
0	0.58																								
1	0.28																								
2	0.23																								
3	0.20																								
4	0.18																								
5	0.16																								
6	0.15																								
7	0.14																								
8	0.13																								
9	0.12																								
3	64	0.962	 <table><caption>Training Loss Data for 3 hidden layers, 64 neurons</caption><thead><tr><th>Epoch</th><th>Training Loss</th></tr></thead><tbody><tr><td>0</td><td>0.55</td></tr><tr><td>1</td><td>0.25</td></tr><tr><td>2</td><td>0.19</td></tr><tr><td>3</td><td>0.16</td></tr><tr><td>4</td><td>0.14</td></tr><tr><td>5</td><td>0.13</td></tr><tr><td>6</td><td>0.12</td></tr><tr><td>7</td><td>0.11</td></tr><tr><td>8</td><td>0.10</td></tr><tr><td>9</td><td>0.09</td></tr></tbody></table>	Epoch	Training Loss	0	0.55	1	0.25	2	0.19	3	0.16	4	0.14	5	0.13	6	0.12	7	0.11	8	0.10	9	0.09
Epoch	Training Loss																								
0	0.55																								
1	0.25																								
2	0.19																								
3	0.16																								
4	0.14																								
5	0.13																								
6	0.12																								
7	0.11																								
8	0.10																								
9	0.09																								

Dari eksperimen yang telah dilakukan, dapat disimpulkan bahwa jumlah neuron serta kedalaman dari neural network (jumlah hidden layer) mempengaruhi performa model. Model pada eksperimen ke-6 memiliki akurasi tertinggi yaitu 0.962 dengan 3 hidden layer dan 64 neuron.

2.2.2. Pengaruh fungsi aktivasi

Variabel Kontrol	
Input Size	784
Hidden Layer Neuron	[64]
Output Neuron	10
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

Fungsi aktivasi	Akurasi	Grafik Loss	Distribusi Bobot	Distribusi Gradien
Linear	0.917			
ReLU	0.945			
Sigmoid	0.908			



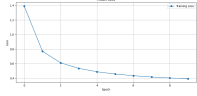
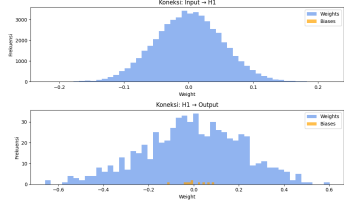
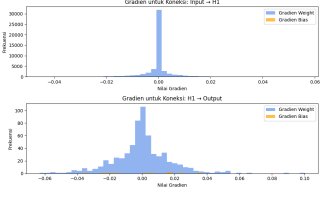
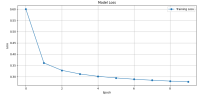
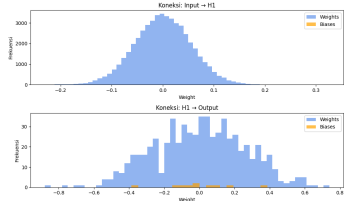
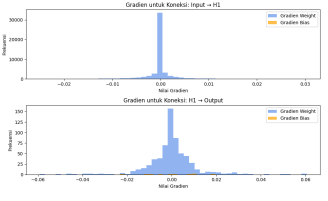
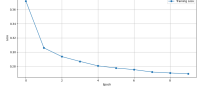
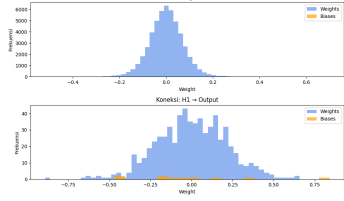
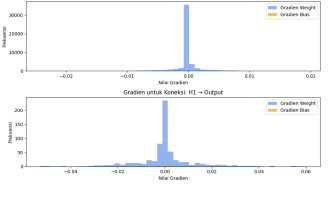
Berdasarkan hasil pengujian, fungsi aktivasi Leaky ReLU memberikan akurasi tertinggi (0.947), diikuti oleh ReLU (0.945), dan tanh (0.938). ReLU dan tanh menunjukkan performa lebih baik karena mampu mengatasi vanishing gradient dibandingkan sigmoid. Grafik loss juga menunjukkan bahwa ReLU dan tanh lebih cepat mencapai konvergensi dibandingkan linear dan sigmoid, yang cenderung lebih lambat dalam pembaruan bobot.

Distribusi bobot dan gradien menunjukkan bahwa ReLU dan tanh memiliki penyebaran yang lebih stabil dibandingkan sigmoid, yang sering mengalami saturasi gradien. Fungsi aktivasi linear kurang optimal karena tidak memperkenalkan non-linearitas dalam model.

2.2.3. Pengaruh *learning rate*

Variabel Kontrol	
Input Size	784
Hidden Layer Neuron	[64]
Output Neuron	10
Activations	ReLU
Loss Function	Categorical Cross Entropy

Weight Initializer	HE
Learning Rate	0.01

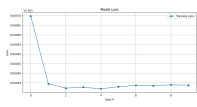
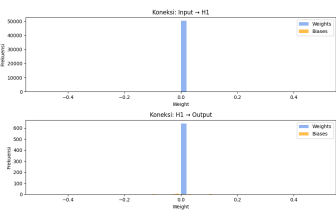
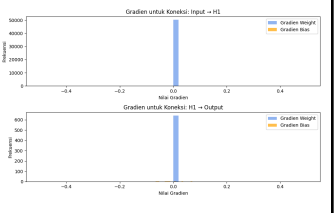
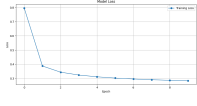
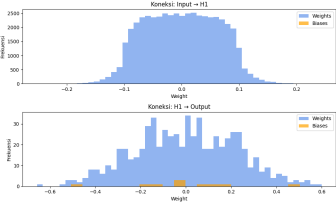
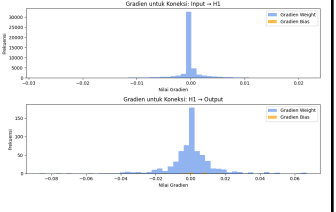
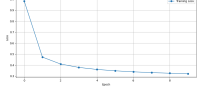
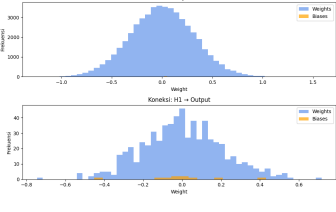
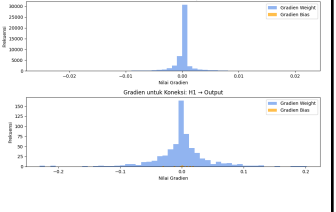
Learning Rate	Akurasi	Grafik Loss	Distribusi Bobot	Distribusi Gradien
0.001	0.892			
0.01	0.919			
0.1	0.910			

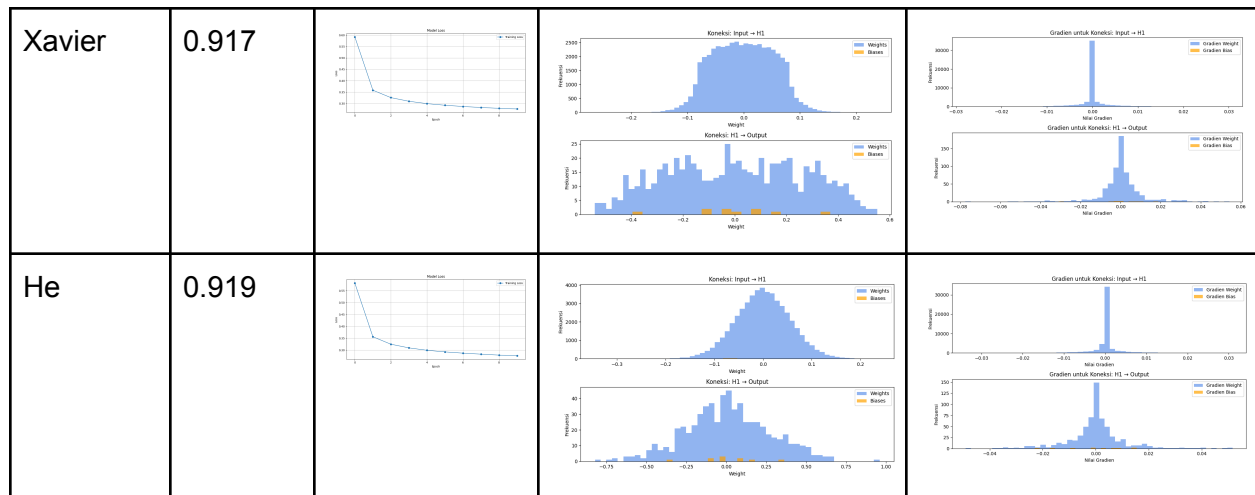
Berdasarkan hasil eksperimen terhadap pengaruh *learning rate*, terlihat bahwa peningkatan *learning rate* cenderung meningkatkan akurasi model. Pada *learning rate* 0.001, model mencapai akurasi 0.892, sementara dengan *learning rate* 0.01 dan 0.1, akurasi meningkat menjadi 0.919 dan 0.910. Grafik *loss* menunjukkan bahwa semakin besar *learning rate*, semakin cepat model mencapai konvergensi. Namun, jika *learning rate* terlalu besar, ada risiko model menjadi tidak stabil atau melewati titik optimal.

Dari distribusi bobot dan gradien, dapat diamati bahwa dengan *learning rate* yang lebih kecil (0.001), distribusi bobot lebih terpusat, sementara pada *learning rate* yang lebih tinggi, distribusi lebih tersebar. Hal ini menunjukkan bahwa *learning rate* yang lebih besar memungkinkan bobot berubah lebih signifikan di setiap iterasi, yang dapat mempercepat pelatihan tetapi juga meningkatkan potensi osilasi. Oleh karena itu, pemilihan *learning rate* harus mempertimbangkan keseimbangan antara kecepatan konvergensi dan stabilitas model.

2.2.4. Pengaruh inisialisasi bobot

Variabel Kontrol	
Input Size	784
Hidden Layer Neuron	[64]
Output Neuron	10
Fungsi Aktivasi	ReLU
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

Inisialisa si Bobot	Akurasi	Grafik Loss	Distribusi Bobot	Distribusi Gradien
Zero	0.114			
Uniform	0.914			
Normal	0.908			

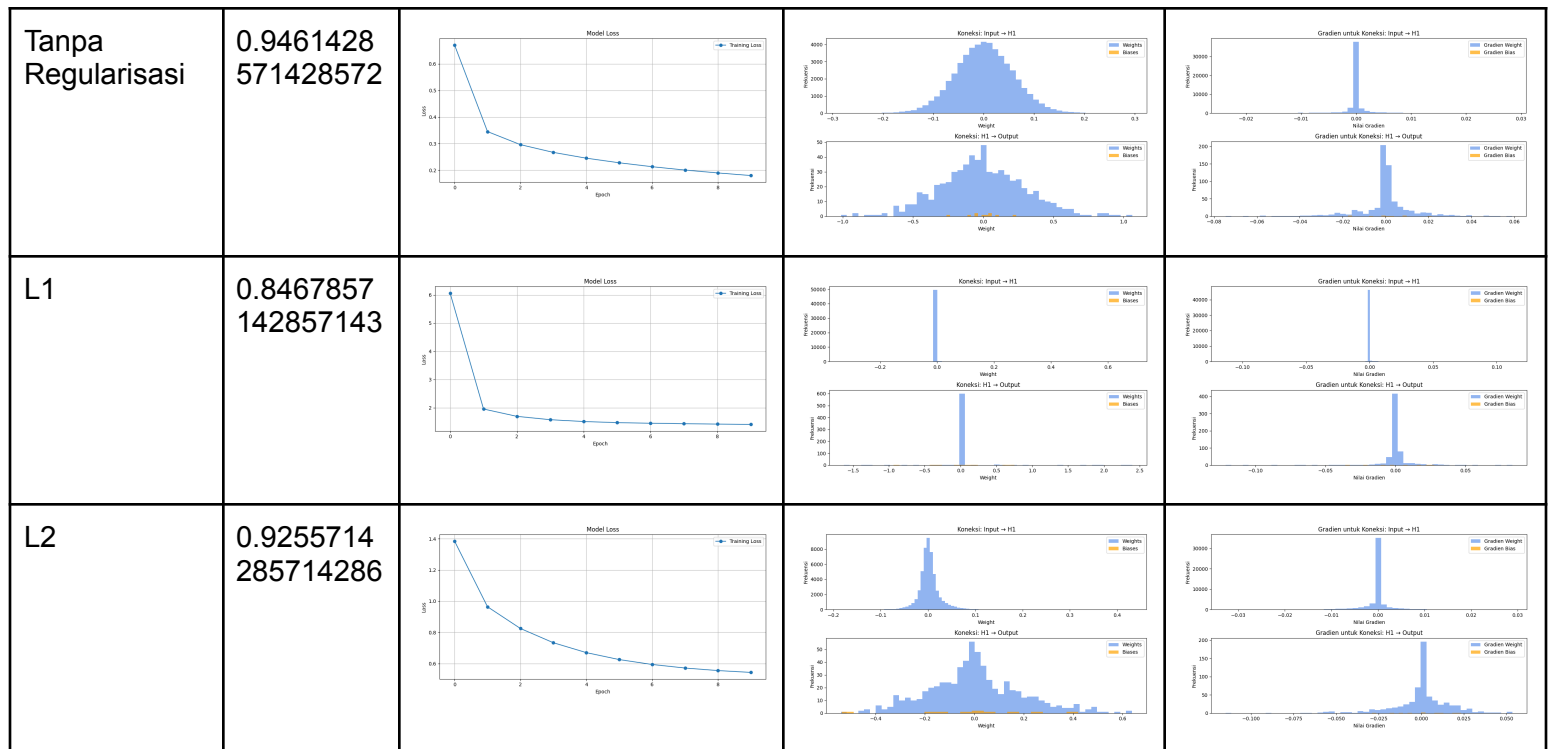


Hasil eksperimen menunjukkan bahwa pemilihan metode inisialisasi bobot sangat mempengaruhi performa model. Inisialisasi bobot nol menghasilkan akurasi sangat rendah (0.114) karena menyebabkan masalah vanishing gradient dan model tidak belajar. Sebaliknya, metode Uniform, Normal, dan Xavier memberikan akurasi tinggi di atas 0.90, dengan distribusi bobot yang lebih bervariasi dan gradien yang stabil. He mencapai akurasi tertinggi (0.919), menunjukkan bahwa metode ini lebih efektif dalam mempercepat konvergensi dan meningkatkan generalisasi model.

2.2.5. Pengaruh regularisasi

Variabel Kontrol	
Input Size	784
Hidden Layer Neuron	[64]
Output Neuron	10
Fungsi Aktivasi	ReLU [Hidden Layer], Softmax [Output Layer]
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

Regularisasi	Akurasi	Grafik Loss	Distribusi Bobot	Distribusi Gradien
--------------	---------	-------------	------------------	--------------------



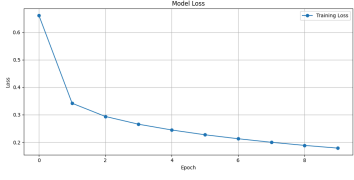
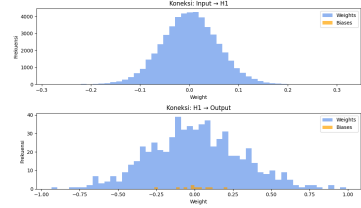
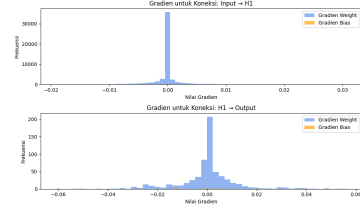
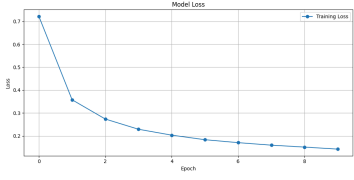
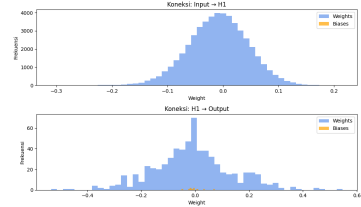
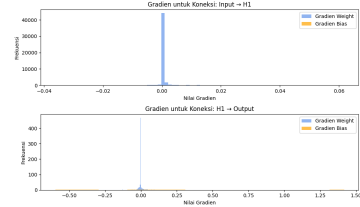
L1 dan L2 regularisasi adalah teknik untuk mencegah overfitting dalam *training* model dengan menambahkan penalti pada *loss function*. L1 regularisasi (Lasso) menambahkan penalti berupa jumlah nilai absolut dari total bobot, mendorong beberapa bobot menjadi nol sehingga menghasilkan model yang lebih sederhana. L2 regularisasi (Ridge) menambahkan penalti berupa jumlah kuadrat dari total bobot, sehingga tidak membuat bobot menjadi nol tapi mengecilkan dan membuat distribusinya menjadi rata.

Hasil perbandingan menunjukkan bahwa model tanpa regularisasi mencapai akurasi tertinggi (94,61%) dengan distribusi bobot yang menyebar lebar mengikuti pola normal. L1 regularization menghasilkan akurasi terendah (84,68%) dan menunjukkan ciri berupa distribusi bobot yang sangat terkonsentrasi pada nilai nol (*sparse model*), sesuai karakteristik Lasso yang mendorong banyak bobot menjadi tepat nol (*fitur tidak dipakai*). Sementara itu, L2 regularization mencapai akurasi yang cukup baik (92,56%) dengan distribusi bobot yang lebih terkontrol dan terkonsentrasi di sekitar nol namun tidak *se-sparse* L1. Grafik loss menunjukkan model tanpa regularisasi turun paling cepat, sementara L1 berada pada nilai yang relatif tinggi. Meskipun model tanpa regularisasi memiliki akurasi training tertinggi, L2 regularization memberikan *balance* antara akurasi dan kompleksitas model, sehingga kemungkinan akan lebih baik dalam generalisasi pada data baru. Parameter lambda pada L1 mungkin terlalu ketat sehingga menyebabkan model terlalu sederhana dan kehilangan fitur penting.

2.2.6. Pengaruh RMSNorm

Variabel Kontrol

Input Size	784
Hidden Layer Neuron	[64]
Output Neuron	10
Fungsi Aktivasi	ReLU
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

RMSNorm	Akurasi	Grafik Loss	Distribusi Bobot	Distribusi Gradien
Tanpa RMSNorm	0.9467857 142857142			
Dengan RMSNorm	0.9535714 285714286			

Hasil percobaan beberapa perbedaan antara model dengan dan tanpa RMSNorm. Pada distribusi bobot, model tanpa RMSNorm memiliki kurva distribusi normal yang lebih lebar di lapisan output dibandingkan dengan model dengan RMSNorm. Hal ini sesuai dengan cara kerja RMSNorm yang membagi nilai dengan akar kuadrat rata-rata, sehingga bobot cenderung terkonsentrasi mendekati nol. Perbedaan juga terlihat pada distribusi gradien, di mana model dengan RMSNorm menunjukkan rentang distribusi yang jauh lebih luas (-1.5 hingga 2.0) dibandingkan model tanpa RMSNorm yang hanya memiliki rentang lebih sempit (-0.100 hingga 0.075). Hal ini mungkin disebabkan oleh Interdependensi Fitur. RMSNorm membuat gradien suatu fitur bergantung pada fitur lain, sehingga menciptakan feedback loop yang memperlebar distribusi

2.2.7. Perbandingan dengan *library sklearn*

Variabel Kontrol	
Input Size	784
Output Neuron	10
Fungsi Aktivasi	ReLU
Loss Function	Categorical Cross Entropy
Weight Initializer	HE
Learning Rate	0.01

Jumlah Hidden Layer	Jumlah Neuron per Layer	SKLearn	From Scratch
2	32	0.9545714285714286	0.9522857142857143
2	64	0.9647142857142857	0.9601428571428572
1	64	0.9530714285714286	0.9468571428571428
3	64	0.9666428571428571	0.9637857142857142

Implementasi FFNN from scratch menunjukkan hasil yang cukup baik karena akurasinya mendekati Sklearn di semua pengujian. Meskipun belum ada yang melebihi Sklearn akurasinya, selisihnya cukup kecil. Hal ini menandakan bahwa implementasi sudah cukup akurat.

Bab 3

Kesimpulan dan Saran

Modul FFNN yang mampu menerima berbagai konfigurasi arsitektur, fungsi aktivasi, inisialisasi bobot, dan teknik regularisasi. Hasil pengujian pada dataset MNIST mengungkapkan beberapa faktor kunci yang mempengaruhi performa model, seperti jumlah hidden layer, fungsi aktivasi, learning rate, metode inisialisasi bobot, dan teknik regularisasi. Dari berbagai eksperimen, ditemukan bahwa arsitektur dengan 3 hidden layer dan 64 neuron menghasilkan akurasi tertinggi (0.962), sedangkan fungsi aktivasi Leaky ReLU menunjukkan performa paling baik dengan akurasi 0.947. Metode inisialisasi bobot He terbukti paling efektif, mencapai akurasi 0.919, dan learning rate 0.01 menunjukkan keseimbangan optimal antara kecepatan konvergensi dan stabilitas model. Selain itu, L2 regularization memberikan hasil yang menjanjikan dalam mencegah overfitting, dengan akurasi 0.926 dan distribusi bobot yang terkontrol. Implementasi from scratch ini tidak hanya berhasil mendekati performa library sklearn, tetapi juga memberikan wawasan yang sangat berharga tentang kompleksitas dan mekanisme internal dari neural network. Perbedaan akurasi yang minimal antara implementasi manual dan library sklearn menunjukkan keberhasilan dalam memahami dan mengimplementasikan konsep-konsep fundamental dalam deep learning, mulai dari forward propagation, backward propagation, hingga weight update.

Untuk pengembangan lebih lanjut, disarankan untuk mengeksplorasi beberapa fitur tambahan dan optimasi. Pertama, implementasi automatic differentiation dapat mempermudah komputasi gradien dan meningkatkan fleksibilitas model. Kedua, penambahan fungsi aktivasi lanjutan seperti GELU atau Swish, serta metode inisialisasi bobot seperti Xavier dan He yang lebih canggih, dapat memberikan wawasan lebih dalam tentang karakteristik jaringan saraf. Selanjutnya, disarankan untuk melakukan eksperimen lebih mendalam dengan teknik regularisasi, khususnya dengan mengoptimasi parameter lambda pada L1 dan L2. Pengujian pada dataset yang lebih beragam dan kompleks akan membantu memvalidasi generalisasi model. Selain itu, implementasi teknik normalisasi lanjutan seperti Layer Normalization atau Batch Normalization dapat memberikan perspektif baru tentang stabilisasi proses pelatihan. Kemudian juga eksplorasi lebih lanjut terhadap hyperparameter tuning, misalnya dengan menggunakan grid search atau algoritma optimasi seperti Bayesian optimization, dapat membantu menemukan konfigurasi model yang paling optimal.

Bab 4

Pembagian Tugas

NIM	Nama	Tugas
13522003	Shafiq Irvansyah	<ul style="list-style-type: none">- Implementasi kelas FFNN- Testing- Laporan- Testing- Save, load, dan visualisasi
13522027	Muhammad Al Thariq Fairuz	<ul style="list-style-type: none">- Implementasi kelas FFNN- Implementasi fungsi aktivasi, regularisasi, weight initializer, dan loss- Laporan
13522067	Randy Verdian	<ul style="list-style-type: none">- Implementasi kelas FFNN- Testing- Laporan

Referensi

Tim Pengajar IF3270. 2025. "Artificial Neural Network (ANN)". Institut Teknologi Bandung. [Artificial Neural Network \(ANN\)](#).

GeeksForGeeks. 2025. "What is a Neural Network?". GeeksForGeeks. [What is a Neural Network?](#)

Rhome. 2019. "Automatic Differentiation". Medium. [Automatic Differentiation](#)

GeeksForGeeks. 2025. "Activation functions in Neural Networks". GeeksForGeeks. [Activation functions in Neural Networks](#)

Benistant, Francis. 2024. "Deep Dive into Deep Learning: Layers, RMSNorm, and Batch Normalization". Medium. [Deep Dive into Deep Learning: Layers, RMSNorm, and Batch Normalization](#)