

# **LAPORAN TUGAS BESAR 1 IF3270 PEMBELAJARAN MESIN**

## ***FEEDFORWARD NEURAL NETWORK***

Diajukan sebagai Pemenuhan Tugas Besar 1



Oleh:

Kelompok 29

Erdianti Wiga Putri Andini	13522053
Shazya Audrea Taufik	13522063
Zahira Dina Amalia	13522085

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2024**

## Daftar Isi

Daftar Isi.....	2
A. Deskripsi Persoalan.....	3
B. Pembahasan.....	3
1. Penjelasan Implementasi.....	3
1. 1. Deskripsi kelas beserta deskripsi atribut dan methodnya.....	3
1. 2. Penjelasan forward propagation.....	8
1. 3. Penjelasan backward propagation dan weight update.....	9
2. Hasil Pengujian.....	11
2. 1. Pengaruh Depth dan Width.....	11
2. 2. Pengaruh Fungsi Aktivasi.....	13
2. 3. Pengaruh Learning Rate.....	18
2. 4. Pengaruh Inisialisasi Bobot.....	21
2. 5. Pengaruh Normalisasi RMSNorm.....	24
2. 6. Perbandingan dengan Library sklearn.....	27
C. Kesimpulan dan Saran.....	28
3.1. Kesimpulan.....	28
3.2. Saran.....	29
D. Pembagian Tugas tiap Anggota Kelompok.....	30
E. Referensi.....	30

## A. Deskripsi Persoalan

Persoalan utama dalam spesifikasi ini adalah bagaimana mengimplementasikan Feedforward Neural Network (FFNN) dari awal, termasuk berbagai elemen penting seperti struktur jaringan, fungsi aktivasi, fungsi loss, serta mekanisme perhitungan gradien dan update bobot menggunakan metode *gradient descent*. Selain itu, persoalan lain yang dihadapi adalah pengaruh berbagai hyperparameter terhadap performa model, seperti kedalaman (*depth*) jaringan, jumlah neuron per layer (*width*), fungsi aktivasi, *learning rate*, serta metode inisialisasi bobot. Pada tugas ini juga diwajibkan untuk menguji dan membandingkan performa model yang diimplementasikan dengan menggunakan library lain, serta menganalisis distribusi bobot dan gradien bobot selama proses *training*. Selain itu, eksperimen tambahan seperti regularisasi dan normalisasi juga menjadi faktor penting yang perlu dipertimbangkan untuk meningkatkan kinerja model.

## B. Pembahasan

### 1. Penjelasan Implementasi

#### 1. 1. Deskripsi kelas beserta deskripsi atribut dan methodnya

##### 1. Kelas Activation

Kelas statik berisi fungsi-fungsi aktivasi dan turunan/derivatifnya.

Berikut adalah deskripsi *method*-nya:

Nama <i>Method</i>	Kegunaan
<code>linear(x)</code>	Fungsi identitas: output = input.
<code>linear_derivative(x)</code>	Turunan dari fungsi linear (konstan 1).
<code>relu(x)</code>	Fungsi ReLU: Mengubah nilai negatif menjadi nol.
<code>relu_derivative(x)</code>	Turunan ReLU: 1 jika $x > 0$ , 0 jika tidak.
<code>sigmoid(x)</code>	Mengubah input ke rentang 0-1 secara smooth.
<code>sigmoid_derivative(x)</code>	Turunan sigmoid.

<code>tanh(x)</code>	Fungsi tanh: output antara -1 dan 1.
<code>tanh_derivative(x)</code>	Turunan tanh.
<code>softmax(x)</code>	Fungsi aktivasi untuk klasifikasi <i>multiclass</i> . Output berbentuk distribusi probabilitas.
<code>softmax_derivative(x)</code>	Dikosongkan. Turunan softmax kompleks dan tergantung loss.
<code>leaky_relu(x, alpha)</code>	ReLU yang memberikan slope kecil di area negatif.
<code>leaky_relu_derivative(x, alpha)</code>	Turunan leaky relu.
<code>elu(x, alpha)</code>	Ekspensial Linear Unit.
<code>elu_derivative(x, alpha)</code>	Turunan elu.

## 2. Kelas *Loss*

Kelas statik untuk fungsi-fungsi *loss* beserta turunannya.

Berikut adalah deskripsi *method*-nya:

Nama <i>Method</i>	Kegunaan
<code>mse(y_true, y_pred, model, l1_lambda, l2_lambda)</code>	Mean Squared Error untuk regresi.
<code>mse_derivative(y_true, y_pred)</code>	Turunan MSE.
<code>binary_cross_entropy(y_true, y_pred, model, l1_lambda, l2_lambda)</code>	Loss untuk klasifikasi biner.
<code>binary_cross_entropy_derivative(y_true, y_pred)</code>	Turunan dari binary cross-entropy.
<code>categorical_cross_entropy(y_true, y_pred, model, l1_lambda, l2_lambda)</code>	Untuk klasifikasi multi kelas (one-hot).
<code>categorical_cross_entropy_derivative(y_true, y_pred)</code>	Turunan dari categorical cross-entropy.
<code>calculate_regularization(model, l1_lambda, l2_lambda)</code>	Untuk melakukan perhitungan regularisasi l1 dan l2

### 3. Kelas WeightInitializer

Kelas statik untuk inisialisasi bobot awal pada layer.

Berikut adalah deskripsi *method*-nya:

Nama <i>Method</i>	Kegunaan
<code>zero_initialization(shape)</code>	Inisialisasi dengan semua nol.
<code>random_uniform(shape, lower_bound=-0.5, upper_bound=0.5, seed=None)</code>	Nilai acak dari distribusi uniform.
<code>random_normal(shape, mean=0.0, variance=0.1, seed=None)</code>	Nilai acak dari distribusi normal.
<code>xavier(shape, fan_in, fan_out, seed=None)</code>	Inisialisasi yang cocok untuk aktivasi sigmoid/tanh. Menjaga variansi tetap stabil saat melalui jaringan dengan aktivasi sigmoid atau tanh.
<code>he(shape, fan_in, seed=None)</code>	Inisialisasi cocok untuk ReLU. Menjaga distribusi variansi tetap saat melewati ReLU, yang mematikan sebagian neuron

### 4. Kelas Layer

Kelas yang mewakili 1 lapisan dalam jaringan saraf.

Berikut adalah deskripsi atributnya:

Nama Atribut	Tipe	Kegunaan
<code>input_size</code>	int	Jumlah neuron pada layer sebelumnya (jumlah input yang diterima layer ini).
<code>output_size</code>	int	Jumlah neuron pada layer ini (jumlah output yang dihasilkan).
<code>activation_name</code>	str	Menyimpan nama fungsi aktivasi sebagai string

activation, activation_derivative	Callable	Menyimpan fungsi aktivasi dan turunannya dari kelas Activation.
weights	array	Matriks bobot dari input ke output neuron, ukuran (input_size, output_size).
bias	array	Bias untuk setiap neuron output, ukuran (1, output_size).
weights_gradient	array	Menyimpan gradien dari loss terhadap bobot, untuk proses pembaruan.
bias_gradient	array	Menyimpan gradien dari loss terhadap bias.
input	array	Input yang diberikan ke layer saat forward pass.
linear_output	array	Output dari operasi linear sebelum aktivasi ( $Wx + b$ ).
output	array	Output akhir setelah aktivasi.

Berikut adalah deskripsi *method*-nya:

Nama <i>Method</i>	Kegunaan
<code>__init__(input_size: int, output_size: int, activation: str = 'linear', weight_initializer: str = 'random_normal', weight_init_params: dict = None)</code>	Inisialisasi layer dan bobot-bias.
<code>forward(x)</code>	Melakukan propagasi maju.

<code>backward(output_gradient, learning_rate)</code>	Propagasi balik dan update bobot.
---	-----------------------------------

#### 5. Kelas FeedForwardNN

Kelas yang mewakili keseluruhan jaringan saraf multilayer.

Berikut adalah deskripsi atributnya:

Nama Atribut	Tipe	Kegunaan
<code>layers</code>	<code>List[Layer]</code>	Layer-layer penyusun jaringan.
<code>layer_dimensions</code>	<code>List[int]</code>	Ukuran layer dari input ke output.
<code>activations</code>	<code>List[str]</code>	Fungsi aktivasi tiap layer.
<code>loss_function, loss_derivative</code>	<code>Callable</code>	Fungsi loss dan turunannya.
<code>loss_name</code>	<code>str</code>	Nama loss yang dipakai.
<code>l1_lambda</code>	<code>float</code>	Koefisien regulasi L1.
<code>l2_lambda</code>	<code>float</code>	Koefisien regulasi L2.

Berikut adalah deskripsi *method*-nya:

Nama <i>Method</i>	Kegunaan
<code>__init__(layer_dimensions: List[int], activations: List[str], loss: str = 'mse', weight_initializer: str = 'random_normal', weight_init_params: dict = None, l1_lambda, l2_lambda)</code>	Membuat jaringan berdasarkan struktur dan parameter.
<code>visualize_nn_graph(self, figsize=(12, 6))</code>	Gambar arsitektur jaringan dengan NetworkX.
<code>visualize_model(figsize)</code>	Menampilkan diagram jaringan saraf dengan node, bobot, dan gradien.

<code>forward(self, x)</code>	Propagasi maju ke seluruh layer.
<code>backward(self, y_true, y_pred, learning_rate)</code>	Backpropagation seluruh jaringan.
<code>train(self, X_train, y_train, X_val=None, y_val=None, batch_size=32, learning_rate=0.01, epochs=100, verbose=1)</code>	Latih jaringan dengan SGD. Menyimpan riwayat loss.
<code>predict(self, X)</code>	Prediksi output dari input.
<code>evaluate(self, X, y_true)</code>	Hitung loss pada dataset.
<code>plot_weight_distribution(self, layers_to_plot=None)</code>	Histogram distribusi bobot layer.
<code>plot_gradient_distribution(self, layers_to_plot=None)</code>	Histogram distribusi gradien layer.
<code>save(filename)</code>	Simpan model ke file .pkl.
<code>load(filename)</code>	Muat model dari file .pkl.

## 1. 2. Penjelasan forward propagation

Forward propagation bertujuan untuk mengalirkan input dari satu layer ke layer berikutnya dalam jaringan saraf. Proses ini menghitung output dari setiap neuron di setiap layer berdasarkan input yang diterima, bobot, bias, dan fungsi aktivasi. Hasil akhir digunakan sebagai prediksi model.

```
def forward(self, x):
    self.input = x
    self.linear_output = np.dot(x, self.weights) + self.bias
    self.output = self.activation(self.linear_output)
    return self.output
```

Langkah-langkah:

### 1. Menerima input

Layer menerima input  $x$ , biasanya berupa matriks 2D dengan dimensi  $[\text{batch\_size}, \text{input\_size}]$ . Nilai ini disimpan untuk digunakan nanti pada saat backpropagation.

### 2. Menghitung linear output



Operasi `np.dot(x, self.weights)` menghitung kombinasi linier dari input dan bobot. Hasilnya kemudian ditambah bias untuk menyesuaikan secara fleksibel output dari neuron. Ini menghasilkan nilai  $z$ , yaitu output dari fungsi linier sebelum fungsi aktivasi diterapkan.

3. Menerapkan fungsi aktivasi

Fungsi aktivasi diterapkan terhadap  $z$ . Fungsi ini menambahkan non-linearitas yang memungkinkan jaringan saraf mempelajari hubungan yang kompleks. Output dari aktivasi ini disimpan sebagai `self.output`.

4. Mengembalikan nilai output

Output ini akan menjadi input untuk layer berikutnya, atau menjadi output akhir jika ini adalah layer terakhir.

### 1. 3. Penjelasan backward propagation dan weight update

Backward propagation menghitung gradien (turunan) dari fungsi kerugian terhadap setiap parameter (bobot dan bias), lalu menggunakan informasi ini untuk menyesuaikan bobot dan bias agar jaringan menjadi lebih akurat.

```
def backward(self, output_gradient, learning_rate):
    if self.activation_name == 'softmax':
        batch_size = output_gradient.shape[0]
        linear_gradient = output_gradient
    else:
        activation_gradient =
self.activation_derivative(self.linear_output)
        linear_gradient = output_gradient * activation_gradient

    self.weights_gradient = np.dot(self.input.T, linear_gradient)
    self.bias_gradient = np.sum(linear_gradient, axis=0,
keepdims=True)

    input_gradient = np.dot(linear_gradient, self.weights.T)

    self.weights -= learning_rate * self.weights_gradient
    self.bias -= learning_rate * self.bias_gradient

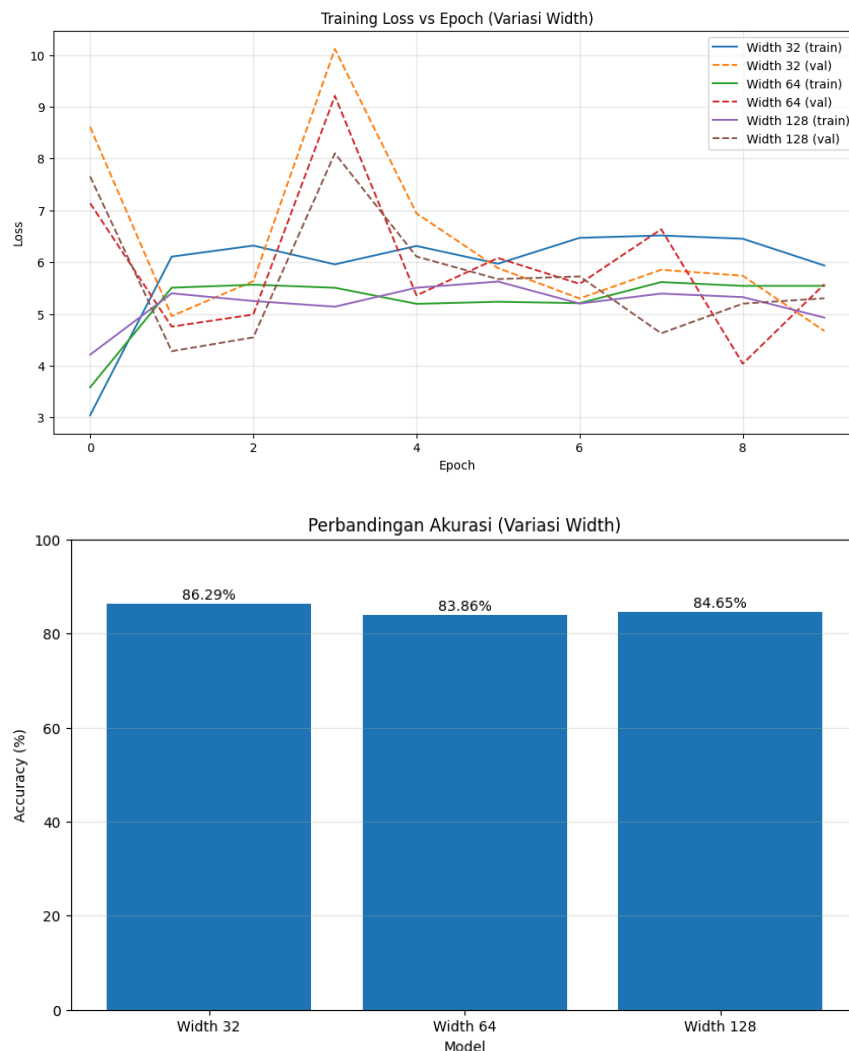
    return input_gradient
```

Langkah-langkah:

1. Menerima gradien dari layer selanjutnya  
Ini adalah turunan dari loss terhadap output layer saat ini. Biasanya berasal dari backward layer di atas atau dari *loss function* (di layer terakhir).
2. Menghitung turunan fungsi aktivasi  
Hitung turunan fungsi aktivasi terhadap input linear  $z$ .
3. Menggabungkan dengan gradien output  
Gradien output dikalikan dengan gradien aktivasi.
4. Hitung gradien terhadap bobot  
Dengan menggunakan aturan rantai, gradien terhadap bobot diperoleh dengan mengalikan input *transpose* dengan gradien linier.
5. Hitung gradien terhadap bias  
Gradien bias dihitung dengan menjumlahkan *linear\_gradient* di seluruh batch.
6. Hitung gradien terhadap input  
Gradien ini akan dikirim ke layer sebelumnya. Diperlukan agar propagasi balik bisa berlanjut ke belakang.
7. Perbarui nilai bobot dan bias  
Setelah gradien dihitung, kita perbarui bobot dan bias menggunakan metode gradient descent agar jaringan belajar dari kesalahan. Bobot dan bias masing-masing dikurangi *learning rate* dikali gradiennya. Learning rate mengontrol seberapa besar langkah perubahan parameter. Jika learning rate terlalu besar, model mudah bisa "loncat" dari solusi optimal. Jika terlalu kecil, maka proses belajar bisa sangat lambat.

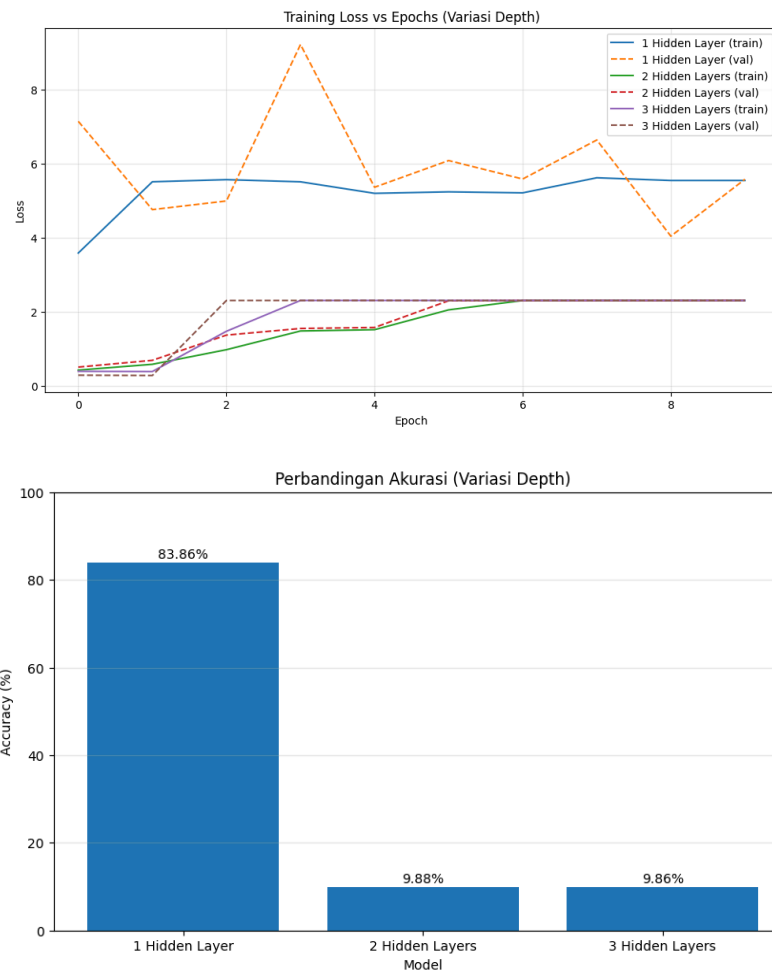
## 2. Hasil Pengujian

### 2.1. Pengaruh Depth dan Width



Berdasarkan grafik *training loss* dan *validation loss*, serta perbandingan akurasi pada eksperimen pengaruh width, terlihat perbedaan antar model. Model dengan **Width 32** memberikan **akurasi tertinggi (86.29%)** dibandingkan dengan model **Width 64 (83.86%)** dan **Width 128 (84.65%)**. Meski jumlah neuron meningkat, ternyata menambah jumlah neuron tidak selalu meningkatkan performa model. Hal ini dapat dilihat dari perbandingan grafik loss di mana model dengan Width 32 memiliki *training loss* dan *validation loss* yang lebih rendah dan lebih stabil dibandingkan dengan model Width 64 dan Width 128. Waktu *training* model dengan Width 32 juga lebih cepat, yaitu 36.54 detik, dibandingkan dengan model lainnya yang

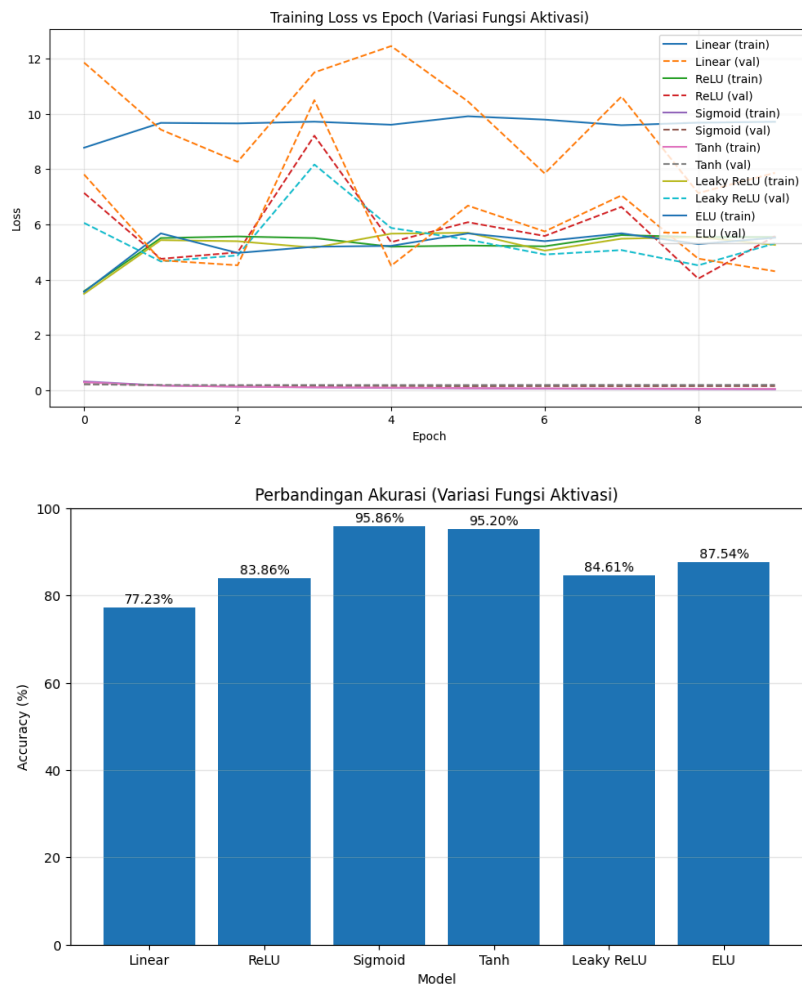
membutuhkan waktu sekitar 45-59 detik. Ini menunjukkan bahwa penambahan lebar jaringan tidak selalu berdampak pada peningkatan performa, bahkan bisa memperburuk stabilitas *training* dan meningkatkan waktu *training*.



Berdasarkan grafik *training loss* dan *validation loss*, serta perbandingan akurasi pada eksperimen pengaruh width, terlihat perbedaan antar model. Model dengan **1 Hidden Layer** menunjukkan **akurasi 83.86%**, yang merupakan nilai terbaik di antara model dengan 2 dan 3 Hidden Layers. Model dengan **2 Hidden Layers** menunjukkan **penurunan akurasi drastis (9.88%)**, sementara model dengan **3 Hidden Layers** juga memiliki **akurasi serupa (9.86%)**. Hal ini menunjukkan kemungkinan adanya masalah *vanishing gradient* atau *overfitting* pada jaringan yang lebih dalam. Grafik *loss* untuk model dengan 2 dan 3 hidden layers menunjukkan fluktuasi yang

sangat besar dan konvergensi yang buruk, yang mengindikasikan bahwa menambah kedalaman jaringan tanpa pengaturan yang tepat dapat menyebabkan kesulitan dalam proses pelatihan. Waktu pelatihan juga meningkat seiring bertambahnya jumlah layer, dari 48.29 detik untuk model dengan 1 hidden layer menjadi 58-59 detik untuk model dengan 2 dan 3 hidden layers.

## 2. 2. Pengaruh Fungsi Aktivasi



Berdasarkan grafik *training loss* dan *validation loss*, serta perbandingan akurasi pada eksperimen pengaruh fungsi aktivasi, dapat diamati perbedaan performa model pada berbagai fungsi aktivasi yang digunakan. Model dengan fungsi aktivasi **Linear** menunjukkan kinerja terendah dengan akurasi **77.23%**. Grafik *loss* untuk model tersebut menunjukkan fluktuasi

yang sangat tinggi, baik pada *training loss* maupun *validation loss*, yang menunjukkan bahwa tidak mampu menangkap kompleksitas data dengan baik dan mengalami kesulitan dalam proses konvergensi.

Fungsi aktivasi **Sigmoid** dan **Tanh** memberikan hasil yang lebih baik dengan masing-masing akurasi **95.86%** dan **95.20%**. Fungsi Sigmoid memiliki output yang terbatas antara 0 dan 1, cocok untuk masalah klasifikasi biner. Sedangkan Tanh memiliki output antara -1 hingga 1, memungkinkan representasi yang lebih baik dari data. Keduanya menunjukkan grafik *loss* yang stabil dan tidak mengalami fluktuasi besar, menunjukkan kemampuan model dalam mencapai konvergensi yang baik.

**ReLU (Rectified Linear Unit)** memperlihatkan peningkatan signifikan dengan akurasi **83.86%**. Grafik *loss* training dan *validation* menunjukkan fluktuasi awal, kemudian mulai stabil di epoch selanjutnya. Karakteristik ReLU yang memotong nilai negatif memberikan kemampuan untuk mengaktivasi neuron secara *sparse*, yang membantu dalam ekstraksi fitur.

**Leaky ReLU** menunjukkan akurasi **84.61%**, sedikit lebih baik dari ReLU standar. Grafik *loss*-nya memperlihatkan kurva yang lebih halus, dengan variasi yang lebih rendah dibandingkan ReLU biasa. Keunggulan Leaky ReLU terletak pada kemampuannya mengatasi masalah "dying ReLU" dengan mengizinkan gradien kecil untuk input negatif.

**ELU (Exponential Linear Unit)** mencapai akurasi **87.54%**, menampilkan performa yang solid. Grafik *loss* ELU menunjukkan karakteristik yang mirip dengan Leaky ReLU, dengan stabilitas yang baik dan kemampuan untuk mengurangi bias aktivasi. Pada hasil run ELU, sistem mendapatkan bahwa terdapat overflow. Ketika overflow terjadi, NumPy menghasilkan peringatan tetapi tetap melanjutkan komputasi. Fungsi `np.where()` masih menghasilkan output numerik yang valid untuk bagian-bagian lain dari data. Gradien untuk parameter lain yang tidak menyebabkan overflow masih dapat diperbarui. Untuk mengatasi hal ini, dapat digunakan parameter  $\alpha$  yang lebih kecil atau learning rate yang lebih kecil.

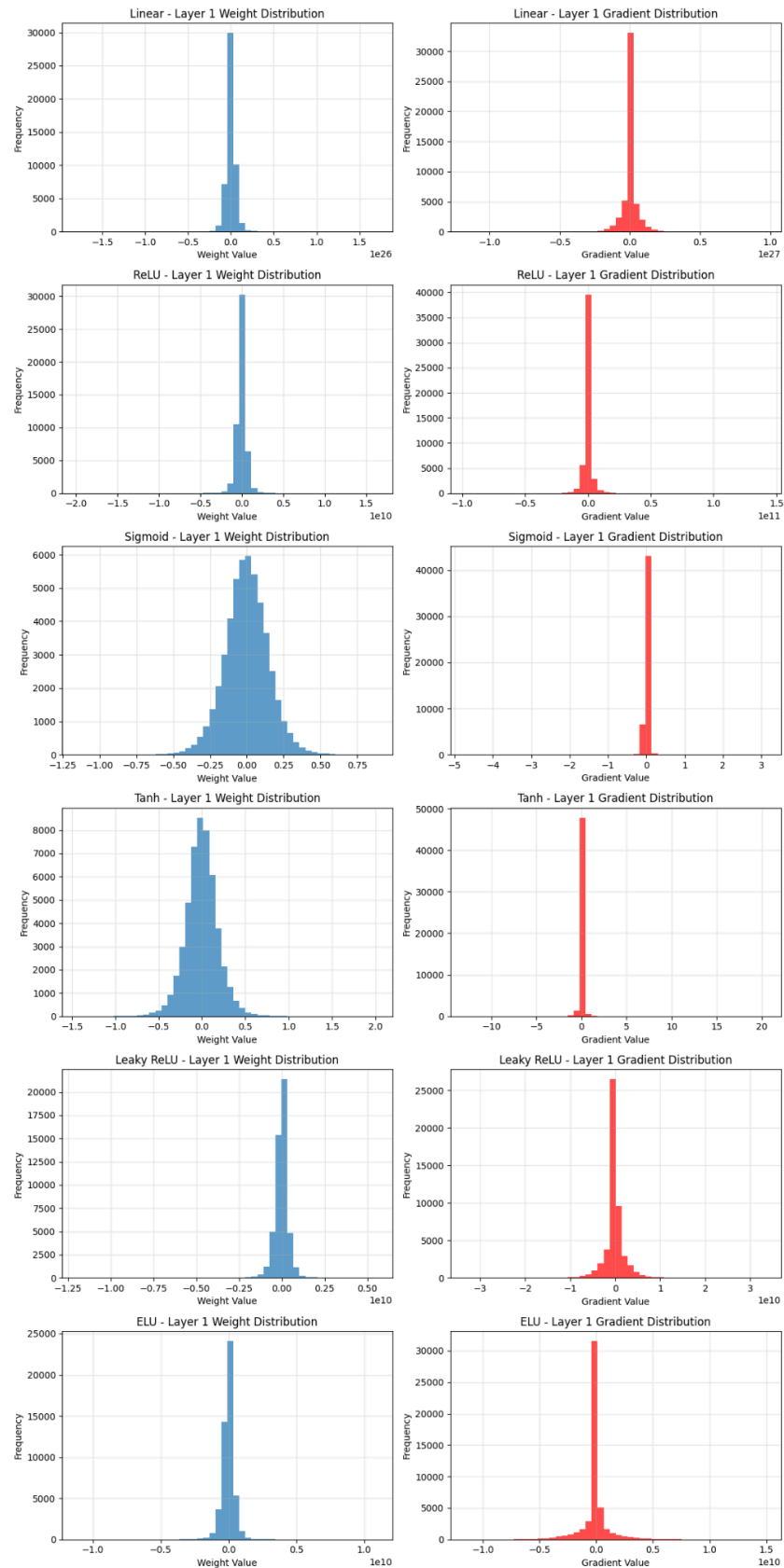
Dalam hal distribusi bobot dan gradien, Linear menunjukkan distribusi *weight* dan *gradient* terlihat sangat sempit dan terpusat di sekitar nol. Hal ini

menjelaskan mengapa model dengan fungsi aktivasi Linear gagal menangkap kompleksitas data, karena setiap neuron memiliki kemampuan transformasi yang sangat terbatas.

Kontras dengan Linear, fungsi Sigmoid dan Tanh menunjukkan distribusi *weight* dan *gradient* yang jauh lebih terstruktur. Distribusi berbentuk lonceng (*bell curve*) yang simetris mengindikasikan kemampuan model untuk melakukan transformasi non-linear dengan terkendali. Pada Sigmoid, distribusi *weight* terbatas antara 0-1, sementara Tanh memiliki rentang -1 hingga 1, memungkinkan representasi fitur yang lebih banyak dan dinamis. Pola distribusi ini sejalan dengan performa akurasi tinggi keduanya, menunjukkan bahwa kendali matematis pada transformasi neuron berkontribusi signifikan terhadap kemampuan generalisasi model.

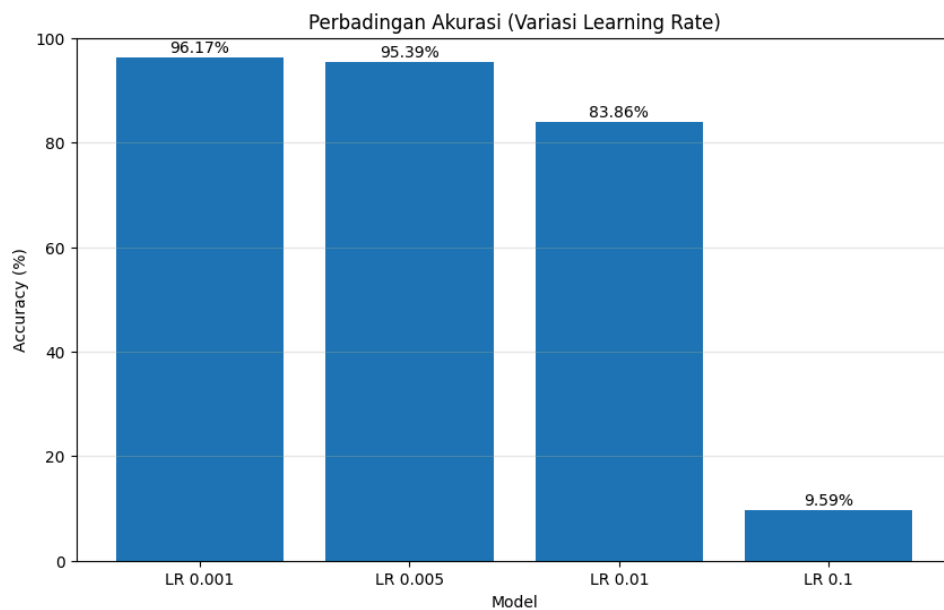
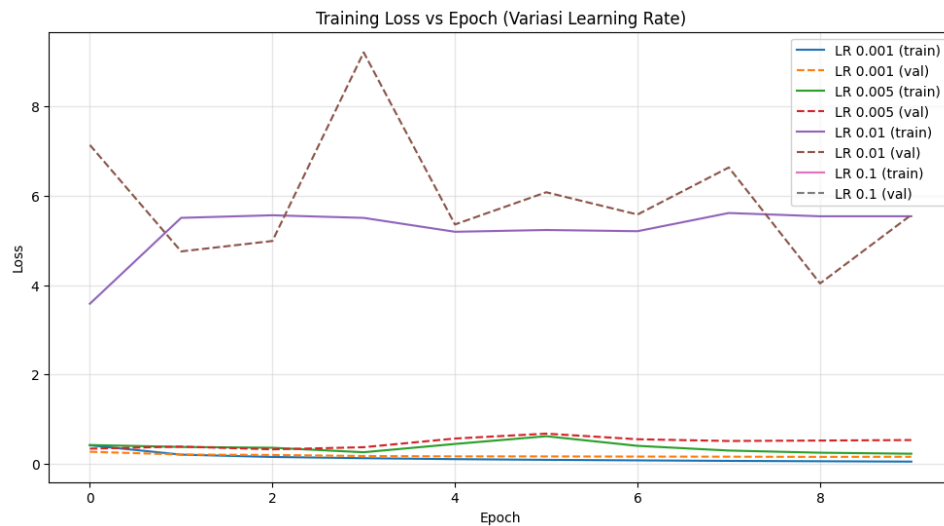
ReLU dan variannya (Leaky ReLU dan ELU) menggunakan pendekatan yang berbeda dalam distribusi *weight* dan *gradient*. ReLU menunjukkan distribusi dengan puncak di sekitar nol namun dengan sebaran yang lebih luas dibandingkan Linear. Keunikan ReLU terletak pada kemampuannya "mematikan" neuron dengan input negatif, menciptakan aktivasi *sparse* yang efektif untuk ekstraksi fitur. Leaky ReLU dan ELU mengembangkan konsep ini lebih lanjut, dengan memberikan jalur gradien kecil pada input negatif, yang terlihat dari distribusi gradien yang lebih halus dan kompleks. Distribusi *weight* dan *gradient* pada Leaky ReLU dan ELU memperlihatkan keunggulan algoritma inisialisasi bobot modern. Kedua fungsi aktivasi ini tidak hanya mencegah masalah *vanishing gradient*, tetapi juga memungkinkan model untuk belajar representasi yang lebih adaptif.

Secara keseluruhan, fungsi aktivasi Sigmoid dan Tanh menunjukkan performa terbaik dengan akurasi masing-masing 95.86% dan 95.20%, didukung oleh grafik loss yang stabil dan distribusi *weight-gradient* yang paling teratur. Kedua fungsi ini secara signifikan unggul dibandingkan fungsi Linear yang hanya mencapai 77.23%, sementara ReLU dan variannya (Leaky ReLU dan ELU) dengan akurasi 83.86%, 84.61%, dan 87.54% menunjukkan peningkatan kemampuan dalam menangkap kompleksitas data melalui aktivasi *sparse* dan penanganan gradien yang lebih canggih.





## 2. 3. Pengaruh Learning Rate



Berdasarkan grafik *training loss* dan *validation loss*, serta perbandingan akurasi pada eksperimen pengaruh *learning rate*, dapat dilihat dengan jelas perbedaan performa model pada berbagai konfigurasi *learning rate*.

Model dengan **learning rate 0.001** menunjukkan kinerja terbaik, terlihat dari akurasi yang tertinggi pada perbandingan tersebut yaitu **96.17%**. Grafik *loss* untuk model ini menunjukkan penurunan yang stabil sepanjang proses *training*, baik pada *training loss* maupun *validation loss*. Hal ini mengindikasikan konvergensi yang efektif, dimana model dapat

menyesuaikan bobot dengan hati-hati tanpa mengalami fluktuasi yang signifikan.

Sementara itu, model dengan **learning rate 0.005** juga menunjukkan performa yang sangat baik dengan akurasi **95.39%**. Meskipun ada sedikit fluktuasi dalam *training loss* dan *validation loss*, model ini masih berhasil mencapai konvergensi yang cukup baik. Konvergensi ini mungkin sedikit lebih cepat daripada learning rate 0.001, tetapi masih dalam batas yang dapat diterima.

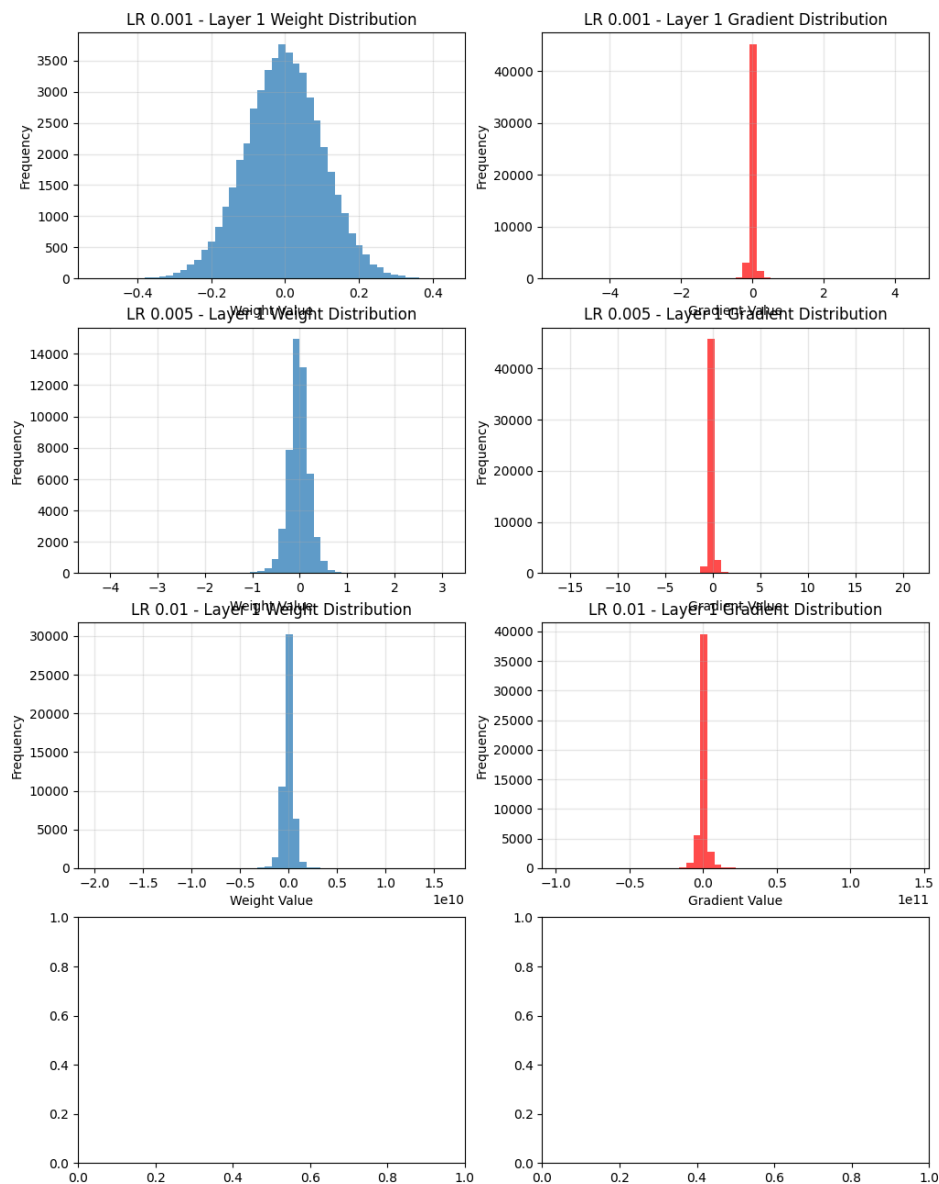
Sebaliknya, model dengan **learning rate 0.01** menunjukkan penurunan performa yang signifikan dengan akurasi hanya sebesar **83.86%**. Pada grafik *loss*, terlihat bahwa *training loss* dan *validation loss* mengalami fluktuasi yang cukup besar. Hal ini menunjukkan bahwa *learning rate* yang terlalu tinggi dapat menyebabkan "lompatan" berlebihan dalam ruang parameter, mengakibatkan model tidak dapat mencapai titik optimal dan kesulitan dalam konvergensi.

Dari segi distribusi bobot dan gradien, model dengan learning rate 0.001 menunjukkan distribusi bobot yang lebih terkendali dan gradien yang relatif kecil, mendukung proses *learning* yang lebih stabil. Sebaliknya, pada learning rate 0.01, distribusi bobot dan gradien menunjukkan nilai yang lebih ekstrem, mengindikasikan bahwa model mungkin mengalami kesulitan dalam menyesuaikan bobot dengan cara yang optimal.

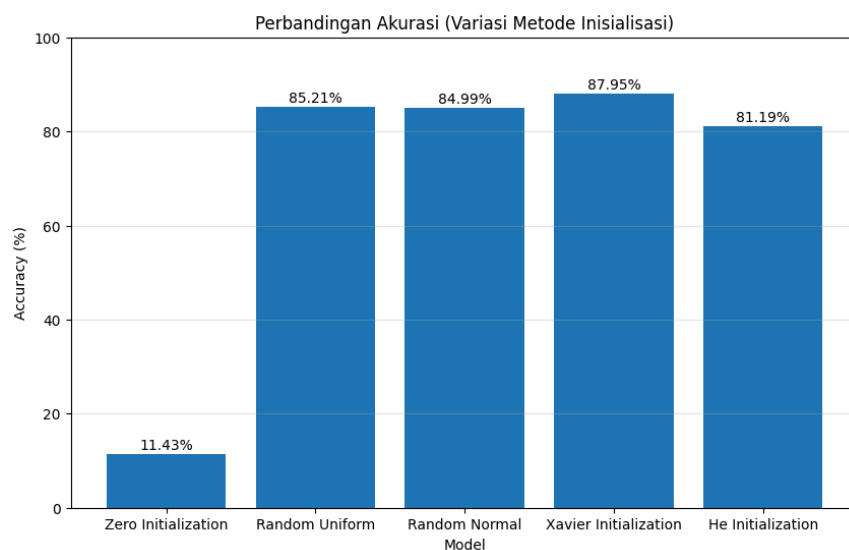
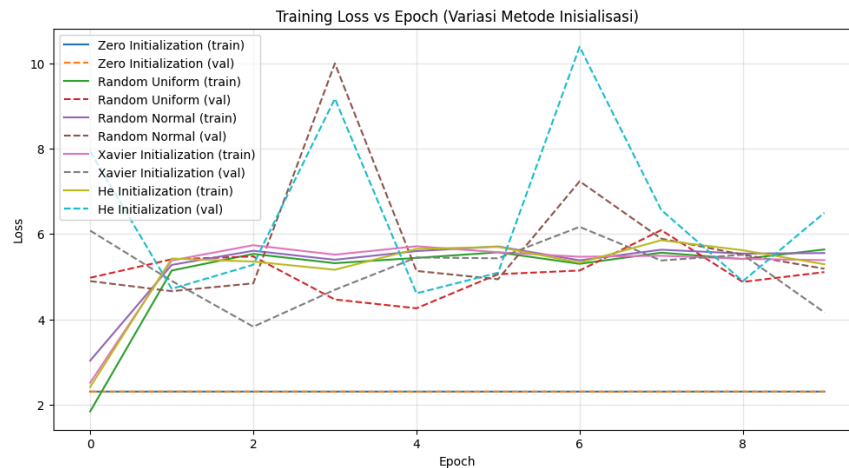
Secara keseluruhan, *learning rate* yang lebih rendah (0.001) memberikan hasil yang lebih stabil dan akurat, sedangkan *learning rate* yang lebih tinggi (0.01) dapat mengganggu proses pembelajaran dan menurunkan performa model.

```
Model dengan learning rate 0.1
d:\ITB\Tugas ML\Tubes 1\Tubes1-IF3270-Kelompok29\src\ffnn_imp.py:45: RuntimeWarning: overflow encountered in subtract
  exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
d:\ITB\Tugas ML\Tubes 1\Tubes1-IF3270-Kelompok29\src\ffnn_imp.py:45: RuntimeWarning: invalid value encountered in subtract
  exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
Epoch 1/10 - Train Loss: nan - Val Loss: nan
Epoch 2/10 - Train Loss: nan - Val Loss: nan
Epoch 3/10 - Train Loss: nan - Val Loss: nan
Epoch 4/10 - Train Loss: nan - Val Loss: nan
Epoch 5/10 - Train Loss: nan - Val Loss: nan
Epoch 6/10 - Train Loss: nan - Val Loss: nan
Epoch 7/10 - Train Loss: nan - Val Loss: nan
Epoch 8/10 - Train Loss: nan - Val Loss: nan
Epoch 9/10 - Train Loss: nan - Val Loss: nan
Epoch 10/10 - Train Loss: nan - Val Loss: nan
Akurasi: 9.59%, Waktu latih: 15.80 detik
```

Selain ketiga learning rate tersebut, kami juga menganalisis model dengan menggunakan learning rate yang cukup besar dibandingkan ketiga learning rate sebelumnya, yaitu 0.1. Pelatihan dengan learning rate besar, dapat menyebabkan overflow. Ketika learning rate terlalu besar, nilai dalam matriks  $x$  bisa menjadi sangat besar. Meskipun ada pengurangan dengan nilai maksimum (`np.max(x, axis=1, keepdims=True)`), jika gradien atau nilai aktivasi sangat besar karena learning rate tinggi, hasilnya tetap bisa melebihi batas numerik floating-point



## 2. 4. Pengaruh Inisialisasi Bobot



Berdasarkan grafik training loss dan validation loss, serta perbandingan akurasi:

- **Zero Initialization** memberikan hasil yang sangat buruk dengan akurasi hanya 11.43%. Hal ini disebabkan karena semua bobot awal bernilai nol, sehingga neuron-neuron dalam layer memiliki output dan gradien yang identik, menyebabkan model tidak bisa belajar secara efektif (masalah simetri).
- **Random Uniform dan Random Normal** menunjukkan akurasi yang cukup baik, masing-masing 85.21% dan 84.99%. Namun, grafik loss pada keduanya tampak fluktuatif dan tidak stabil, terutama pada

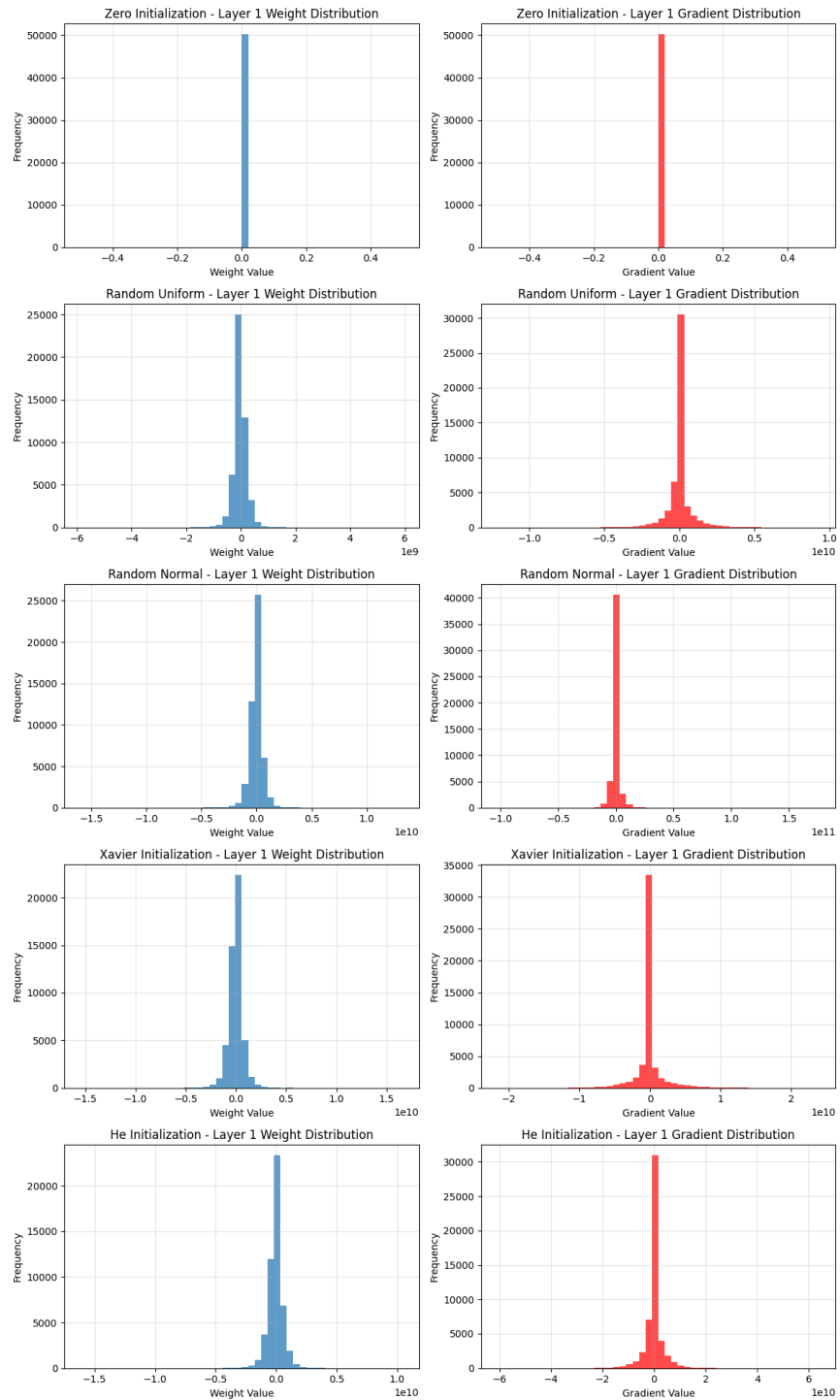
validation loss. Hal ini menunjukkan bahwa model bisa belajar, tapi terkadang mengalami kesulitan konvergen secara konsisten.

- **Xavier** Initialization memberikan hasil terbaik dengan akurasi tertinggi 87.95% dan loss yang cukup stabil. Inisialisasi ini memang dirancang untuk fungsi aktivasi seperti sigmoid/tanh atau dalam kasus saya, relu-softmax, dengan menyesuaikan skala bobot agar gradien tidak menghilang atau meledak.
- **He** Initialization menghasilkan akurasi 81.19%. Meski cukup baik, namun performanya masih di bawah Xavier. Walau penggunaan metode inisialisasi He lebih disarankan untuk aktivasi ReLU, tetapi dalam arsitektur dan parameter saya, dapat dilihat Xavier lebih optimal.

Adapun terkait distribusi bobot dan gradien.

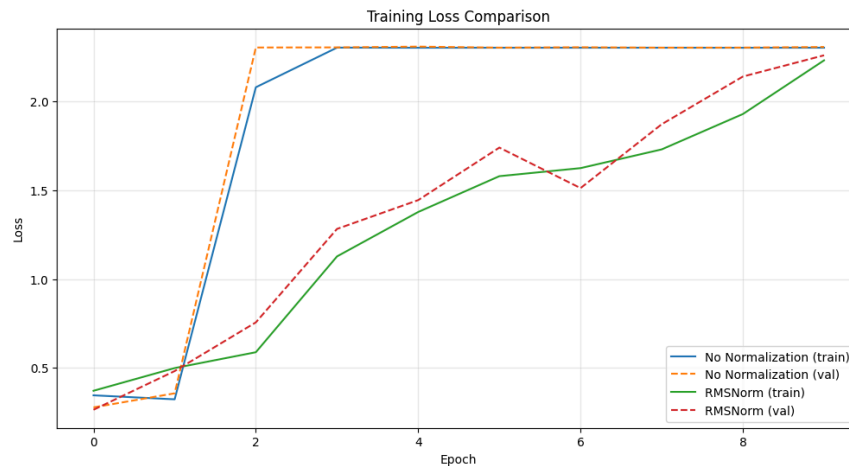
- **Zero Initialization** memiliki distribusi bobot dan gradien yang sangat sempit di sekitar nol, membuktikan bahwa tidak terjadi pembelajaran yang bermakna.
- **Random Uniform** dan **Random Normal** memiliki sebaran yang lebih luas, menunjukkan variasi bobot yang baik, tetapi terkadang terlalu ekstrem (outlier besar), dan menyebabkan fluktuasi learning curve.
- **Xavier dan He Initialization** menghasilkan distribusi bobot dan gradien yang cukup simetris dan terkontrol, mendukung konvergensi yang lebih baik.

Metode inisialisasi bobot sangat mempengaruhi proses pelatihan dan akurasi model. Xavier Initialization menjadi yang terbaik dengan akurasi tertinggi dan pelatihan yang stabil. Zero Initialization tidak efektif dan sebaiknya dihindari karena menghambat proses belajar. Random Uniform Normal cukup baik namun kurang stabil, sedangkan He Initialization juga memberikan hasil yang cukup baik, meski masih di bawah Xavier untuk arsitektur dan data yang digunakan.



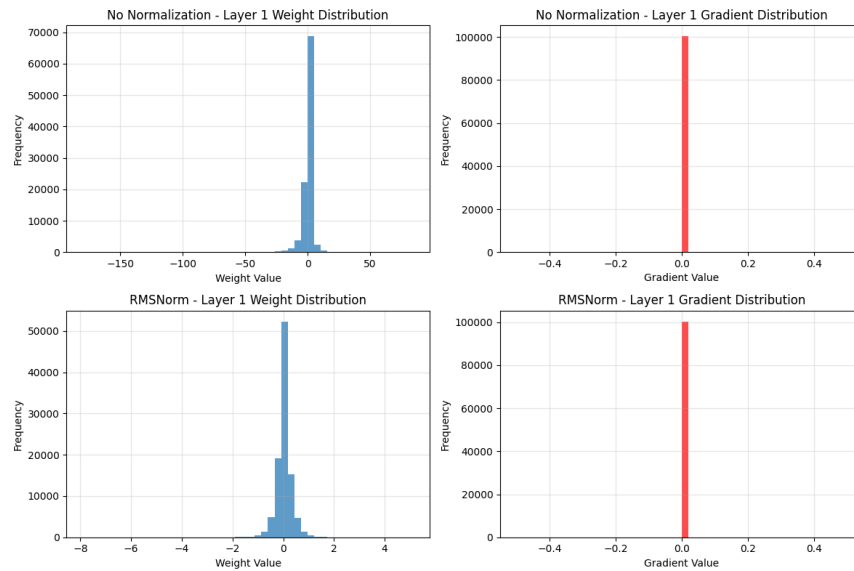
## 2. 5. Pengaruh Normalisasi RMSNorm

Hasil eksperimen yang telah dilakukan menunjukkan perbedaan yang cukup signifikan dalam performa dua model Feed Forward Neural Network (FFNN) yang diuji, satu dengan normalisasi RMSNorm dan satu tanpa normalisasi. Berikut ini analisis dari hasil yang didapatkan:



### 1) Loss Training dan Validasi:

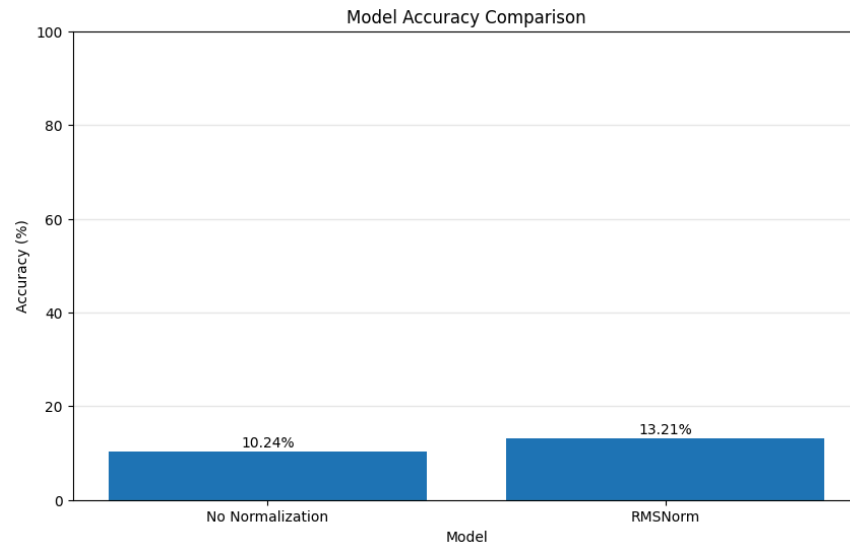
- Model **Tanpa Normalisasi**: Dari grafik, terlihat adanya lonjakan loss yang besar pada epoch ketiga yang kemudian stabil di angka tinggi (sekitar 2.3) untuk sisa epochs. Ini mengindikasikan bahwa model FFNN mengalami kesulitan dalam proses belajar, mungkin karena vanishing atau exploding gradients yang tidak dikendalikan oleh normalisasi.
- Model **dengan RMSNorm**: Ada peningkatan loss yang lebih bertahap dan tidak sebesar pada model tanpa normalisasi. Ini menunjukkan bahwa RMSNorm mungkin membantu mengurangi masalah gradients, tetapi model tetap tidak mencapai konvergensi yang baik, mungkin karena adanya overfitting atau pilihan parameter yang kurang tepat.



## 2) Distribusi Bobot dan Gradien:

- Distribusi Bobot:** Pada model tanpa normalisasi, distribusi bobot sangat terkonsentrasi dekat nol, yang bisa mengindikasikan kurangnya *diversity* dalam pembelajaran bobot. Pada model dengan RMSNorm, distribusi lebih tersebar walaupun masih cukup terpusat, menunjukkan kemungkinan efek stabilisasi dari RMSNorm.
- Distribusi Gradien:** Baik model tanpa normalisasi maupun dengan RMSNorm menunjukkan distribusi gradien yang sangat terpusat di nilai yang sangat rendah, mengindikasikan adanya masalah dalam propagasi gradien yang efektif, yang bisa jadi disebabkan oleh aktivasi yang tidak efisien atau inisialisasi yang kurang tepat.





### 3) Akurasi Model:

Model tanpa normalisasi mencapai akurasi yang sangat rendah (10.24%), sedangkan model dengan RMSNorm hanya sedikit lebih baik (13.21%). Ini menunjukkan bahwa kedua model memiliki kesulitan yang signifikan dalam mempelajari pemetaan yang tepat dari input ke output, yang mungkin disebabkan oleh arsitektur atau parameter model.

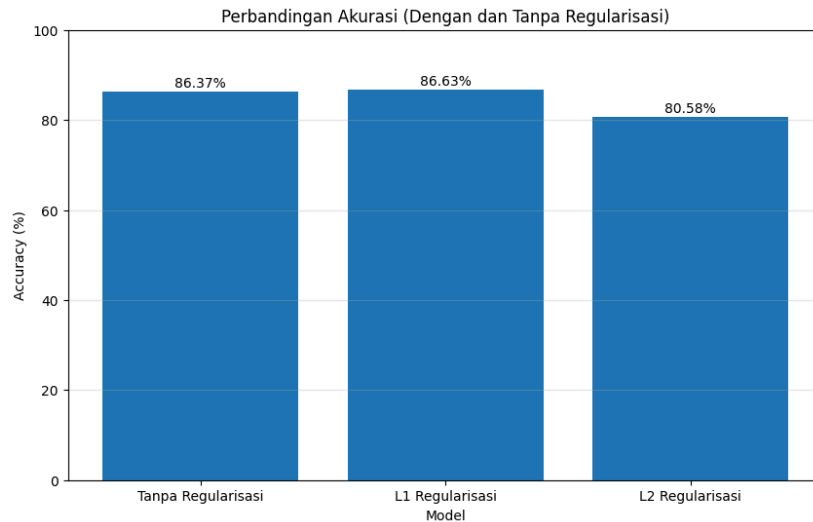
### 4) Waktu Pelatihan:

```
No Normalization Model:  
Accuracy: 10.24%  
Training Time: 52.26 seconds  
  
RMSNorm Model:  
Accuracy: 13.21%  
Training Time: 36.29 seconds
```

Model dengan RMSNorm memiliki waktu pelatihan yang lebih singkat dibandingkan dengan model tanpa normalisasi. Ini menunjukkan bahwa penggunaan RMSNorm mungkin membantu dalam efisiensi perhitungan meskipun belum berhasil meningkatkan akurasi dengan signifikan.

## 2. 6. Pengaruh Regularisasi

Berdasarkan eksperimen yang dilakukan terhadap penggunaan regularisasi, didapatkan grafik perbandingan akurasi berikut.

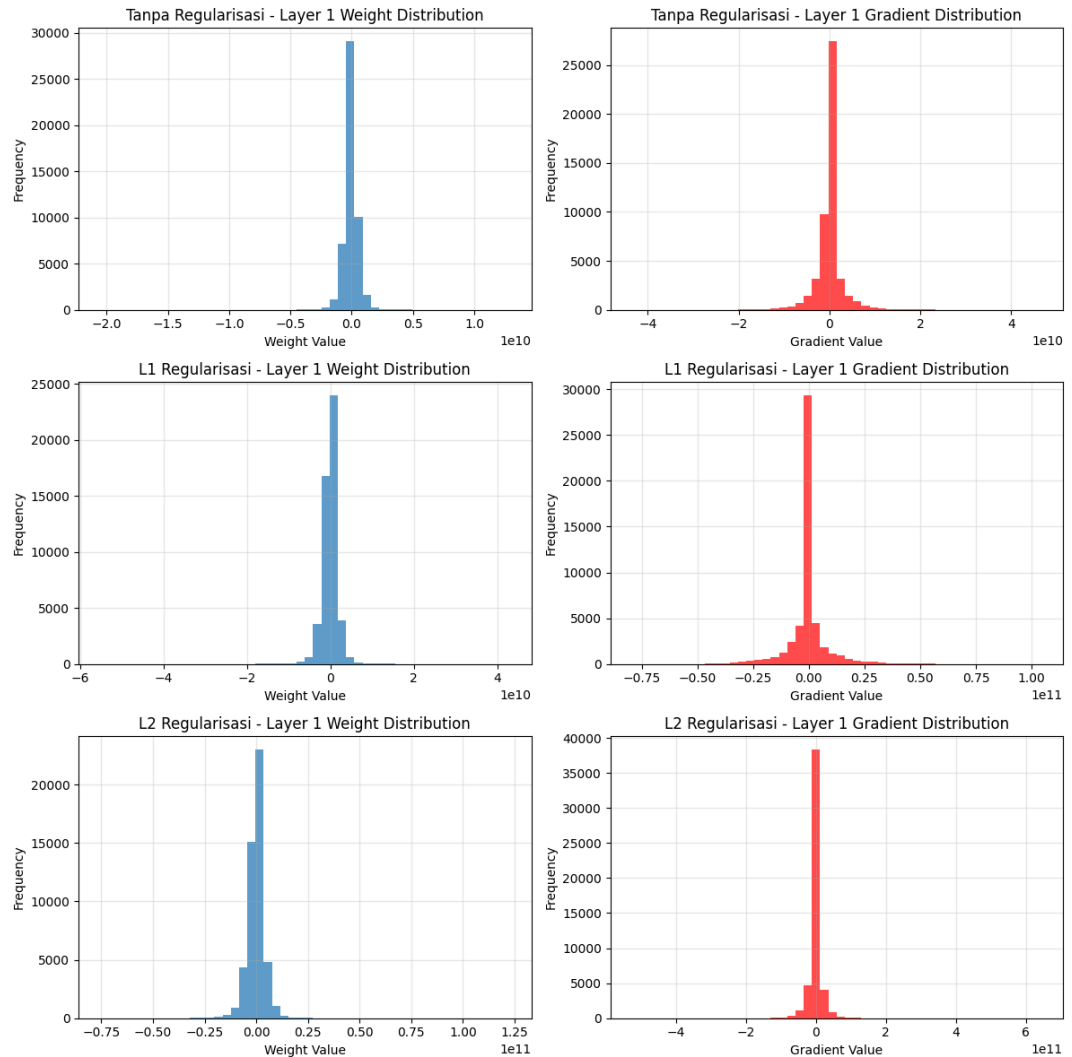


Grafik menunjukkan bahwa model tanpa regularisasi mencapai akurasi sebesar 86.37%, sedangkan penggunaan L1 regularisasi sedikit meningkatkan akurasi menjadi 86.63%. Sebaliknya, penggunaan L2 regularisasi justru menurunkan akurasi secara signifikan menjadi 80.58%. Hal ini menunjukkan bahwa L1 regularisasi memiliki potensi untuk membantu generalisasi model pada data validasi, meskipun dampaknya tidak terlalu besar, sementara L2 kemungkinan menyebabkan underfitting akibat penalti yang terlalu besar terhadap bobot, sehingga model kesulitan dalam mempelajari pola kompleks dalam data.

Selain akurasi, pengaruh regularisasi juga terlihat jelas pada grafik perbandingan loss antara data training dan validasi.



Model tanpa regularisasi menunjukkan tren loss yang relatif stabil, dengan gap yang kecil antara training dan validation loss. Namun, baik L1 maupun L2 mengalami peningkatan loss secara drastis (exploding loss) setelah epoch ke-8. Fenomena ini mengindikasikan adanya ketidakstabilan numerik dalam proses pembelajaran, kemungkinan besar disebabkan oleh nilai bobot dan gradien yang menjadi sangat besar karena nilai parameter regularisasi ( $\lambda$ ) yang tidak optimal. Gap antara training dan validation loss pada L1 mencapai lebih dari 39 miliar, sedangkan pada L2 mencapai nilai luar biasa tinggi sekitar 471 kuintiliun, memperkuat dugaan bahwa regularisasi justru memperburuk performa dalam konfigurasi ini.

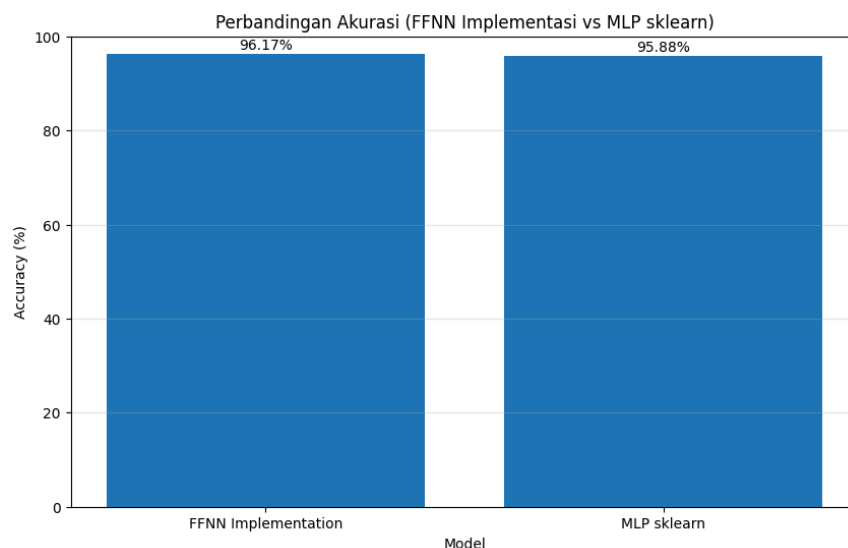


Distribusi bobot dan gradien juga memperlihatkan perbedaan yang mencolok. Tanpa regularisasi, bobot dan gradien cenderung terdistribusi secara simetris dan terkonsentrasi di sekitar nol. Pada model dengan L1, distribusi bobot mulai menunjukkan pola sparsity—yaitu banyak bobot mendekati nol—namun gradiennya menjadi lebih tersebar luas. Sementara pada L2, distribusi bobot dan gradien tampak jauh lebih melebar dan ekstrem, mengindikasikan terjadinya numerical explosion selama pelatihan. Visualisasi struktur jaringan mendukung temuan ini, dengan model L1 dan L2 memperlihatkan bobot yang sangat besar atau kecil di beberapa layer, berbeda dengan model tanpa regularisasi yang menunjukkan visualisasi lebih stabil dan seimbang.

Secara keseluruhan, meskipun regularisasi secara teori dapat membantu mengendalikan kompleksitas model dan mencegah overfitting, hasil eksperimen ini menegaskan bahwa pemilihan nilai lambda sangat krusial. Dalam kasus ini, model tanpa regularisasi justru menghasilkan performa terbaik karena pelatihannya lebih stabil dan mampu melakukan generalisasi dengan baik pada data validasi. Hal ini menunjukkan bahwa regularisasi tidak selalu meningkatkan performa model diperlukan penyesuaian konfigurasi agar regularisasi bekerja dengan optimal.

## 2. 7. Perbandingan dengan Library sklearn

Hasil perbandingan akurasi implementasi FFNN dan MLP sklearn bisa dilihat dari gambar berikut.



FFNN yang diimplementasikan menunjukkan akurasi yang lebih tinggi, yakni 96.17%. Hal ini menunjukkan bahwa model dengan parameter yang ditetapkan pada model FFNN ini efektif dalam menangani data yang digunakan untuk pelatihan dan pengujian. Fungsi aktivasi 'relu' dan 'softmax' serta inisialisasi bobot 'random normal' yang digunakan tampaknya cocok untuk dataset ini, memberikan kelebihan terhadap model MLP sklearn yang mencapai akurasi sedikit lebih rendah pada 95.88%.

Selain itu, model FFNN juga menunjukkan efisiensi waktu yang lebih baik dengan waktu pelatihan yang lebih cepat dibandingkan dengan MLP

sklearn, menandakan bahwa pendekatan model FFNN yang dibangun mungkin lebih optimal dalam hal komputasi. Ini menjadi penting, terutama dalam aplikasi di mana waktu pelatihan yang cepat sangat diperlukan. Namun, perlu dicatat bahwa model sklearn belum mencapai konvergensi dalam jumlah iterasi yang ditetapkan, yang menunjukkan potensi untuk peningkatan lebih lanjut. Meningkatkan jumlah iterasi atau menyesuaikan parameter lain pada model MLP sklearn mungkin akan membawa performa yang setara atau bahkan lebih baik dari implementasi FFNN dan ini adalah area yang layak untuk dijelajahi lebih lanjut untuk kesempatan berikutnya.

## C. Kesimpulan dan Saran

### 3.1. Kesimpulan

Pada tugas besar ini, kami berhasil mengimplementasikan Feedforward Neural Network (FFNN) dari awal, yang mencakup berbagai elemen penting seperti struktur jaringan, fungsi aktivasi, fungsi loss, serta mekanisme perhitungan gradien dan pembaruan bobot menggunakan metode gradient descent. Beberapa eksperimen dilakukan untuk menganalisis pengaruh berbagai hyperparameter terhadap performa model, seperti kedalaman jaringan (depth), jumlah neuron per layer (width), fungsi aktivasi, learning rate, inisialisasi bobot, dan normalisasi RMSNorm.

- **Pengaruh Depth dan Width:**

Model dengan **Width 32** memberikan hasil terbaik dengan akurasi **86.29%**, sementara model dengan **Width 64** dan **Width 128** memberikan hasil yang sedikit lebih rendah, yaitu **83.86%** dan **84.65%**. Hal ini menunjukkan bahwa menambah jumlah neuron tidak selalu menghasilkan peningkatan yang signifikan dalam performa, bahkan bisa memperburuk stabilitas training dan memperlambat waktu pelatihan.

Untuk kedalaman jaringan (depth), model dengan **1 Hidden Layer** menunjukkan akurasi **83.86%**, sedangkan menambah layer menyebabkan penurunan drastis dalam akurasi, dengan model **2 Hidden Layers** dan **3 Hidden Layers** hanya mencatatkan akurasi sekitar **9.88%** dan **9.86%**. Ini mengindikasikan adanya masalah seperti **vanishing gradient** atau **overfitting** pada jaringan yang lebih dalam.

- **Pengaruh Fungsi Aktivasi:**

Dalam hal fungsi aktivasi, **Sigmoid** dan **Tanh** memberikan akurasi terbaik dengan masing-masing **95.86%** dan **95.20%**. Fungsi **Linear** menunjukkan kinerja yang sangat buruk dengan akurasi **77.23%**, sedangkan **ReLU** dan variannya (**Leaky ReLU** dan **ELU**) menunjukkan hasil yang lebih baik namun masih di bawah **Sigmoid** dan **Tanh**. **ELU** memberikan performa terbaik di antara varian **ReLU** dengan akurasi **87.54%**.

- **Pengaruh Learning Rate:**

Eksperimen menunjukkan bahwa **learning rate 0.001** memberikan hasil terbaik dengan **akurasinya 96.17%** dan **loss yang lebih stabil**. **Learning rate 0.01** menyebabkan penurunan performa yang signifikan dengan **akurasi hanya 83.86%**. Ini menegaskan bahwa learning rate yang lebih rendah dapat menghasilkan konvergensi yang lebih baik dan lebih stabil. Learning rate yang sangat kecil bisa menyebabkan proses pelatihan untuk menjadi lebih lambat, sehingga perlu ditemukan learning rate yang tepat sesuai dengan kebutuhan. Learning rate yang terlalu besar juga bisa menyebabkan overflow.

- **Pengaruh Inisialisasi Bobot:**

Inisialisasi **Xavier** memberikan hasil terbaik dengan akurasi **87.95%**, sementara **Zero Initialization** menghasilkan performa terburuk dengan akurasi hanya **11.43%**. Inisialisasi bobot sangat berpengaruh pada proses pelatihan dan akurasi model.

- **Pengaruh Normalisasi RMSNorm:**

Model dengan normalisasi RMSNorm menunjukkan performa yang lebih stabil, meskipun akurasi akhir hanya sedikit lebih baik dibandingkan model tanpa normalisasi. Hal ini menunjukkan bahwa RMSNorm dapat membantu mengurangi masalah gradien, meskipun masih ada tantangan dalam mencapai konvergensi yang optimal.

- **Pengaruh Regularisasi:**

Penggunaan regularisasi menunjukkan dampak yang bervariasi terhadap performa model. Model tanpa regularisasi memberikan hasil terbaik dengan akurasi 86.37% dan loss yang stabil antara data latih dan validasi. Sementara itu, L1 regularisasi sedikit meningkatkan akurasi menjadi 86.63%, namun menyebabkan instabilitas pada loss akibat nilai gradien yang besar. L2 regularisasi justru menurunkan akurasi ke 80.58% dan mengalami *exploding loss* secara signifikan. Distribusi bobot dan gradien pada model L1 dan L2 juga terlihat lebih menyebar dibandingkan model tanpa regularisasi. Hasil ini menunjukkan bahwa regularisasi tidak selalu



meningkatkan performa model, terutama jika parameter lambda tidak disesuaikan dengan tepat.

- **Perbandingan dengan Library sklearn:**

Model FFNN yang diimplementasikan menunjukkan akurasi lebih tinggi (**96.17%**) dibandingkan dengan MLP dari **sklearn** (**95.88%**), menunjukkan bahwa model yang dibangun dari awal lebih efektif dalam menangani data yang digunakan. Selain itu, waktu pelatihan model FFNN juga lebih cepat, menandakan bahwa model FFNN yang dibangun memiliki kelebihan dalam efisiensi komputasi.

### 3.2. Saran

Saran yang dapat kami berikan untuk pembaca atau peneliti selanjutnya adalah:


- Melakukan eksperimen lebih lanjut dengan *batch normalization* atau *dropout* untuk mengatasi *vanishing gradient* dan *overfitting*.
- Menggunakan *learning rate decay* untuk meningkatkan konvergensi.
- Menerapkan He initialization pada model dengan ReLU untuk hasil yang lebih baik.
- Eksplorasi penggunaan fungsi aktivasi lain untuk perbaikan lebih lanjut.
- Melakukan perbandingan lebih lanjut dengan sklearn MLPClassifier menggunakan grid search untuk pencarian hyperparameter yang optimal.
- Melakukan eksplorasi mengenai metode mengatasi overflow.

### D. Pembagian Tugas tiap Anggota Kelompok

NIM	Nama	Tugas
13522053	Erdianti Wiga Putri Andini	Linear, Sigmoid, Random Distribusi Uniform, MSE, Forward, Backward, Visualisasi Network, Normalisasi RMSNorm, Laporan
13522063	Shazya Audrea Taufik	Binary Cross Entropy, Random Distribusi Normal, Softmax, Forward, Backward, Kode pengujian, Visualisasi gradien dan bobot, Laporan

13522085	Zahira Dina Amalia	Categorical cross entropy, Zero weight initialization, ReLU, Hyperbolic, He, Xavier, Leaky ReLU, eLU, Forward, Backward, Laporan
----------	--------------------	--

## E. Referensi

-  The spelled-out intro to neural networks and backpropagation: building mi...
- <https://www.jasonosajima.com/forwardprop>
- <https://www.jasonosajima.com/backprop>
- <https://numpy.org/doc/2.2/>
- [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
- [https://math.libretexts.org/Bookshelves/Calculus/Calculus\\_\(OpenStax\)/14%3ADifferentiation\\_of\\_Functions\\_of\\_Several\\_Variables/14.05%3A\\_The\\_Chain\\_Rule\\_for\\_Multivariable\\_Functions](https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/14%3ADifferentiation_of_Functions_of_Several_Variables/14.05%3A_The_Chain_Rule_for_Multivariable_Functions)
- <https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/>
- <https://douglasorr.github.io/2021-11-autodiff/article.html>