

**LAPORAN TUGAS BESAR 1**  
**IF3270 Pembelajaran Mesin**  
**Kelompok 17**  
**“Feedforward Neural Network”**



**Dosen:**

Dr. Fariska Zakhralativa Ruskanda, S.T., M.T.

**Oleh:**

Owen Tobias Sinurat (13522131)

Ahmad Thoriq Saputra (13522141)

Muhammad Fatihul Irhab (13522143)

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**SEMESTER II TAHUN 2023/2024**

# DESKRIPSI PERSOALAN

## Spesifikasi

Implementasikan suatu modul FFNN yang memenuhi ketentuan-ketentuan 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**. Pilihan fungsi aktivasi yang harus diimplementasikan adalah sebagai berikut:

Nama Fungsi Aktivasi	Definisi Fungsi
Linear	$Linear(x) = x$
ReLU	$ReLU(x) = \max(0, x)$
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$
Hyperbolic Tangent (tanh)	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Softmax	Untuk vector $\vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ , $softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$

- FFNN yang diimplementasikan dapat **menerima fungsi loss** dari model tersebut. Pilihan loss function yang harus diimplementasikan adalah sebagai berikut:

Nama Fungsi Loss	Definisi Fungsi
------------------	-----------------

<a href="#">MSE</a>	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
<a href="#">Binary Cross-Entropy</a>	$\mathcal{L}_{BCE} = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$ <p> <math>y_i</math> = Actual binary label (0 or 1)  <math>\hat{y}_i</math> = Predicted value of <math>y_i</math>  <math>n</math> = Batch size </p>
<a href="#">Categorical Cross-Entropy</a>	$\mathcal{L}_{CCE} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C (y_{ij} \log \hat{y}_{ij})$ <p> <math>y_{ij}</math> = Actual value of instance <math>i</math> for class <math>j</math>  <math>\hat{y}_{ij}</math> = Predicted value of <math>y_{ij}</math>  <math>C</math> = Number of classes  <math>n</math> = Batch size </p>

- Catatan:
  - Binary cross-entropy merupakan kasus khusus categorical cross-entropy dengan kelas sebanyak 2
  - Log yang digunakan merupakan logaritma natural (logaritma dengan basis e)
- Terdapat mekanisme untuk **inisialisasi bobot** tiap neuron (termasuk bias). Pilihan metode inisialisasi bobot yang harus diimplementasikan adalah sebagai berikut:
  - **Zero initialization**
  - Random dengan distribusi **uniform**.
    - Menerima parameter lower bound (batas minimal) dan upper bound (batas maksimal)
    - Menerima parameter seed untuk reproducibility
  - Random dengan distribusi **normal**.
    - Menerima parameter mean dan variance
    - Menerima parameter seed untuk reproducibility

- Instance model yang diinisialisasikan harus bisa **menyimpan bobot** tiap neuron (termasuk bias)
- Instance model yang diinisialisasikan harus bisa **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**. (Format graf dibebaskan)
- Instance model memiliki method untuk **menampilkan distribusi bobot** dari tiap layer.
  - Menerima masukan berupa list of integer (bisa disesuaikan ke struktur data lain sesuai kebutuhan) yang menyatakan layer mana saja yang distribusinya akan di-plot
- Instance model memiliki method untuk **menampilkan distribusi gradien bobot** dari tiap layer.
  - Menerima masukan berupa list of integer (bisa disesuaikan ke struktur data lain sesuai kebutuhan) yang menyatakan layer mana saja yang distribusinya akan di-plot
- Instance model memiliki method untuk **save** dan **load**
- Model memiliki implementasi **forward propagation** dengan ketentuan sebagai berikut:
  - Dapat menerima input berupa **batch**.
- Model memiliki implementasi **backward propagation** untuk menghitung perubahan gradien:
  - Dapat menangani perhitungan perubahan gradien untuk input data **batch**.
  - Gunakan konsep **chain rule** untuk menghitung gradien tiap bobot terhadap loss function.
  - Berikut merupakan **turunan pertama** untuk setiap fungsi aktivasi:

Nama Fungsi Aktivasi	Turunan Pertama
Linear	$\frac{d(\text{Linear}(x))}{dx} = 1$
ReLU	$\frac{d(\text{ReLU}(x))}{dx} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$
Sigmoid	$\frac{d(\sigma(x))}{dx} = \sigma(x)(1 - \sigma(x))$

Hyperbolic Tangent (tanh)	$\frac{d(\tanh(x))}{dx} = \left( \frac{2}{e^x - e^{-x}} \right)^2$
Softmax	<p>Untuk vector <math>\vec{x} = (x_1, \dots, x_n) \in \mathbb{R}^n</math>,</p> $\frac{d(\text{softmax}(\vec{x}))}{d\vec{x}} = \begin{bmatrix} \frac{\partial(\text{softmax}(\vec{x})_1)}{\partial x_1} & \dots & \frac{\partial(\text{softmax}(\vec{x})_1)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial(\text{softmax}(\vec{x})_n)}{\partial x_1} & \dots & \frac{\partial(\text{softmax}(\vec{x})_n)}{\partial x_n} \end{bmatrix}$ <p>Dimana untuk <math>i, j \in \{1, \dots, n\}</math>,</p> $\frac{\partial(\text{softmax}(\vec{x})_i)}{\partial x_j} = \text{softmax}(\vec{x})_i (\delta_{i,j} - \text{softmax}(\vec{x})_j)$ $\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$

- Berikut merupakan **turunan pertama** untuk setiap fungsi loss terhadap bobot suatu FFNN (lanjutkan sisanya menggunakan chain rule):

Nama Fungsi Loss	Definisi Fungsi
<a href="#">MSE</a>	$\frac{\partial \mathcal{L}_{MSE}}{\partial W} = -\frac{2}{n} \sum_{i=1}^n \frac{\partial \mathcal{L}_{MSE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial W} = -\frac{2}{n} \sum_{i=1}^n (y_i - \hat{y}_i) \frac{\partial \hat{y}_i}{\partial W}$
<a href="#">Binary Cross-Entropy</a>	$\frac{\partial \mathcal{L}_{BCE}}{\partial W} = -\frac{1}{n} \sum_{i=1}^n \frac{\partial \mathcal{L}_{BCE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial W} = -\frac{1}{n} \sum_{i=1}^n \frac{\hat{y}_i - y_i}{\hat{y}_i(1 - \hat{y}_i)} \frac{\partial \hat{y}_i}{\partial W}$
<a href="#">Categorical Cross-Entropy</a>	$\frac{\partial \mathcal{L}_{CCE}}{\partial W} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^c \frac{\partial \mathcal{L}_{CCE}}{\partial \hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial W} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^c \frac{y_{ij}}{\hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial W}$

- Model memiliki implementasi **weight update** dengan menggunakan **gradient descent** untuk memperbarui bobot berdasarkan gradien yang telah dihitung, berikut persamaannya:

$$W_{new} = W_{old} - \alpha \left( \frac{\partial \mathcal{L}}{\partial W_{old}} \right)$$

$\alpha$  = Learning rate


- Implementasi untuk pelatihan model harus memenuhi ketentuan berikut:
  - Dapat menerima parameter berikut:
    - Batch size
    - Learning rate
    - Jumlah epoch
    - Verbose
      - Verbose 0 berarti tidak menampilkan apa-apa selama pelatihan
      - Verbose 1 berarti hanya menampilkan progress bar beserta dengan kondisi training loss dan validation loss saat itu
  - Proses pelatihan mengembalikan **histori dari proses pelatihan** yang berisi **training loss** dan **validation loss tiap epoch**.
- Lakukan **pengujian** terhadap implementasi FFNN dengan ketentuan sebagai berikut:
  - Analisis pengaruh beberapa hyperparameter sebagai berikut:
    - Pengaruh **depth (banyak layer)** dan **width (banyak neuron per layer)**
      - Pilih 3 variasi kombinasi width (depth tetap) dan 3 variasi depth (width semua layer tetap)
      - Bandingkan hasil akhir prediksinya
      - Bandingkan grafik loss pelatihnannya
    - Pengaruh **fungsi aktivasi** hidden layer
      - Lakukan untuk setiap fungsi aktivasi yang diimplementasikan **kecuali softmax**.
      - Bandingkan hasil akhir prediksinya
      - Bandingkan grafik loss pelatihnannya
      - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
    - Pengaruh **learning rate**
      - Lakukan 3 variasi learning rate (nilainya dibebaskan)
      - Bandingkan hasil akhir prediksinya
      - Bandingkan grafik loss pelatihnannya
      - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
    - Pengaruh **inisialisasi bobot**

- Lakukan untuk setiap metode inisialisasi bobot yang diimplementasikan
  - Bandingkan hasil akhir prediksinya
  - Bandingkan grafik loss pelatihannya
  - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
- Analisis perbandingan hasil prediksi dengan [library sklearn MLP](#)
  - Lakukan satu kali pelatihan dengan hyperparameter yang sama untuk kedua model
  - Hyperparameter yang digunakan dibebaskan
  - Bandingkan hasil akhir prediksinya saja
- Gunakan dataset berikut untuk menguji model: [mnist\\_784](#)
  - Gunakan method [fetch\\_openml](#) dari sklearn untuk memuat dataset
  - Berikut contoh untuk memuat dataset: [Contoh](#)
- Akan ada beberapa **test case** yang akan diberikan oleh tim asisten (menyusul).
- Pengujian dilakukan di file .ipynb terpisah

## Spesifikasi Bonus

Berikut merupakan beberapa spesifikasi bonus yang dapat Anda kerjakan:

- Implementasikan FFNN dengan menggunakan **automatic differentiation**.
  - Metode ini merupakan metode yang umum digunakan untuk perhitungan gradien pada library-library untuk deep learning seperti PyTorch atau TensorFlow.
  - Jika ingin mengimplementasikan bonus ini, sangat disarankan untuk mengimplementasikan dari sejak awal mengerjakan tugas supaya tidak terlalu banyak perubahan dari implementasi biasa.
  - Referensi video:
 

 The spelled-out intro to neural networks and backpropagation: building...
- Implementasikan minimal 2 fungsi aktivasi lain yang sering digunakan.
- Implementasikan 2 metode inisialisasi bobot berikut:
  - Xavier initialization
  - He initialization
- Implementasikan metode regularisasi L1 dan L2, kemudian lakukan analisis untuk beberapa hal berikut:
  - Lakukan eksperimen masing-masing 1 kali untuk model tanpa regularisasi, dengan regularisasi L1, dan dengan regularisasi L2.
  - Bandingkan hasil akhir prediksinya
  - Bandingkan grafik loss pelatihannya

- Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model
- Implementasikan metode normalisasi RMSNorm, kemudian lakukan analisis untuk beberapa hal berikut:
  - Lakukan eksperimen masing-masing 1 kali untuk model tanpa normalisasi dan dengan normalisasi.
  - Bandingkan hasil akhir prediksinya
  - Bandingkan grafik loss pelatihannya
  - Bandingkan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model



# PEMBAHASAN

## A. Penjelasan Implementasi

1. Deskripsi kelas beserta deskripsi atribut dan methodnya

### Atribut

layer_sizes	Daftar ukuran setiap layer dalam jaringan saraf.
n_layers	Jumlah layer dalam jaringan saraf.
activation_functions	Daftar fungsi aktivasi yang digunakan pada setiap layer.
loss_function	Fungsi loss yang digunakan untuk mengukur kesalahan prediksi.
weights	Daftar matriks bobot untuk setiap layer.
biases	Daftar vektor bias untuk setiap layer.
gradients_w	Daftar gradien untuk bobot yang dihitung selama backpropagation.
gradients_b	Daftar gradien untuk bias yang dihitung selama backpropagation.
logger	Objek logger untuk mencatat aktivitas dan metrik jaringan saraf.
z_values	Nilai pre-aktivasi yang dihitung selama forward pass.
a_values	Nilai post-aktivasi (output) yang dihitung selama forward pass.
history	Kamus yang berisi riwayat metrik pelatihan seperti train_loss dan val_loss.

### Method

init()	Menginisialisasi jaringan saraf dengan parameter yang ditentukan
--------	--

<code>_setup_logging</code>	Mengatur sistem logging untuk jaringan saraf
<code>_initialize_weights</code>	Menginisialisasi bobot dan bias menggunakan metode yang ditentukan
<code>_activation_forward</code>	Menerapkan fungsi aktivasi pada nilai input Z
<code>_activation_backward</code>	Menghitung gradien fungsi aktivasi selama backpropagation
<code>_compute_loss</code>	Menghitung nilai loss antara prediksi dan nilai sebenarnya
<code>_compute_loss_gradient</code>	Menghitung gradien fungsi loss
<code>forward</code>	Melakukan forward pass untuk input X, menghitung output jaringan saraf
<code>backward</code>	Melakukan backward pass untuk menghitung gradien berdasarkan nilai sebenarnya
<code>update_weights</code>	Memperbarui bobot dan bias berdasarkan gradien yang dihitung
<code>fit</code>	Melatih jaringan saraf menggunakan data pelatihan
<code>predict</code>	Menghasilkan prediksi untuk input X menggunakan jaringan saraf yang sudah dilatih
<code>display_model</code>	Method kosong untuk menampilkan struktur model (belum diimplementasikan)
<code>plot_weight_distribution</code>	Method kosong untuk memvisualisasikan distribusi bobot (belum diimplementasikan)
<code>plot_gradient_distribution</code>	Method kosong untuk memvisualisasikan distribusi gradien (belum diimplementasikan)
<code>save_model</code>	Method kosong untuk menyimpan model ke file (belum diimplementasikan)

load_model	Method kosong untuk memuat model dari file (belum diimplementasikan)
plot_training_history	Method kosong untuk memvisualisasikan riwayat pelatihan (belum diimplementasikan)

## 2. Penjelasan forward propagation

Forward propagation merupakan tahap awal dalam proses operasional jaringan syaraf tiruan yang bertujuan menghasilkan prediksi berdasarkan data masukan. Dalam implementasi kelas **FFNN**, proses ini diatur dalam metode **forward(self, X: np.ndarray) -> np.ndarray**. Proses dimulai dengan menerima masukan **X**, yang berupa matriks dengan dimensi **(n\_samples, n\_features)**, mewakili jumlah sampel dan fitur pada data. Masukan ini disimpan sebagai aktivasi awal dalam daftar **self.a\_values** dengan **A = X**. Selanjutnya, untuk setiap lapisan dari indeks 0 hingga **self.n\_layers - 2**, masukan dari lapisan sebelumnya (**A**) dikalikan dengan matriks bobot (**self.weights[i]**) dan ditambahkan dengan vektor bias (**self.biases[i]**), menghasilkan nilai pra-aktivasi **Z = np.dot(A, W) + b**. Nilai **Z** ini dicatat dalam **self.z\_values** untuk keperluan backward propagation. Kemudian, **Z** diproses melalui fungsi aktivasi yang telah ditentukan dalam **self.activation\_functions[i]**, seperti sigmoid, relu, atau softmax, menggunakan metode **\_activation\_forward**, untuk menghasilkan aktivasi baru **A** yang disimpan kembali ke **self.a\_values** dan digunakan sebagai masukan bagi lapisan berikutnya.

Proses berulang ini berlangsung hingga mencapai lapisan terakhir, di mana keluaran akhir (**A**) dikembalikan sebagai hasil prediksi (**y\_pred**). Untuk mendukung analisis, setiap langkah dilengkapi dengan logging yang mencatat informasi seperti dimensi masukan, bobot, dan bias, serta statistik nilai **Z** dan **A**. Implementasi ini memungkinkan fleksibilitas dalam pemilihan fungsi aktivasi untuk setiap lapisan, sehingga jaringan dapat disesuaikan dengan kebutuhan tugas tertentu. Secara keseluruhan, forward propagation dalam kode ini dirancang untuk mengalirkan informasi dari masukan ke keluaran secara efisien, menjadi dasar bagi proses pembelajaran lebih lanjut.

## 3. Penjelasan backward propagation

Backward propagation adalah tahap penting dalam pelatihan jaringan saraf yang bertujuan menghitung gradien fungsi kerugian terhadap parameter jaringan, yaitu bobot dan bias, untuk mengoptimalkan model.

Dalam kelas **FFNN**, proses ini diimplementasikan melalui metode **backward(self, y\_true: np.ndarray) -> None**. Tahap awal melibatkan inisialisasi gradien bobot (**self.gradients\_w**) dan bias (**self.gradients\_b**) dengan nilai nol untuk setiap lapisan, memastikan tidak ada residu dari perhitungan sebelumnya. Gradien awal terhadap keluaran (**dA**) dihitung menggunakan metode **\_compute\_loss\_gradient** berdasarkan prediksi (**self.a\_values[-1]**) dan target (**y\_true**), dengan perhitungan yang disesuaikan menurut jenis fungsi kerugian, seperti **mse**, **binary\_crossentropy**, atau **categorical\_crossentropy**. Sebagai contoh, untuk kombinasi **categorical\_crossentropy** dan aktivasi softmax, gradien dihitung langsung sebagai  $(y_{pred} - y_{true}) / m$ , dengan **m** sebagai jumlah sampel, memanfaatkan efisiensi matematis dari kombinasi tersebut.

Proses propagasi mundur dilakukan dengan iterasi dari lapisan terakhir (**self.n\_layers - 2**) hingga lapisan pertama (indeks 0). Untuk setiap lapisan, gradien terhadap nilai pra-aktivasi (**dZ**) dihitung melalui metode **\_activation\_backward**, yang menerapkan turunan fungsi aktivasi yang sesuai, seperti turunan relu yang menghasilkan 0 untuk  $Z \leq 0$  dan 1 untuk  $Z > 0$ . Gradien bobot dihitung dengan  $np.dot(A_{prev}.T, dZ) / m$ , di mana **A\_prev** adalah aktivasi lapisan sebelumnya dari **self.a\_values**, sedangkan gradien bias diperoleh dengan  $np.sum(dZ, axis=0, keepdims=True) / m$ . Jika lapisan bukan lapisan pertama, gradien **dA** untuk lapisan sebelumnya diperbarui dengan  $np.dot(dZ, self.weights[layer].T)$ . Logging mencatat norma gradien untuk memantau stabilitas, memastikan proses berjalan dengan baik. Backward propagation ini berhasil menghitung kontribusi setiap parameter terhadap kerugian, menjadi langkah kunci sebelum pembaruan bobot.

#### 4. Penjelasan weight update

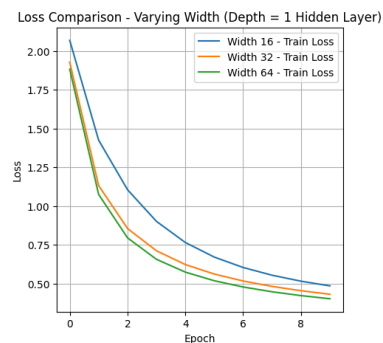
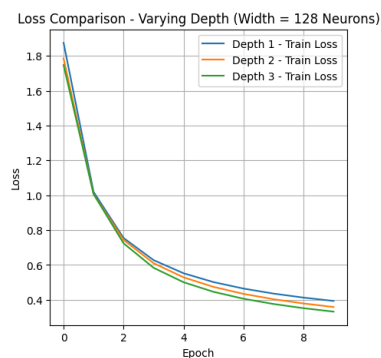
Pembaruan bobot adalah tahap akhir dalam siklus pelatihan yang bertujuan menyesuaikan parameter jaringan berdasarkan gradien yang dihitung, sehingga model dapat meminimalkan fungsi kerugian. Dalam kelas **FFNN**, proses ini diimplementasikan pada metode **update\_weights(self, learning\_rate: float) -> None**. Setelah gradien bobot dan bias diperoleh dari backward propagation, pembaruan dilakukan dengan mengurangi nilai bobot (**self.weights[i]**) dan bias (**self.biases[i]**) masing-masing dengan produk dari laju pembelajaran (**learning\_rate**) dan gradien yang sesuai (**self.gradients\_w[i]** dan **self.gradients\_b[i]**). Secara matematis, ini dapat ditulis sebagai **self.weights[i] -= learning\_rate \* self.gradients\_w[i]** dan **self.biases[i] -= learning\_rate \* self.gradients\_b[i]**. Laju pembelajaran menjadi parameter kritis yang menentukan besarnya langkah pembaruan; pemilihan nilai yang tepat penting untuk menghindari divergensi dan konvergensi yang terlalu lambat.

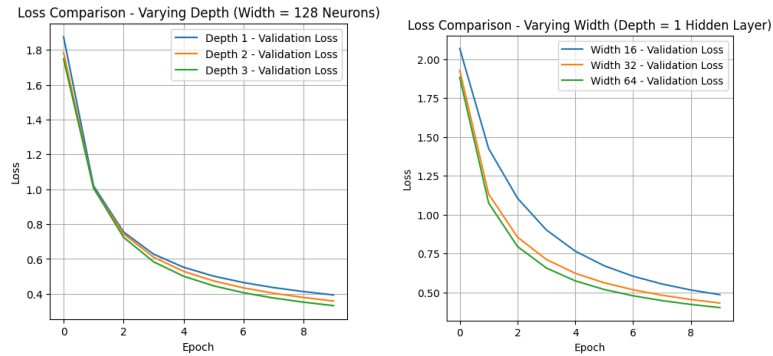
Proses pembaruan ini mengikuti prinsip gradient descent, di mana parameter digeser ke arah yang berlawanan dengan gradien untuk mengurangi kerugian. Dalam konteks metode fit, pembaruan bobot dilakukan setelah setiap batch data diproses melalui forward dan backward propagation, memungkinkan pembelajaran iteratif selama sejumlah epoch yang ditentukan. Implementasi ini menunjukkan struktur yang sederhana namun efektif, memastikan bahwa bobot dan bias diperbarui secara konsisten untuk meningkatkan performa model. Dengan demikian, kombinasi forward propagation, backward propagation, dan weight update dalam kelas FFNN membentuk siklus pembelajaran yang terintegrasi, mendukung pelatihan jaringan saraf secara menyeluruh sesuai dengan tugas yang diberikan.

## B. Hasil Pengujian

### 1. Pengaruh depth dan width

- Perbandingan Akurasi
  - Variasi Depth (Width = 128):
    - Depth 1 : 0.8903
    - Depth 2 : 0.8989
    - Depth 3 : 0.9016
  - Variasi Width (Depth = 1):
    - Width 16 : 0.8716
    - Width 32 : 0.8815
    - Width 64 : 0.8890
- Grafik Training Loss dan Validation Loss





- Analisis

Akurasi meningkat dengan seiring bertambahnya hidden layer (depth). Peningkatan ini menunjukkan bahwa model dengan depth lebih dalam mampu menangkap pola yang lebih kompleks dalam data, yang berkontribusi pada performa yang lebih baik.

Akurasi meningkat dengan seiring bertambahnya jumlah neuron per layer (width). Ini menunjukkan bahwa model dengan width lebih besar dapat mempelajari fitur yang lebih kaya dari data, sehingga meningkatkan performa.

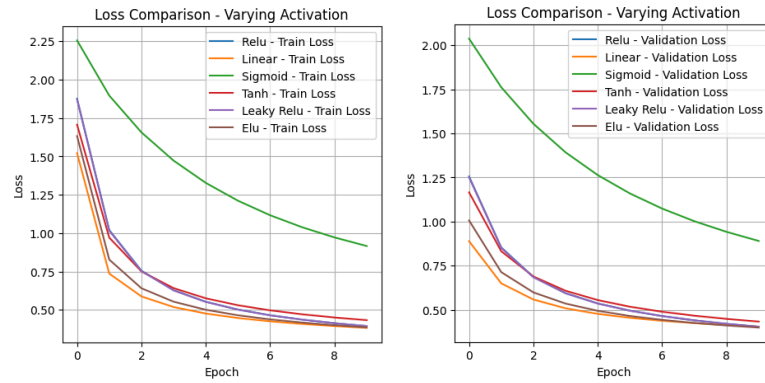
Semua kurva menunjukkan penurunan tajam pada awal pelatihan (epoch 0 hingga 4), diikuti oleh dengan kestabilan setelah epoch 4. Ini menunjukkan bahwa model cepat menyesuaikan bobotnya dengan data pada fase awal pelatihan.

## 2. Pengaruh fungsi aktivasi

- Perbandingan Akurasi

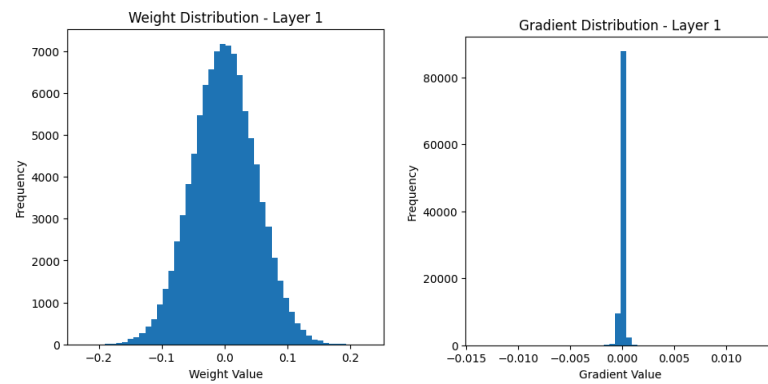
- Relu : 0.8906
- Linear : 0.8894
- Sigmoid : 0.8199
- Tanh : 0.8766
- Leaky Relu : 0.8897
- Elu : 0.8884

- Grafik Training Loss dan Validation Loss

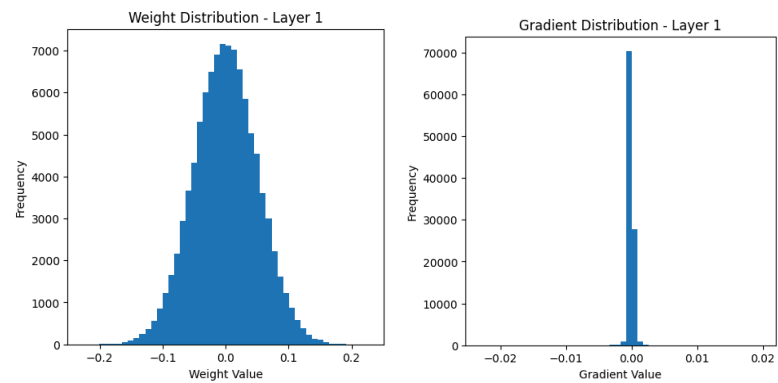


- Distribusi Bobot dan Gradien Bobot

- Relu

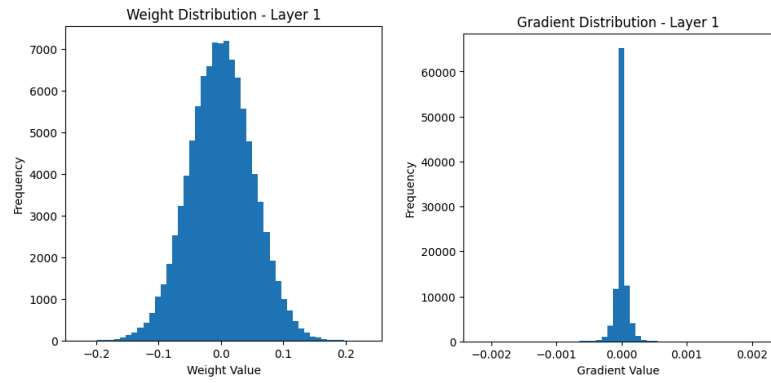


- Linear

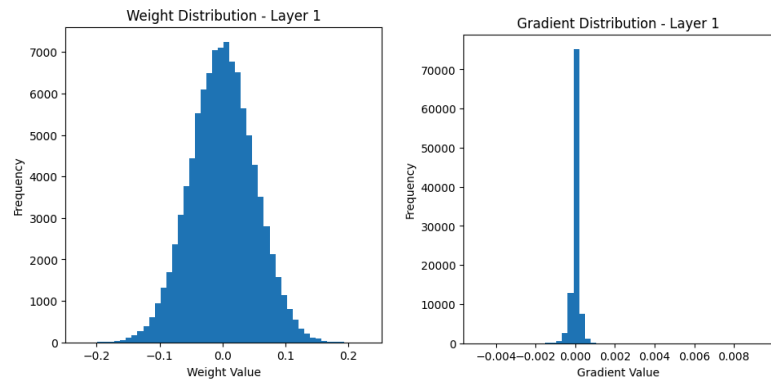


- Sigmoid

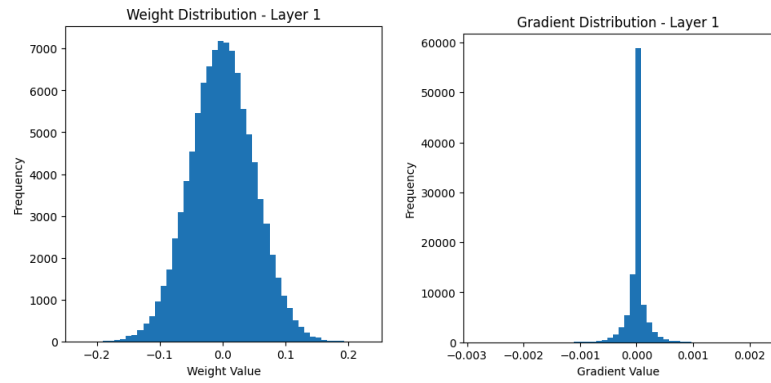




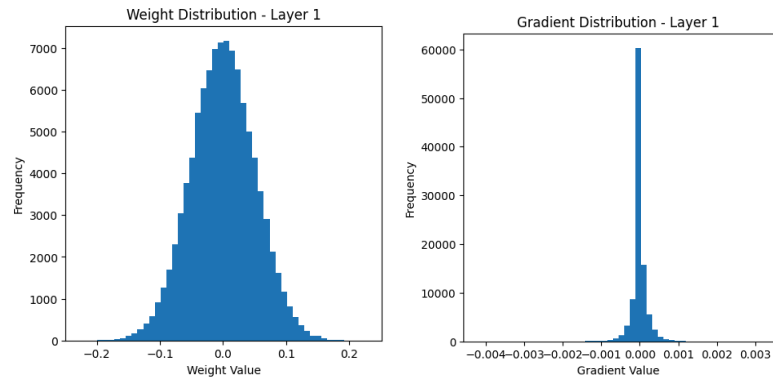
○ Tanh



○ LeakyRelu



○ Elu



- Analisis

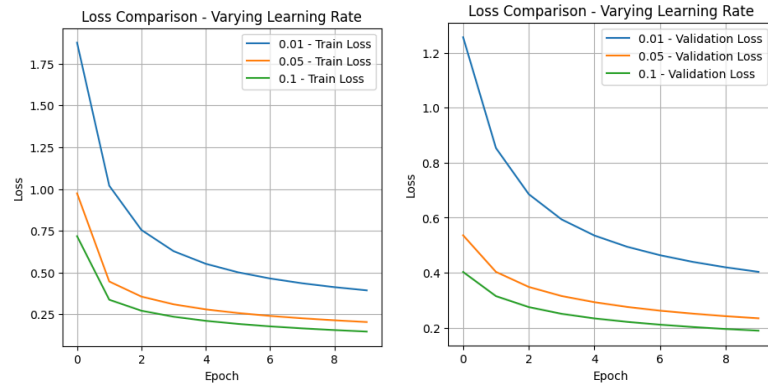
ReLU, Leaky ReLU, Elu, dan Linear menunjukkan akurasi yang sangat tinggi dan mirip, dengan ReLU mencapai nilai tertinggi (0.8906). Sigmoid memiliki akurasi terendah (0.8199), jauh di bawah fungsi aktivasi lainnya. Hal ini menunjukkan bahwa Sigmoid kurang efektif untuk dataset ini.

Semua fungsi aktivasi kecuali Sigmoid menunjukkan penurunan loss yang cepat dan stabil, mencapai nilai akhir sekitar 0.75 pada epoch 10. Ini menunjukkan bahwa ReLU, Linear, Tanh, Leaky ReLU, dan ELU efektif dalam mengoptimalkan model. Sementara sigmoid memiliki penurunan loss yang lebih lambat dan berhenti pada nilai yang lebih tinggi (~1.0), yang konsisten dengan akurasi terendahnya (0.8199).

Distribusi bobot yang menyerupai distribusi normal dengan rata-rata mendekati 0 menunjukkan bahwa inisialisasi bobot yang digunakan telah berhasil menciptakan distribusi yang seimbang. Distribusi gradien yang sangat terkonsentrasi di sekitar 0 menunjukkan bahwa sebagian besar gradien bobot sangat kecil, hal ini menunjukkan bahwa pelatihan model cenderung stabil, tetapi gradien yang sangat kecil dapat memperlambat pembelajaran pada layer yang lebih depth.

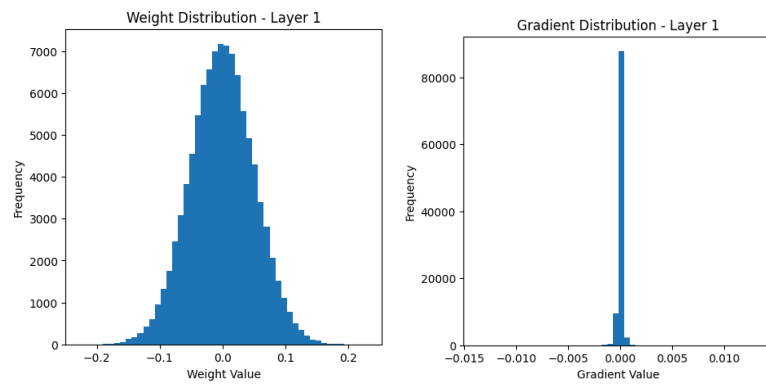
### 3. Pengaruh learning rate

- Perbandingan Akurasi
  - Learning Rate 0.01 : 0.8906
  - Learning Rate 0.05 : 0.9355
  - Learning Rate 0.1 : 0.9490
- Grafik Training Loss dan Validation Loss

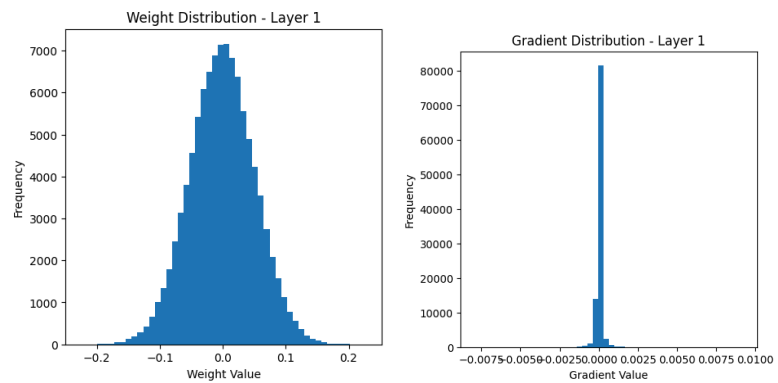


- Distribusi Bobot dan Gradien Bobot

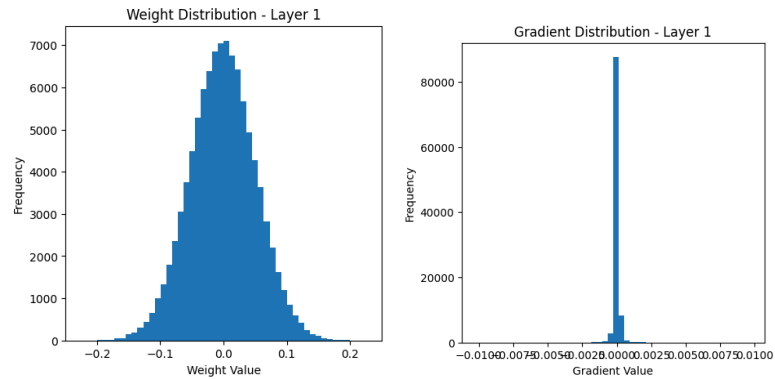
- Learning Rate 0.01



- Learning Rate 0.05



- Learning Rate 0.1



- Analisis

Akurasi meningkat seiring dengan peningkatan learning rate. Learning rate 0.1 menghasilkan akurasi tertinggi (0.9490), diikuti oleh 0.05 (0.9355), dan 0.01 (0.8906), hal ini menunjukkan bahwa meskipun learning rate yang lebih tinggi terus meningkatkan akurasi, keuntungan tambahan semakin mengecil seiring learning rate bertambah besar.

Semua kurva menunjukkan penurunan loss yang signifikan pada awal pelatihan (epoch 0 hingga 2), diikuti oleh penurunan yang lebih lambat hingga stabil pada epoch 10. Learning rate 0.01 memiliki penurunan yang kurang baik jika dibandingkan dengan learning rate yang lain (perbedaanya cukup jauh).

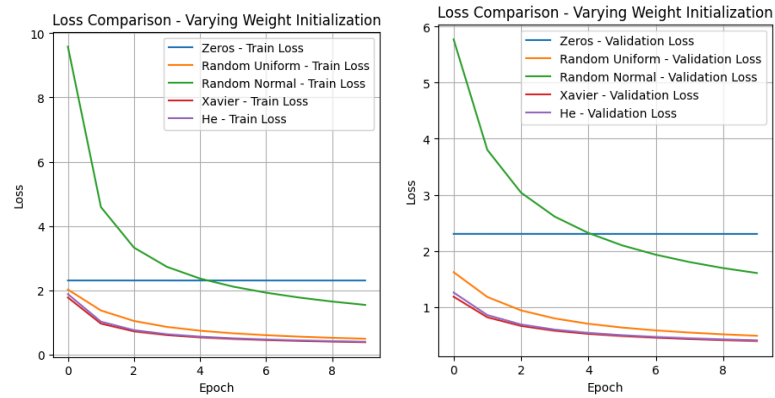
Distribusi bobot yang menyerupai distribusi normal dengan rata-rata mendekati 0 menunjukkan bahwa inisialisasi bobot yang digunakan telah berhasil menciptakan distribusi yang seimbang. Distribusi gradien yang sangat terkonsentrasi di sekitar 0 menunjukkan bahwa sebagian besar gradien bobot sangat kecil, hal ini menunjukkan bahwa pelatihan model cenderung stabil, tetapi gradien yang sangat kecil dapat memperlambat pembelajaran pada layer yang lebih depth.

#### 4. Pengaruh inisialisasi bobot

- Perbandingan Akurasi

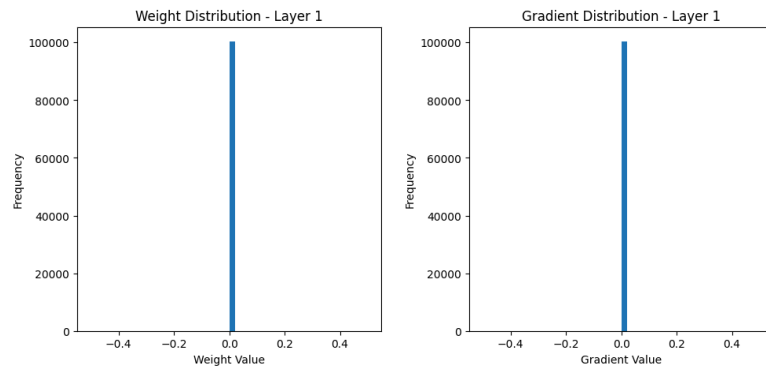
- Zeros : 0.1143
- Random Uniform : 0.8709
- Random Normal : 0.8269
- Xavier : 0.8921
- He : 0.8902

- Grafik Training Loss dan Validation Loss

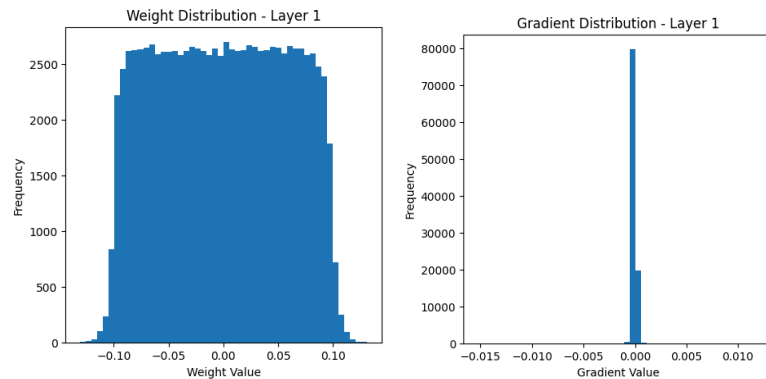


- Distribusi Bobot dan Gradien Bobot

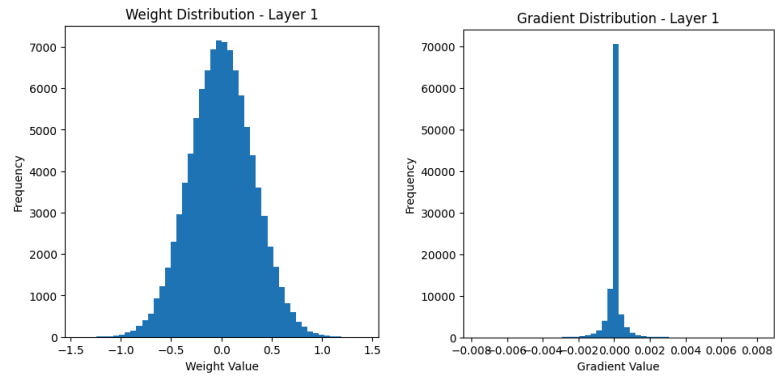
- Zeros



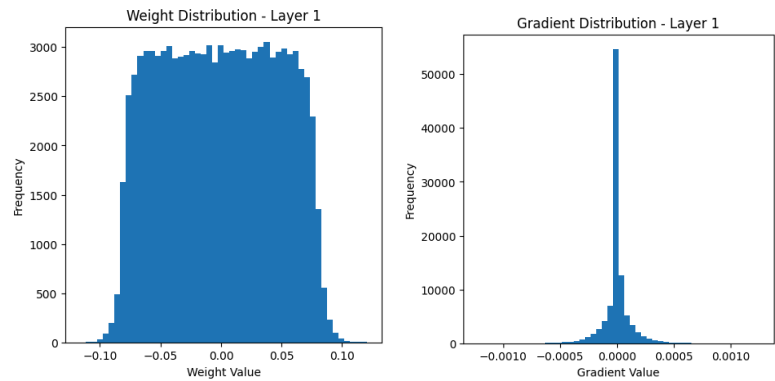
- Random Uniform



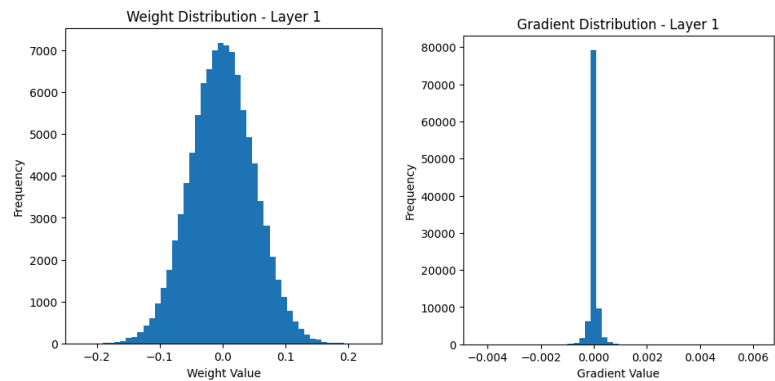
- Random Normal



- Xavier



- He



- Analisis

Xavier, He, dan Random Uniform menunjukkan akurasi yang sangat tinggi dan mirip, dengan Xavier mencapai nilai tertinggi (0.8921). Zeros memiliki akurasi terendah (0.1143), jauh di bawah fungsi aktivasi lainnya. Hal ini menunjukkan bahwa inisialisasi semua bobot ke nol menyebabkan masalah simetri, di mana neuron-neuron dalam layer yang sama belajar fitur yang identik, sehingga model gagal menangkap pola yang bermakna.

Inisialisasi bobot dengan metode Zeros menghasilkan loss yang tinggi dan stagnan, sehingga mendapatkan akurasi yang buruk (0.1143), karena gradien yang seragam atau hilang menyebabkan pembelajaran tidak efektif, sedangkan metode Xavier dan He memberikan penurunan loss yang paling signifikan dan stabil, dengan He mencapai loss terendah ( $\sim 0.5$ ) pada epoch 8, yang berkaitan dengan akurasi tertinggi (0.8921 dan 0.8902), menunjukkan bahwa kedua metode ini mendukung pelatihan yang optimal.

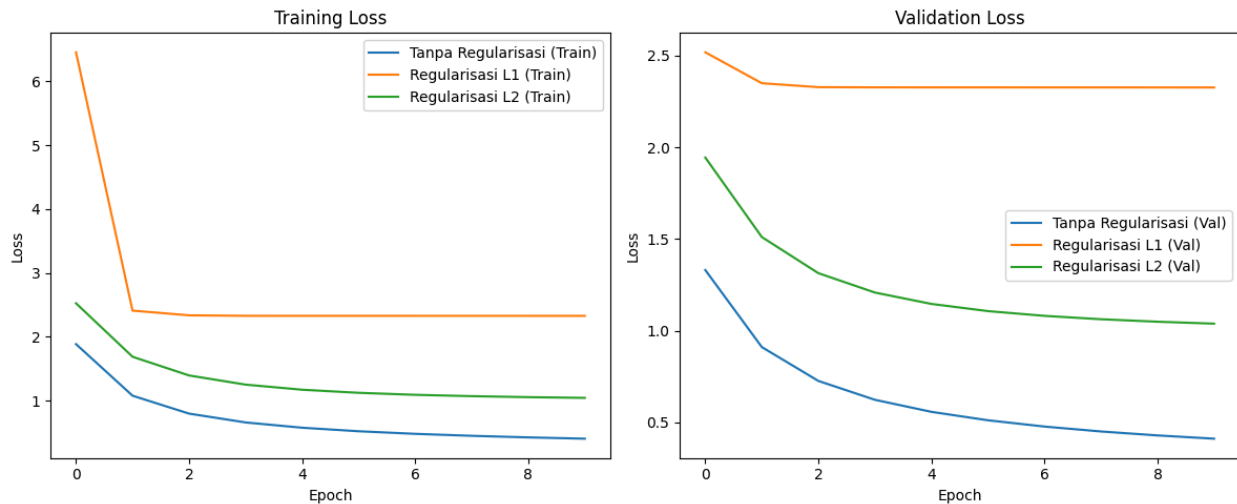
Distribusi bobot pada metode Zeros menunjukkan inisialisasi yang seragam yang menyebabkan simetri dan kegagalan dalam melakukan pembelajaran, sehingga menyebabkan akurasi rendah (0.1143). Distribusi bobot pada metode Random Uniform dan Xavier menunjukkan puncak lebar, mencerminkan inisialisasi yang seimbang dan mendukung pembelajaran efektif dengan akurasi yang baik. Distribusi bobot pada metode Random Normal dan He menunjukkan puncak tajam di sekitar 0, mencerminkan inisialisasi yang seimbang dimana mengarah ke nilai 0 sehingga membuat akurasi cukup baik.

## 5. Pengaruh regularisasi

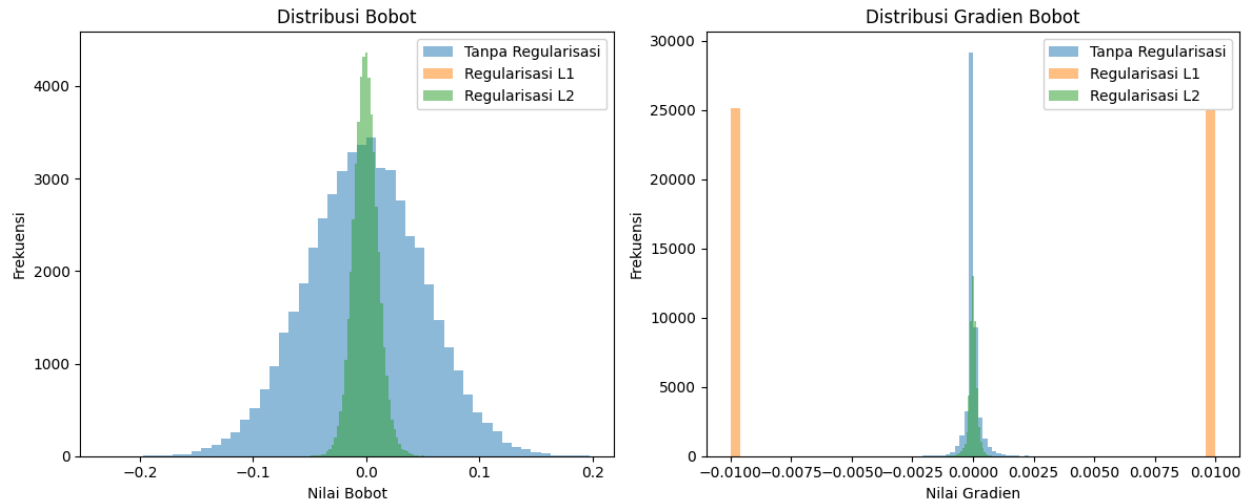
- Perbandingan Akurasi

- Tanpa regularisasi : 0.8890
- Regularisasi L1 : 0.1143
- Regularisasi L2 : 0.8394

- Grafik Loss dan Validation



- Distribusi Bobot dan Gradien Bobot



- Analisis

Hasil akurasi dari pengujian menunjukkan model tanpa regularisasi unggul dengan akurasi 0.8890, diikuti oleh regularisasi L2 dengan akurasi 0.8394, sementara regularisasi L1 hanya mencapai 0.1143, mengindikasikan bahwa lambda 0.01 pada L1 terlalu besar sehingga menyebabkan underfitting pada dataset MNIST. Model tanpa regularisasi dan L2 mampu belajar dengan baik, sedangkan L1 gagal menangkap pola yang signifikan akibat penalti bobot yang berlebihan.

Grafik training dan validation loss mengungkapkan bahwa model tanpa regularisasi dan L2 mengalami penurunan loss yang stabil hingga mendekati 0.25 dan 0.50, menunjukkan pembelajaran yang efektif tanpa overfitting berlebihan, sedangkan model L1 memiliki loss yang tetap tinggi di sekitar 2.25 untuk training dan 1.50 untuk validation sepanjang epoch, memperkuat indikasi underfitting. Hal ini menunjukkan bahwa regularisasi L1 dengan pengaturan saat ini tidak cocok untuk arsitektur model ini pada dataset MNIST.

Distribusi bobot dan gradien bobot menunjukkan efek regularisasi yang berbeda: L1 menghasilkan bobot yang sangat sparse (banyak nol) dengan gradien hampir nol, sedangkan tanpa regularisasi dan L2 memiliki distribusi bobot yang lebih lebar dan gradien yang bervariasi, mendukung pembelajaran yang lebih aktif. Secara keseluruhan, L2 lebih seimbang dibandingkan L1, tetapi model tanpa regularisasi sudah optimal untuk kasus ini; menurunkan lambda L1 ke 0.001 atau memperdalam arsitektur model dapat membantu meningkatkan performa.



## 6. Perbandingan dengan library sklearn

- Perbandingan Akurasi
  - FFNN Akurasi: 0.8483
  - sklearn MLP Akurasi: 0.9678
- Analisis

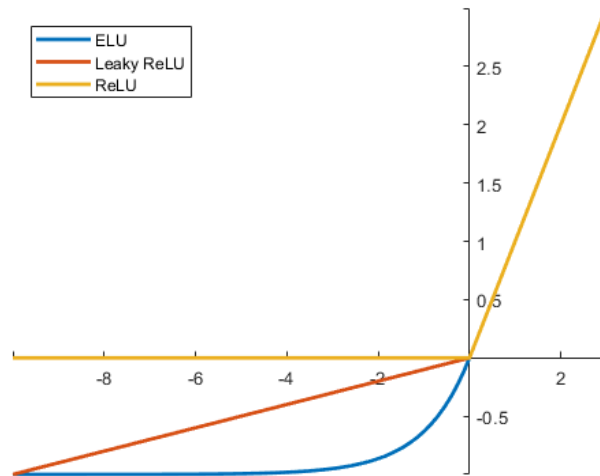
Implementasi FFNN dari scratch mencapai akurasi (0.8483), yang menunjukkan kemampuan model dalam menangkap pola pada dataset MNIST cukup baik, tetapi masih kalah dibandingkan scikit-learn MLPClassifier yang mencapai akurasi (0.9647) dengan selisih akurasi sebesar (0.1195).

Scikit-learn MLPClassifier menggunakan algoritma optimasi canggih seperti Adam atau L-BFGS, sedangkan implementasi FFNN dari scratch hanya mengimplementasikan gradient sederhana. Implementasi dari FFNN yang di buat dari scratch memungkinkan banyak implementasi yang kurang efisien jika dibandingkan oleh Scikit-learn MLPClassifier. Hal-hal ini dan beberapa hal optimasi maupun fitur yang tidak ada di FFNN dari scratch lah yang membuat terjadi perbedaan akurasi antara kedua model tersebut.

# BONUS

## Fungsi Aktivasi

→ Implementasikan minimal 2 fungsi aktivasi lain yang sering digunakan.



### Leaky ReLU activation function

Leaky ReLU adalah variasi dari ReLU yang menyelesaikan masalah “dead ReLU”. Fungsi aktivasi ini mengenalkan gradien negatif, sehingga neuron masih bisa memiliki nilai bukan nol walaupun inputnya negatif. Variasi ini menangani permasalahan neuron mati atau tidak responsif pada ReLU biasa.

Fungsi Leaky ReLU terdefinisi sebagai berikut:

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

Dalam persamaan di atas,  $x$  merepresentasikan input ke neuron atau lapisan neuron, dan  $a$  merepresentasikan konstanta positif kecil. Jika  $x$  positif, perlakuannya sama seperti ReLU biasa, mengembalikan  $x$ . Akan tetapi, jika  $x$  nya negatif, nilai yang dikembalikan adalah  $ax$  dibandingkan 0.

### Exponential Linear Unit (ELU) activation function

Exponential Linear Unit (ELU) adalah fungsi aktivasi yang ditujukan untuk overcome limitasi dari fungsi ReLU biasa, seperti masalah neuron mati dan saturasi nilai negatif. ELU memperkenalkan fungsi diferensial yang *smoothly saturates* input negatif dan memberikan output tidak 0 untuk input negatif.

Fungsi Leaky ELU terdefinisi sebagai berikut:

$$f(x) = x \text{ if } x \geq 0$$

$$f(x) = a(e^x - 1) \text{ if } x < 0$$

Di persamaan ini,  $x$  merepresentasikan input ke neuron atau lapisan neuron, dan  $a$  merepresentasikan konstanta positif yang mengatur gradien untuk input negatif.

## Metode inisialisasi bobot

→ Implementasikan 2 metode inisialisasi bobot berikut yaitu Xavier dan He.

### 1. Inisialisasi Xavier

Metode inisialisasi Xavier, yang diperkenalkan oleh Glorot dan Bengio, dengan tujuan untuk menjaga variansi aktivasi dan gradien agar tetap konsisten di seluruh layer. Metode ini sangat cocok digunakan pada fungsi aktivasi seperti tanh atau sigmoid, yang memiliki nilai ekstrim.

Bobot diinisialisasi dengan distribusi seragam (uniform distribution) dalam rentang  $[-limit, limit]$  dengan limit bernilai  $\sqrt{\frac{6}{inputSize + outputSize}}$

Contoh implementasi:

```
elif method.lower() == "xavier":  
    limit = np.sqrt(6 / (input_size + output_size))  
    W = np.random.uniform(low=-limit, high=limit, size=(input_size, output_size))  
    b = np.zeros((1, output_size))
```

### 2. Inisialisasi He

Metode inisialisasi He, yang diperkenalkan oleh He et al. Dibuat khusus untuk layer yang menggunakan fungsi aktivasi ReLU (Rectified Linear Unit) atau variannya. ReLU memiliki sifat non-linear yang hanya mengaktifkan neuron dengan nilai positif, sehingga membutuhkan inisialisasi yang mempertahankan variansi aktivasi hanya berdasarkan input.

Bobot diinisialisasi dengan distribusi normal (gaussian distribution) dengan rata-rata 0 dan deviasi standar yang bernilai  $\sqrt{\frac{2}{inputSize}}$

Contoh implementasi:

```
elif method.lower() == "he":  
    std = np.sqrt(2 / input_size)  
    W = np.random.normal(loc=0.0, scale=std, size=(input_size, output_size))  
    b = np.zeros((1, output_size))
```

## KESIMPULAN & SARAN

Berdasarkan analisis yang telah dilakukan terhadap implementasi FFNN dari scratch, terdapat beberapa hal yang dapat disimpulkan.

1. Variasi depth dan width menunjukkan peningkatan akurasi seiring bertambahnya kedalaman (dari 0.8903 pada Depth 1 menjadi 0.9016 pada Depth 3) dan lebar (dari 0.8716 pada Width 16 menjadi 0.8890 pada Width 64), meskipun terdapat diminishing returns yang mengindikasikan batas optimal dalam konfigurasi model.
2. Fungsi aktivasi seperti Relu, Leaky Relu, dan Elu memberikan performa terbaik (akurasi ~0.89) dibandingkan Sigmoid (0.8199), yang menderita vanishing gradient, sementara Tanh menunjukkan hasil menengah (0.8766).
3. Learning rate yang lebih tinggi (0.1) menghasilkan akurasi tertinggi (0.9490) dan konvergensi loss tercepat (~0.3), dibandingkan 0.05 (0.9355) dan 0.01 (0.8906).
4. Inisialisasi bobot dengan metode Xavier (0.8921) dan He (0.8902) unggul dibandingkan Random Uniform (0.8709), Random Normal (0.8269), dan Zeros (0.1143), dengan Zeros menunjukkan kegagalan total akibat simetri.
5. Perbandingan dengan scikit-learn MLPClassifier menunjukkan keterbatasan implementasi FFNN (0.8483) dibandingkan akurasi 96.78% dari MLPClassifier, menyoroti keunggulan optimasi dan fitur tambahan pada library tersebut.

Terdapat beberapa saran untuk meningkatkan performa FFNN dari scratch:

1. Pertimbangkan untuk mengoptimalkan kombinasi depth dan width (misalnya, Depth 3 dengan Width 128 dan learning rate 0.1) berdasarkan hasil terbaik sebelumnya, serta mengevaluasi generalisasi dengan validasi loss untuk mencegah overfitting.
2. Gunakan fungsi aktivasi Relu atau Leaky Relu untuk mengatasi vanishing gradient, terutama pada layer yang depth, dan hindari Sigmoid pada hidden layer.
3. Pilih inisialisasi Xavier atau He untuk mendukung pelatihan yang lebih baik, dan hindari Zeros untuk mencegah simetri, tambahkan analisis distribusi gradien pada layer yang lebih depth untuk mendeteksi masalah vanishing gradient lebih lanjut.
4. Implementasikan algoritma-algoritma lainnya yang dapat membuat model FFNN dari scratch lebih optimal untuk menutup perbedaan performa dengan scikit-learn, serta optimalkan implementasi modelnya untuk mendekati akurasi MLPClassifier.

## PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522131	Owen Tobias Sinurat	<ul style="list-style-type: none"><li>• Membuat kelas FFNN dalam Python</li><li>• Menangani struktur jaringan (jumlah layer &amp; neuron per layer)</li><li>• Implementasi fungsi aktivasi (Linear, ReLU, Sigmoid, Tanh, Softmax, Leaky ReLU, ELU)</li><li>• Implementasi forward propagation</li><li>• Implementasi fungsi loss (MSE, Binary Cross-Entropy, Categorical Cross-Entropy)</li><li>• Implementasi regularisasi L1 dan L2</li></ul>
13522141	Ahmad Thoriq Saputra	<ul style="list-style-type: none"><li>• Implementasi backward propagation dengan chain rule</li><li>• Menghitung gradien bobot &amp; bias tiap layer</li><li>• Implementasi weight update menggunakan gradient descent</li><li>• Memastikan model bisa menerima batch input</li><li>• Implementasi fungsi save &amp; load model</li><li>• Melatih model FFNN menggunakan hyperparameter yang sama dengan sklearn MLPClassifier</li><li>• Membuat perbandingan regularisasi L1 dan L2.</li></ul>
13522143	Muhammad Fatihul Irbab	<ul style="list-style-type: none"><li>• Implementasi inisialisasi bobot (Zero, Uniform, Normal, Xavier, He)</li><li>• Menulis notebook pengujian (.ipynb)</li><li>• Menguji model dengan dataset MNIST_784</li><li>• Menampilkan grafik loss training &amp; distribusi bobot</li><li>• Membandingkan hasil prediksi FFNN dan sklearn MLPClassifier</li><li>• Menyusun tabel/visualisasi perbandingan hasil model</li></ul>

## REFERENSI

- [But what is a neural network? | Deep learning chapter 1](#)
- [Backpropagation, step-by-step | DL3](#)
- [Activation functions in Neural Networks - GeeksforGeeks.](#)
- [Weight Initialization for Deep Learning Neural Networks - MachineLearningMastery.com](#)
- [Matplotlib](#)