

**Tugas Besar 1 IF3270 Pembelajaran Mesin**  
**Feedforward Neural Network**



**Kelompok 65**

13522051	Kharris Khisunica
13522079	Emery Fathan Zwageri
13522089	Abdul Rafi Radityo Hutomo

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2024**

# DESKRIPSI PERSOALAN

## A. Tujuan

Tugas Besar I pada kuliah IF3270 Pembelajaran Mesin agar peserta kuliah mendapatkan wawasan tentang bagaimana cara mengimplementasikan Feedforward Neural Network (FFNN). Pada tugas ini, peserta kuliah akan ditugaskan untuk mengimplementasikan FFNN from scratch.

## B. Deskripsi Persoalan

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: Linear, ReLU, Sigmoid, Hyperbolic Tangent, dan Softmax
- FFNN yang diimplementasikan dapat menerima fungsi loss dari model tersebut. Pilihan loss function yang harus diimplementasikan adalah sebagai berikut: Minimum Square Error (MSE), Binary Cross-Entropy (BCE), dan Categorical Cross-Entropy (CCE).
- 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.

- 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.
- Model memiliki implementasi weight update dengan menggunakan gradient descent untuk memperbarui bobot berdasarkan gradien yang telah dihitung.
- 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.

# PEMBAHASAN

## A. Penjelasan Forward Propagation

Forward propagation adalah proses perhitungan dalam Neural Network di mana input akan melewati setiap layer untuk menghasilkan output berupa hasil Accuracy. Setiap perhitungan dalam sebuah neuron terdiri dari bobot (**W**) dan bias (**b**). Perhitungan linier untuk suatu input **x** diberikan oleh:

$$z = xW^T + b$$

di mana:

- **x** adalah input dengan bentuk (**batch\_size, n\_input**),
- **W** adalah bobot dengan bentuk (**n\_neurons, n\_input**),
- **b** adalah bias dengan bentuk (**n\_neurons,**).

Hasil **z** memiliki bentuk (**batch\_size, n\_neurons**), yang kemudian diteruskan ke tahap berikutnya.

Setelah itu, nilai **z** dilewatkan ke fungsi aktivasi, yang bertujuan untuk membuat jaringan non linear. Hasil dari fungsi aktivasi ini merupakan output dari layer tersebut dan dapat diteruskan ke layer berikutnya atau digunakan sebagai Accuracy akhir.

## B. Penjelasan Backward Propagation dan Weight Update

Backward propagation adalah proses utama dalam pelatihan neural network yang digunakan untuk menyesuaikan bobot dan bias berdasarkan error yang dihasilkan selama forward propagation. Dengan menerapkan aturan rantai, backward propagation menghitung gradien dari fungsi loss terhadap setiap parameter model, sehingga bobot dapat diperbarui untuk meminimalkan error.

Backward propagation bekerja dengan menyebarkan error dari layer output ke layer-layer sebelumnya. Proses ini memungkinkan model untuk belajar dari error tersebut

dan meningkatkan akurasi dalam memprediksi data baru. Setelah gradien dihitung, pembaruan bobot dilakukan menggunakan algoritma optimasi seperti Stochastic Gradient Descent (SGD) atau Adam.

Pada tugas besar ini, kami mengimplementasikan optimizer SGD yang memiliki proses pembaruan weight sebagai berikut

$$w = w - \eta \frac{\partial E}{\partial w}$$

Dengan E adalah error dan w adalah weight.

## C. Deskripsi Kelas dan Implementasi

### 1. Implementasi Loss function

Deskripsi	File untuk menyimpan fungsi loss yang mungkin untuk digunakan dalam pembuatan FFNN, yakni fungsi loss MSE, BCE, dan CCE.
-----------	--

Nama Fungsi	def mse_loss(pred, target) -> Value
Deskripsi Fungsi	Mengembalikan nilai fungsi loss MSE
Source Code	<pre>def mse_loss(pred: Value, target: Value) -&gt; Value:     diff = pred + (-target)     return Value(np.mean(diff.data ** 2))</pre>

Nama Fungsi	def bce_loss(pred, target) -> Value
Deskripsi Fungsi	Fungsi nilai fungsi loss BCE

Source Code	<pre>def bce_loss(pred: Value, target: Value) -&gt; Value:     """     Binary Cross Entropy Loss:     loss = - [ target * log(pred) + (1 - target) * log(1 - pred) ], rata-rata elemen.     Pastikan pred bernilai antara (0,1).     """     one = Value(np.ones_like(pred.data))     loss = -(target * pred.log() + (one + (-target)) * ((one + (-pred)).log()))     return mean(loss)</pre>
-------------	---

Nama Fungsi	def cce_loss(pred, target) -> Value
Deskripsi Fungsi	Fungsi nilai fungsi loss CCE
Source Code	<pre>def cce_loss(pred: Value, target: Value) -&gt; Value:     """     Categorical Cross Entropy Loss:     Asumsikan pred berisi probabilitas (output softmax) dengan bentuk (batch, num_classes)     dan target adalah one-hot encoding dengan bentuk yang sama.     loss = - mean( sum( target * log(pred), axis=1 ) )     """     loss = -(target * pred.log())     loss_sum = sum_axis(loss, axis=1) # bentuk (batch, 1)     return mean(loss_sum)</pre>

## 2. Implementasi weight initialization

Deskripsi	File untuk menyimpan implementasi inisialisasi bobot untuk digunakan dalam membentuk FFNN. Jenis inisialisasi bobot yang diimplementasikan adalah inisialisasi wajib dan inisialisasi bonus: Inisialisasi dengan bobot 0, inisialisasi dengan bobot acak seragam, inisialisasi dengan bobot acak normal, inisialisasi He, dan inisialisasi Xavier.
-----------	--

Nama Fungsi	def zero_init(n_inputs, n_outputs)
Deskripsi Fungsi	Menginisialisasi semua bobot neuron dengan nilai 0
Source Code	<pre>def zero_init(n_inputs, n_outputs):     return np.zeros((n_outputs, n_inputs))</pre>

Nama Fungsi	def uniform_init(n_inputs, n_outputs, lower=-1.0, upper= 1.0,
-------------	---

	seed=None)
Deskripsi Fungsi	Menginisialisasi bobot neuron secara acak dan uniform berdasarkan seed dan batas atas dan bawah.
Source Code	<pre>def uniform_init(n_inputs, n_outputs, lower=-1.0, upper=1.0, seed=None):     if seed is not None:         rd.seed(seed)     return rd.uniform(lower, upper, size=(n_outputs, n_inputs))</pre>

Nama Fungsi	def normal_init(n_inputs, n_outputs, mean=0.0, variance=1.0, seed=None)
Deskripsi Fungsi	Menginisialisasi bobot neuron secara acak dan berdasarkan distribusi normal berdasarkan mean, standar deviasi, dan seed.
Source Code	<pre>def normal_init(n_inputs, n_outputs, mean=0.0, variance=1.0, seed=None):     if seed is not None:         rd.seed(seed)     std_dev = np.sqrt(variance)     return rd.normal(mean, std_dev, (n_outputs, n_inputs))</pre>

Nama Fungsi	Def he_init(n_inputs, n_outputs)
Deskripsi Fungsi	Menginisialisasi bobot neuron berdasarkan inisialisasi He
Source Code	<pre>def he_init(n_inputs, n_outputs):     return np.sqrt(2.0 / n_inputs) * rd.randn(n_outputs, n_inputs)</pre>

Nama Fungsi	def xavier_init(n_inputs, n_outputs)
Deskripsi Fungsi	Menginisialisasi bobot neuron berdasarkan inisialisasi Xavier
Source Code	<pre>def xavier_init(n_inputs, n_outputs):     limit = np.sqrt(6.0 / (n_inputs + n_outputs))     return rd.uniform(-limit, limit, (n_outputs, n_inputs))</pre>

### 3. Implementasi kelas Value

Deskripsi	Kelas value berfungsi sebagai dasar dari neuron dalam FFNN
-----------	--

	yang akan dibangun.
--	---------------------

Nama Fungsi	<code>__init__(self, data: np.ndarray   list)</code>
Deskripsi Fungsi	Fungsi initsialisasi untuk Value.
Source Code	<pre>class Value:     def __init__(self, data: Union[np.ndarray, list]):         self.data = np.array(data, dtype=np.float32)         self.grad = None         self._backward = lambda: None         self._prev = set()         self.label = None         self._op = None</pre>

Nama Fungsi	<code>__getitem__(self, idx)</code>
Deskripsi Fungsi	Mengembalikan elemen pada indeks tertentu dari data
Source Code	<pre>self._op = None def __getitem__(self, idx):     # Pastikan hasil slicing selalu 2D     sliced = self.data[idx]     if sliced.ndim == 1:         sliced = np.atleast_2d(sliced)     return Value(sliced)</pre>

Nama Fungsi	<code>__repr__(self)</code>
Deskripsi Fungsi	Mengembalikan representasi string dari data
Source Code	<pre>def __repr__(self) :     return str(self.data)</pre>

Nama Fungsi	<code>backward(self, grad=None)</code>
Deskripsi Fungsi	Menghitung backpropagation dari gradien yang mengalir ke node ini



Source Code	<pre> def backward(self, grad=None):     if grad is None:         grad = np.ones_like(self.data)     self.grad = grad if self.grad is None else self.grad + grad      # Topological sort to determine correct backward order     topo = []     visited = set()     def build_topo(t):         if t not in visited:             visited.add(t)             for child in t._prev:                 build_topo(child)             topo.append(t)     build_topo(self)      for t in reversed(topo):         t._backward() </pre>
-------------	--

Nama Fungsi	zero_grad(self)
Deskripsi Fungsi	Me reset gradien menjadi 0
Source Code	<pre> def zero_grad(self):     """Reset gradients to zero."""     self.grad = np.zeros_like(self.data) </pre>

Nama Fungsi	__add__(self,other)
Deskripsi Fungsi	Operator “+” overloading dan mendefinisikan backward propagationnya.
Source Code	<pre> def __add__(self, other):     if not isinstance(other, Value):         other = Value(other)     out = Value(self.data + other.data)     out._prev = {self, other}     def _backward():         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + out.grad         other.grad = (other.grad if other.grad is not None else np.zeros_like(other.data)) + out.grad     out._backward = _backward     out._op = '+'     return out </pre>

Nama Fungsi	__radd__(self,other)
-------------	----------------------

Deskripsi Fungsi	Operator “+” overloading untuk Value di sebelah kanan operator.
Source Code	<pre>def __radd__(self, other) :     return self + other</pre>

Nama Fungsi	__sub__(self, other)
Deskripsi Fungsi	Operator “-” overloading dengan memanfaatkan __add__
Source Code	<pre>def __sub__(self, other):     return self.__add__(-other)</pre>

Nama Fungsi	__rsub__(self, other)
Deskripsi Fungsi	Operator “-” overloading untuk kasus Value di sebelah kanan operator.
Source Code	<pre>def __rsub__(self, other) :     return (-self) + other</pre>

Nama Fungsi	unbroadcast(self, grad, shape)
Deskripsi Fungsi	Menyesuaikan ukuran gradien agar sesuai dengan ukuran asli operand yang di-broadcast dalam operasi elemen-wise
Source Code	<pre>def unbroadcast(self, grad, shape):     """     Reduces grad sehingga memiliki bentuk yang sama dengan `shape`.     Fungsi ini melakukan penjumlahan (sum) pada axis yang tidak sesuai.     """     # Jika grad memiliki dimensi lebih dari shape yang diinginkan, jumlahkan sumbu ekstra     while grad.ndim &gt; len(shape):         grad = grad.sum(axis=0)     # Periksa tiap axis, jika tidak sesuai, jumlahkan pada axis tersebut     for i, dim in enumerate(shape):         if grad.shape[i] != dim:             grad = grad.sum(axis=i, keepdims=True)     return grad</pre>

Nama Fungsi	__mul__(self, other)
Deskripsi Fungsi	Operator “*” overloading dan mengimplementasi backpropagation nya.

Source Code	<pre> def __mul__(self, other):     if not isinstance(other, Value):         other = Value(other)     out = Value(self.data * other.data)     out._prev = {self, other}     def _backward():         # Hitung gradien dasar untuk masing-masing operand         grad_self = other.data * out.grad         grad_other = self.data * out.grad          # Unbroadcast gradien sesuai dengan shape asli dari self.data dan other.data         grad_self = self.unbroadcast(grad_self, self.data.shape)         grad_other = self.unbroadcast(grad_other, other.data.shape)          self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + grad_self         other.grad = (other.grad if other.grad is not None else np.zeros_like(other.data)) + grad_other     out._backward = _backward     out._op = '*'     return out </pre>
-------------	--

Nama Fungsi	reciprocal(self)
Deskripsi Fungsi	Mengembalikan 1/self.
Source Code	<pre> def reciprocal(self):     out = Value(1.0 / self.data)     out._prev = {self}      def _backward():         grad_input = -out.data * out.data * out.grad         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + grad_input      out._backward = _backward     out._op = 'reciprocal'     return out </pre>

Nama Fungsi	__truediv__(self, other)
Deskripsi Fungsi	Operator “/” overloading
Source Code	<pre> def __truediv__(self, other):     if not isinstance(other, Value):         other = Value(other)      return self * other.reciprocal() </pre>

Nama Fungsi	__rtruediv__(self, other)
Deskripsi Fungsi	Operator “/” overloading untuk kasus Value berada di sebelah kanan operator
Source Code	<pre> def __rtruediv__(self, other):     return Value(other) * self.reciprocal() </pre>

Nama Fungsi	<code>__gt__(self, other)</code>
Deskripsi Fungsi	Operator “>” overloading
Source Code	<pre>def __gt__(self, other):     if not isinstance(other, Value):         other = Value(other)      out = Value(np.where(self.data &gt; other.data, 1.0, 0.0))     out._prev = {self, other}      def _backward():         pass      out._backward = _backward     out._op = '&gt;'     return out</pre>

Nama Fungsi	<code>__len__(self)</code>
Deskripsi Fungsi	Mengembalikan jumlah elemen
Source Code	<pre>def __len__(self):     return len(self.data)</pre>

Nama Fungsi	<code>matmul(self, other)</code>
Deskripsi Fungsi	Melakukan perkalian matriks
Source Code	<pre>def matmul(self, other):     out = Value(self.data.dot(other.data))     out._prev = {self, other}     def _backward():         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + out.grad.dot(other.data.T)         other.grad = (other.grad if other.grad is not None else np.zeros_like(other.data)) + self.data.T.dot(out.grad)     out._backward = _backward     out._op = 'matmul'     return out</pre>

Nama Fungsi	<code>relu(self)</code>
Deskripsi Fungsi	Melakukan ReLU pada self dan juga mendefinisikan backpropagationnya

Source Code	<pre>def relu(self):     out = Value(np.maximum(0, self.data))     out._prev = {self}     def _backward():         grad = (self.data &gt; 0).astype(np.float32)         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + grad * out.grad     out._backward = _backward     out._op = "relu"     return out</pre>
-------------	--

Nama Fungsi	log(self)
Deskripsi Fungsi	Melakukan log pada self dan menangani nilai $< \text{eps}$ ( $1\text{e-}7$ )
Source Code	<pre>def log(self):     eps = 1e-7     out = Value(np.log(np.maximum(self.data, eps)))     out._prev = {self}     def _backward():         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + (1 / np.maximum(self.data, eps)) * out.grad     out._backward = _backward     out._op = 'log'     return out</pre>

Nama Fungsi	__neg__(self)
Deskripsi Fungsi	Mengembalikan hasil negasi value
Source Code	<pre>def __neg__(self):     return self * (-1)</pre>

Nama Fungsi	transpose(self)
Deskripsi Fungsi	Mengembalikan transpose dari data
Source Code	<pre>def transpose(self):     out = Value(self.data.T)     out._prev = {self}     def _backward():         self.grad = (self.grad if self.grad is not None else np.zeros_like(self.data)) + out.grad.T     out._backward = _backward     out._op = 'transpose'     return out</pre>

Nama Fungsi	T(self)
Deskripsi Fungsi	Properti untuk mendapatkan hasil transpose
Source Code	<pre>@property def T(self):     return self.transpose()</pre>

Nama Fungsi	mean(t:Value) -> Value
Deskripsi Fungsi	Menghitung nilai rata-rata dari Value
Source Code	<pre>def mean(t: Value) -&gt; Value:     data = np.mean(t.data)     out = Value(data)     out._prev = {t}     def _backward():         grad = np.ones_like(t.data) * (1 / t.data.size) * out.grad         t.grad = (t.grad if t.grad is not None else np.zeros_like(t.data)) + grad     out._backward = _backward     out._op = 'mean'     return out</pre>

Nama Fungsi	sum_axis(t:Value, axis:int) -> Value
Deskripsi Fungsi	Menjumlahkan data pada usatu axis
Source Code	<pre>def sum_axis(t: Value, axis: int) -&gt; Value:     data = np.sum(t.data, axis=axis, keepdims=True)     out = Value(data)     out._prev = {t}     def _backward():         grad = out.grad * np.ones_like(t.data)         t.grad = (t.grad if t.grad is not None else np.zeros_like(t.data)) + grad     out._backward = _backward     out._op = 'sum_axis'     return out</pre>

Nama Fungsi	trace(root:Value) -> Tuple[set[Value], Set[Tuple[Value, Value]]]
Deskripsi Fungsi	Melacak node dan edge dalam computational graph dari Value yang diberikan
Source Code	<pre>def trace(root: Value) -&gt; Tuple[Set[Value], Set[Tuple[Value, Value]]]:     nodes, edges = set(), set()      def build(v: Value) -&gt; None:         if v not in nodes:             nodes.add(v)             for child in v._prev:                 edges.add((child, v))                 build(child)      build(root)     return nodes, edges</pre>

Nama Fungsi	draw_dot(root:Value) -> Diagram
-------------	---------------------------------

Deskripsi Fungsi	Menghasilkan visualisasi computational graph dari Value dengan Digraph
Source Code	<pre> def draw_dot(root: Value) -&gt; Digraph:     dot = Digraph(format='svg', graph_attr={'rankdir': 'LR'})     nodes, edges = trace(root)      for n in nodes:         name = str(id(n))         label = n.label if hasattr(n, "label") and n.label else "const " + n.__repr__()         dot.node(name, label=f"{label}   data {str(n.data)}   grad {str(n.grad)}", shape='record')         if hasattr(n, '_op') and n._op:             op_name = name + n._op             dot.node(op_name, label=n._op, shape='circle')             dot.edge(op_name, name)      for n1, n2 in edges:         name1 = str(id(n1))         name2 = str(id(n2))         if hasattr(n2, '_op') and n2._op:             op_name = name2 + n2._op             dot.edge(name1, op_name)         else:             dot.edge(name1, name2)      return dot </pre>

#### 4. Implementasi activation

Deskripsi	File untuk menyimpan implementasi fungsi aktivasi untuk digunakan dalam membentuk FFNN. Jenis fungsi aktivasi yang diimplementasi adalah fungsi aktivasi wajib dan fungsi aktivasi bonus: Linear, tanh, ReLU, Sigmoid, Softmax, Leaky ReLU, dan Swish.
-----------	--

Nama Fungsi	def exp(x: Value)
Deskripsi Fungsi	Implementasi fungsi eksponen untuk keperluan fungsi aktivasi.
Source Code	<pre> def exp(x: Value):     t = np.exp(x.data)     out = Value(t)     out._prev = {x}      def _backward():         if x.grad is None:             x.grad = np.zeros_like(x.data)             x.grad += t * out.grad      out._backward = _backward     out._op = 'exp'     return out </pre>

Nama Fungsi	def linear(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi linear
Source Code	<pre>def linear(x:Value):     return x</pre>

Nama Fungsi	def tanh(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi tanh
Source Code	<pre>def tanh(x: Value):     exp_x = exp(x)     exp_neg_x = 1 / exp_x     return (exp_x - exp_neg_x) / (exp_x + exp_neg_x)</pre>

Nama Fungsi	def relu(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi ReLU
Source Code	<pre>def relu(x: Value):     return x * (x.data &gt; 0)</pre>

Nama Fungsi	def leaky_relu(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi leaky ReLU
Source Code	<pre>def leaky_relu(x: Value, alpha=0.01):     out = x * (x.data &gt; 0) + alpha * x * (x.data &lt;= 0)     return out</pre>

Nama Fungsi	def swish(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi swish



Source Code	<pre>def swish(x: Value):     return x * sigmoid(x)</pre>
-------------	---

Nama Fungsi	def sigmoid(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi sigmoid
Source Code	<pre>def sigmoid(x: Value):     return 1 / (1 + exp(-x))</pre>

Nama Fungsi	def softmax(x: Value)
Deskripsi Fungsi	Implementasi fungsi aktivasi softmax
Source Code	<pre>def softmax(x: Value):     max_x = Value(np.max(x.data, axis=1, keepdims=True))     exp_x = exp(x - max_x)     sum_exp = Value(np.sum(exp_x.data, axis=1, keepdims=True))     out = exp_x / sum_exp      out._prev = {x}      def _backward():         # Compute softmax Jacobian-vector product         if x.grad is None:             x.grad = np.zeros_like(x.data)          # Gradient computation for softmax         # s * (δ<sub>ij</sub> - s<sub>j</sub>)         # out.data is the softmax probabilities         # out.grad is the incoming gradient         grad = out.data * (out.grad - np.sum(out.data * out.grad, axis=1, keepdims=True))          x.grad += grad      out._backward = _backward     out._op = 'softmax'     return out</pre>

## 5. Implementasi Layer

Deskripsi	Class yang berfungsi sebagai layer dari sebuah FFNN.
-----------	--

Nama Fungsi	<pre>__init__(self, n_inputs: int, n_neurons: int,          activation: Callable[[Value], Value] = linear,          weight_init: Callable[..., np.ndarray] = lambda n_in,          n_out: np.zeros((n_out, n_in)),</pre>
-------------	--

	weight_init_kwargs: Optional[Dict] = None, rmsnorm: bool = False, eps: float = 1e-8)
Deskripsi Fungsi	Fungsi inisialisasi untuk Layer
Source Code	<pre>def __init__(self, n_inputs: int, n_neurons: int,               activation: Callable[[Value], Value] = linear,               weight_init: Callable[[..., np.ndarray]] = lambda n_in, n_out: np.zeros((n_out, n_in)),               weight_init_kwargs: Optional[Dict] = None,               rmsnorm: bool = False,               eps: float = 1e-8):     if weight_init_kwargs is None:         weight_init_kwargs = {}      self.activation = activation     self.W = Value(data=weight_init(n_inputs, n_neurons, **weight_init_kwargs))     self.b = Value(data=np.zeros(n_neurons))     self.rmsnorm = rmsnorm     self.eps = eps     if self.rmsnorm:         self.gamma = Value(np.random.randn(1, n_neurons))     else:         self.gamma = None</pre>

Nama Fungsi	parameters(self) -> List[Value]
Deskripsi Fungsi	Mengembalikan list of bobot dan nilai dari Self
Source Code	<pre>def parameters(self) -&gt; List[Value]:     return [self.W, self.b]</pre>

Nama Fungsi	rmsnorm_func(self, x:Value) -> Value
Deskripsi Fungsi	Melakukan RMS dan dinormalisasi untuk x
Source Code	<pre>def rmsnorm_func(self, x: Value) -&gt; Value:     ms = Value(np.mean(x.data ** 2, axis=1, keepdims=True))     rms = Value(np.sqrt(ms.data + self.eps))     normed = x / rms     if self.gamma is not None:         normed = normed * self.gamma     return normed</pre>

Nama Fungsi	__call__(self, x:Value) -> Value
Deskripsi Fungsi	Menerapkan operasi linier $z = xW^T + b$ , dengan opsional normalisasi RMS, lalu menerapkan fungsi aktivasi.

Source Code	<pre>def __call__(self, x: Value) -&gt; Value:     z = x.matmul(self.W.T) + self.b     if self.rmsnorm:         z = self.rmsnorm_func(z)     return self.activation(z)</pre>
-------------	--

## 6. Implementasi FFNN

Deskripsi	Class yang berfungsi sebagai layer dari sebuah FFNN.
-----------	--

Nama Fungsi	<pre>def __init__(     self,     layers : list[Layer] = None,     layer_sizes: List[int] = None,     activations: List[Callable[[Value], Value]] = None,     loss_fn: Callable[[Value, Value], Value] = None,     weight_init: Callable[[int, int], Value] = zero_init,     lr: float = 0.01, )</pre>
Deskripsi Fungsi	Fungsi inisialisasi untuk FFNN
Source Code	<pre>def __init__(     self,     layers : list[Layer] = None,     layer_sizes: List[int] = None,     activations: List[Callable[[Value], Value]] = None,     loss_fn: Callable[[Value, Value], Value] = None,     weight_init: Callable[[int, int], Value] = zero_init,     lr: float = 0.01, ):     # assert len(activations) == len(layer_sizes) - 1,     # "Each layer (except input) must have an activation function"      self.layers = [         Layer(             n_inputs=layer_sizes[i],             n_neurons=layer_sizes[i + 1],             activation=activations[i],             weight_init=weight_init         )         for i in range(len(layer_sizes) - 1)     ] if layers is None else layers     self.learning_rate = lr     self.loss_fn = loss_fn</pre>

Nama Fungsi	<code>__call__(self, x: Value) -&gt; Value</code>
Deskripsi Fungsi	Melakukan forward pass untuk semua layer
Source Code	<pre>def __call__(self, x: Value) -&gt; Value:     for layer in self.layers:         x = layer(x)     return x</pre>

Nama Fungsi	<code>parameters(self) -&gt; List[Value]</code>
Deskripsi Fungsi	Mengembalikan semua parameter dari setiap layer
Source Code	<pre>def parameters(self) -&gt; List[Value]:     params = []     for layer in self.layers:         params.extend(layer.parameters())     return params</pre>

Nama Fungsi	<code>backward(self, loss: Value)</code>
Deskripsi Fungsi	Melakukan backward pass dengan menambah regulasi (L1 dan/atau L2) sebelum melakukan backward pass
Source Code	<pre>def backward(self, loss: Value):     L_new = loss      for layer in self.layers:         for param in layer.parameters():             if self.lambda_l1 &gt; 0:                 L_new += self.lambda_l1 * param.abs().sum()              if self.lambda_l2 &gt; 0:                 L_new += self.lambda_l2 * (param * param).sum()      L_new.backward()</pre>

Nama Fungsi	<code>update_weights(self)</code>
Deskripsi Fungsi	Memperbaharui bobot menggunakan gradien hasil perhitungan backward pass

Source Code	<pre> def update_weights(self):     for param in self.parameters():         grad = param.grad          if grad.shape != param.data.shape:             grad = grad.sum(axis=0)          param.data -= self.learning_rate * grad </pre>
-------------	---

Nama Fungsi	<pre> def train(     self,     training_data,     training_target,     max_epoch,     error_threshold,     batch_size,     validation_data=None,     validation_target=None,     verbose=False ) </pre>
-------------	---

Deskripsi Fungsi	Melatih FFNN dengan validasi optimal
------------------	--------------------------------------

Source Code	<pre> training_loss_history = [] validation_loss_history = []  for epoch in range(max_epoch):     training_loss = self._train_epoch(training_data, training_target, batch_size)     training_loss_history.append(training_loss)      validation_loss = None     if validation_data is not None and validation_target is not None:         validation_loss = self._validate(validation_data, validation_target)         validation_loss_history.append(validation_loss)      if verbose:         output_str = f"Epoch {epoch + 1}/{max_epoch}: Training Loss = {training_loss}"         if validation_loss is not None:             output_str += f", Validation Loss = {validation_loss}"         print(output_str)      if training_loss &lt;= error_threshold:         break  return {     'training_loss_history': training_loss_history,     'validation_loss_history': validation_loss_history } </pre>
-------------	--

Nama Fungsi	<code>_train_epoch(self, training_data: Value, training_target: Value, batch_size: int)</code>
-------------	--

Deskripsi Fungsi	Melakukan pelatihan secara mini batch untuk 1 epoch
Source Code	<pre> def _train_epoch(self, training_data : Value, training_target:Value, batch_size: int):     total_loss = 0     num_samples = len(training_data)     num_batches = (num_samples + batch_size - 1) // batch_size # menangani batch terakhir yang      for i in range(0, num_samples, batch_size):          batch_data = training_data[i:i+batch_size]         batch_target = training_target[i:i+batch_size]          # Lakukan forward pass untuk batch         output = self(batch_data)          # Hitung loss untuk batch         loss = self.loss_fn(output, batch_target)         total_loss += loss.data          # Backward pass dan update weights         # Reset gradien untuk setiap parameter         for param in self.parameters():             param.zero_grad()         self.backward(loss)         self.update_weights()      # Rata-rata loss per batch     average_loss = total_loss / num_batches     return average_loss </pre>

Nama Fungsi	_validate(self, validation_data: Value, validation_target:Value)
Deskripsi Fungsi	Menghitung average loss dari data validasi
Source Code	<pre> def _validate(self, validation_data: Value, validation_target : Value):     total_loss = 0     for x, y in zip(validation_data, validation_target):         output = self(x)         loss = self.loss_fn(output, y)         total_loss += loss.data     return total_loss / len(validation_data) </pre>

Nama Fungsi	save(self, file_path:str)
Deskripsi Fungsi	Menyimpan model ke file JSON
Source Code	<pre> def save(self, file_path: str):     model_data = {         "layer_sizes": [layer.W.data.shape[1] for layer in self.layers] + [self.layers[-1].W.data.shape[0]],         "weights": [layer.W.data.tolist() for layer in self.layers],         "biases": [layer.b.data.tolist() for layer in self.layers],         "rmsnorm": [layer.rmsnorm for layer in self.layers],         "gamma": [layer.gamma.data.tolist() if layer.rmsnorm else None for layer in self.layers],         "activations": [layer.activation.__name__ for layer in self.layers],         "loss_fn": self.loss_fn.__name__ if self.loss_fn else None,         "learning_rate": self.learning_rate     }     with open(file_path, "w") as f:         json.dump(model_data, f, indent=4) </pre>

Nama Fungsi	load(cls, file_path:str)
Deskripsi Fungsi	Memuat model FFNN dari file JSON
Source Code	<pre> @classmethod def load(cls, file_path: str):     with open(file_path, "r") as f:         model_data = json.load(f)      layer_sizes = model_data["layer_sizes"]     activations = [activations_map[name] for name in model_data["activations"]]     loss_fn = loss_fn_map.get(model_data["loss_fn"], None)     layers = []      for i in range(len(layer_sizes) - 1):         layer = Layer(             n_inputs=layer_sizes[i],             n_neurons=layer_sizes[i + 1],             activation=activations[i],             weight_init=zero_init,             rmsnorm=model_data["rmsnorm"][i]         )         layer.W.data = np.array(model_data["weights"][i])         layer.b.data = np.array(model_data["biases"][i])         if model_data["rmsnorm"][i]:             layer.gamma.data = np.array(model_data["gamma"][i])         layers.append(layer)      return cls(layers=layers, loss_fn=loss_fn, lr=model_data["learning_rate"]) </pre>

## 7. Implementasi visualize

Deskripsi	Class yang berfungsi untuk memvisualisasi perbandingan fungsi loss antara training dan validasi, serta memvisualisasi distribusi bobot dan gradient dalam setiap layer FFNN.
-----------	--

Nama Fungsi	def plot_training_comparison(training_history)
Deskripsi Fungsi	Fungsi untuk plotting perbandingan fungsi loss antara training dan validasi

Source Code	<pre>def plot_training_comparison(training_history):     plt.figure(figsize=(10, 6))     plt.plot(         training_history['training_loss_history'],         label='Loss on Training Data',         color='blue'     )     plt.plot(         training_history['validation_loss_history'],         label='Loss on Validation Data',         color='red',         linestyle='--'     )     plt.title('Training Loss Comparison')     plt.xlabel('Epoch')     plt.ylabel('Loss')     plt.legend()     plt.tight_layout()     plt.show()</pre>
-------------	---

Nama Fungsi	def plot_weight_distribution(ffnn: FFNN)
Deskripsi Fungsi	Fungsi untuk plotting distribusi bobot dan gradien dari setiap layer
Source Code	<pre>def plot_weight_distribution(ffnn: FFNN):     num_layers = len(ffnn.layers)      fig, axes = plt.subplots(num_layers, 2, figsize=(12, 4 * num_layers))      if num_layers == 1:         axes = [axes]      for i, layer in enumerate(ffnn.layers):         weights = layer.W.data.flatten()         gradients = layer.W.grad.flatten() if layer.W.grad is not None else None          axes[i][0].hist(weights, bins=30, color='blue', alpha=0.7, edgecolor='black')         axes[i][0].set_title(f"Layer {i+1} - Weight Distribution")         axes[i][0].set_xlabel("Weight Value")         axes[i][0].set_ylabel("Frequency")          if gradients is not None:             axes[i][1].hist(gradients, bins=30, color='red', alpha=0.7, edgecolor='black')             axes[i][1].set_title(f"Layer {i+1} - Gradient Distribution")             axes[i][1].set_xlabel("Gradient Value")             axes[i][1].set_ylabel("Frequency")         else:             axes[i][1].axis("off")      plt.tight_layout()     plt.show()</pre>

## 8. Implementasi draw\_FFNN

Deskripsi	Class yang berfungsi untuk memvisualisasi graph FFNN
-----------	--



Nama Fungsi	def visualized_FFNN(ffnn: FFNN)
Deskripsi Fungsi	Fungsi untuk memvisualisasikan graph FFNN
Source Code	<pre> def visualize_FFNN(ffnn: FFNN):     nodes, edges = [], []      n_inputs = ffnn.layers[0].W.data.shape[1]     input_nodes = [f"x{i+1}" for i in range(n_inputs)]      y_offset = 0     max_nodes_per_layer = max([len(layer.W.data) for layer in ffnn.layers] + [n_inputs])     layer_height = max_nodes_per_layer * 150      layer_start_y = y_offset + (layer_height - n_inputs * 150) / 2     prev_layer_nodes = input_nodes     nodes.extend([{"data": {"id": node, "label": node}, "classes": "input", "position": {"x": 0, "y": i * 150 + layer_start_y}} for i, node in enumerate(input_nodes)])      layer_distance = 100 * max_nodes_per_layer     for layer_idx, layer in enumerate(ffnn.layers):         n_neurons = layer.W.data.shape[0]         sum_nodes = [f"S{layer_idx + 1}_N{n_idx + 1}" for n_idx in range(n_neurons)]         act_nodes = [f"A{layer_idx + 1}_N{n_idx + 1}" for n_idx in range(n_neurons)]          layer_start_y = y_offset + (layer_height - n_neurons * 150) / 2          nodes.extend([{"data": {"id": node, "label": node}, "classes": "sum", "position": {"x": (layer_idx + 1) * layer_distance, "y": i * 150 + layer_start_y}} for i, node in </pre>

```

enumerate(sum_nodes)])
        nodes.extend([{"data": {"id": node,
"label": node}, "classes": "activation",
"position": {"x": (layer_idx + 1) *
layer_distance + 150, "y": i * 150 +
layer_start_y}} for i, node in
enumerate(act_nodes)])

        for neuron_idx in range(n_neurons):
            for prev_idx, prev_node in
enumerate(prev_layer_nodes):
                weight_value =
layer.W.data[neuron_idx, prev_idx]
                weight_grad = layer.W.grad
                label_extension =
f"\ndw={weight_grad[neuron_idx, prev_idx]}" if
weight_grad is not None else ""
                edges.append({
                    "data": {"id":
f"{prev_node}-{sum_nodes[neuron_idx]}", "source":
prev_node, "target": sum_nodes[neuron_idx],
"weight":
f"w={weight_value:.2f}{label_extension}",
                    "classes": "hidden-label"
                })

                bias_value = layer.b.data[neuron_idx]
                bias_grad = layer.b.grad
                label_extension =
f"\ndb={np.sum(bias_grad, axis=0)[neuron_idx]}"
if bias_grad is not None else ""
                bias_node =
f"b{layer_idx}_N{neuron_idx}"
                nodes.append({"data": {"id":
bias_node, "label": bias_node}, "classes":
"bias", "position": {"x": (layer_idx + 1) *
layer_distance - (layer_distance) // 2, "y":
neuron_idx * 150 + layer_start_y}})
                edges.append({

```

```

        "data": {"id":
f"{bias_node}-{sum_nodes[neuron_idx]}", "source":
bias_node, "target": sum_nodes[neuron_idx],
"weight":
f"b={bias_value:.2f}{label_extension}",
        "classes": "hidden-label"
    })

    edges.append({
        "data": {"id":
f"{sum_nodes[neuron_idx]}-{act_nodes[neuron_idx]}",
        "source": sum_nodes[neuron_idx], "target":
act_nodes[neuron_idx], "weight":
layer.activation.__name__,
        "classes": "hidden-label"
    })

    prev_layer_nodes = act_nodes

    output_nodes = [f"y{i+1}" for i in
range(len(prev_layer_nodes))]
    layer_start_y = y_offset + (layer_height -
len(prev_layer_nodes) * 150) / 2
    nodes.extend([{"data": {"id": node, "label":
node}, "classes": "output", "position": {"x":
(len(ffnn.layers) + 1) * layer_distance, "y": i *
150 + layer_start_y}} for i, node in
enumerate(output_nodes)])

    for prev_node, out_node in
zip(prev_layer_nodes, output_nodes):
        edges.append({
            "data": {"id":
f"{prev_node}-{out_node}", "source": prev_node,
"target": out_node, "weight": ""},
            "classes": "hidden-label"
        })

    app = dash.Dash(__name__)

```

```

app.layout = html.Div([
    cyto.Cytoscape(
        id='ffnn-graph',
        layout={'name': 'preset'},
        style={'width': '100%', 'height':
'800px'},
        elements=nodes + edges,
        stylesheet=[
            {"selector": "node", "style": {
                "content": "data(label)",
                "text-valign": "center",
                "text-halign": "center",
                "color": "white",
                "width": "60px",
                "height": "60px",
                "font-size": "18px",
                "border-width": "2px",
                "border-color": "black"
            }},
            {"selector": ".input", "style":
{"background-color": "#FFA500"}},
            {"selector": ".sum", "style":
{"background-color": "#2ca02c"}},
            {"selector": ".activation",
"style": {"background-color": "#1f77b4"}},
            {"selector": ".output", "style":
{"background-color": "#FF4500"}},
            {"selector": ".bias", "style":
{"background-color": "#d62728"}},
            {"selector": "edge", "style": {
                "curve-style": "bezier",
                "target-arrow-shape":
"triangle",
                "label": "",
                "font-size": "0px"
            }},
            {"selector": ".show-label",
"style": {

```

```

        "label": "data(weight)",
        "font-size": "14px",
        "color": "black",
        "text-background-opacity": 1,
        "text-background-color":
"white",
        "text-border-opacity": 1,
        "text-border-width": 1,
        "text-border-color": "black"
    }}
    ]
),
html.Div(id='edge-data')
])

@app.callback(
    Output('ffnn-graph', 'stylesheet'),
    Input('ffnn-graph', 'tapEdgeData'),
    State('ffnn-graph', 'stylesheet')
)
def display_edge_data(edge_data, stylesheet):
    if edge_data:
        edge_id = edge_data['id']
        found = False
        for style in stylesheet:
            if style['selector'] == f'edge[id
= "{edge_id}"]':
                if 'label' in style['style']
and style['style']['label'] == 'data(weight)':
                    style['style']['label'] =
''
style['style']['font-size'] = '0px'
                else:
                    style['style'] = {
                        "label":
"data(weight)",
                        "font-size": "14px",
                        "color": "black",

```

```

"text-background-opacity": 1,

"text-background-color": "white",

"text-border-opacity": 1,
                                "text-border-width":
1,
                                "text-border-color":
"black"
                                }
                                found = True
                                break
                                if not found:
                                    stylesheet.append({
                                        "selector": f'edge[id =
"{edge_id}"]',
                                        "style": {
                                            "label": "data(weight)",
                                            "font-size": "14px",
                                            "color": "black",

"text-background-opacity": 1,
                                "text-background-color":
"white",
                                "text-border-opacity": 1,
                                "text-border-width": 1,
                                "text-border-color":
"black"
                                }
                                })
                                return stylesheet
                                else:
                                    return stylesheet

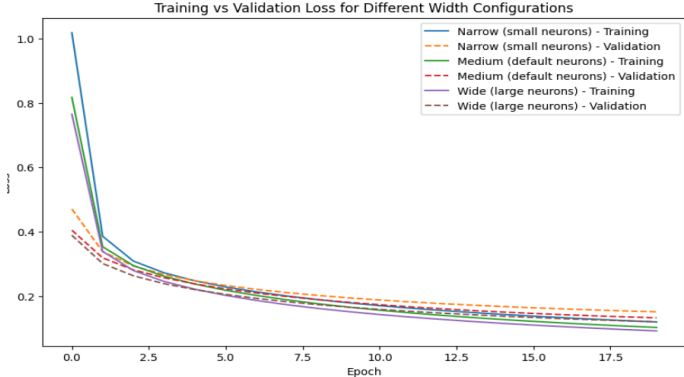
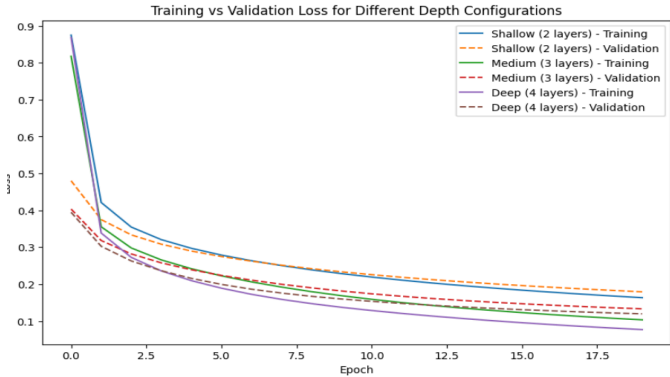
                                return app

```

## D. Hasil Pengujian dan Analisis

Input feature sebanyak 784 (default dataset MNIST 784), output layer 10(default label), semua nya menggunakan categorical cross entropy loss.

### 1. Pengaruh Depth dan Width

max_epoch=20, error_threshold=0.01, batch_size=64, lr=0.01			
Depth Tetap			
Depth	Width	Accuracy(%)	Loss Graph
3	64,32	95.69	
	128,64	96.15	
	256,128	96.47	
Width Tetap			
Width	Depth	Accuracy(%)	Loss Graph
128	2	95.01	
128,64	3	96.20	
128,64,32	4	96.48	

Penambahan width dan depth dapat meningkatkan akurasi karena model dengan kapasitas yang lebih besar mampu menangkap pola-pola yang lebih kompleks [3]. Namun, perlu diperhatikan bahwa peningkatan depth dan width juga dapat meningkatkan risiko

overfitting, terutama jika tidak diimbangi dengan jumlah data yang cukup atau strategi regularisasi yang tepat [3][4]. Untuk mengatasi hal ini, berbagai teknik seperti dropout, regularisasi (misalnya L1 atau L2), dan skip connection dapat diterapkan guna meningkatkan kemampuan generalisasi model [3][4].

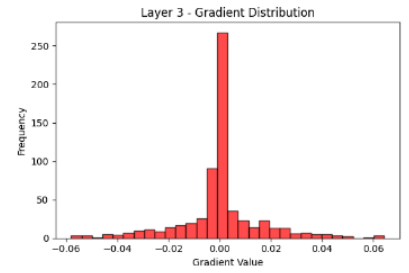
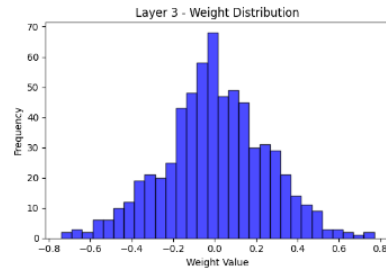
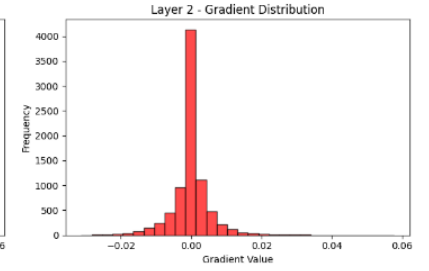
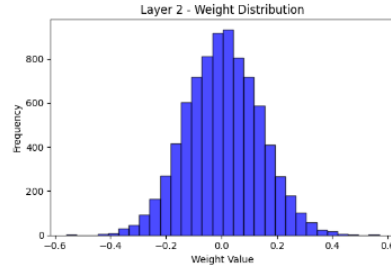
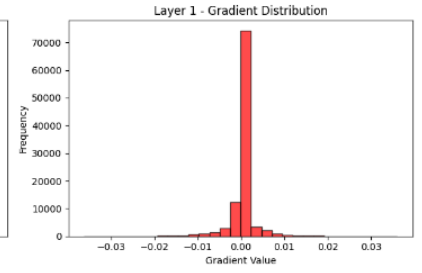
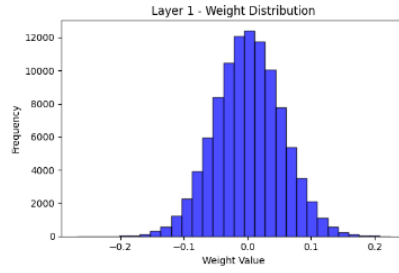
## 2. Pengaruh fungsi aktivasi hidden layer

Layer width= {128,64} max_epoch=10, error_threshold=0.01, batch_size=64, lr=0.01		
Activation	Accuracy(%)	Loss Graph
ReLU	94.91	<p>Training vs Validation Loss for Different Activation Functions</p> <p>The graph displays the training and validation loss for six different activation functions: Linear, Tanh, ReLU, Leaky ReLU, Sigmoid, and Swish. The x-axis represents the number of epochs (0 to 10), and the y-axis represents the loss (0.0 to 2.0). The legend indicates that solid lines represent training loss and dashed lines represent validation loss for each function. ReLU (purple solid line) shows the lowest training loss, reaching approximately 0.2 by epoch 10. Leaky ReLU (pink solid line) follows, reaching approximately 0.3. Sigmoid (yellow solid line) and Swish (orange solid line) show higher training losses, reaching approximately 0.6 and 0.5 respectively. Linear (blue solid line) and Tanh (green solid line) show the highest training losses, reaching approximately 1.0 and 0.8 respectively. The validation losses for all functions are relatively low, staying below 0.5 throughout the training process.</p>
Leaky_Re Lu	95.05	
Linear	91.91	
Tanh	94.09	
Sigmoid	86.02	
Swish	94.23	
Weight and Gradient Distribution		



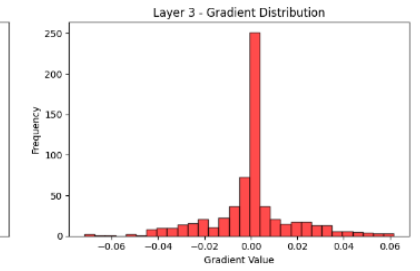
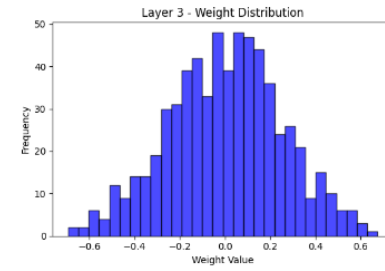
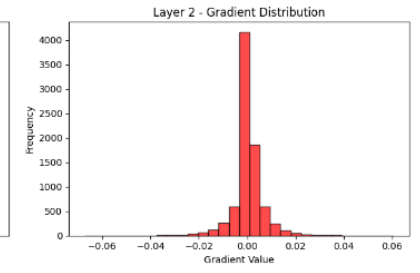
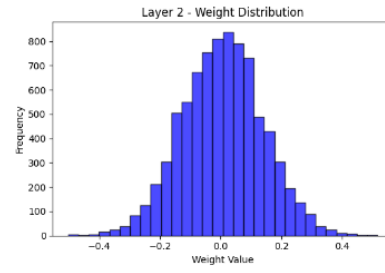
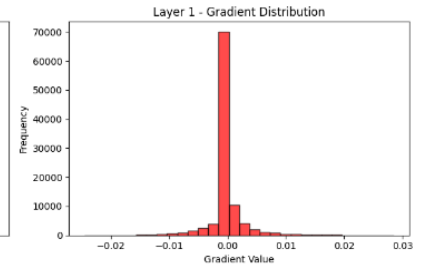
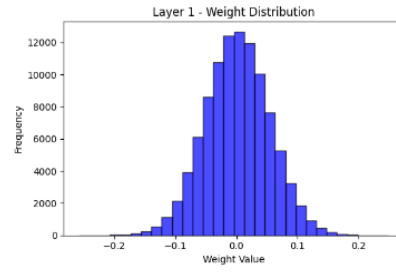
# ReLU

Weight Distribution for ReLU



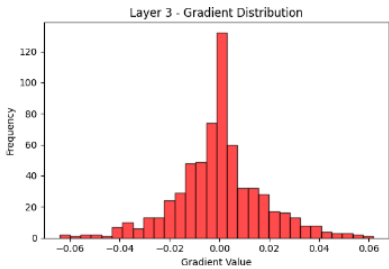
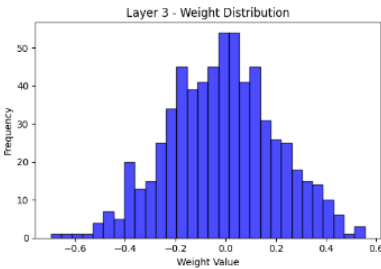
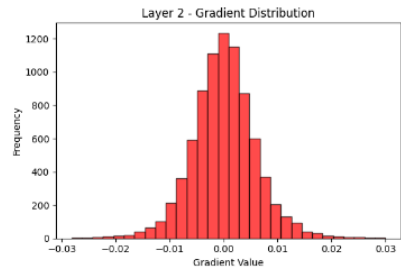
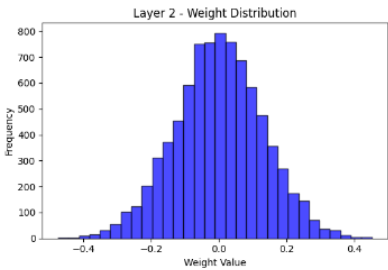
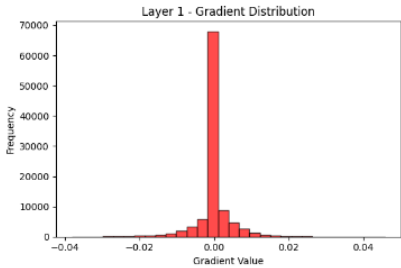
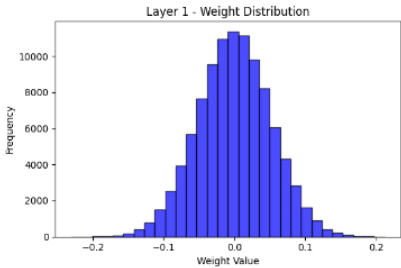
# Leaky\_ReLu

Weight Distribution for Leaky ReLU



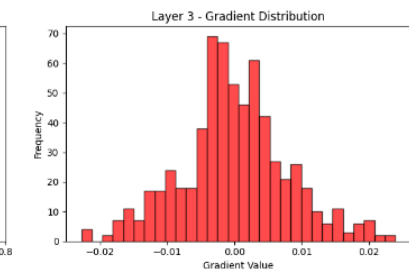
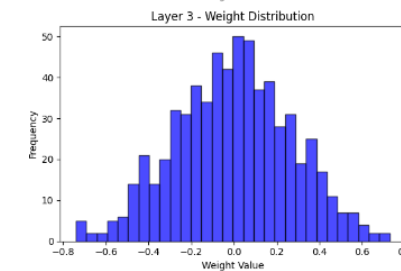
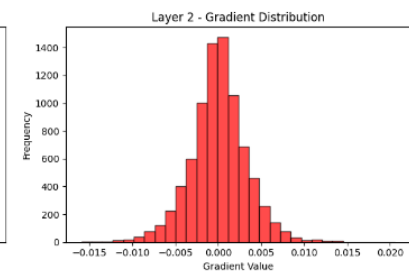
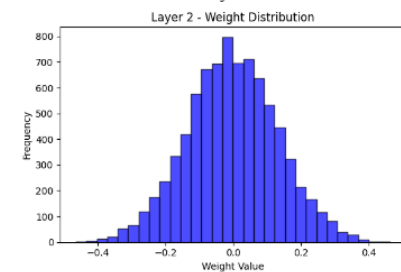
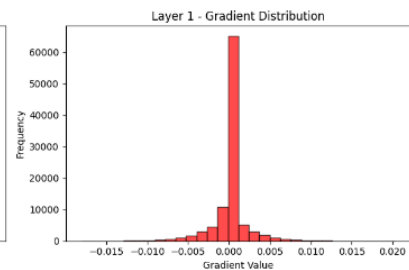
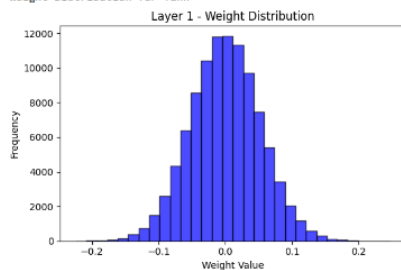
# Linear

Weight Distribution for Linear



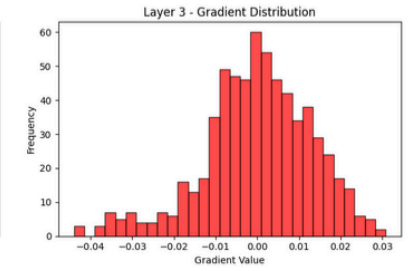
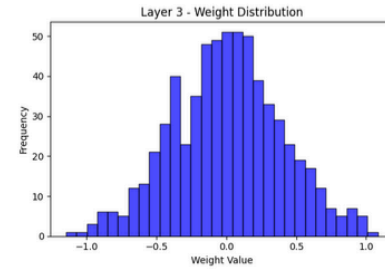
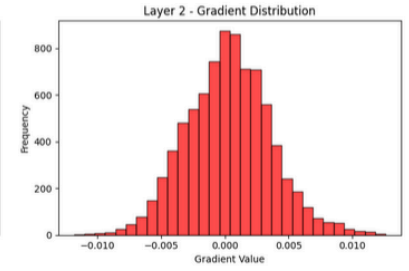
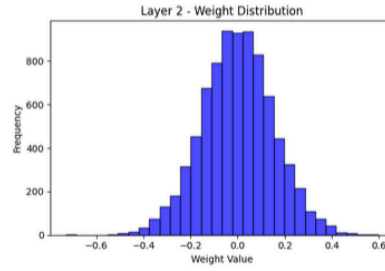
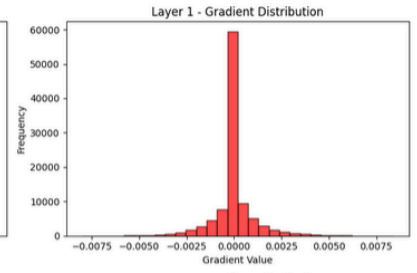
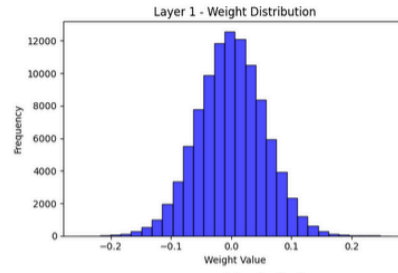
# Tanh

Weight Distribution for Tanh

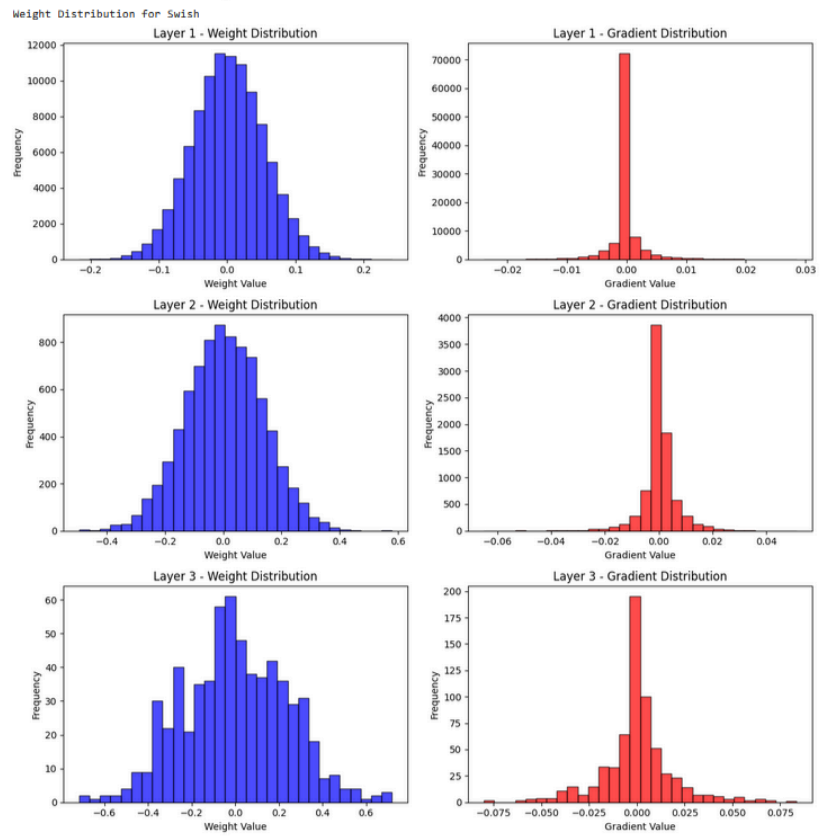


# Sigmoid

Weight Distribution for Sigmoid



## Swish



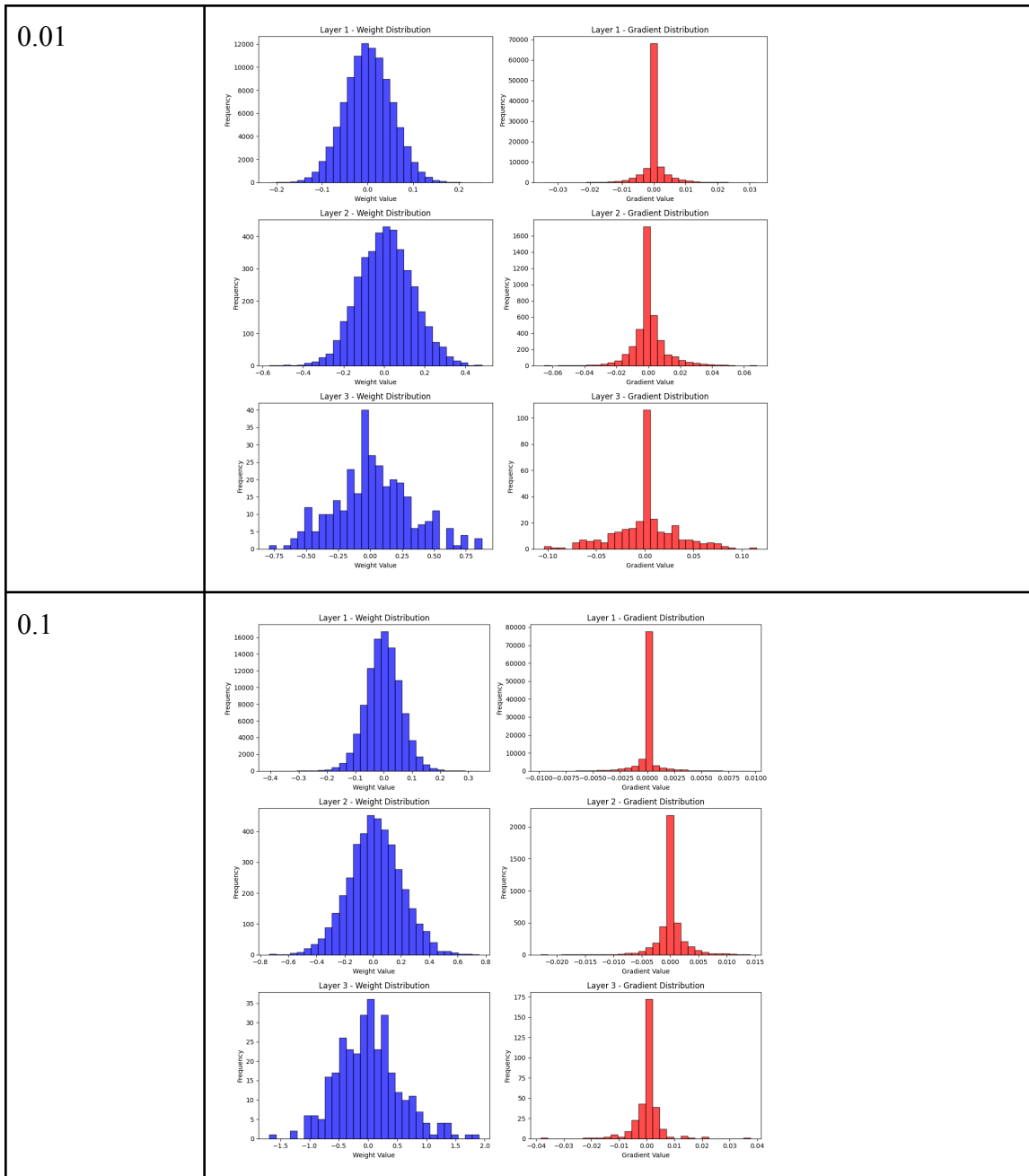
Dari hasil pengujian, dapat disimpulkan bahwa fungsi aktivasi leaky relu, memiliki performa yang paling baik dibanding dengan fungsi aktivasi lainnya. Hal ini karena kemampuannya mengatasi masalah vanishing gradient, yang sering terjadi pada fungsi aktivasi seperti Sigmoid atau ReLU standar. Dalam ReLU, unit yang memiliki output negatif akan menghasilkan gradien nol, menyebabkan beberapa neuron menjadi tidak aktif secara permanen selama pelatihan. Namun, Leaky ReLU mengatasi masalah ini dengan memberikan slope kecil untuk nilai negatif, memungkinkan gradien tetap mengalir meskipun input negatif. Selain itu, dibandingkan dengan Sigmoid dan Tanh, Leaky ReLU memiliki sifat tidak terbatas untuk nilai positif, sehingga dapat menangani rentang nilai yang lebih luas.

### 3. Pengaruh learning rate

Layer Width = {128,32}, hidden\_activation = ReLU, output = softmax  
 Max\_epoch = 20, error threshold=0.01, batch size = 64

Learning Rate	Accuracy(%)	Loss Graph
0.001	91.01	
0.01	96.16	
0.1	97.58	

Training Loss Comparison	
0.001	<div> <div> <p>Layer 1 - Weight Distribution</p> </div> <div> <p>Layer 1 - Gradient Distribution</p> </div> <div> <p>Layer 2 - Weight Distribution</p> </div> <div> <p>Layer 2 - Gradient Distribution</p> </div> <div> <p>Layer 3 - Weight Distribution</p> </div> <div> <p>Layer 3 - Gradient Distribution</p> </div> </div>



Learning rate merupakan hyperparameter yang krusial dalam pelatihan model, karena secara langsung mengontrol besarnya langkah (step size) dalam proses optimisasi. Jika learning rate ditetapkan sangat kecil, model akan melakukan pembaruan parameter secara perlahan. Meskipun hal ini dapat menghasilkan konvergensi yang stabil, model memerlukan jumlah epoch yang lebih banyak untuk mencapai titik minimum dari fungsi loss, yang dapat meningkatkan waktu pelatihan secara signifikan. Di sisi lain, learning

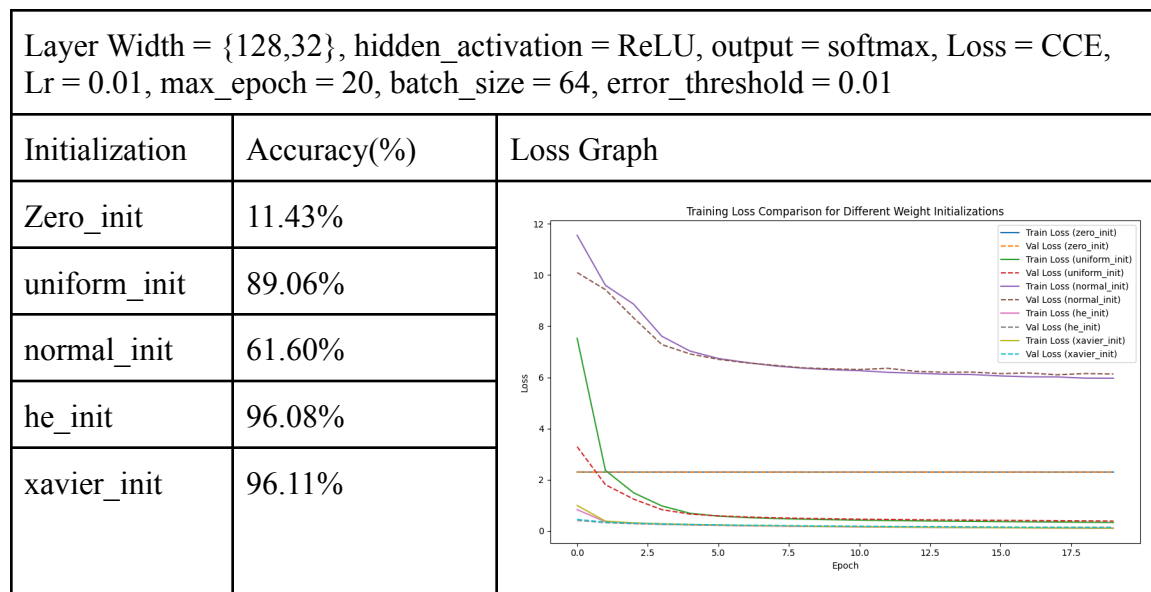


rate yang terlalu besar mempercepat pembelajaran karena setiap pembaruan parameter memiliki dampak yang lebih besar. Namun, peningkatan ini sering disertai dengan risiko ketidakstabilan; pembaruan yang terlalu agresif dapat menyebabkan overshooting, di mana parameter melampaui titik optimal, atau bahkan menyebabkan osilasi dan kegagalan konvergensi.

Oleh karena itu, pemilihan learning rate yang tepat merupakan trade-off antara kecepatan pembelajaran dan kestabilan konvergensi. Strategi seperti adaptive learning rate (misalnya, algoritma Adam atau RMSprop) dan penjadwalan learning rate (learning rate scheduling) sering digunakan untuk mengatasi permasalahan ini, sehingga model dapat secara dinamis menyesuaikan kecepatan pembelajaran selama proses training dan mencapai kinerja yang optimal [3].

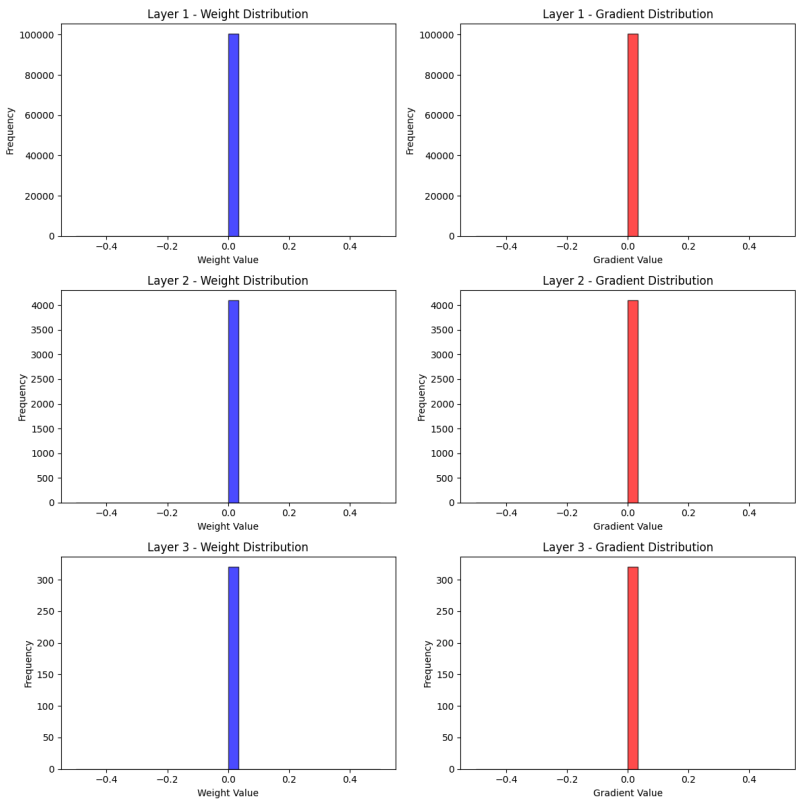
Learning Rate juga perlu disesuaikan dengan ukuran batch, misal batch yang kecil cenderung lebih baik menggunakan learning rate yang kecil karena perlu kestabilan dalam belajar karena estimasi gradiennya memiliki varians yang tinggi [3].

#### 4. Pengaruh inisialisasi bobot

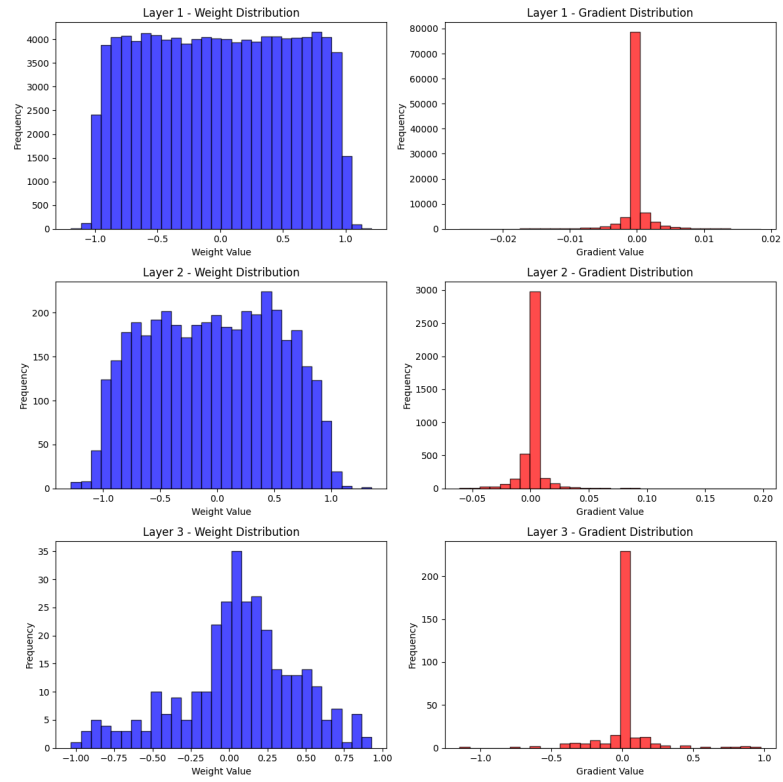


Distribusi bobot dan gradien

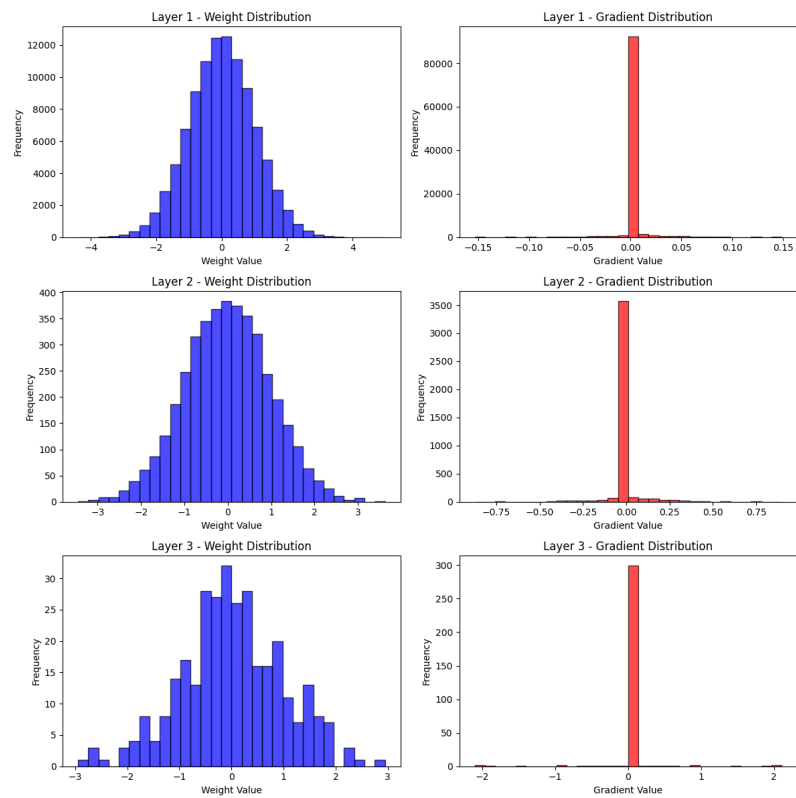
Zero\_init



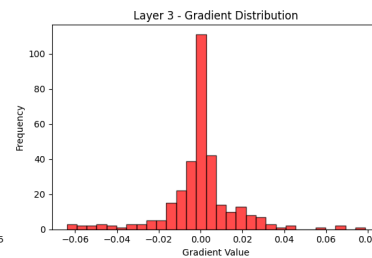
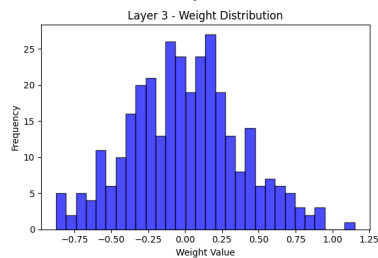
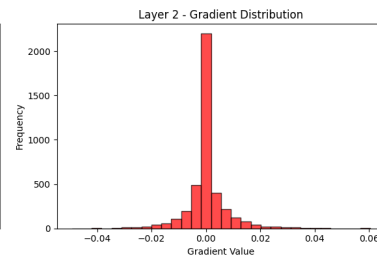
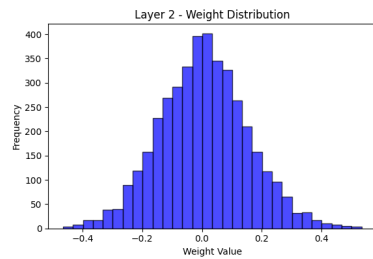
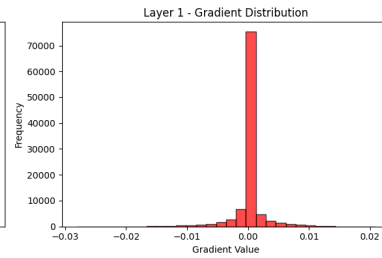
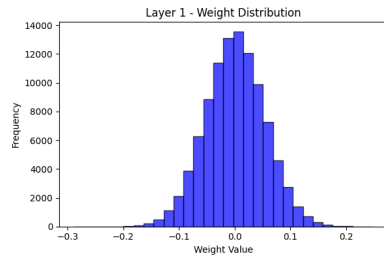
uniform\_init



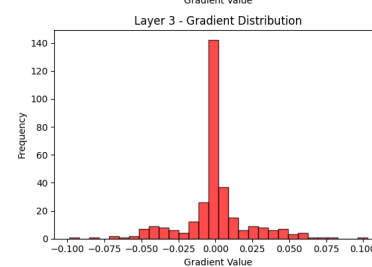
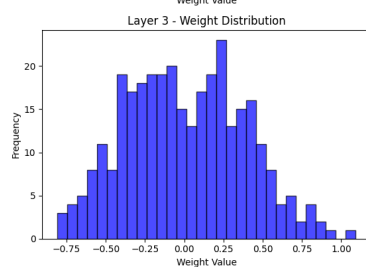
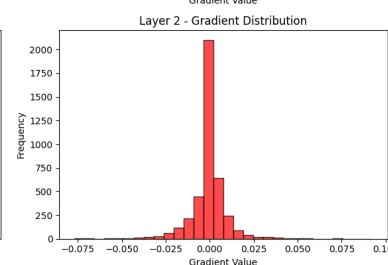
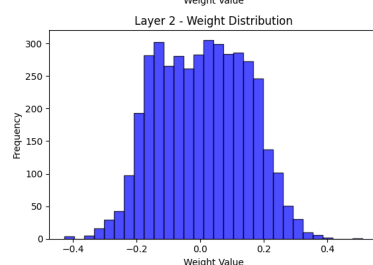
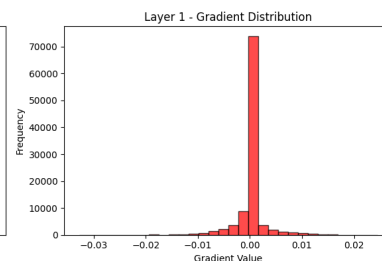
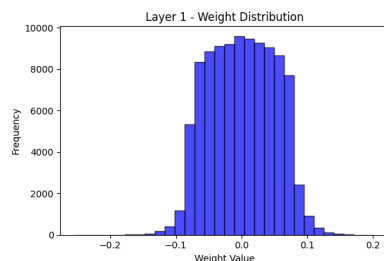
normal\_init



he\_init



xavier\_init

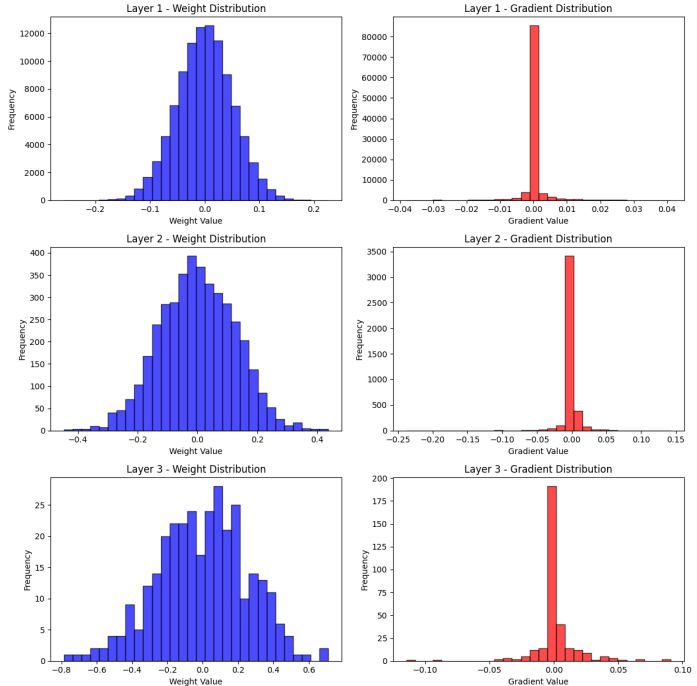
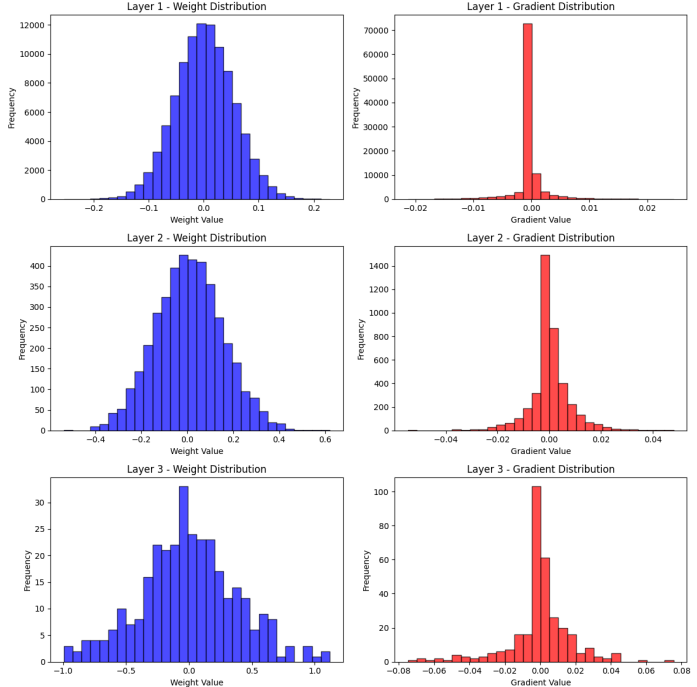


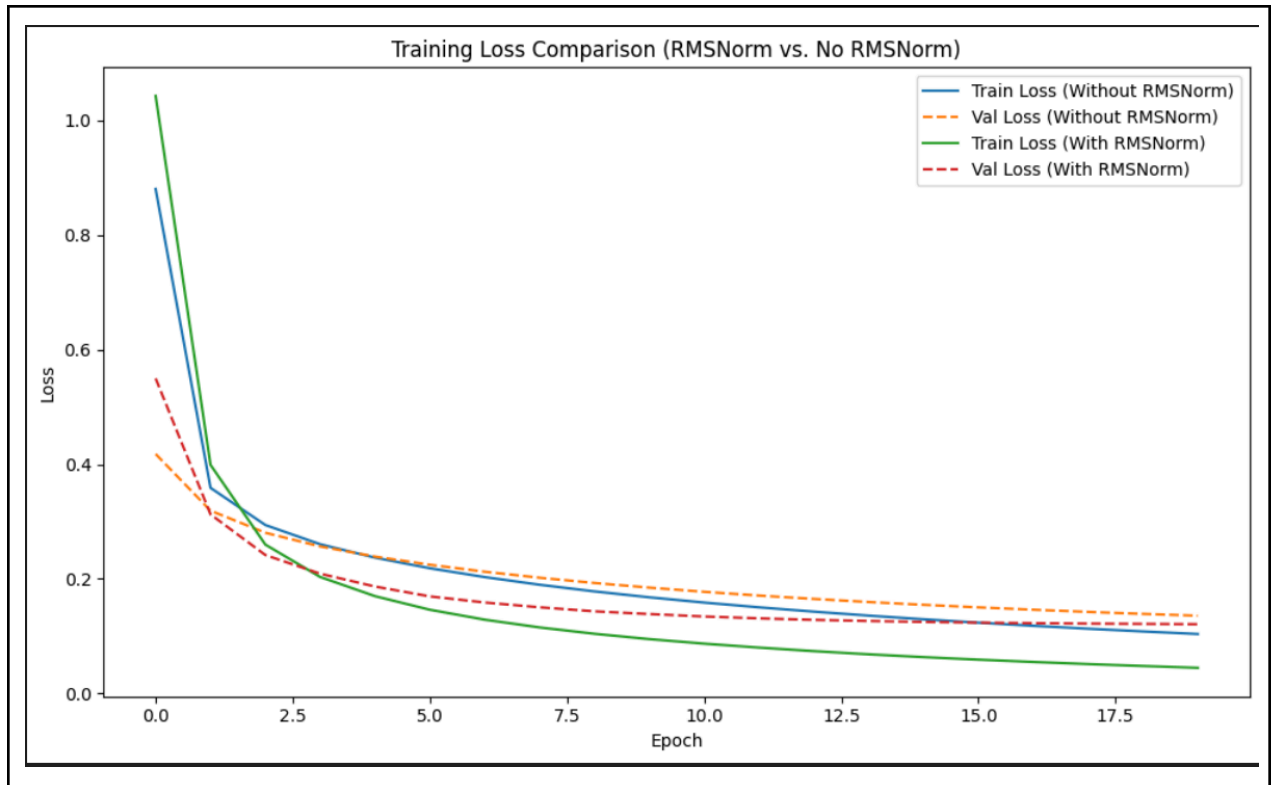
Dari hasil pengujian berbagai metode inisialisasi bobot, ditemukan bahwa tiap metode menghasilkan akurasi yang berbeda-beda, dengan metode Xavier dan He memberikan akurasi tertinggi. Hal ini terjadi karena inisialisasi bobot yang tepat sangat berpengaruh pada kekonvergenan dan pelatihan [3]. Inisialisasi yang optimal dapat menjaga distribusi aktivasi dan gradien di seluruh jaringan, sehingga mencegah masalah vanishing dan exploding gradients. Xavier initialization dirancang untuk menjaga variansi output agar tetap konsisten di setiap layer, sedangkan He initialization mengoptimalkan inisialisasi khusus untuk layer dengan fungsi aktivasi ReLU, yang rentan terhadap saturasi. Dengan menggunakan metode inisialisasi seperti ini, model dapat mencapai konvergensi yang lebih cepat dan stabil, yang pada akhirnya menghasilkan akurasi yang lebih tinggi [3].

Dapat dilihat juga untuk zero\_init semua weight 0 dan setelah pelatihan distribusi weight juga 0 karena gradiennya 0 sehingga akan selalu 0 dan mengandalkan nilai bias.

##### 5. Pengaruh RMSNorm

<i>Layer Width = {128,32}, hidden_activation = ReLU, output = softmax, batch_size = 64, lr = 0.01, init = he_init</i>		
	<i>Accuracy (%)</i>	<i>Weight and Gradient Distribution</i>

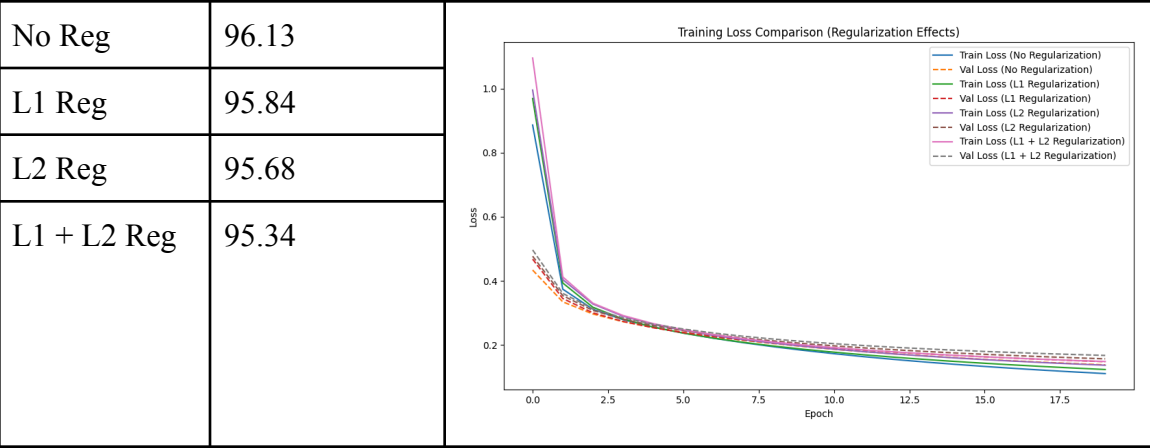
<p><i>RMSNorm</i></p>	<p>96.61</p>	
<p><i>Without RMSNorm</i></p>	<p>96.29</p>	
<p><i>Loss Graph</i></p>		



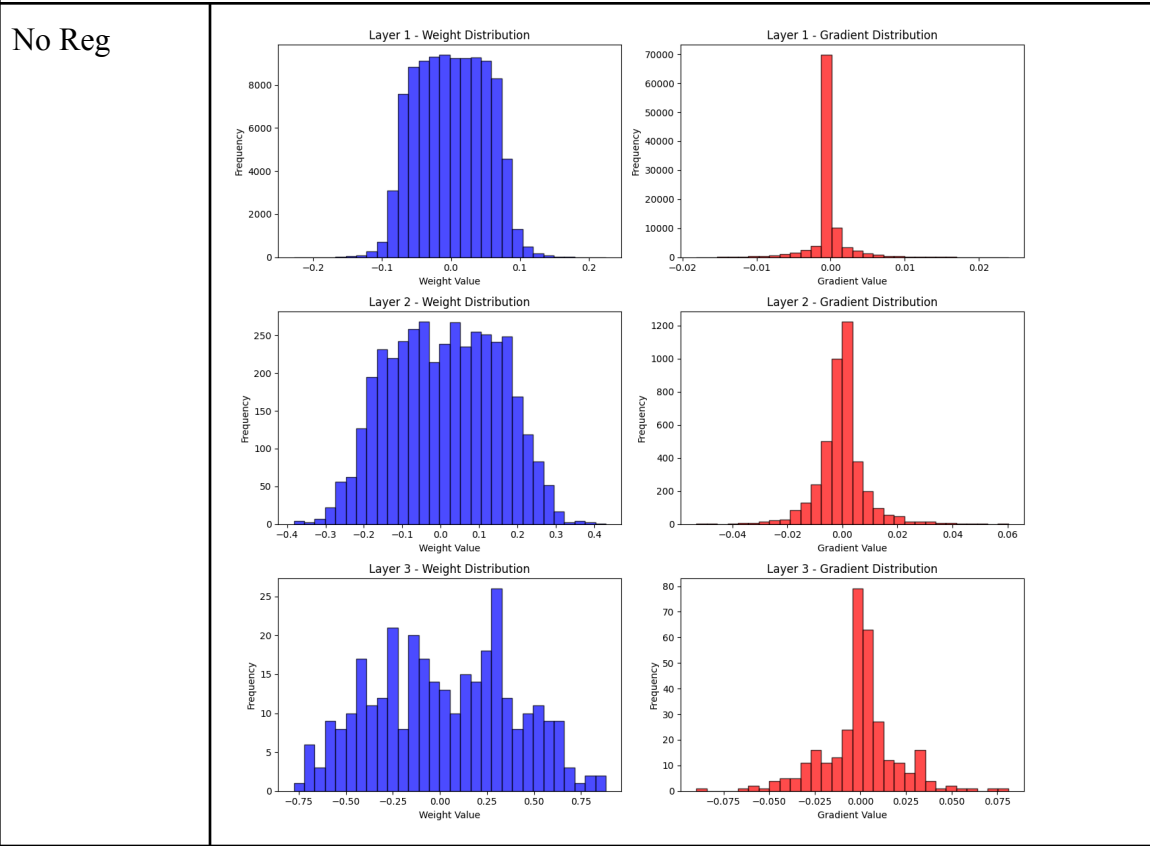
Dapat dilihat dari weight distribution dari hasil pengujian ini dengan menggunakan RMSNorm distribusi weight lebih banyak yang mendekati 0, ini dikarenakan efek dari RMSNorm yang membuat pembelajaran menjadi lebih stabil dan lebih cepat konvergen, sehingga pada akhir pelatihan gradient banyak yang sangat kecil menandakan model stabil dalam pelatihan. Dapat dilihat juga bahwa dengan RMSNorm pembelajaran lebih stabil antara train dan validationnya lebih seimbang. Dari hasil pengujian kami didapat bahwa penggunaan RMSNorm meningkatkan akurasi validasi, hal ini dikarenakan hasil aktivasi di setiap layer dinormalisasi sehingga dapat kurang sensitif terhadap *Invariance*.

## 6. Pengaruh Regularisasi L1 dan L2

<i>Layer = {128,32}, hidden_activation = leaky ReLU, output = softmax, weight_init = Xavier, lr = 0.01, loss = CCE</i>		
Regularisasi	Accuracy(%)	Loss Graph

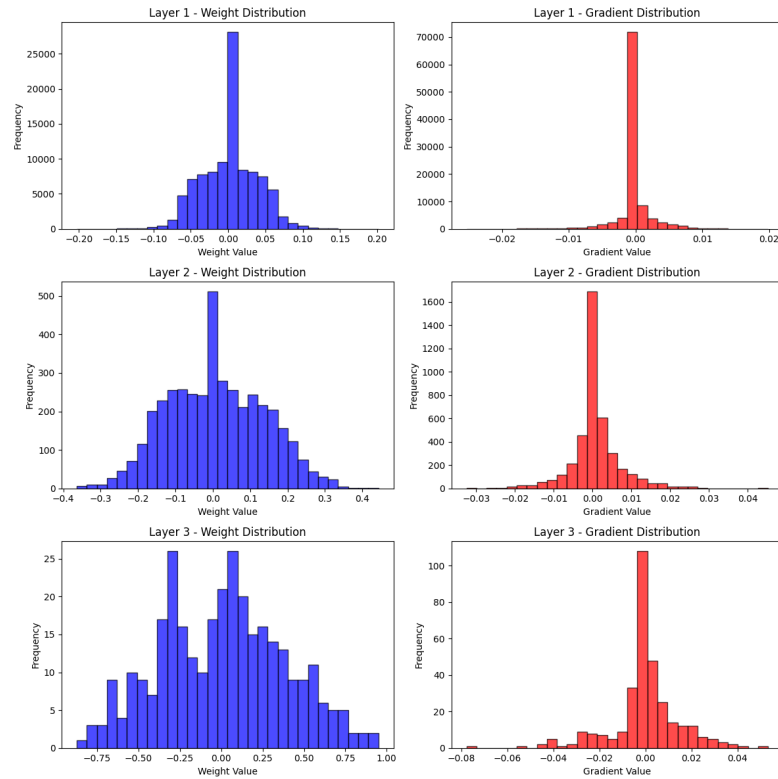


Distribusi bobot dan gradien

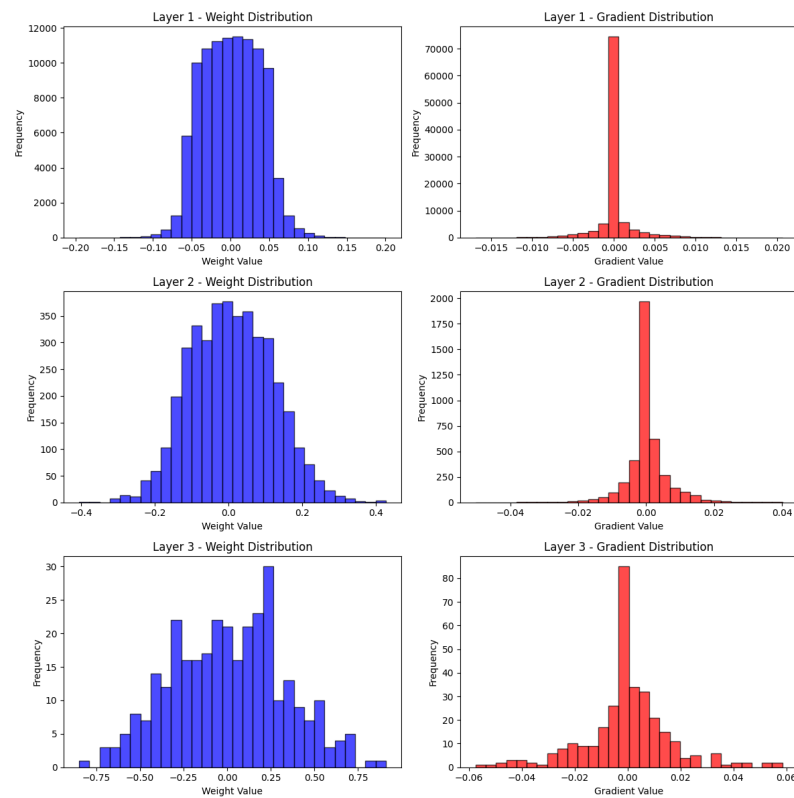




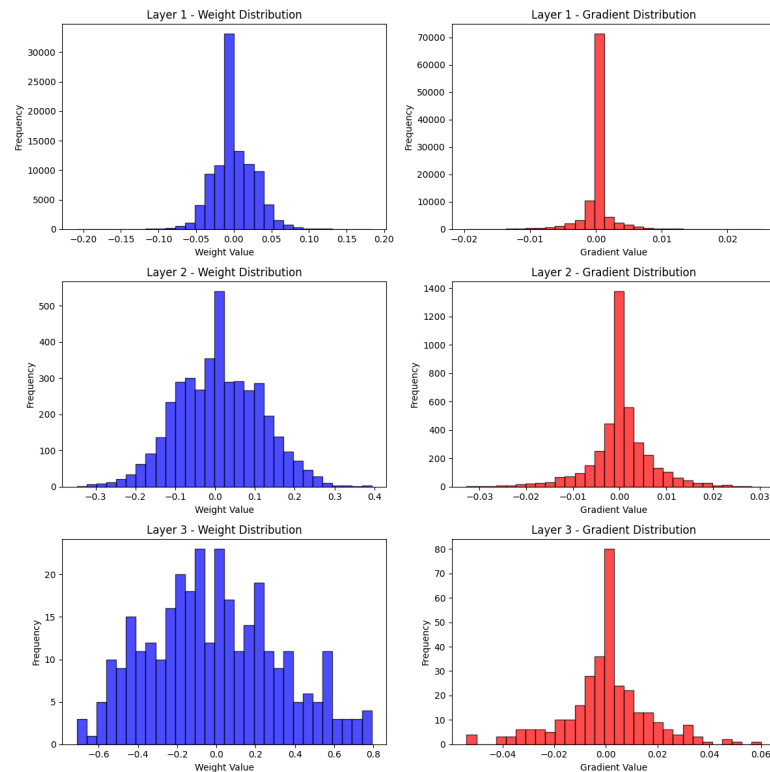
## L1 Reg



## L2 Reg



## L1 + L2 Reg



Hasil yang diamati menunjukkan bahwa meskipun regularisasi membantu mengendalikan kompleksitas model, terdapat trade-off dalam akurasi, di mana network dengan regularisasi memiliki akurasi yang lebih rendah dibandingkan tanpa regularisasi. Pada regularisasi L1, banyak bobot yang sangat mendekati nol, yang menunjukkan bahwa metode ini mendorong sparsity dalam network. Sementara itu, pada regularisasi L2, bobot tidak didekati ke nol, tetapi terdistribusi lebih merata. Hal ini terjadi karena regularisasi L2 lebih mendorong diskriminasi bobot secara proporsional, tanpa benar-benar menghilangkan kontribusi fitur tertentu.

Jika digambarkan ke dalam euclidean space, grafik dari L2 memiliki tepi yang bundar, sedangkan L1 patah yang menunjukkan beberapa weight bisa menjadi 0 (*Feature Selection*) [3].

## 7. Perbandingan dengan library sklearn

<i>Layer = {128,32}, hidden_activation = leaky ReLU, output = softmax, weight_init = Xavier, lr = 0.01, loss = CCE, max epoch = 20, batch size = 64</i>	
Metode	Accuracy(%)
Sklearn MLP	97.83
My FFNN	95.94

MLP Scikit-Learn memiliki banyak optimisasi bawaan yang meningkatkan performa. Fungsi aktivasi ReLU yang digunakan berbeda dan lebih stabil dalam konvergensi. Optimizer seperti Adam yang merupakan gabungan dari GD with Momentum dan RMSProp dengan penyesuaian learning rate otomatis membuat training lebih efisien. Regularisasi L2 bawaan membantu menghindari overfitting. Inisialisasi bobot yang lebih optimal mempercepat training. Selain itu, implementasi berbasis NumPy dan Cython meningkatkan efisiensi komputasi sehingga waktu training dengan sklearn juga jauh lebih singkat.

# KESIMPULAN DAN SARAN

## A. Kesimpulan

Dari hasil eksplorasi yang ada, pemahaman kami mengenai cara kerja dari Feed Forward Neural Network menjadi bertambah. Kami menemukan bahwa penambahan width dan depth dari layer akan menambah akurasi namun penambahan ini juga memiliki resiko untuk overfitting. Kami juga menemukan bahwa untuk dataset yang kami teliti dan hyperparameter yang kami tetapkan, fungsi aktivasi Leaky ReLU memiliki akurasi paling tinggi, yakni 95.05% yang diikuti oleh ReLU. Kami juga mendapati bahwa fungsi aktivasi dengan akurasi terburuk adalah Sigmoid dengan akurasi 86.02%. Terkait learning rate, kami menemukan bahwa untuk max epoch = 20, learning rate berbanding lurus dengan akurasi. Namun, untuk learning rate yang terlalu besar akan membuat model overshoot yang mengakibatkan nilai loss menjadi divergen.

Terkait inisialisasi bobot, kami menemukan bahwa metode Xavier initialization memiliki akurasi yang terbaik, yakni 96.11% yang diikuti oleh He initialization, yakni 96.08%. Selain itu, metode inisialisasi bobot terburuk adalah zero initialization, yakni 11.43%. Terkait pengaruh regularisasi L1 dan L2, kami menemukan regularisasi mengakibatkan penurunan pada akurasi, walau tidak terlalu signifikan. Tanpa regularisasi vs L1 vs L2 vs L1 + L2 (96.13% vs 95.84% vs 95.68% vs 95.34%).

Terkait pengaruh RMSNorm, kami menemukan bahwa RMSNorm menghasilkan akurasi yang lebih baik jika dibandingkan tanpa menggunakan RMSNorm (96.61% vs 96.29%) dan RMSNorm memberikan pembelajaran yang lebih stabil.

Jika dibandingkan dengan fungsi bawaan dari sklearn, model kami memiliki akurasi yang kurang (97.83% vs 95.94%). Hal ini dikarenakan MLP Scikit-learn memiliki banyak optimasi bawaan, implementasi fungsi aktivasi ReLU yang berbeda, dan optimizer lainnya yang mempengaruhi efektifitas dan efisiensi komputasi.

## B. Saran

Saran yang bisa diberikan adalah untuk memberikan usaha yang lebih baik lagi untuk tugas kedepannya agar dapat memberikan hasil yang lebih baik lagi.

## PEMBAGIAN KERJA

NIM	Pembagian Kerja
13522051	laporan, readme, Value, Autograd, test.
13522079	Value, Autograd, Layer, loss, init, test, RMSNorm, activation, laporan
13522089	FFNN, Value, Autograd, init, test, visualization, regularization, activation, laporan

## REFERENSI

- [1] Goodfellow, I., Bengio, Y., & LeCun, Y. (2016). *Deep Learning*. MIT Press
- [2] PyTorch Documentation. (n.d.). Diakses dari <https://pytorch.org/docs/stable/index.html>