

# Laporan Tugas Besar 1 IF3270

## Pembelajaran Mesin



### **Kelompok 50**

Bagas Sambega Rosyada - 13522071

Elijah Darrelshane S - 13522097

Indraswara Galih Jayanegara - 13522119

**Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung**

## DAFTAR ISI

<b>Deskripsi Persoalan</b>	<b>3</b>
Feedforward Neural Network	3
Forward Propagation	3
Backward Propagation	4
<b>Pembahasan</b>	<b>5</b>
1. Penjelasan Implementasi	5
A. Deskripsi Kelas, Atribut, dan Method	5
Kelas FFNN	5
File LossFunction.py	6
File Visualize.py	6
File WeightGenerator.py	7
Kelas Layer	7
Kelas Scalar	7
B. Penjelasan Forward Propagation	8
C. Penjelasan Backward Propagation dan Weight Update	9
2. Hasil Pengujian	10
Pengaruh depth dan width	10
Pengaruh fungsi aktivasi	16
Pengaruh Learning Rate	24
Pengaruh Inisialisasi Bobot	26
Pengaruh Regularization	29
Perbandingan Sklearn	35
<b>Kesimpulan &amp; Saran</b>	<b>37</b>
Kesimpulan	37
Saran	37
<b>Pembagian Tugas</b>	<b>38</b>
<b>Referensi</b>	<b>39</b>

# Deskripsi Persoalan

## Feedforward Neural Network

Feedforward Neural Network (FFNN) adalah salah satu arsitektur yang ada pada *deep learning*. *Neural Network* terdiri dari *input layer*, setidaknya satu *hidden layer*, dan *output layer*. Setiap node terkoneksi dengan node dari layer sebelum maupun layer setelahnya. Feedforward propagation sendiri adalah sebuah proses dalam *neural network* yang mana input data dimasukkan ke dalam *network layer* untuk men-generate input.

1. **Input Layer:** Input data dimasukkan ke dalam input layer neural network
2. **Hidden Layer:** Data akan diproses dari satu hidden layer ke hidden layer sampai dengan output layer. Pada setiap output yang dihasilkan akan dimasukkan ke dalam *activation function* yang kemudian hasilnya digunakan untuk input layer berikutnya.

$$Z = W \cdot X + b$$

3. **Output Layer:** Final output dihasilkan yang nantinya akan menjadi prediksi

Proses pembelajaran dilakukan setelah proses di atas selesai dilakukan, dengan mencari nilai turunan negatif yang meminimalkan *error* (*gradient descent*). Proses pembelajaran ini disebut sebagai *backward propagation*.

## Forward Propagation

Forward propagation adalah proses di mana data input melewati *Neural Network* dari lapisan input menuju output tanpa adanya perhitungan *feedback*. Proses ini digunakan untuk menghasilkan prediksi berdasarkan **bobot** dan **bias** yang telah ditetapkan dalam model.

Proses ini dimulai dengan menghitung nilai *net* dari layer sebelumnya dengan formula:

$$Z = W \cdot X + b$$

dengan Z adalah nilai *net* untuk layer saat ini, W adalah bobot, X adalah input dari layer sebelumnya, dan b adalah bias. W merupakan sebuah matriks bobot berukuran  $m \times n$  yang terdiri dari bobot-bobot dari  $n$  neuron di layer sebelumnya ke  $m$  neuron di layer selanjutnya. Sementara X merupakan matriks berukuran  $n \times 1$  yang berisi nilai-nilai dari layer sebelumnya. Perkalian  $W \cdot X$  dapat dihitung dengan menggunakan perkalian matriks.

Setelah nilai *net* didapat, nilai tersebut akan dimasukkan ke suatu fungsi aktivasi  $f(x)$  untuk mendapatkan nilai *output* dari layer tersebut.

## Backward Propagation

Backward Propagation adalah sebuah method untuk melatih *neural network*. Goal-nya adalah mengurangi perbedaan antara **prediksi dari model** dan **prediksi aktual**. Hal ini dapat dicapai dengan mengatur bobot dan bias dari setiap layer *neural network*. Secara garis besar Backward propagation adalah sebuah teknik pada *deep learning* yang digunakan untuk meng-update bobot dan bias pada setiap neuron pada *neural network*.

Backward propagation melakukan *update weight* dengan meminimalkan turunan dari *loss* terhadap *weight* tersebut. Perhitungan *backward propagation* dilakukan dengan melakukan *update weight* berdasarkan formula berikut,

$$w = w - \eta \frac{dL}{dw}$$

dengan nilai turunan didapat dengan menggunakan *chain rule* sebagai berikut,

Untuk *output layer*:

$$\frac{dL}{dw} = \frac{dL}{doutput} \frac{doutput}{dnet} \frac{dnet}{dw}$$

Sementara untuk *hidden layer*, nilai turunan didapat dengan menggunakan formula berikut,

$$\frac{dL}{dw} = \frac{dL}{dnet} \frac{dnet}{dw}$$

dengan  $dnet/dw$  disesuaikan dengan kontribusi dari *weight* tersebut ke layer selanjutnya.

# Pembahasan

## 1. Penjelasan Implementasi

### A. Deskripsi Kelas, Atribut, dan Method

#### Kelas FFNN

kelas ini adalah kelas utama dari model yang kami buat. Atribut yang ada pada kelas ini diantaranya adalah sebagai berikut.

```
x: np.ndarray | List -> feature dari dataset
y: np.ndarray | List -> target dari dataset
layers: List[int] -> layer
activations: List[str] | str -> fungsi aktivasi
weight_method: str -> inisialisasi weight method
loss_function: str -> loss function
batch_size: int -> ukuran dari batch
epochs: int -> jumlah epoch
learning_rate: float -> learning rate dari model
mean: Optional[int | float] -> rata-rata untuk inisialisasi weight dengan metode normal distribution
variance: Optional[int | float] -> varians untuk inisialisasi weight dengan metode normal distribution
lower_bound: Optional[int | float] -> batas bawah untuk inisialisasi weight dengan metode uniform distribution
upper_bound: Optional[int | float] -> batas atas untuk inisialisasi weight dengan metode uniform distribution
seed: Optional[int | float] -> untuk reproducible pada inisialisasi weight
verbose: Optional[bool] -> atribut untuk menampilkan output pelatihan
randomize: Optional[bool] -> memilih apakah data akan di-random atau tidak saat dilatih
```

Beberapa method yang ada pada kelas FFNN adalah

- **initialize\_weights** method ini digunakan untuk menginisiasi bobot dari keseluruhan layer di dalamnya terdapat beberapa pilihan inisiasi seperti, *zero initialization*, *normal distribution*, *xavier initialization*, *He initialization*, dan *uniform distribution*.
- **net** method ini digunakan untuk menghitung net dari layer ke-N
- **activate** method activate digunakan untuk menghitung hasil output dari layer ke dalam *activation function*. Terdapat 5 activation function yaitu ReLU, sigmoid, hyperbolic tangent, linear, dan softmax.

- **loss** method loss digunakan untuk menghitung *loss* yang dihasilkan dari pelatihan yang dilakukan oleh model. Terdapat 3 *loss function* yaitu, mse, binary cross entropy, dan categorical cross entropy.
- **\_zero\_gradients** method ini digunakan untuk mereset seluruh gradient menjadi 0 sebelum melakukan pemrosesan baru pada batch yang baru.
- **fit** method fit digunakan untuk melatih model dengan data yang sudah dimasukkan sebagai input. Pada method ini akan dilakukan [forward propagation](#) dan [backward propagation](#)
- **predict\_single** method ini digunakan untuk memprediksi *single data* sesuai dengan hasil dari pelatihan pada method fit
- **save\_model** untuk menyimpan model dalam format file .pkl
- **load\_model** untuk memakai model dari file dengan format .pkl

### File LossFunction.py

Loss function yang kami implementasikan tidak berupa dalam bentuk kelas atau hanya dalam bentuk fungsi-fungsi pada umumnya saja. Terdapat 3 fungsi dalam file **LossFunction.py** yaitu, [mse](#), [binary\\_cross\\_entropy](#), dan [categorical\\_cross\\_entropy](#).

1. Mean Square Error (MSE)  
MSE adalah fungsi yang menghitung rata-rata dari selisih kelas target dengan kelas yang diprediksi oleh model. MSE dirumuskan sebagai berikut,

$$L = \frac{1}{n} \sum (y_{predicted} - y_{target})^2$$

Fungsi MSE bagus digunakan untuk menghitung nilai *loss* atau error untuk prediksi bertipe regresi (numerik kontinu)

2. Binary Cross Entropy  
Binary cross entropy adalah metode untuk menghitung loss function untuk dataset yang memiliki target tipe biner (2 kelas saja).
3. Categorical Cross Entropy  
Fungsi mse digunakan untuk menghitung loss function *mean squared error*. Fungsi [binary\\_cross\\_entropy](#) digunakan untuk menghitung loss function binary cross entropy. Terakhir

### File Visualize.py

Visualisasi dari model kami menggunakan graphviz. Pada file **visualize.py** terdapat dua fungsi yaitu `trace` dan `draw_dot`. Fungsi **trace** digunakan untuk menghubungkan semua nodes dari *children* ke *parent*-nya. Fungsi **draw\_dot** digunakan untuk menggambarkan graph.

## File WeightGenerator.py

Pada file ini kami mengimplementasikan fungsi-fungsi yang nantinya akan digunakan untuk inisialisasi bobot. Terdapat 6 Fungsi berbeda untuk inisialisasi bobot.

- **zero\_initialization** Fungsi ini menginisialisasi semua bobot menjadi bernilai 0.
- **one\_initialization** Fungsi ini menginisialisasi semua bobot menjadi bernilai 1.
- **random\_uniform\_distribution** Fungsi ini menginisialisasi dengan distribusi *uniform*. Inisialisasinya dibatasi dengan adanya parameter `lower_bound` dan `upper_bound`.
- **normal\_distribution** Fungsi ini menginisialisasi bobot dengan distribusi normal dengan menerima parameter `mean`, `variance`, dan `seed` (parameter ini digunakan untuk reproducible bobot)
- **xavier\_initialization** Fungsi ini menginisialisasi bobot dengan algoritma xavier. Pada fungsi ini menerima parameter `seed` yang digunakan untuk reproducible inisiasi bobot.
- **he\_initialization** Fungsi ini menginisialisasi bobot dengan algoritma he. Pada Fungsi ini menerima parameter `seed` juga untuk reproducible.

## Kelas Layer

Implementasi dari satu layer untuk Feed Forward Neural Network. Kelas Layer terdapat pada file `Layer.py` dan berisi nilai `weights` dan `bias` untuk layer tersebut, dan juga fungsi aktivasi yang digunakan pada layer tersebut. Kelas ini menyimpan beberapa fungsi yaitu:

- **Initialize\_weights**, melakukan inisialisasi bobot dan bias untuk satu layer menggunakan metode inisialisasi seperti He, Xavier, zero, dan normal
- **activate**, menghitung nilai output menggunakan fungsi aktivasi seperti softmax, linear, ReLu, dan sigmoid
- **net**, menghitung nilai net atau jumlah perkalian dari bobot dengan nilai input
- **forward**, melakukan forward propagation dengan memanggil fungsi `net` dan `activate` untuk layer tersebut.

## Kelas Scalar

Kelas ini digunakan untuk menyimpan nilai skalar dan gradiennya. Kelas ini terletak pada file **Autograd.py**. Kelas ini digunakan dalam perhitungan otomatisasi diferensiasi (auto differentiation) yang memungkinkan *backward propagation* untuk menghitung turunan.

Kelas ini berperan sebagai sebuah *container* untuk menyimpan nilai dan juga turunan dari nilai tersebut. Kelas ini mengimplementasikan *magic function* milik Python, yaitu fungsi-fungsi yang bisa dikonversi menjadi operator. Misalnya fungsi `__add__`, yang merepresentasikan operasi penjumlahan dengan menggunakan

objek Scalar lainnya. Pada fungsi `__add__(self, other)`, fungsi ini akan dipanggil secara otomatis saat terjadi penjumlahan dua objek Scalar seperti `scalarA + scalarB`.

Beberapa method yang ada pada kelas ini.

- **get\_value** Getter yang digunakan untuk mengambil value
- **get\_backward** Caller yang digunakan untuk memanggil fungsi `_backward()`
- **set\_backward** Method setter pada fungsi `backward`
- **do\_backward** Method untuk melakukan backward propagation
- **get\_parents** Method untuk mengembalikan parents dari Scalar
- **get\_operation** Method untuk mengembalikan fungsi operasi yang membuat scalar value
- **\_\_add\_\_** Method penjumlahan antara dua objek Scalar.
- **\_\_mul\_\_** Method untuk perkalian antara dua objek Scalar.
- **\_\_pow\_\_** Method untuk pemangkatan dari value
- **\_\_neg\_\_** Method untuk negasi
- **\_\_sub\_\_** Method untuk pengurangan
- **\_\_rsub\_\_** Method pengurangan (`other - self`)
- **\_\_radd\_\_** Method penjumlahan (`other + self`)
- **\_\_rmul\_\_** Method operator perkalian (`other * self`)
- **\_\_truediv\_\_** Method operator pembagian (`self / other`)
- **\_\_rtruediv\_\_** Method operator pembagian (`other / self`)
- **\_\_repr\_\_** Method untuk merepresentasikan string objek Scalar
- **log** Method untuk menghitung *natural logarithm* dari Scalar
- **\_\_mod\_\_** Method modulo
- **\_\_rmod\_\_** Method module (`self + other`)
- **relu** Method untuk fungsi aktivasi relu
- **sigmoid** Method untuk fungsi aktivasi sigmoid
- **linear** Method untuk fungsi aktivasi linear
- **tanh** Method untuk fungsi aktivasi hyperbolic tangent
- **backward** Method untuk propagasi menghitung gradien dari nilai ini terhadap semua nilai dalam computational graph yang berkontribusi terhadapnya.

## B. Penjelasan Forward Propagation

Forward propagation pada model kami dilakukan pada method **fit** pada kelas **FFNN**. Hal ini dilakukan pada bagian ini dari method **fit** pada kelas **FFNN**



```

    for i in batch_indices:
        # Forward pass
        for j in range(len(self.layers)):
            if j == 0:
                self.layer_net[i][j] = self.net(self.weights[0], [self.x[i]], self.bias[0], j)
            else:
                self.layer_net[i][j] = self.net(self.weights[j], self.layer_output[i][j-1],
self.bias[j], j)

                self.layer_output[i][j] = self.activate(self.activations[j], self.layer_net[i][j])

        # Calculate loss
        self.loss_values[i] = self.loss(self.loss_function, [self.y[i]], self.layer_output[i][-1][0])
        batch_loss += self.loss_values[i].value
        self.loss_values[i].backward()

```

Dalam cuplikan kode, method **net** dijalankan terlebih dahulu untuk menghitung nilai *net* (aggregate linear) pada suatu layer. Nilai *net* yang diperoleh kemudian diproses oleh method **activate** guna menerapkan fungsi aktivasi yang telah ditentukan. Hasil aktivasi inilah yang menjadi output dari layer tersebut dan sekaligus berperan sebagai input bagi layer berikutnya jika bukan merupakan layer output (layer terakhir).

### C. Penjelasan Backward Propagation dan Weight Update

Backward propagation pada model kami dilakukan pada method **fit** pada kelas **FFNN** bersamaan dengan forward propagation

```

    for i in batch_indices:
        # Forward pass
        for j in range(len(self.layers)):
            if j == 0:
                self.layer_net[i][j] = self.net(self.weights[0], [self.x[i]], self.bias[0], j)
            else:
                self.layer_net[i][j] = self.net(self.weights[j], self.layer_output[i][j-1],
self.bias[j], j)

                self.layer_output[i][j] = self.activate(self.activations[j], self.layer_net[i][j])

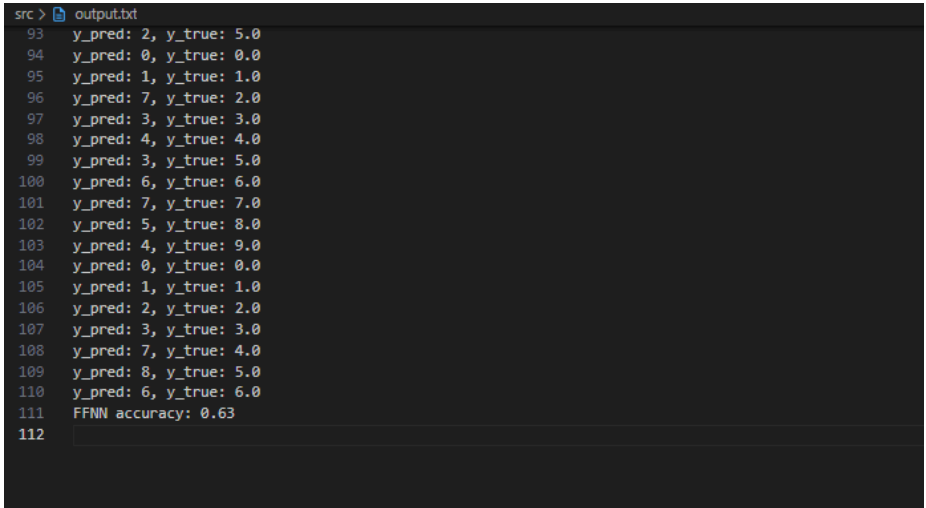
        # Calculate loss
        self.loss_values[i] = self.loss(self.loss_function, [self.y[i]], self.layer_output[i][-1][0])
        batch_loss += self.loss_values[i].value
        self.loss_values[i].backward()

```

Backward propagation disini dipanggil pada *self.loss\_values[i].backward()* yang kemudian setelah batch\_indices sudah selesai akan diperbarui nilai dari bobot dan bias.

## 2. Hasil Pengujian

Pengaruh *depth* dan *width*

Variasi Depth (Width: 5 neuron)	
<b>Constant Hyperparameters</b> <ul style="list-style-type: none"><li>• Width : 5</li><li>• Training data : 5000</li><li>• Test data : 50</li><li>• Learning rate : 0.01</li><li>• Error function : MSE</li><li>• Weight initialization : Xavier</li><li>• Seed : 42</li><li>• Epochs : 2</li></ul>	
Depth: 2 layers Input → ReLu → Softmax	<b>Hasil akhir prediksi</b>  <pre>src &gt; output.txt 93 y_pred: 2, y_true: 5.0 94 y_pred: 0, y_true: 0.0 95 y_pred: 1, y_true: 1.0 96 y_pred: 7, y_true: 2.0 97 y_pred: 3, y_true: 3.0 98 y_pred: 4, y_true: 4.0 99 y_pred: 3, y_true: 5.0 100 y_pred: 6, y_true: 6.0 101 y_pred: 7, y_true: 7.0 102 y_pred: 5, y_true: 8.0 103 y_pred: 4, y_true: 9.0 104 y_pred: 0, y_true: 0.0 105 y_pred: 1, y_true: 1.0 106 y_pred: 2, y_true: 2.0 107 y_pred: 3, y_true: 3.0 108 y_pred: 7, y_true: 4.0 109 y_pred: 8, y_true: 5.0 110 y_pred: 6, y_true: 6.0 111 FFNN accuracy: 0.63 112</pre>
	<b>Grafik Loss Pelatihan</b>



Depth: 3 layers  
 Input → ReLu →  
 ReLu → Softmax

### Hasil akhir prediksi

```
src > output1.txt
43 y_pred: 7, y_true: 2.0
44 y_pred: 3, y_true: 3.0
45 y_pred: 9, y_true: 4.0
46 y_pred: 3, y_true: 5.0
47 y_pred: 6, y_true: 6.0
48 y_pred: 7, y_true: 7.0
49 y_pred: 3, y_true: 8.0
50 y_pred: 0, y_true: 9.0
51 y_pred: 0, y_true: 0.0
52 y_pred: 1, y_true: 1.0
53 y_pred: 7, y_true: 2.0
54 y_pred: 3, y_true: 3.0
55 y_pred: 7, y_true: 4.0
56 y_pred: 7, y_true: 5.0
57 y_pred: 6, y_true: 6.0
58 FFNN accuracy: 0.58
59
```

### Grafik Loss Pelatihan



Depth: 4 layers  
 Input → ReLu →  
 ReLu → ReLu →  
 Softmax

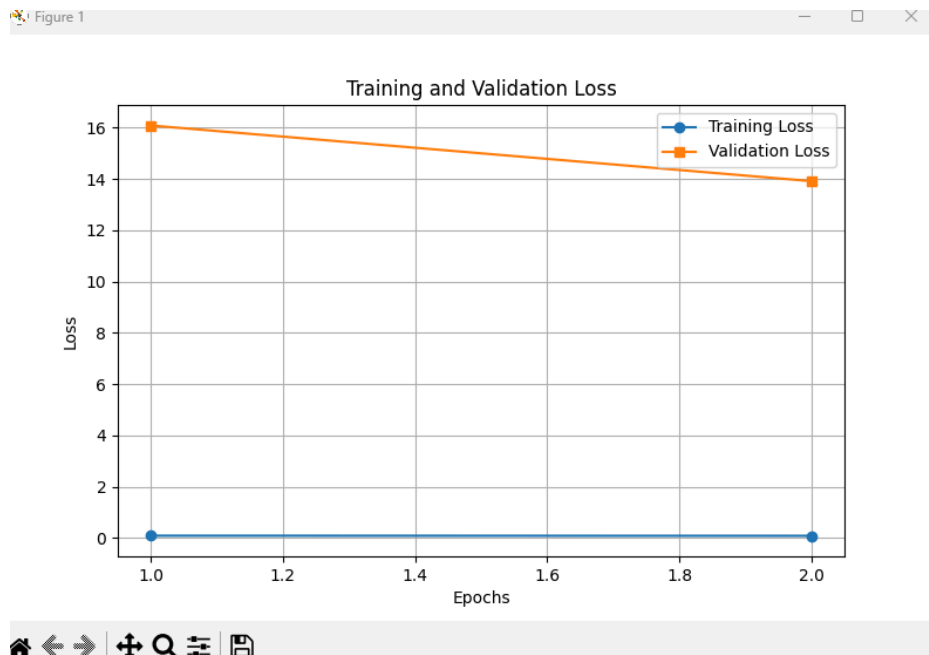
### Hasil akhir prediksi

```

3 y_pred: 1, y_true: 2.0
4 y_pred: 1, y_true: 3.0
5 y_pred: 6, y_true: 4.0
6 y_pred: 1, y_true: 5.0
7 y_pred: 6, y_true: 6.0
8 y_pred: 1, y_true: 7.0
9 y_pred: 1, y_true: 8.0
0 y_pred: 0, y_true: 9.0
1 y_pred: 0, y_true: 0.0
2 y_pred: 1, y_true: 1.0
3 y_pred: 1, y_true: 2.0
4 y_pred: 1, y_true: 3.0
5 y_pred: 9, y_true: 4.0
6 y_pred: 9, y_true: 5.0
7 y_pred: 6, y_true: 6.0
8 FFNN accuracy: 0.38
9

```

### Grafik Loss Pelatihan



Analisis:  
 makin besar depth pada model akan mengakibatkan akurasi model makin turun.  
 Alasan makin turun karena bobot yang dipakai tidak se-diverse itu sehingga  
 mengakibatkan model makin turun

Variasi Width  
 (Depth: 3 layer)  
 Input → ReLU → softmax

#### Constant Hyperparameters

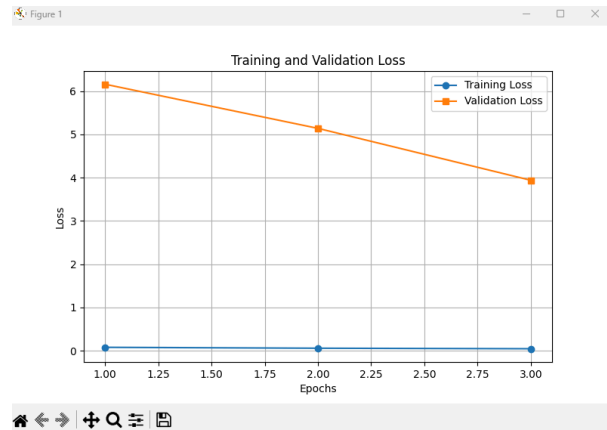
- Depth : 3
- Training data : 2500
- Test data : 50
- Learning rate : 0.01
- Error function : MSE
- Batch size : 10
- Weight initialization : Xavier
- Seed : 42
- Epochs : 3

Width: 5

**Hasil akhir prediksi**

```
y_pred: 4, y_true: 4.0
5 y_pred: 3, y_true: 5.0
7 y_pred: 6, y_true: 6.0
8 y_pred: 7, y_true: 7.0
9 y_pred: 5, y_true: 8.0
0 y_pred: 4, y_true: 9.0
1 y_pred: 0, y_true: 0.0
2 y_pred: 1, y_true: 1.0
3 y_pred: 2, y_true: 2.0
4 y_pred: 3, y_true: 3.0
5 y_pred: 4, y_true: 4.0
6 y_pred: 6, y_true: 5.0
7 y_pred: 6, y_true: 6.0
8 FFNN accuracy: 0.72
9
```

## Grafik Loss Pelatihan

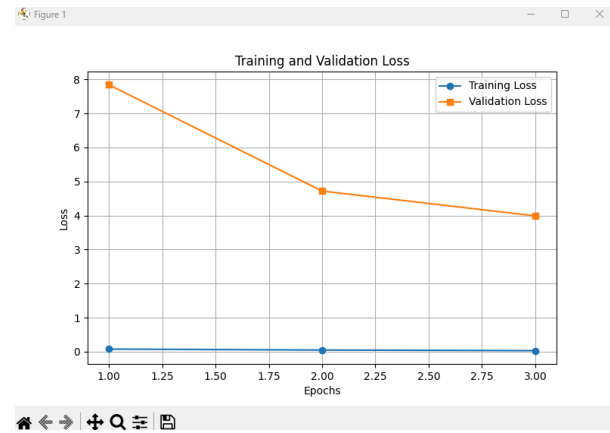


Width: 7

## Hasil akhir prediksi

```
src > output-width-2.txt
43 y_pred: 8, y_true: 2.0
44 y_pred: 8, y_true: 3.0
45 y_pred: 4, y_true: 4.0
46 y_pred: 5, y_true: 5.0
47 y_pred: 6, y_true: 6.0
48 y_pred: 7, y_true: 7.0
49 y_pred: 8, y_true: 8.0
50 y_pred: 4, y_true: 9.0
51 y_pred: 0, y_true: 0.0
52 y_pred: 1, y_true: 1.0
53 y_pred: 2, y_true: 2.0
54 y_pred: 3, y_true: 3.0
55 y_pred: 4, y_true: 4.0
56 y_pred: 5, y_true: 5.0
57 y_pred: 6, y_true: 6.0
58 FFNN accuracy: 0.82
59
```

## Grafik Loss Pelatihan

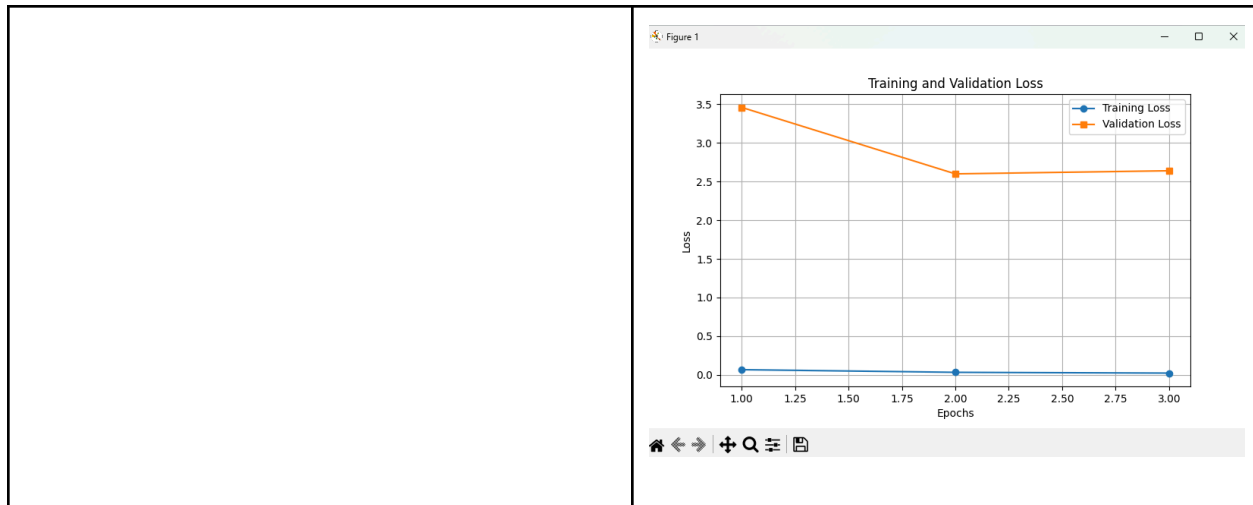


Width: 10

## Hasil akhir prediksi

```
40 y_pred: 2, y_true: 3.0
41 y_pred: 0, y_true: 0.0
42 y_pred: 1, y_true: 1.0
43 y_pred: 7, y_true: 2.0
44 y_pred: 8, y_true: 3.0
45 y_pred: 4, y_true: 4.0
46 y_pred: 5, y_true: 5.0
47 y_pred: 6, y_true: 6.0
48 y_pred: 7, y_true: 7.0
49 y_pred: 8, y_true: 8.0
50 y_pred: 4, y_true: 9.0
51 y_pred: 0, y_true: 0.0
52 y_pred: 1, y_true: 1.0
53 y_pred: 2, y_true: 2.0
54 y_pred: 3, y_true: 3.0
55 y_pred: 4, y_true: 4.0
56 y_pred: 8, y_true: 5.0
57 y_pred: 6, y_true: 6.0
58 FFNN accuracy: 0.78
59
```

## Grafik Loss Pelatihan



Analisis:

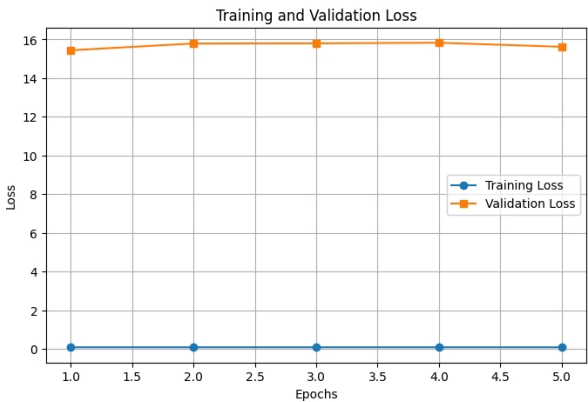
Semakin besar besar widthnya akan semakin memperbagus hasil akurasi, tapi pada percobaan ketiga akurasinya turun daripada width pada percobaan kedua.

## Pengaruh fungsi aktivasi

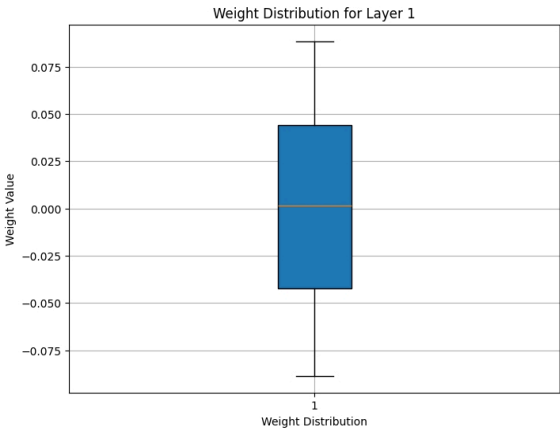
Variasi Fungsi Aktivasi	
<b>Constant Hyperparameters</b>	
• Width	: 5
• Depth	: 3
• Training data	: 1000
• Validation data	: 100
• Test data	: 50
• Learning rate	: 0.01
• Error function	: MSE
• Weight initialization	: Xavier
• Seed	: 42
• Epochs	: 5



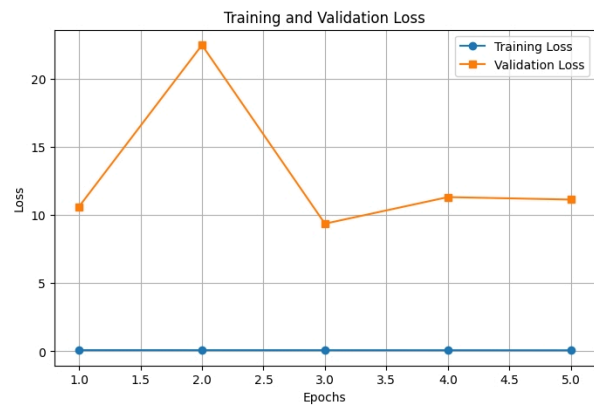
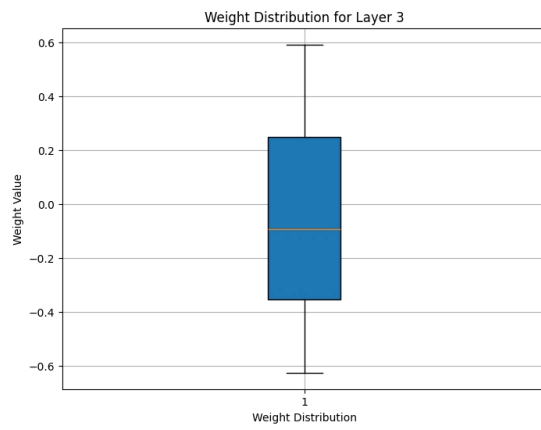
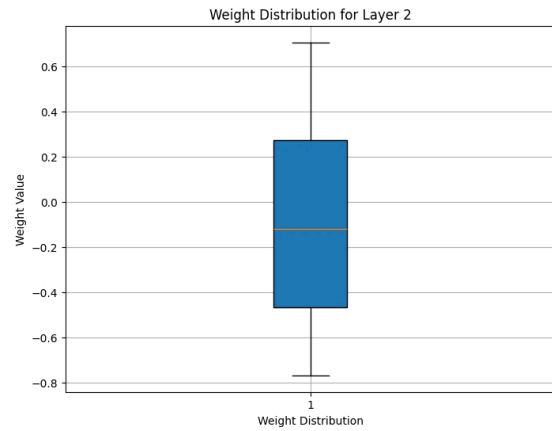
Sigmoid  
Input → Sigmoid → Sigmoid



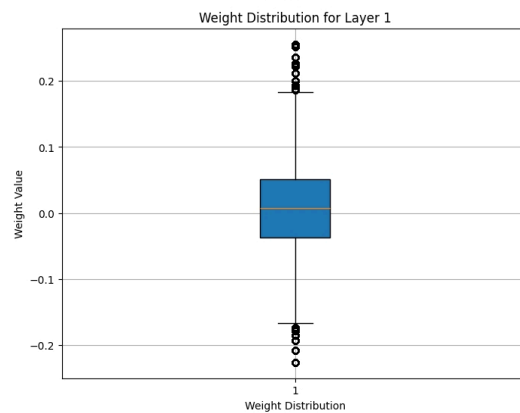
y\_pred: 7, y\_true: 4.0  
y\_pred: 7, y\_true: 1.0  
y\_pred: 7, y\_true: 7.0  
y\_pred: 7, y\_true: 2.0  
y\_pred: 7, y\_true: 6.0  
y\_pred: 7, y\_true: 5.0  
y\_pred: 4, y\_true: 0.0  
y\_pred: 7, y\_true: 1.0  
y\_pred: 7, y\_true: 2.0  
y\_pred: 7, y\_true: 3.0  
y\_pred: 7, y\_true: 4.0  
y\_pred: 6, y\_true: 5.0  
y\_pred: 6, y\_true: 6.0  
y\_pred: 7, y\_true: 7.0  
y\_pred: 7, y\_true: 8.0  
y\_pred: 7, y\_true: 9.0  
y\_pred: 7, y\_true: 0.0  
y\_pred: 7, y\_true: 1.0  
y\_pred: 7, y\_true: 2.0  
y\_pred: 7, y\_true: 3.0  
y\_pred: 7, y\_true: 4.0  
y\_pred: 6, y\_true: 5.0  
y\_pred: 7, y\_true: 6.0  
Accuracy: 0.14



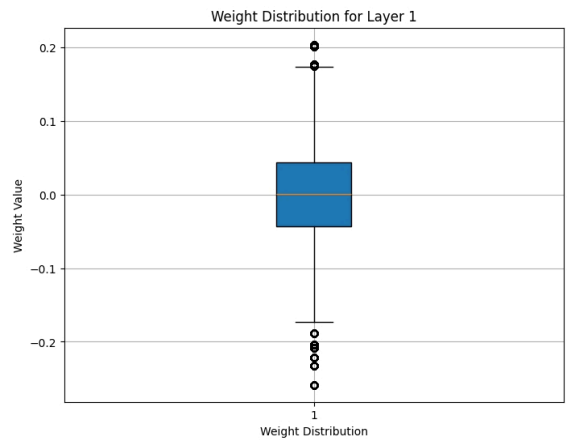
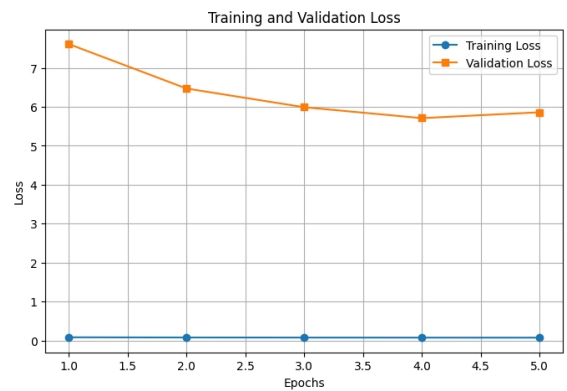
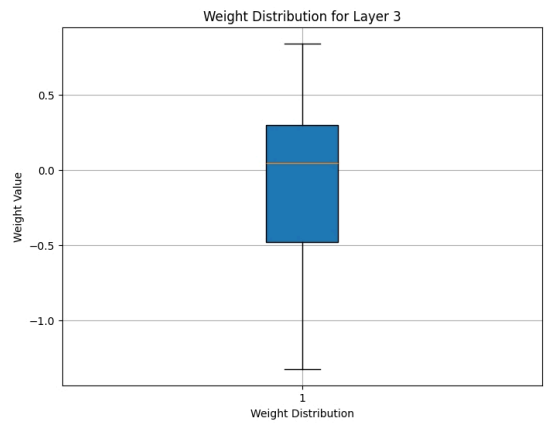
ReLu  
Input → ReLu → ReLu

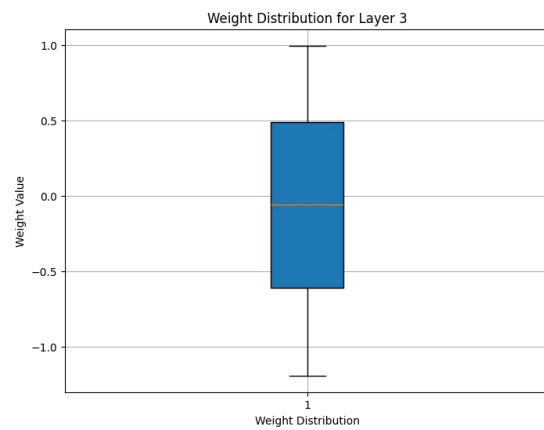
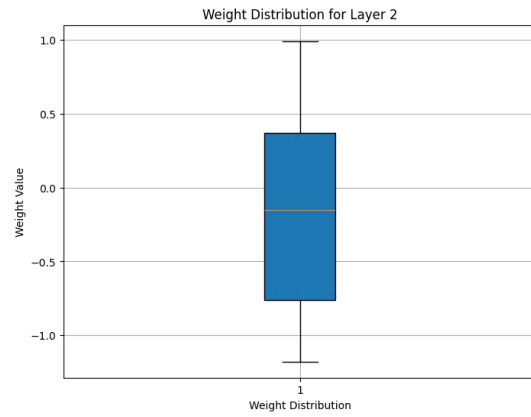


y\_pred: 4, y\_true: 2.0  
y\_pred: 4, y\_true: 4.0  
y\_pred: 4, y\_true: 9.0  
y\_pred: 6, y\_true: 4.0  
y\_pred: 7, y\_true: 3.0  
y\_pred: 6, y\_true: 6.0  
y\_pred: 4, y\_true: 4.0  
y\_pred: 3, y\_true: 1.0  
y\_pred: 7, y\_true: 7.0  
y\_pred: 5, y\_true: 2.0  
y\_pred: 4, y\_true: 6.0  
y\_pred: 4, y\_true: 5.0  
y\_pred: 4, y\_true: 0.0  
y\_pred: 3, y\_true: 1.0  
y\_pred: 5, y\_true: 2.0  
y\_pred: 3, y\_true: 3.0  
y\_pred: 7, y\_true: 4.0  
y\_pred: 3, y\_true: 5.0  
y\_pred: 6, y\_true: 6.0  
y\_pred: 5, y\_true: 7.0  
y\_pred: 3, y\_true: 8.0  
y\_pred: 4, y\_true: 9.0  
y\_pred: 0, y\_true: 0.0  
y\_pred: 3, y\_true: 1.0  
y\_pred: 5, y\_true: 2.0  
y\_pred: 3, y\_true: 3.0  
y\_pred: 7, y\_true: 4.0  
y\_pred: 7, y\_true: 5.0  
y\_pred: 6, y\_true: 6.0  
Accuracy: 0.36



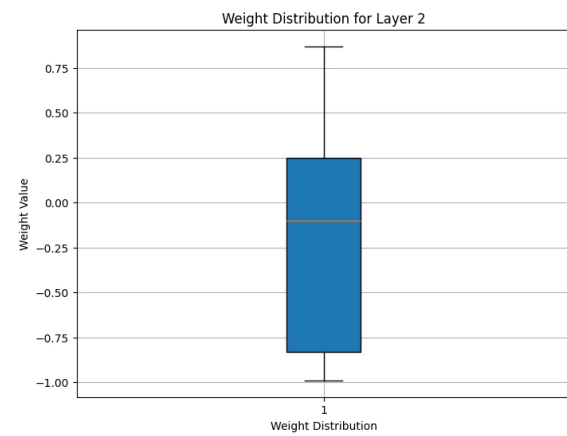
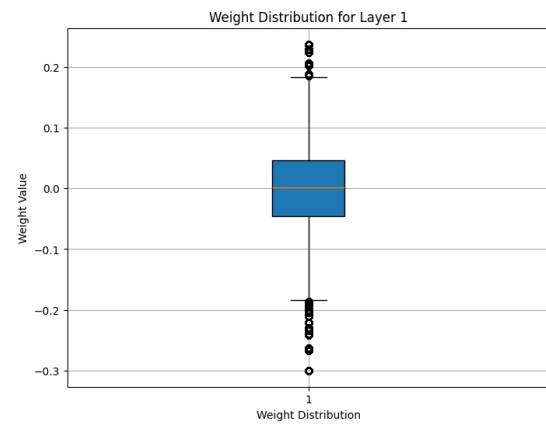
Tanh  
Input → Tanh → Tanh

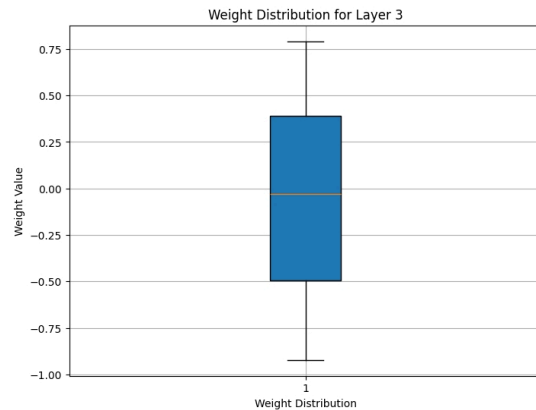




```
y_pred: 4, y_true: 4.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 7.0
y_pred: 5, y_true: 2.0
y_pred: 7, y_true: 6.0
y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 8, y_true: 8.0
y_pred: 4, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 5, y_true: 5.0
y_pred: 6, y_true: 6.0
Accuracy: 0.7
```

Linear  
Input → Linear → Linear





```

y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 8, y_true: 8.0
y_pred: 4, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 5, y_true: 5.0
y_pred: 6, y_true: 6.0
Accuracy: 0.7

```

#### Analisis:

Berdasarkan penggunaan berbagai jenis fungsi aktivasi, linear menghasilkan akurasi tertinggi, diikuti tanh, lalu ReLu dan sigmoid. Hal ini dapat disebabkan oleh fungsi linear yang akan melakukan update weight saat backpropagation, dengan langsung memperhatikan nilai selisih dari target dan hasil prediksi. Hal ini menyebabkan update weight tidak dimanipulasi oleh fungsi-fungsi lain dan langsung merepresentasikan perubahan yang dibutuhkan. Untuk fungsi tanh, nilai output yang dapat dihasilkan berkisar antara -1 dan 1, sehingga masih tetap memungkinkan nilai output yang dihasilkan negatif dan lebih sesuai untuk penghitungan gradiennya. Sementara itu ReLu tidak memungkinkan nilai negatif, tapi output-nya bisa bernilai berapapun selain negatif. Hal ini dapat menyebabkan jika terdapat output yang bernilai negatif, akan selalu dipaksa menjadi bernilai 0 dan menyebabkan tidak terjadi perubahan gradien. Untuk fungsi sigmoid, nilainya lebih strict yaitu antara 0 dan 1, sehingga perubahan yang dilakukan hanya sedikit

demi sedikit saja, terbatas pada angka di antara 0 dan 1, sehingga update weight juga hanya akan dipengaruhi oleh angka-angka kecil, sementara untuk kasus klasifikasi multi-class dengan nilai 0-9, hal ini menyebabkan update weight berjalan sangat lambat dan bahkan terlalu dipaksakan ke angka 1 atau 0 jika nilai gradien seharusnya adalah besar/negatif.

## Pengaruh Learning Rate

### Learning Rate

#### Constant Hyperparameters

- Width : 5
- Depth : 3
- Training data : 1.000
- Test data : 20
- Error function : MSE
- Weight initialization : Xavier
- Seed : 42
- Epochs : 2
- Activation Function: Relu, Sigmoid, Softmax

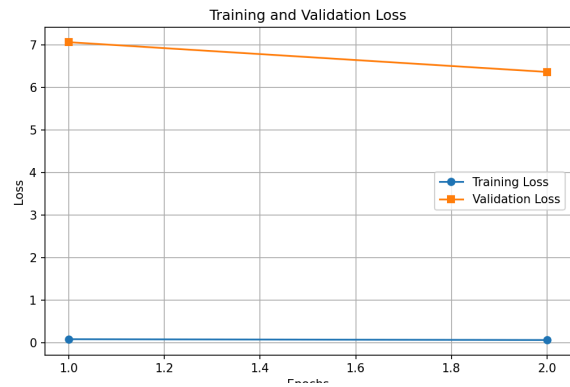
Learning rate: 0.01



```
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 7, y_true: 3.0
y_pred: 0, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 6.0
FFNN accuracy: 0.2
```

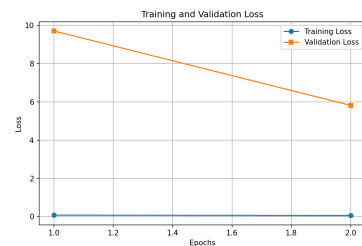


Learning rate: 0.05



```
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 4, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
FFNN accuracy: 0.55
```

Learning rate: 0.1



```
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 0, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 2, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 8, y_true: 5.0
y_pred: 6, y_true: 6.0
FFNN accuracy: 0.6
```

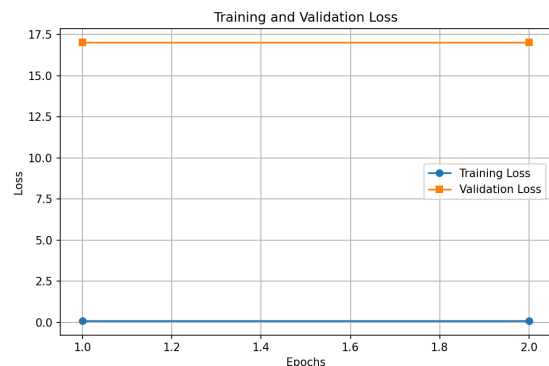
Analisis:

Ditemukan bahwa semakin besar learning rate, maka semakin besar juga akurasi. Hal ini disebabkan karena learning rate yang lebih besar membantu memperbesar perubahan bobot sehingga model bisa belajar lebih cepat. Dalam kata lain, semakin besar learning rate, semakin besar juga gradien perubahan loss.

Namun, perlu diperhatikan bahwa testing ini dilakukan dengan model yang relatif kecil dan epoch yang sedikit saja sehingga sebenarnya ada juga risiko bahwasanya perubahan weight membuat model “melompati” titik minimum karena terlalu fluktuatif.

## Pengaruh Inisialisasi Bobot

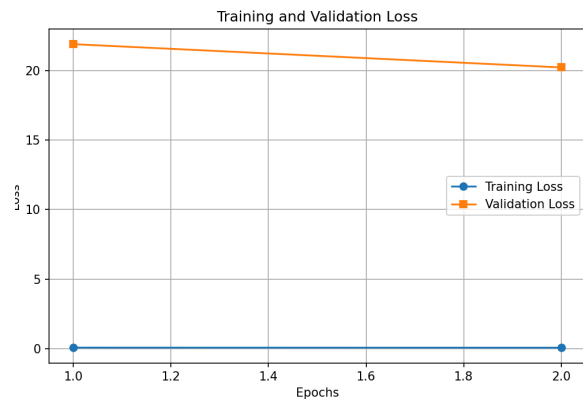
Weight Generation	
<b>Constant Hyperparameters</b>	
● Width	: 5
● Depth	: 3
● Training data	: 1000
● Test data	: 20
● Learning rate	: 0.01
● Error function	: MSE
● Seed	: 42
● Epochs	: 2

Zero	 <table><caption>Training and Validation Loss Data</caption><tr><th>Epochs</th><th>Training Loss</th><th>Validation Loss</th></tr><tr><td>1.0</td><td>0.0</td><td>17.5</td></tr><tr><td>2.0</td><td>0.0</td><td>17.5</td></tr></table>	Epochs	Training Loss	Validation Loss	1.0	0.0	17.5	2.0	0.0	17.5
Epochs	Training Loss	Validation Loss								
1.0	0.0	17.5								
2.0	0.0	17.5								

```
y_pred: 7, y_true: 9.0
y_pred: 7, y_true: 0.0
y_pred: 7, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 7, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 7, y_true: 6.0
FFNN accuracy: 0.05
```

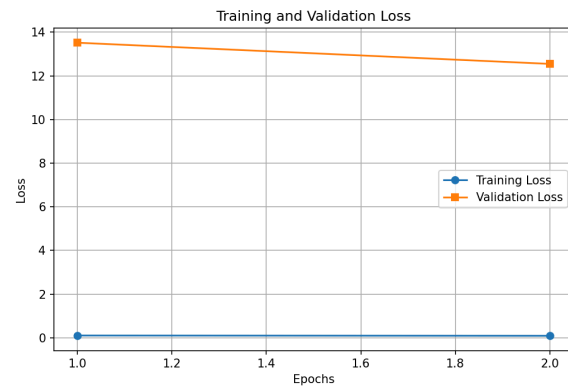
Uniform

\*Lower\_bound = 0  
Upper\_bound = 1



```
y_pred: 9, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 9, y_true: 2.0
y_pred: 9, y_true: 3.0
y_pred: 9, y_true: 4.0
y_pred: 9, y_true: 5.0
y_pred: 9, y_true: 6.0
FFNN accuracy: 0.15
```

Normal  
\*mean = 0  
Variance = 1



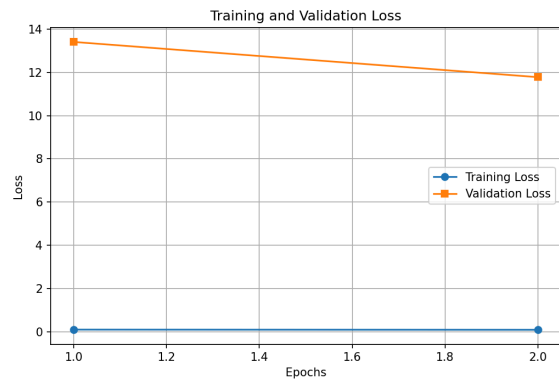
```
y_pred: 3, y_true: 1.0
y_pred: 3, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 3, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 7, y_true: 9.0
y_pred: 3, y_true: 0.0
y_pred: 7, y_true: 1.0
y_pred: 3, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 7, y_true: 6.0
FFNN accuracy: 0.2
```

Xavier



```
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 7, y_true: 3.0
y_pred: 0, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 6.0
FFNN accuracy: 0.2
```

He



```
y_pred: 7, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 7, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 7, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 1, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 7, y_true: 6.0
FFNN accuracy: 0.2
```

Analisis:

Setelah dilakukan pengujian, ditemukan bahwa urutan metode inisialisasi bobot berdasarkan akurasi sebagai berikut:

Zero < Uniform < Normal = Xavier = He

Hal ini dikarenakan inisialisasi bobot menggunakan zero menghasilkan error yang tinggi tapi gradien yang kecil sehingga sangat rawan terjebak dalam sebuah local minimum. Metode uniform sangat dipengaruhi oleh parameter lower bound dan upper bound yang digunakan, begitu pula metode normal sangat dipengaruhi oleh parameter mean dan variance. Hal ini jelas terlihat ketika kita membandingkan metode xavier yang didasarkan pada uniform distribution serta metode He yang didasarkan pada normal distribution. Ketika dipilih parameter yang ideal, terlihat loss berkurang dan akurasi meningkat.

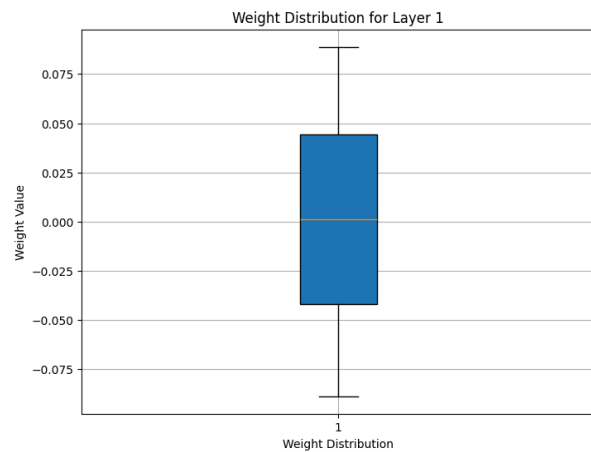
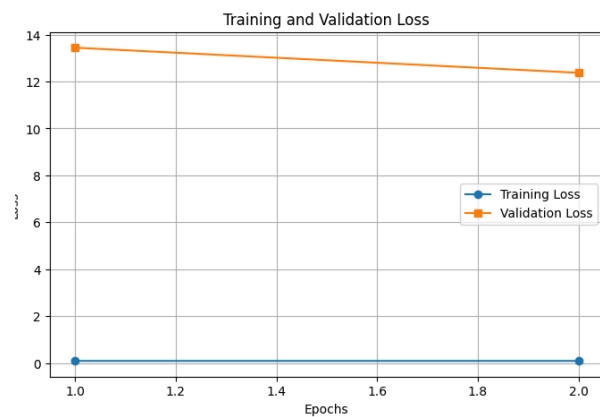
## Pengaruh Regularization

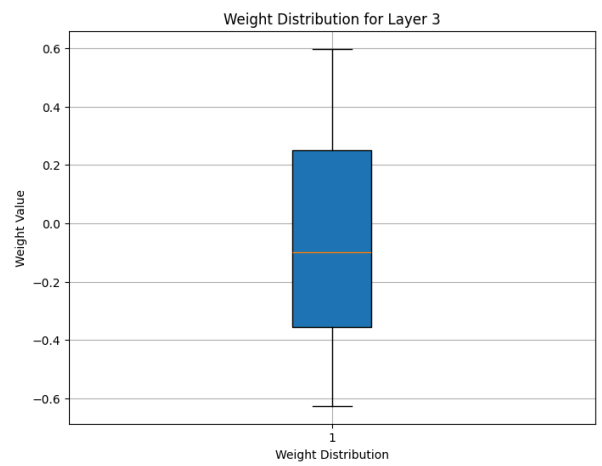
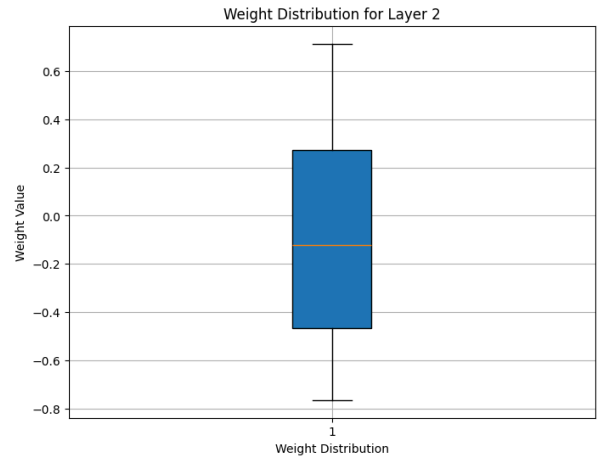
Learning Rate

**Constant Hyperparameters**

- Width : 5
- Depth : 3
- Training data : 1.000
- Test data : 20
- Learning rate : 0.01
- Error function : MSE
- Weight initialization : Xavier
- Seed : 42
- Epochs : 2
- Activation Function: Relu, Sigmoid, Softmax

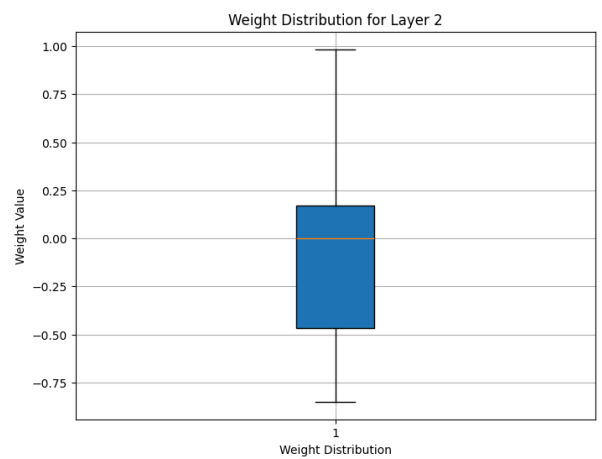
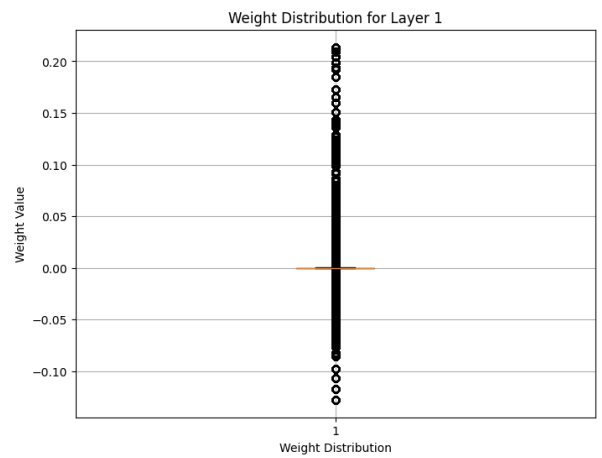
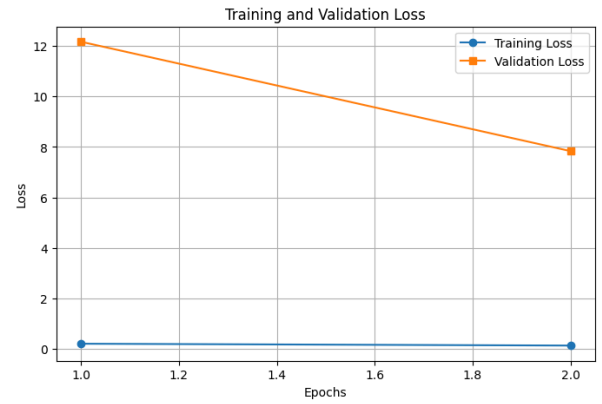
Tidak memakai L1 & L2



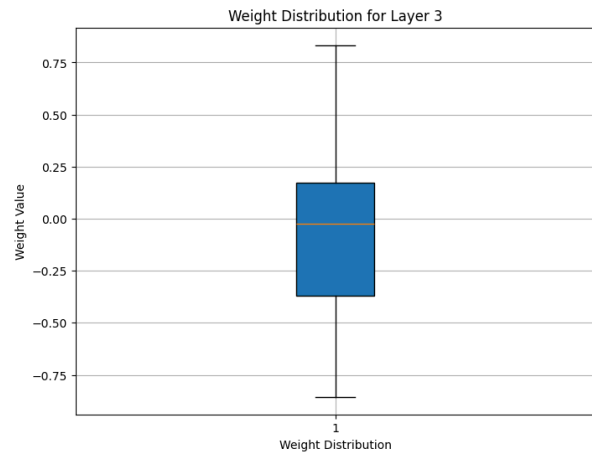


```
y_pred: 4, y_true: 7.0
y_pred: 1, y_true: 2.0
y_pred: 4, y_true: 6.0
y_pred: 4, y_true: 5.0
y_pred: 4, y_true: 0.0
y_pred: 4, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 1, y_true: 3.0
y_pred: 1, y_true: 4.0
y_pred: 6, y_true: 5.0
y_pred: 4, y_true: 6.0
y_pred: 1, y_true: 7.0
y_pred: 4, y_true: 8.0
y_pred: 4, y_true: 9.0
y_pred: 4, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 1, y_true: 3.0
y_pred: 1, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 4, y_true: 6.0
Accuracy: 0.16
```

Memakai  $L1 = 0.001$





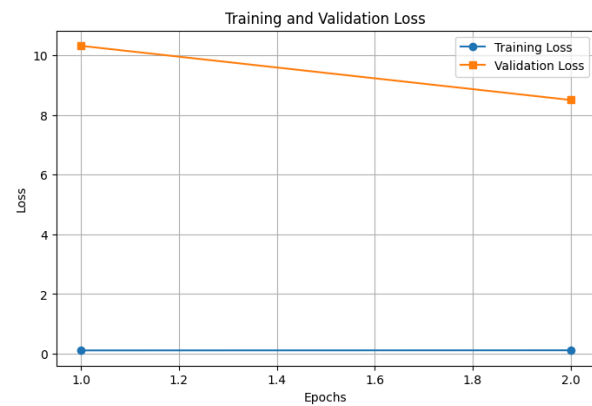


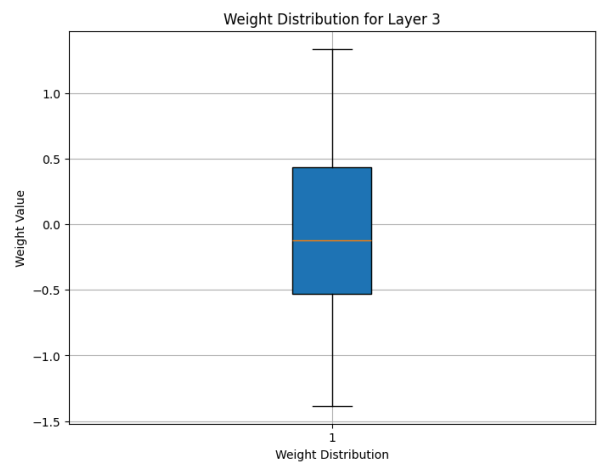
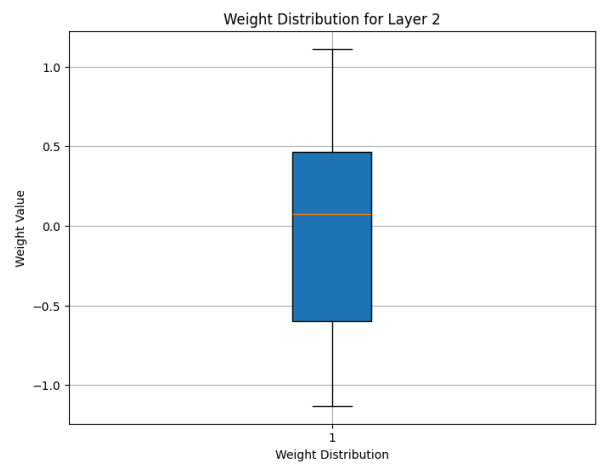
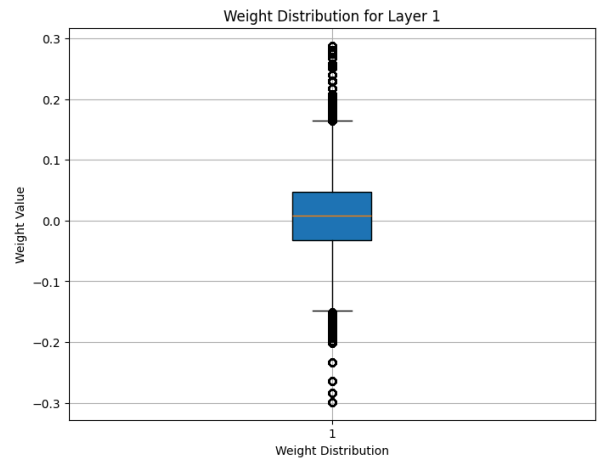
```

y_pred: 7, y_true: 6.0
y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 1, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 4, y_true: 6.0
y_pred: 1, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 4, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 1, y_true: 2.0
y_pred: 1, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 4, y_true: 6.0
Accuracy: 0.38

```

Memakai L2 = 0.001





	<pre> y_pred: 7, y_true: 6.0 y_pred: 7, y_true: 5.0 y_pred: 0, y_true: 0.0 y_pred: 1, y_true: 1.0 y_pred: 1, y_true: 2.0 y_pred: 1, y_true: 3.0 y_pred: 4, y_true: 4.0 y_pred: 7, y_true: 5.0 y_pred: 4, y_true: 6.0 y_pred: 1, y_true: 7.0 y_pred: 3, y_true: 8.0 y_pred: 4, y_true: 9.0 y_pred: 0, y_true: 0.0 y_pred: 1, y_true: 1.0 y_pred: 1, y_true: 2.0 y_pred: 1, y_true: 3.0 y_pred: 7, y_true: 4.0 y_pred: 7, y_true: 5.0 y_pred: 4, y_true: 6.0 Accuracy: 0.38 </pre>
<p>Analisis:</p> <p>Berdasarkan percobaan, penggunaan L1 dan L2 sangat meningkatkan akurasi, dari 0.12 menjadi 0,38 dengan L1 = 0.001 dan L2 = 0.001. Namun dari sisi loss, L1 mengalami perbaikan loss value lebih baik dibandingkan L2, nilai validation loss L1 lebih menurun drastis dibandingkan penurunan loss memakai L2. Hal ini karena L2 menghitung kuadrat dari nilai weight, sehingga selisih update weight nantinya juga akan jauh lebih besar dibandingkan pada L1.</p>	

## Perbandingan Sklearn

Sklearn (Input → ReLu → ReLu → Softmax)	
<b>Constant Hyperparameters</b> <ul style="list-style-type: none"> <li>• Width : 5</li> <li>• Depth : 3</li> <li>• Training data : 1000</li> <li>• Test data : 20</li> <li>• Learning rate : 0.1</li> <li>• Error function : MSE</li> <li>• Weight initialization : Xavier</li> <li>• Seed : 42</li> <li>• Epochs : 2</li> </ul>	

## Self-implementation

Learning rate = 0.01

```
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 7, y_true: 3.0
y_pred: 0, y_true: 4.0
y_pred: 7, y_true: 5.0
y_pred: 0, y_true: 6.0
FFNN accuracy: 0.2
```

Learning rate = 0.1

```
y_pred: 1, y_true: 1.0
y_pred: 7, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 4, y_true: 4.0
y_pred: 3, y_true: 5.0
y_pred: 6, y_true: 6.0
y_pred: 7, y_true: 7.0
y_pred: 3, y_true: 8.0
y_pred: 0, y_true: 9.0
y_pred: 0, y_true: 0.0
y_pred: 1, y_true: 1.0
y_pred: 2, y_true: 2.0
y_pred: 3, y_true: 3.0
y_pred: 7, y_true: 4.0
y_pred: 8, y_true: 5.0
y_pred: 6, y_true: 6.0
FFNN accuracy: 0.6
```

## Sklearn

Learning rate = 0.01

```
Classification Report:
              precision    recall  f1-score   support

    0.0         0.40      1.00     0.57         2
    1.0         1.00      1.00     1.00         2
    2.0         0.00      0.00     0.00         3
    3.0         0.00      0.00     0.00         2
    4.0         0.00      0.00     0.00         2
    5.0         0.40      0.67     0.50         3
    6.0         0.00      0.00     0.00         3
    7.0         0.25      1.00     0.40         1
    8.0         0.00      0.00     0.00         1
    9.0         0.25      1.00     0.40         1

 accuracy          0.40      20
 macro avg         0.23      0.47     0.29      20
```

Learning rate = 0.1

```
Classification Report:
              precision    recall  f1-score   support

    0.0         0.00      0.00     0.00         2.0
    1.0         0.00      0.00     0.00         2.0
    2.0         0.00      0.00     0.00         3.0
    3.0         0.00      0.00     0.00         2.0
    4.0         0.00      0.00     0.00         2.0
    5.0         0.00      0.00     0.00         3.0
    6.0         0.00      0.00     0.00         3.0
    7.0         0.00      0.00     0.00         1.0
    8.0         0.00      0.00     0.00         1.0
    9.0         0.00      0.00     0.00         1.0

 accuracy          0.00      20.0
 macro avg         0.00      0.00     0.00      20.0
 weighted avg      0.00      0.00     0.00      20.0
```

Analisis:

Ditemukan bahwa model implementasi sklearn menghasilkan hasil yang lebih akurat pada learning rate yang lebih kecil, sedangkan model self-implementation menghasilkan hasil yang lebih akurat pada learning rate yang lebih besar. Hal ini kemungkinan disebabkan oleh model sklearn dibuat lebih sensitif terhadap learning rate untuk memfasilitasi pembelajaran yang lebih cepat. Dengan demikian, model sklearn lebih cenderung untuk *overshoot* titik minima ketika epoch rendah tetapi learning rate tinggi.

# Kesimpulan & Saran

## Kesimpulan

Tugas besar kali ini membantu kami untuk memahami *Feedforward Neural Network* karena pada tugas ini kami membuat implementasi FFNN dari *scratch*. Berdasarkan implementasi yang telah dibuat, seluruh parameter baik *learning rate*, fungsi aktivasi, fungsi loss, banyak neuron, banyak layer dan regularisasi berpengaruh terhadap kinerja dan akurasi model. Penggunaan parameter yang tepat dapat meningkatkan akurasi dari model tersebut juga.

## Saran

Bagi orang-orang yang ingin melanjutkan apa yang sudah kami buat, kalian bisa melakukan eksplorasi yang lebih jauh untuk mendapatkan hasil yang lebih baik. Selain itu, bisa dilakukan juga *refactor* dari kode yang kami buat untuk mendapatkan hasil yang lebih efektif dan efisien. Terlebih lagi jika pembaca memiliki waktu “lebih” bisa melakukan training dengan parameter, depth, dan width yang lebih banyak sehingga dapat menghasilkan hasil yang lebih baik. Terakhir, saat melakukan implementasi sebaiknya menggunakan resources yang sudah ada agar tidak memenuhi memori dan memperlambat waktu pelatihan. Sebaiknya saat melakukan revert version cek lagi fitur apa saja yang diubah agar saat proses dari revert memiliki fitur yang sama dengan fitur sebelum revert. Selain itu, seharusnya struktur kelas utama dapat lebih dioptimasi lagi terutama jika dataset yang digunakan untuk train sangat besar.

# Pembagian Tugas

13522071	Struktur FFNN, automatic differentiation, <i>backward propagation</i> , dan Laporan
13522097	Inisialisasi weight (+ bonus Xavier dan He), one-hot encoding, <i>backward propagation</i> , dan Laporan
13522119	<i>Forward propagation</i> , <i>loss function</i> , <i>activation function</i> , turunan fungsi loss dan activation, L1 & L2 Regularization, Save and Load mode, dan Laporan

# Referensi

- <https://www.jasonosajima.com/forwardprop>
- <https://www.jasonosajima.com/backprop>
- <https://numpy.org/doc/2.2/>
- [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
- <https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization>
- PPT Kelas IF3270