Implementasi *Forward Propagation* untuk *Feed Forward Neural Network*

Diajukan sebagai pemenuhan tugas I

Oleh

13521116 Juan Christopher Santoso

13521135 Nicholas Liem

13521139 Nathania Calista Djunaedi

13521162 Antonio Natthan Krishna

Dosen Pengampu : Fariska Zakhralativa Ruskanda, S.T.,M.T.

IF3270 - Pembelajaran Mesin

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Bab I
Penjelasan Implementasi

1.1. Implementasi Fast Feed Neural Network

Untuk mengimplementasikan semua algoritma pembelajaran yang ada, penulis membuat sebuah kelas (*class FastFeedNeuralNetwork* (FFNN)) yang menggambarkan bagaimana sebuah FFNN bekerja. Kelas FFNN memiliki beberapa *method* yang nantinya dapat digunakan untuk membantu model membuat sebuah prediksi. Implementasi serta metode - metode yang terdapat pada kelas FFNN dapat dilihat pada gambar di bawah ini

```
class FastFeedNeuralNetwork:
    def __init__(self, layers, weights, input_data):
        self.input_data = np.array(input_data) if not isinstance(input_data, np.ndarray)
else input_data
        self.layers = layers
        self.weights = [np.array(w) for w in weights]
        self.graph_visualizer = GraphNetworkVisualizer(layers, weights, input_data)

    def prepend_bias(self, activations):
        return np.insert(activations, 0, 1, axis=1)


    def predict(self):
        activations = self.input_data
        activationFunc = ActivationFunction()

        for i in range(len(self.layers)):
            activations_with_bias = self.prepend_bias(activations)
            net_input = np.dot(activations_with_bias, self.weights[i])

            # Update nilai activations dengan menggunakan fungsi aktivasi yang
diinginkan
            activation_function_str = self.layers[i]['activation_function']
            match activation_function_str:
```

```
            case "relu":
                activations = activationFunc.relu(net_input)
            case "softmax":
                activations = activationFunc.softmax(net_input)
            case "sigmoid":
                activations = activationFunc.sigmoid(net_input)
            case "linear":
                activations = activationFunc.linear(net_input)

    return activations
```

Kelas *FastFeedNeuralNetwork* memiliki beberapa *method* yang akan dijelaskan pada tabel di bawah ini

Method	**Deskripsi**
init	Inisialisasi kelas dan menerima 2 parameter, yaitu *layers* dan *weights.*
prepend_bias	Memasukkan nilai bias pada input
predict	Metode yang dapat dipanggil oleh program untuk memberikan prediksi. Di dalam *method* ini, akan dipanggil fungsi aktivasi yang sesuai dengan *input* dari pengguna.

Inti dari algoritma *Fast Feed Neural Network* terletak pada metode *predict*. Berikut adalah prosedur dari fungsi tersebut:

1. Menyimpan input_data pada variabel *activations*
2. Untuk setiap layer yang ada kita akan iterasi sub-prosedur di bawah ini
    a. Menambahkan bias (bernilai 1) pada input_data untuk setiap iterasi
    b. Menghitung nilai net_input dengan melakukan dot product antara input yang sudah diappend dengan bias dengan setiap nilai weight yang berkoresponden

c. Berdasarkan fungsi aktivasi pada layer tersebut, ditentukan pilihannya dan dihitung nilai aktivasi yang baru

3. Mengembalikan nilai atau hasil output terakhir

Dalam mengimplementasikan fungsi aktivasi yang digunakan dalam kelas FFNN, dibuatlah sebuah kelas yang menyimpan seluruh algoritma fungsi aktivasi tersebut, yakni kelas *ActivationFunction.* Kelas *ActivationFunction* memiliki beberapa *method* yang akan dijelaskan pada tabel di bawah ini

Method	**Deskripsi**
linear	Fungsi aktivasi untuk membangkitkan prediksi nilai dengan cara metode *linear*
softmax	Fungsi aktivasi untuk membangkitkan prediksi nilai dengan cara metode *softmax*
relu	Fungsi aktivasi untuk membangkitkan prediksi nilai dengan cara metode *relu*
sigmoid	Fungsi aktivasi untuk membangkitkan prediksi nilai dengan cara metode *sigmoid*

1.2. Implementasi Linear

Fungsi aktivasi untuk fungsi linear adalah sebagai berikut

```
def linear(self, x):
    return x
```

1.3. Implementasi ReLU

Fungsi aktivasi untuk fungsi ReLU adalah sebagai berikut

```
def relu(self, x):
    return np.maximum(0, x)
```

1.4. Implementasi Sigmoid

Fungsi aktivasi untuk fungsi Sigmoid adalah sebagai berikut

```
def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))
```

1.5. Implementasi Softmax

Fungsi aktivasi untuk fungsi Softmax adalah sebagai berikut

```
def softmax(self, x):
    expX = np.exp(x - np.max(x, axis=1, keepdims=True))
    return expX / np.sum(expX, axis=1, keepdims=True)
```

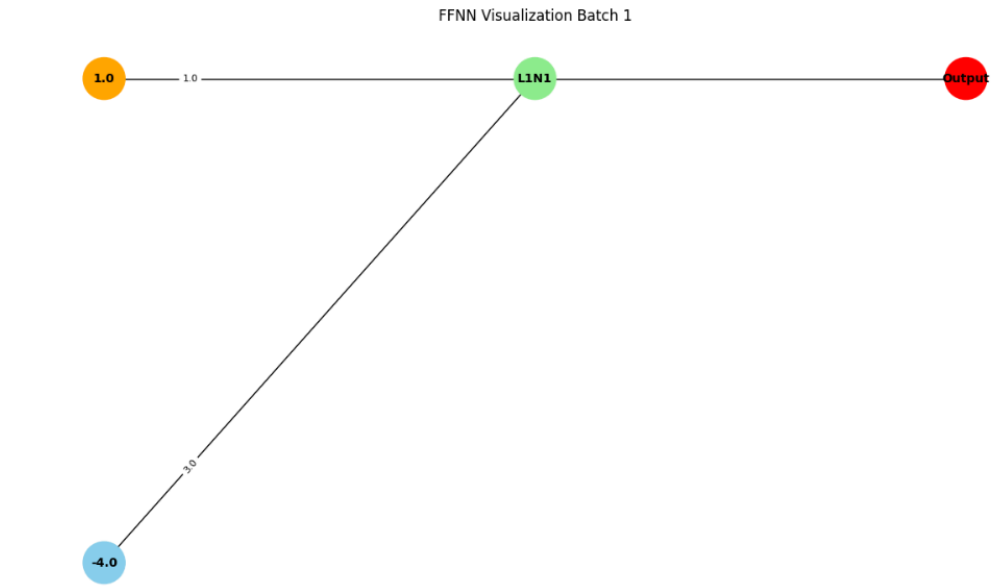
Bab II
Hasil Pengujian

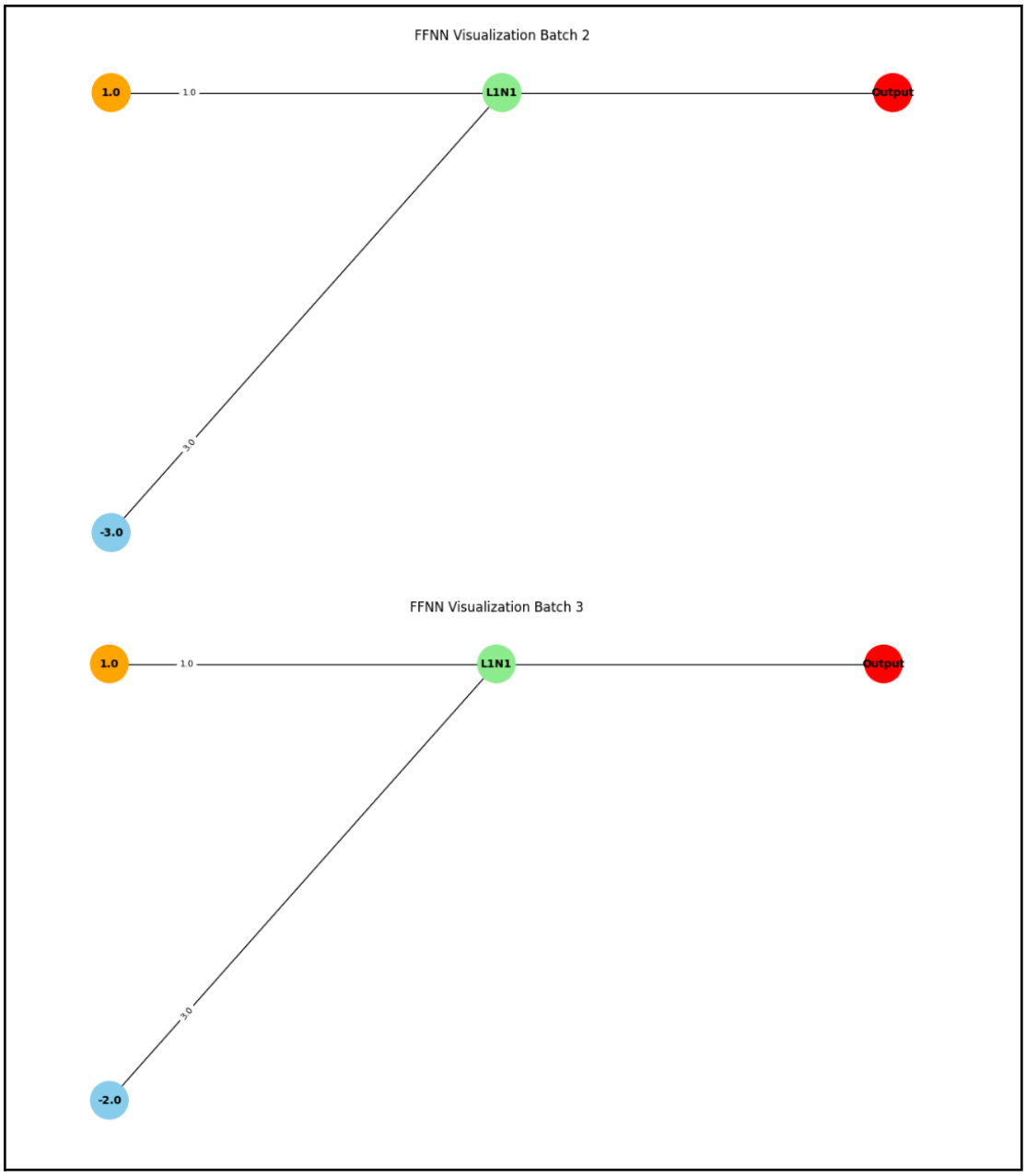
2.1. Hasil Pengujian *Test Case* Linear

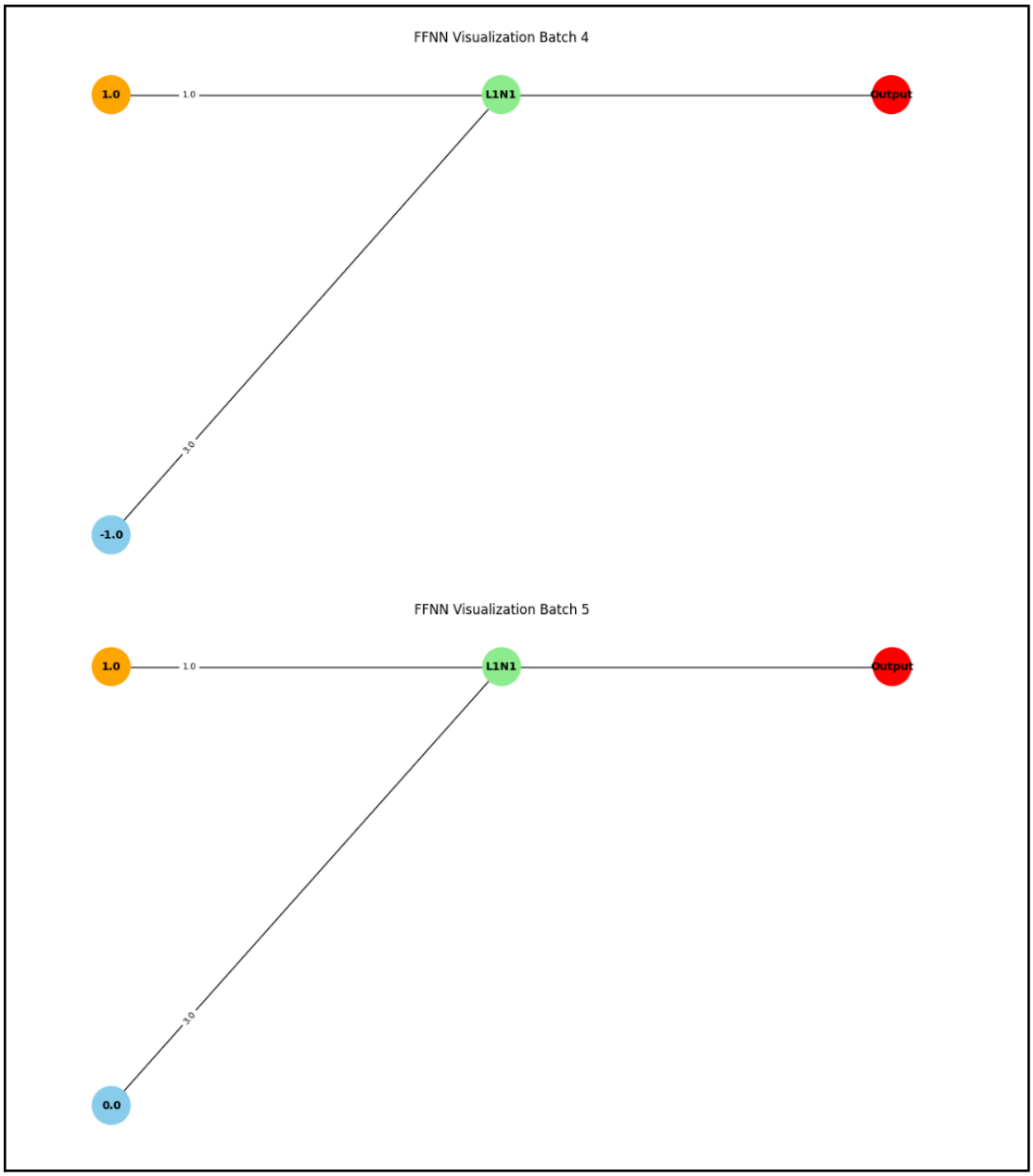
Output (Log)

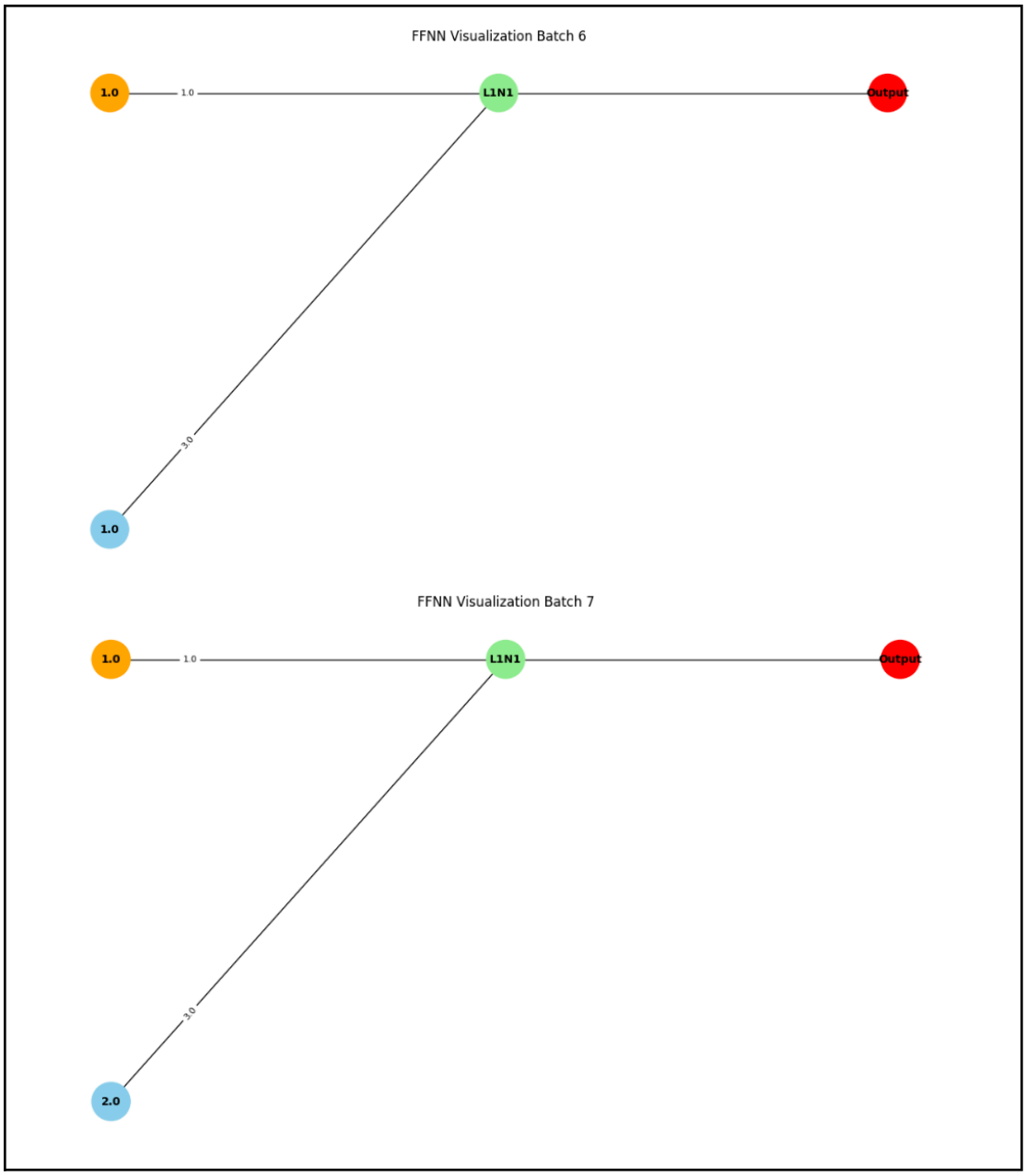
```
Output: [[-11.]
 [ -8.]
 [ -5.]
 [ -2.]
 [  1.]
 [  4.]
 [  7.]
 [ 10.]
 [ 13.]
 [ 16.]]
Sum of Squared Errors (SSE): 0.0
```

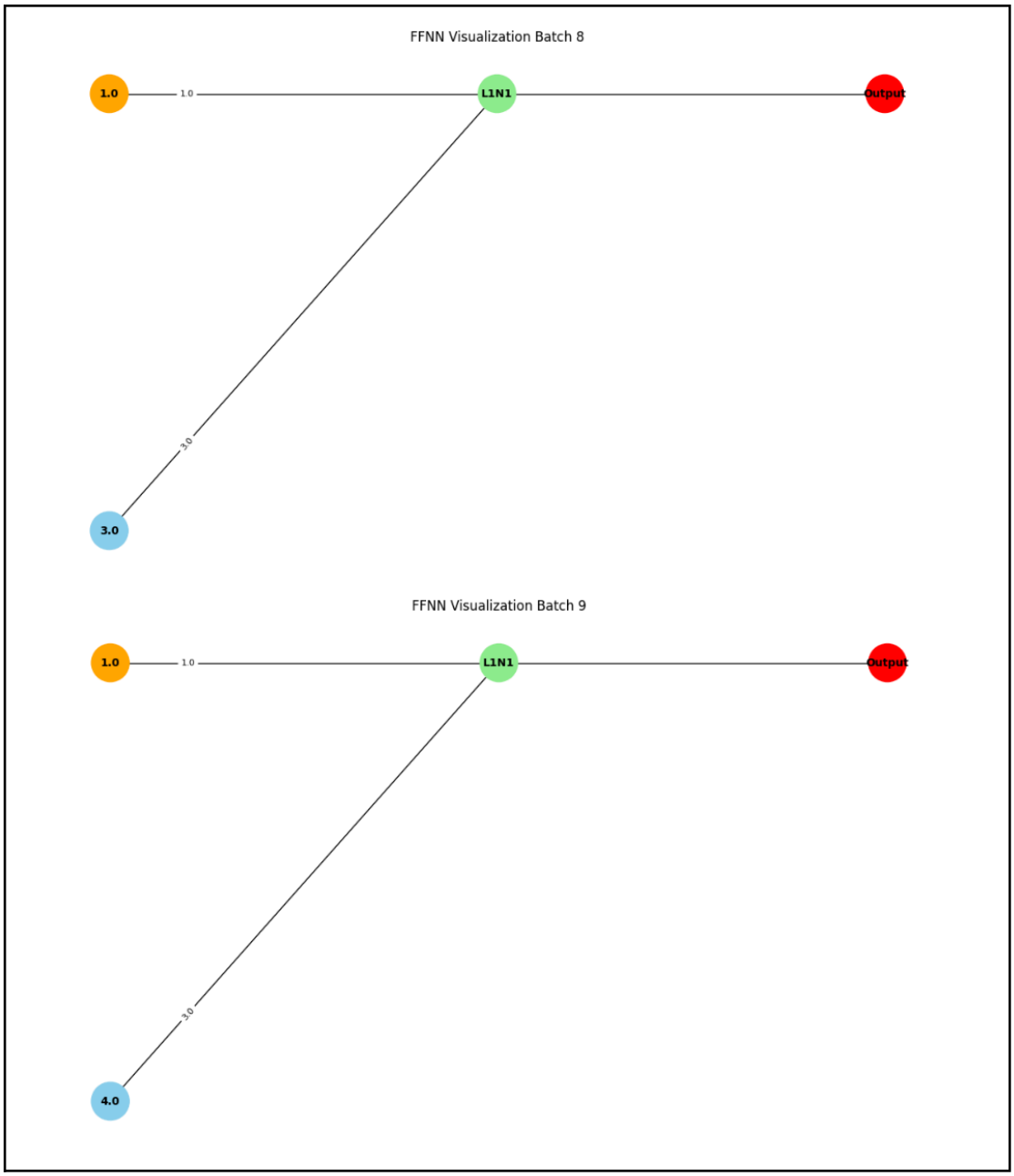
Output (Network)

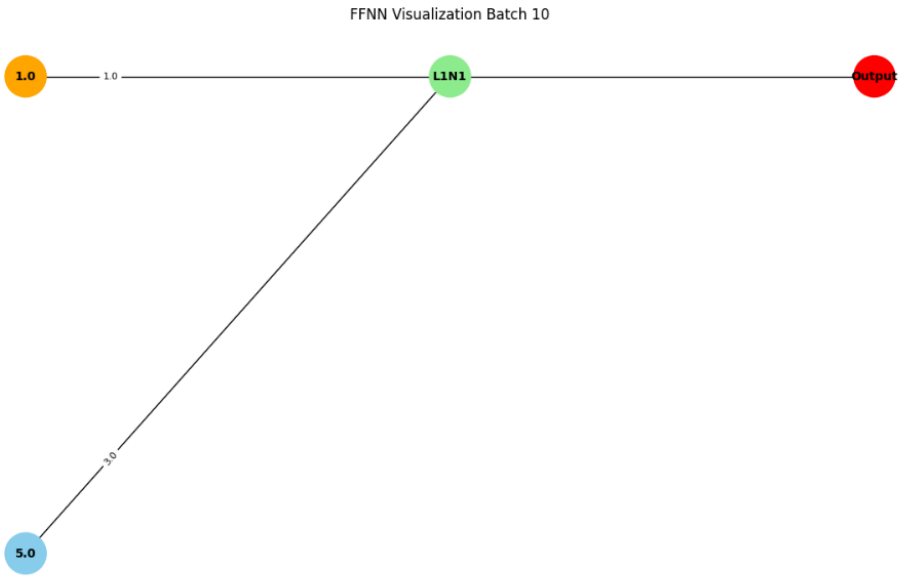



FFNN Visualization Batch 2
1.0
1.0
L1N1
Output
-3.0
3.0
FFNN Visualization Batch 3
1.0
1.0
L1N1
Output
-2.0
3.0


FFNN Visualization Batch 4
1.0
L1N1
Output
-1.0
1.0
3.0
FFNN Visualization Batch 5
1.0
L1N1
Output
0.0
1.0
3.0


FFNN Visualization Batch 6
1.0
1.0
L1N1
Output
3.0
1.0
FFNN Visualization Batch 7
1.0
1.0
L1N1
Output
3.0
2.0


FFNN Visualization Batch 8
1.0
L1N1
Output
3.0
1.0
3.0
FFNN Visualization Batch 9
1.0
L1N1
Output
4.0
1.0
3.0



Penjelasan

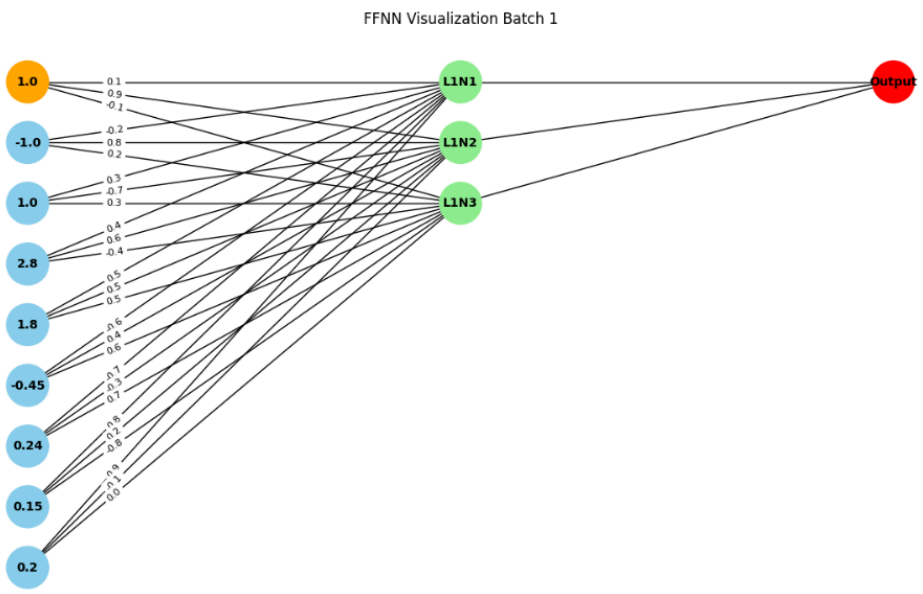
Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan terbagi menjadi 10 *batch* yang berbeda. Untuk setiap *batch* terdapat 1 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 1 *layer* dengan jumlah *node* adalah 1 *node*. Terakhir, model akan menghasilkan satu *node* output.

2.2. Hasil Pengujian *Test Case* Softmax

Output (Log)

```
Output: [[0.76439061 0.21168068 0.02392871]]
Sum of Squared Errors (SSE): 1.2639167711800341e-1
```

Output (Network)



Penjelasan

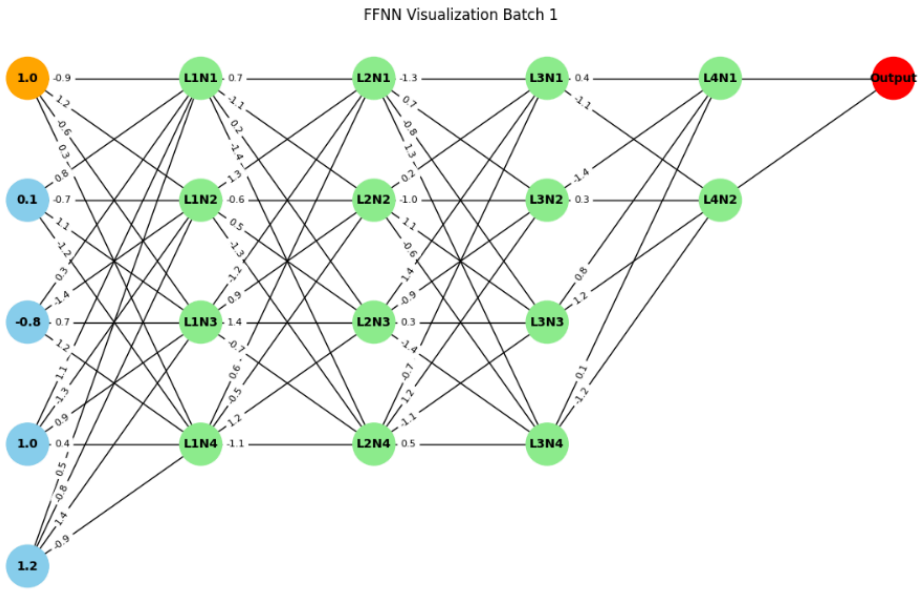
Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan hanya terdapat 1 *batch*. Untuk *batch* tersebut, terdapat 8 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 1 *layer* dengan jumlah *node* adalah 3 *node*. Terakhir, model akan menghasilkan satu *node* output.

2.3. Hasil Pengujian *Test Case* Multilayer Softmax

Output (Log)

Output: [[0.7042294 0.2957706]]
Sum of Squared Errors (SSE): 1.942282085801731e-19

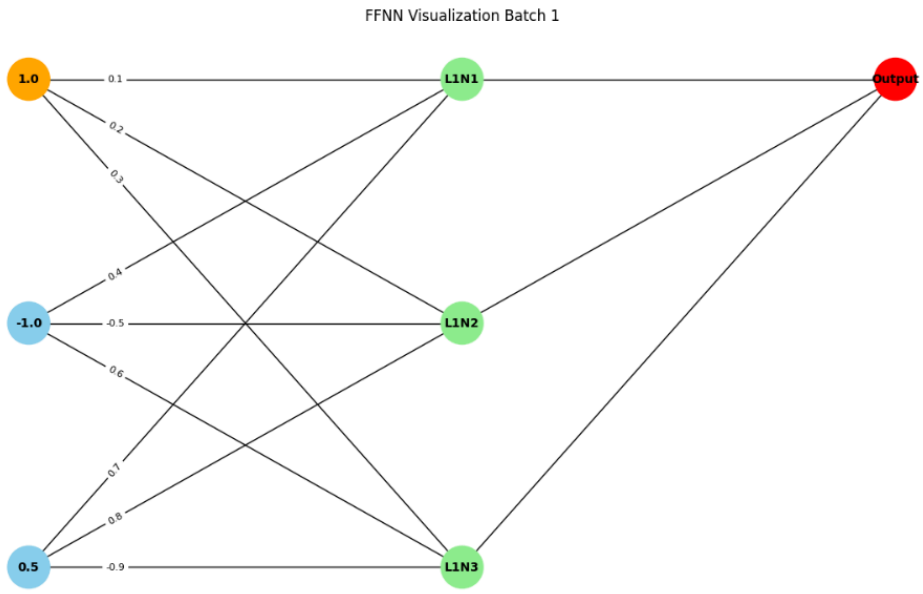
Output (Network)



Penjelasan

Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan hanya terdapat 1 *batch*. Untuk *batch* tersebut, terdapat 4 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 3 *layer* dengan jumlah *node* masing-masing adalah 3, 3, 3, dan 2 *node*. Terakhir, model akan menghasilkan satu *node* output.

2.4. Hasil Pengujian *Test Case* ReLU

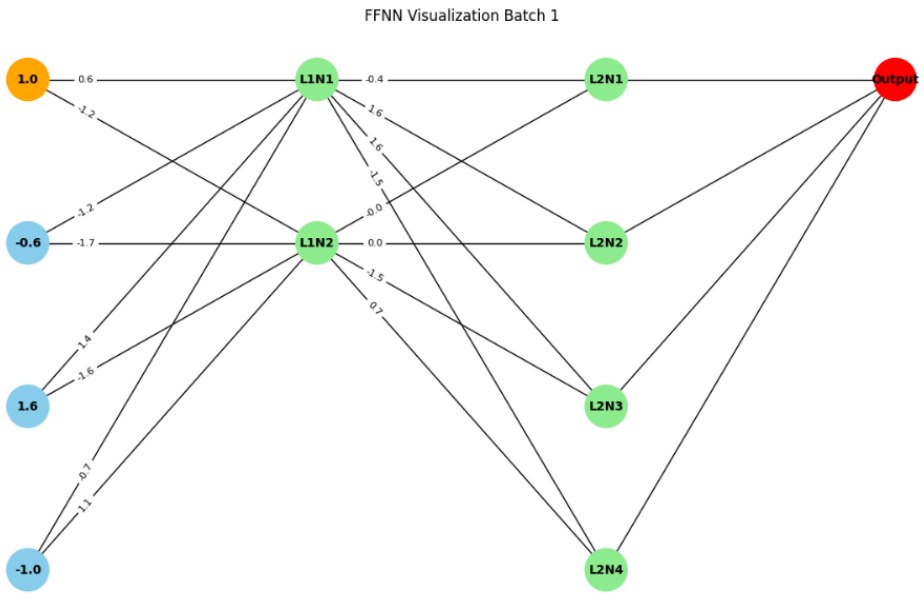
Output (Log)
Output: [[0.05 1.1 0.]] Sum of Squared Errors (SSE): 4.8148248609680896e-33
Output (Network)

Penjelasan
Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan hanya terdapat 1 *batch*. Untuk *batch* tersebut, terdapat 2 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 1 *layer* dengan jumlah *node* adalah 3 *node*. Terakhir, model akan menghasilkan satu *node* output.

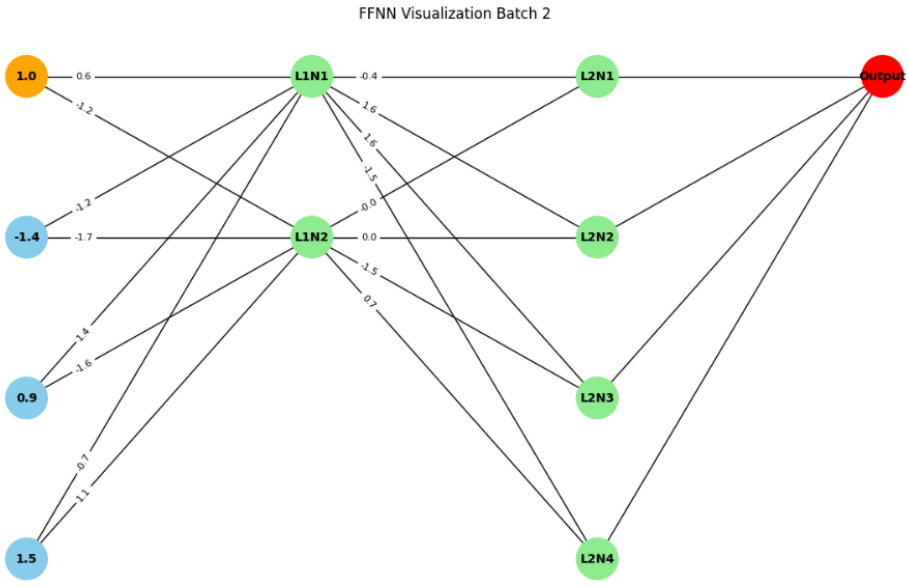
2.5. Hasil Pengujian *Test Case* Sigmoid

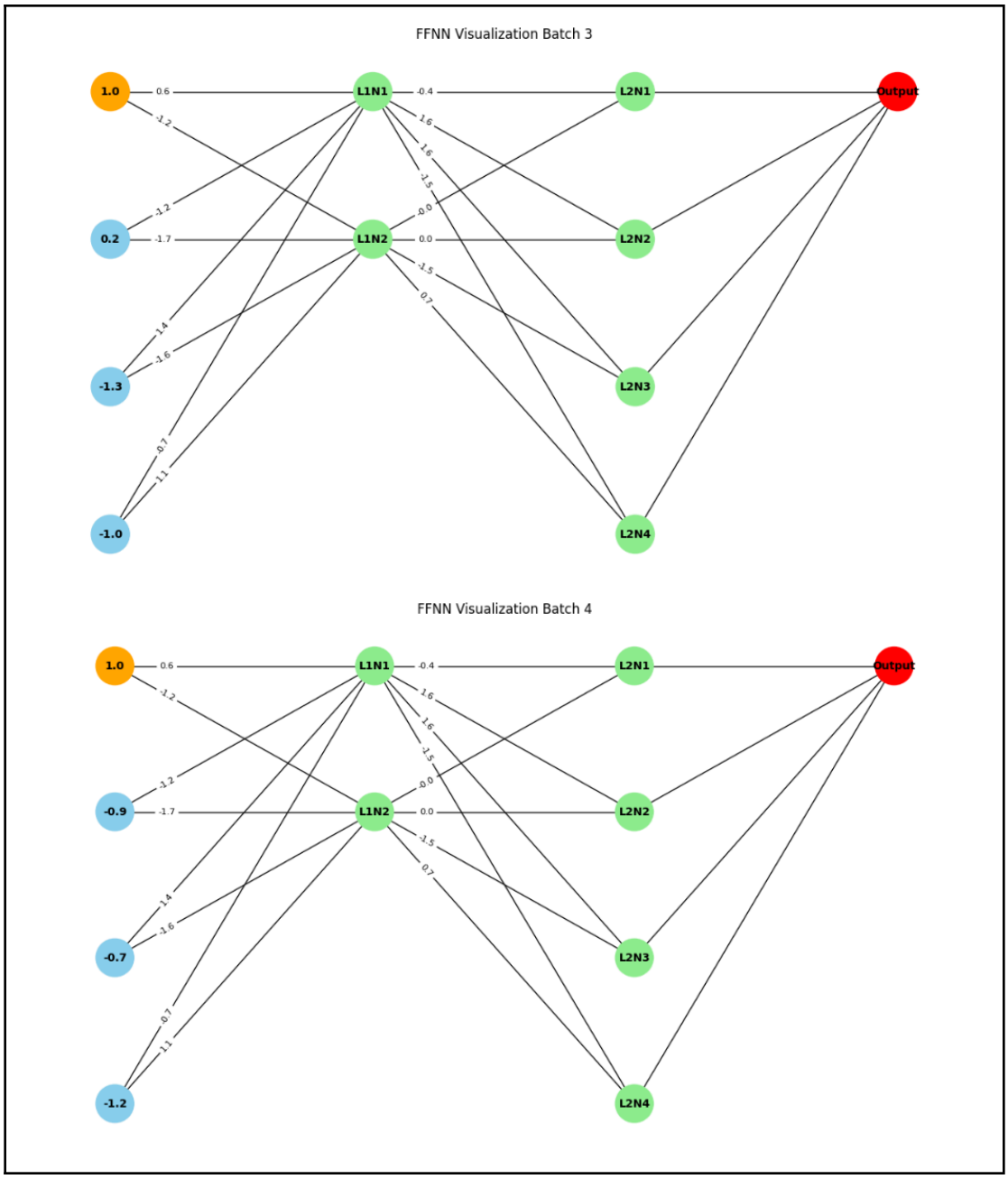
Output (Log)

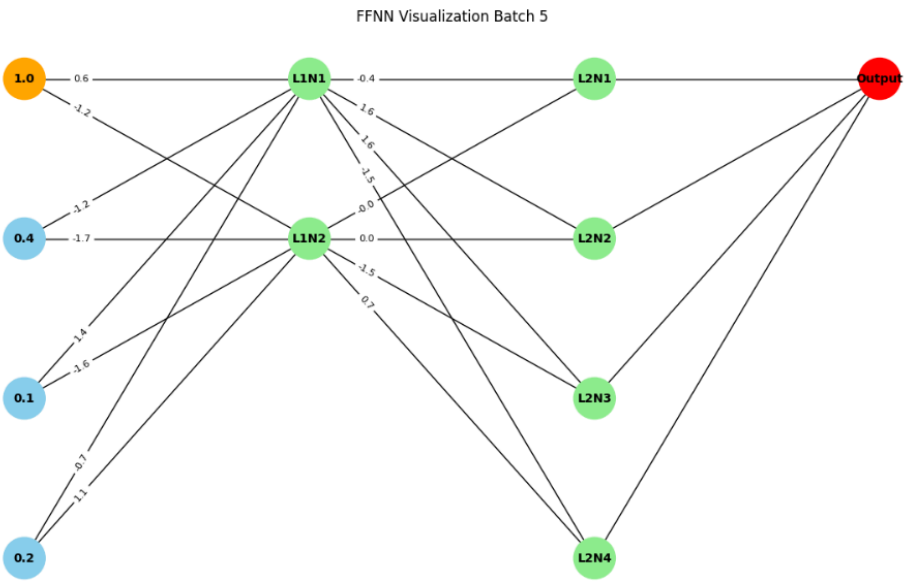
```
Output: [[0.41197346 0.8314294  0.53018536 0.31607396]
 [0.78266141 0.80843631 0.55350518 0.64278501]
 [0.58987524 0.82160954 0.75436518 0.34919895]
 [0.6722004  0.81660439 0.59020258 0.50870988]
 [0.47322841 0.82808466 0.69105452 0.29358323]]
Sum of Squared Errors (SSE): 2.1756063320477236e-16
```

Output (Network)





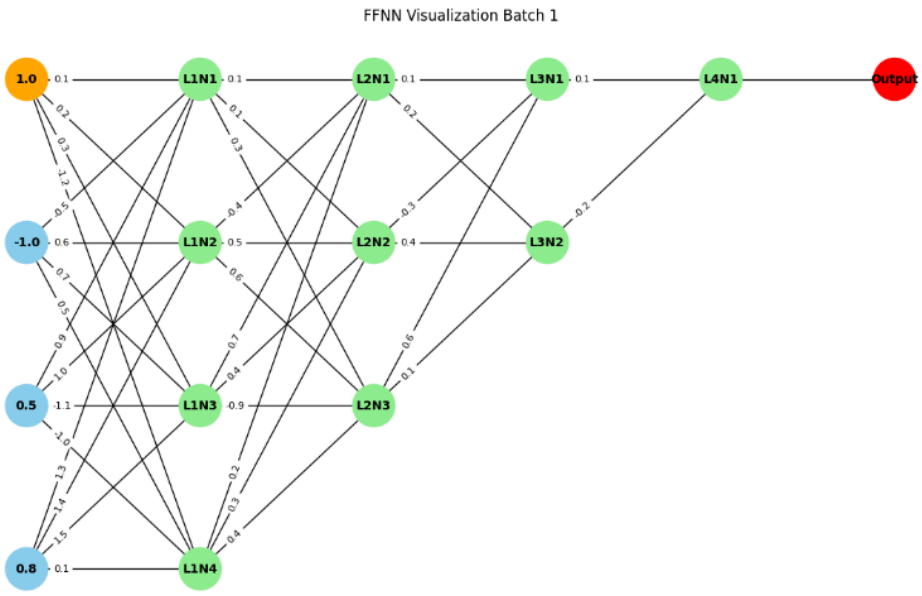

FFNN Visualization Batch 3
1.0
0.2
-1.3
-1.0
L1N1
L1N2
L2N1
L2N2
L2N3
L2N4
Output
FFNN Visualization Batch 4
1.0
-0.9
-0.7
-1.2
L1N1
L1N2
L2N1
L2N2
L2N3
L2N4
Output



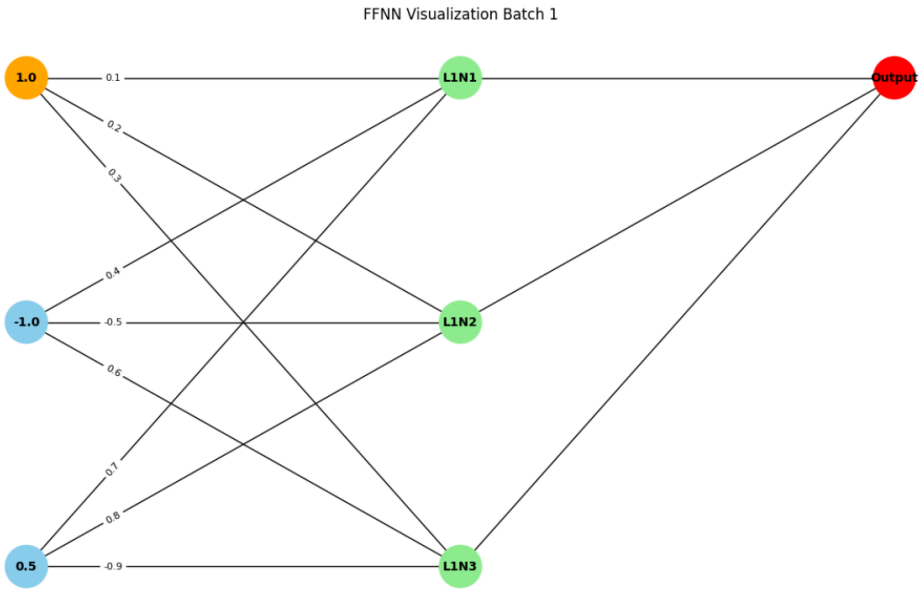
Penjelasan

Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan terbagi menjadi 5 *batch* yang berbeda. Untuk setiap *batch* terdapat 3 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 2 *layer* dengan masing-masing jumlah *node* adalah 2 dan 4 *node*. Terakhir, model akan menghasilkan satu *node* output.

2.6. Hasil Pengujian *Test Case* Multilayer

Output (Log)
Output: [[0.4846748]] Sum of Squared Errors (SSE): 3.1555532735024718e-18
Output (Network)

Penjelasan
Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan hanya terdapat 1 *batch*. Untuk *batch* tersebut, terdapat 3 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 4 *layer* dengan masing-masing jumlah *node* adalah 4, 3, 2, dan 1 *node*. Terakhir, model akan menghasilkan satu *node* output.

2.7. Hasil Pengujian *Test Case* Init

Output (Log)
Output: [[0.05 1.1 0.]] Sum of Squared Errors (SSE): 4.8148248609680896e-33
Output (Network)

Penjelasan
Dalam visualisasi tersebut, dapat dilihat bahwa input yang digunakan hanya terdapat 1 *batch*. Untuk *batch* tersebut, terdapat 2 nilai input dan 1 nilai bias untuk jumlah *node input*. Banyak *layer* perantara adalah 1 *layer* dengan jumlah *node* adalah 3 *node*. Terakhir, model akan menghasilkan satu *node* output.

Bab III
Perbandingan dengan Perhitungan Manual

Perhitungan manual dilakukan dengan bantuan *tools* Excel yang dapat diakses pada link ini.

3.1. Perbandingan Hasil Pengujian *Test Case* Linear

LINEAR

	b	x	w0	w1	Sum	Output (After Activation
1	1	-4	1	3	-11	-11
2	1	-3	1	3	-8	-8
3	1	-2	1	3	-5	-5
4	1	-1	1	3	-2	-2
5	1	0	1	3	1	1
6	1	1	1	3	4	4
7	1	2	1	3	7	7
8	1	3	1	3	10	10
9	1	4	1	3	13	13
10	1	5	1	3	16	16

3.2. Perbandingan Hasil Pengujian *Test Case* ReLU

RELU

	b	x1	x2	w0	w1	w2	Sum	Output (After Activation
O1	1	1.5	-0.45	0.1	0.47	1.1	0.31	0.31
O2	1	1.5	-0.45	0.2	-0.6	-1.3	-0.115	0
O3	1	1.5	-0.45	0.3	0.2	0.5	0.375	0.375

3.3. Perbandingan Hasil Pengujian *Test Case* Sigmoid

SIGMOID

	b	x1	x2	x3	w0.01	w0.02	w0.11	w0.12	w0.21	w0.22	w0.31	w0.32	Sum L11	Sum L12	Output L11	Output L12
1	1	-0.6	1.6	-1	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	4.26	-3.84	0.9860743597	0.02104134702
2	1	-1.4	0.9	1.5	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	2.49	1.39	0.9234378026	0.8005922432
3	1	0.2	-1.3	-1	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	-0.76	-0.56	0.3186462662	0.3635474597
4	1	-0.9	-0.7	-1.2	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	1.54	0.13	0.8234647252	0.5324543064
5	1	0.4	0.1	0.2	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	0.12	-1.82	0.5299640518	0.139433873

w1.01	w1.02	w1.03	w1.04	w1.11	w1.12	w1.13	w1.14	w1.21	w1.22	w1.23	w1.24	Sum L21	Sum L22	Sum L23	Sum L24	Output L21	Output L22	Output L23	Output L24
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.355813171	1.595791731	0.120888460	0.771873523	0.4119734556	0.8314293994	0.5301853633	0.3160739649
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	1.281243711	1.439881551	0.214843296	0.587472499	0.7826614091	0.8084363083	0.5535051761	0.6427850098
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.363449665	1.527290508	1.122030601	0.622562186	0.5898752435	0.8216095373	0.7543651777	0.3491989467
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.718154043	1.493509139	0.364802912	0.034843059	0.6722003954	0.816604391	0.5902025844	0.5087098836
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.107188866	1.572113225	0.805053922	0.878044192	0.473228411	0.8280846566	0.6910545249	0.2935832342

3.4. Perbandingan Hasil Pengujian *Test Case* Softmax

SOFTMAX

	b	x1	x2	x3	x4	x5	x6	x7	x8	w0	w1	w2	w3	w4	w5	w6	w7	w8	Sum	Output (After Activation
O1	1	-1	1	2.8	1.8	-0.45	0.24	0.15	0.2	0.1	-0.2	0.3	0.4	0.5	-0.6	-0.7	0.8	0.9	3.022	0.7643906087
O2	1	-1	1	2.8	1.8	-0.45	0.24	0.15	0.2	0.9	0.8	-0.7	0.6	0.5	0.4	-0.3	0.2	-0.1	1.738	0.2116806829
O3	1	-1	1	2.8	1.8	-0.45	0.24	0.15	0.2	-0.1	0.2	0.3	-0.4	0.5	0.6	0.7	-0.8	0	-0.442	0.0239287084

3.5. Perbandingan Hasil Pengujian *Test Case* Multilayer Softmax

MULTILAYER SOFTMAX

LAYER 0->1

	b	x1	x2	x3	x4	w0	w1	w2	w3	w4	Sum	Output
L11	1	0.1	-0.8	1	1.2	-0.9	0.8	0.3	1.1	0.5	0.64	0.64
L12	1	0.1	-0.8	1	1.2	1.2	-0.7	-1.4	-1.3	-0.8	-0.01	0
L13	1	0.1	-0.8	1	1.2	-0.6	1.1	0.7	0.9	1.4	1.53	1.53
L14	1	0.1	-0.8	1	1.2	0.3	-1.2	1.2	0.4	-0.9	-1.46	0

LAYER 1->2

	b	x1	x2	x3	x4	w0	w1	w2	w3	w4	Sum	Output
L21	1	0.64	0	1.53	0	0.7	1.3	-1.2	0.6	1	2.45	2.45
L22	1	0.64	0	1.53	0	-1.1	-0.6	0.9	-0.5	-0.4	-2.249	0
L23	1	0.64	0	1.53	0	0.2	0.5	1.4	1.2	0.8	2.356	2.356
L24	1	0.64	0	1.53	0	-1.4	-1.3	-0.7	-1.1	-1	-3.915	0

LAYER 2->3

	b	x1	x2	x3	x4	w0	w1	w2	w3	w4	Sum	Output
L31	1	2.45	0	2.356	0	-1.3	0.2	1.4	-0.7	0.9	-2.4592	0
L32	1	2.45	0	2.356	0	0.7	-1	-0.9	1.2	-0.7	1.0772	1.0772
L33	1	2.45	0	2.356	0	-0.8	1.1	0.3	-1.1	1.3	-0.6966	0
L34	1	2.45	0	2.356	0	1.3	-0.6	-1.4	0.5	-0.8	1.008	1.008

LAYER 3->4

	b	x1	x2	x3	x4	w0	w1	w2	w3	w4	Sum	Output
O1	1	0	1.0772	0	1.008	0.4	-1.4	0.8	0.1	1.2	2.47136	0.7042293997
O2	1	0	1.0772	0	1.008	-1.1	0.3	1.2	-1.2	1.4	1.60384	0.2957706003

3.6. Perbandingan Hasil Pengujian *Test Case* Multilayer

MULTILAYER SOFTMAX

LAYER 0->1

	b	x1	x2	x3		w0	w1	w2	w3		Sum	Output
L11	1	-1	0.5	0.8		0.1	-0.5	0.9	1.3		2.09	2.09
L12	1	-1	0.5	0.8		0.2	0.6	1	1.4		1.22	1.22
L13	1	-1	0.5	0.8		0.3	0.7	-1.1	1.5		0.25	0.25
L14	1	-1	0.5	0.8		-1.2	0.5	-1	0.1		-2.12	0

LAYER 1->2

	b	L11	L12	L13	L14	w0	w1	w2	w3	w4	Sum	Output
L21	1	2.09	1.22	0.25	0	0.1	-0.4	0.7	0.2	-0.1	0.168	0.168
L22	1	2.09	1.22	0.25	0	0.1	0.5	0.4	0.3	0.2	1.708	1.708
L23	1	2.09	1.22	0.25	0	0.3	0.6	-0.9	0.4	0.1	0.556	0.556

LAYER 2->3

	b	L21	L22	L23		w0	w1	w2	w3		Sum	Output
L31	1	0.168	1.708	0.556		0.1	-0.3	0.6	0.1		1.13	1.13
L32	1	0.168	1.708	0.556		0.2	0.4	0.1	-0.4		0.2156	0.2156

LAYER 3->4

	b	L31	L32			w0	w1	w2			Sum	Output
O1	1	1.13	0.2156			0.1	-0.2	0.3			-0.06132	0.4846748018

Bab IV
Pembagian Tugas Anggota Kelompok

Nama	NIM	Tugas
Juan Christopher Santoso	13521116	FFNN Class, Graph Visualization
Nicholas Liem	13521135	JSON Parser, FFNN Class, Graph Visualization
Nathania Calista Djunaedi	13521139	FFNN Class, Graph Visualization
Antonio Natthan Krishna	13521162	FFNN Class, Graph Visualization, Perhitungan Manual

Lampiran

Berikut adalah pranala *repository* yang digunakan untuk menyimpan program yang telah dikembangkan: https://github.com/NicholasLiem/IF3270_FP_FFNN .