

# **Laporan Tugas Besar**

## **IF3270 - Pembelajaran Mesin**

Artificial Neural Network Part 1:  
Feed Forward Neural Network

Dipersiapkan oleh

Angger Ilham Amanullah	13521001
Kelvin Rayhan Alkarim	13521005
Ditra Rizqa Amadia	13521019
Bernardus Willson	13521021



## I. Penjelasan implementasi

Berikut ini merupakan implementasi serta penjelasan algoritma utama yang telah kami buat.

```
1  class FFNN:
2      # Initialize model
3      def __init__(self, model):
4          try:
5              self._layers = model["case"]["model"]["layers"]
6              self._weights = model["case"]["weights"]
7              self._inputs = model["case"]["input"]
8              self._outputs = []
9              self._outputs
10             self._expect = model["expect"]
11         except:
12             print(f"[X] Error initialiazing model")
13             print("")
14
15             Driver.terminate(1)
16
17             print(f"[✓] Successfully initialized model")
18             print(f"Number of features: {model['case']['model']['input_size']}")
19             print(f"Number of layers: {len(self._layers)}")
20             print(f"Number of data: {len(self._inputs)}")
21             print("")
```

Hal pertama yang dilakukan setelah membaca input .json adalah menginisiasi class FFNN. Berikut adalah penjelasan setiap atributnya:

- **layers** : atribut ini mempresentasikan lapisan-lapisan neuron dengan jumlah neuron pada setiap layer dan jenis fungsi aktivasi pada layer tersebut (dapat berupa relu, sigmoid, linear, dan softmax).
- **weights** : atribut ini mempresentasikan bobot yang digunakan untuk menghubungkan jaringan neuron antar layer dengan layer selanjutnya.
- **inputs** : atribut ini digunakan untuk menerima data nilai input yang akan diproses.
- **outputs** : atribut ini digunakan untuk menyimpan hasil dari pemrosesan Feed Forward Neural Network.

- `expect` : atribut ini digunakan untuk menyimpan hasil output yang diharapkan dari proses jaringan neuron yang telah terjadi. Selain itu, atribut ini juga menyimpan nilai maksimum dari Sum of Squared Errors (SSE) yang akan digunakan sebagai batas galat dari output yang dihasilkan.

```
1  # Build the network
2  def build(self):
3
4      # Calculate output for each input
5      for i in range(len(self._inputs)):
6
7          print(f"[●] Evaluating data {i + 1} =====")
8          print("")
9
10         _in = self._inputs[i]
11
12         # Build each layer
13         for j in range(len(self._layers)):
14
15             # Create layer
16             layer = Layer(j, self._layers[j], self._weights[j], _in)
17
18             # Build layer
19             _in = layer.build()
20
21         self._outputs.append(_in)
22
23         print(f"[✓] Successfully evaluated data {i + 1}")
24         print(f"Input: {self._inputs[i]}")
25         print(f"Output: {self._outputs[i]}")
26         print("")
```

Setiap input akan dievaluasi / diproses dengan membuat objek 'Layer' pada setiap layer. Objek layer dibuat dengan menggunakan konfigurasi layer, bobot yang sesuai, dan data masukan dari iterasi sebelumnya, karena seperti yang sudah dijelaskan di atas bahwa setiap layer akan saling terhubung dengan layer selanjutnya.


```

1  class Layer:
2      # Initialize layer
3      def __init__(self, id, layer, weight, input):
4          try:
5              self._id = id
6              self._num_of_neuron = layer["number_of_neurons"]
7              self._activation_function = layer["activation_function"]
8              self._weight = weight
9              self._input = input
10             self._input = np.insert(self._input, 0, 1) # Append bias of 1
11             self._sigma = np.zeros(self._num_of_neuron)
12             self._output = np.zeros(self._num_of_neuron)
13         except:
14             print(f"[X] Error initialiazing layer")
15             print("")
16
17             Driver.terminate(1)

```

Lakukan inisiasi layer dengan membentuk beberapa atribut. Berikut adalah penjelasan setiap atributnya:

- id: atribut ini mempresentasikan urutan layer yang sedang ditinjau.
- num\_of\_neuron: atribut ini merepresentasikan jumlah neuron yang ada di layer tersebut.
- Activation\_function: atribut ini merepresentasikan jenis activation function yang digunakan pada layer tersebut.
- weight : atribut ini mempresentasikan bobot yang digunakan untuk menghubungkan jaringan neuron antar layer dengan layer selanjutnya.
- input : atribut ini merepresentasikan data input (dapat berupa data input pertama kali, atau hasil dari fungsi aktivasi pada layer sebelumnya).
- sigma : atribut ini merepresentasikan jumlah bobot setelah dikalikan dengan data input berdasarkan atribut 'input'.
- output: atribut ini merepresentasikan data yang dihasilkan setelah melalui pemrosesan pada layer tersebut.




```

1  # Build the layer
2  def build(self):
3      print(f"[●] Building layer {self._id}")
4
5      # For each neuron, calculate its activation
6      for i in range(self._num_of_neuron):
7          self._calculate_sigma(i)
8          self._calculate_activation(i)
9
10     print(f"[✓] Successfully built layer {self._id}")
11     print(f"Number of neurons: {self._num_of_neuron}")
12     print(f"Activation function: {self._activation_function}")
13     for i in range(len(self._output)):
14         print(f"h{self._id}{i}: {self._output[i]}")
15     print("")
16
17     return self._output

```

Setelah menginisiasi layer, program ini akan melakukan perhitungan sigma dan activation untuk setiap neuron dengan penggunaan for loop dan pemanggilan fungsi calculate\_sigma dan calculate\_activation.



```

1  # Calculate sigma
2  def _calculate_sigma(self, i):
3      for j in range(len(self._input)):
4          self._sigma[i] += self._input[j] * self._weight[j][i]

```

Fungsi ini digunakan untuk menghitung sigma pada setiap neuron dengan menggunakan rumus  $w \cdot x$  atau  $w_0 \cdot bias + w_1 \cdot x_1 + \dots + w_n \cdot x_n$

```
1 # Calculate activation
2 def _calculate_activation(self, i):
3     if self._activation_function == 'relu':
4         self._output[i] = max(0, self._sigma[i])
5     elif self._activation_function == 'sigmoid':
6         self._output[i] = 1 / (1 + np.exp(-self._sigma[i]))
7     elif self._activation_function == 'linear':
8         self._output[i] = self._sigma[i]
9     elif self._activation_function == 'softmax':
10        exp_values = np.exp(self._sigma - np.max(self._sigma))
11        self._output = exp_values / np.sum(exp_values)
```

Setelah mendapatkan nilai sigma, hitung nilai aktivasi sesuai dengan fungsi aktivasinya:

- relu:  $\max(0, \sigma)$
- sigmoid:  $\frac{1}{1 + e^{-(\sigma)}}$
- linear:  $\sigma$
- softmax:  $\frac{e^{\sigma}}{\sum_i e^{\sigma(i)}}$

Hasil dari fungsi aktivasi kemudian akan dipakai menjadi input pada layer selanjutnya. Kemudian proses-proses di atas diulang secara iteratif sampai mencapai output layer. Hasil fungsi aktivasi pada output layer inilah yang menjadi hasil dari pemrosesan FFNN.

## II. Hasil pengujian

### 1. relu.json

```
[✓] Successfully loaded model from 'models/relu.json'

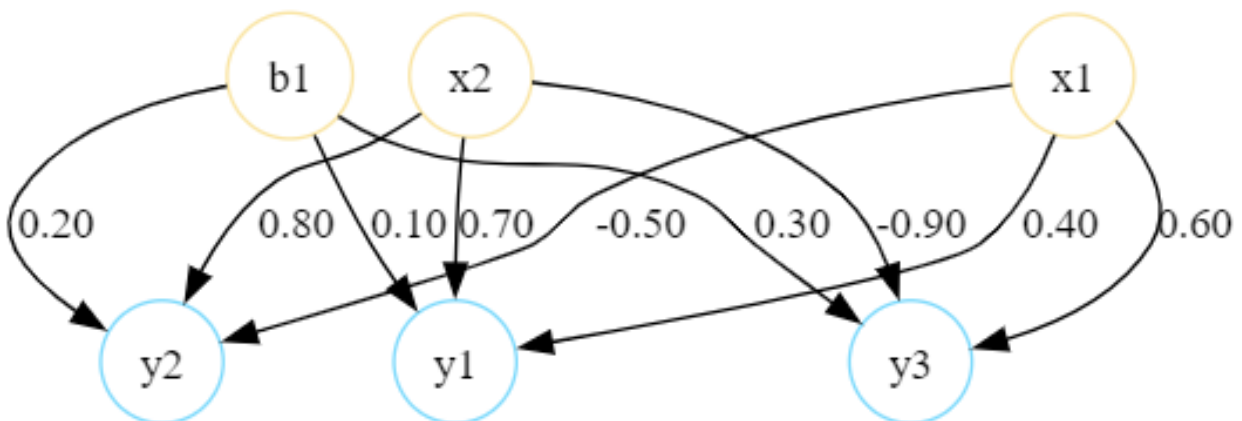
[✓] Successfully initialized model
Number of features: 2
Number of layers: 1
Number of data: 1

[•] Evaluating data 1 =====

[•] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 3
Activation function: relu
h00: 0.04999999999999993
h01: 1.1
h02: 0.0

[✓] Successfully evaluated data 1
Input: [-1.0, 0.5]
Output: [0.05 1.1 0. ]

[•] Asserting output =====
Test status: PASS
Test result: 1/1
```



## 2. multilayer.json

```
[✓] Successfully loaded model from 'models/multilayer.json'

[✓] Successfully initialized model
Number of features: 3
Number of layers: 4
Number of data: 1

[●] Evaluating data 1 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 4
Activation function: relu
h00: 2.09
h01: 1.22
h02: 0.25000000000000002
h03: 0.0

[●] Building layer 1
[✓] Successfully built layer 1
Number of neurons: 3
Activation function: relu
h10: 0.16800000000000004
h11: 1.7080000000000002
h12: 0.55599999999999998

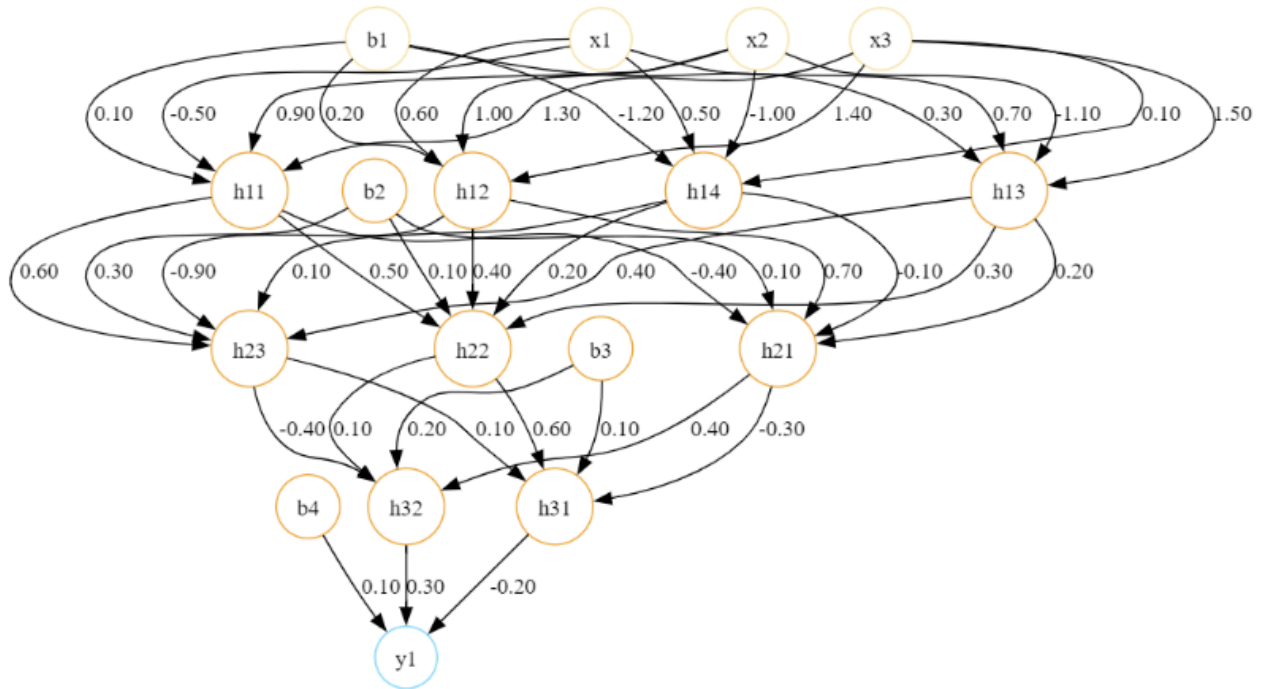
[●] Building layer 2
[✓] Successfully built layer 2
Number of neurons: 2
Activation function: relu
h20: 1.1300000000000003
h21: 0.21560000000000012

[●] Building layer 3
[✓] Successfully built layer 3
Number of neurons: 1
Activation function: sigmoid
h30: 0.4846748017763877

[✓] Successfully evaluated data 1
Input: [-1.0, 0.5, 0.8]
Output: [0.4846748]

[●] Asserting output =====
Test status: PASS
Test result: 1/1
```





### 3. linear.json

```
[✓] Successfully loaded model from 'models/linear.json'

[✓] Successfully initialized model
Number of features: 1
Number of layers: 1
Number of data: 10

[●] Evaluating data 1 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 1
Activation function: linear
h00: -11.0

[✓] Successfully evaluated data 1
Input: [-4.0]
Output: [-11.]

[●] Evaluating data 2 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 1
Activation function: linear
```

h00: -8.0

[✓] Successfully evaluated data 2

Input: [-3.0]

Output: [-8.]

[●] Evaluating data 3 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: -5.0

[✓] Successfully evaluated data 3

Input: [-2.0]

Output: [-5.]

[●] Evaluating data 4 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: -2.0

[✓] Successfully evaluated data 4

Input: [-1.0]

Output: [-2.]

[●] Evaluating data 5 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: 1.0

[✓] Successfully evaluated data 5

Input: [0.0]

Output: [1.]

[●] Evaluating data 6 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: 4.0

[✓] Successfully evaluated data 6

Input: [1.0]

Output: [4.]

[●] Evaluating data 7 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: 7.0

[✓] Successfully evaluated data 7

Input: [2.0]

Output: [7.]

[●] Evaluating data 8 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: 10.0

[✓] Successfully evaluated data 8

Input: [3.0]

Output: [10.]

[●] Evaluating data 9 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

h00: 13.0

[✓] Successfully evaluated data 9

Input: [4.0]

Output: [13.]

[●] Evaluating data 10 =====

[●] Building layer 0

[✓] Successfully built layer 0

Number of neurons: 1

Activation function: linear

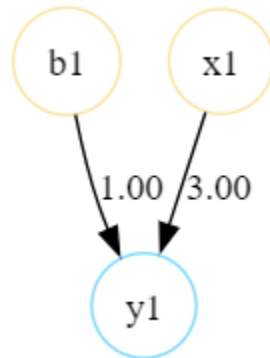
h00: 16.0

[✓] Successfully evaluated data 10

Input: [5.0]

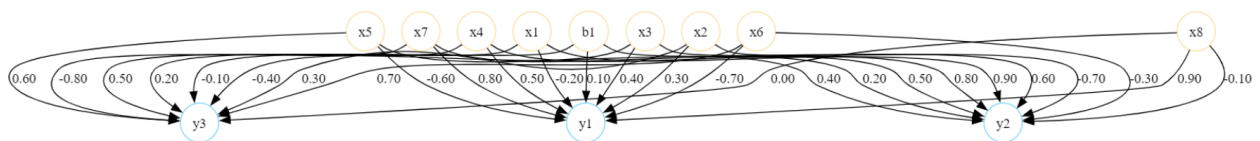
Output: [16.]

```
[●] Asserting output =====  
Test status: PASS  
Test result: 10/10
```



#### 4. softmax.json

```
[✓] Successfully loaded model from 'models/softmax.json'  
  
[✓] Successfully initialized model  
Number of features: 8  
Number of layers: 1  
Number of data: 1  
  
[●] Evaluating data 1 =====  
  
[●] Building layer 0  
[✓] Successfully built layer 0  
Number of neurons: 3  
Activation function: softmax  
h00: 0.7643906087005896  
h01: 0.21168068289764494  
h02: 0.023928708401765492  
  
[✓] Successfully evaluated data 1  
Input: [-1.0, 1.0, 2.8, 1.8, -0.45, 0.24, 0.15, 0.2]  
Output: [0.76439061 0.21168068 0.02392871]  
  
[●] Asserting output =====  
Test status: PASS  
Test result: 1/1
```



## 5. sigmoid.json

```
[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 2
Activation function: sigmoid
h00: 0.9234378026481879
h01: 0.8005922431513315

[●] Building layer 1
[✓] Successfully built layer 1
Number of neurons: 4
Activation function: sigmoid
h10: 0.7826614091150284
h11: 0.8084363083194981
h12: 0.5535051760955713
h13: 0.6427850098146376

[✓] Successfully evaluated data 2
Input: [-1.4, 0.9, 1.5]
Output: [0.78266141 0.80843631 0.55350518 0.64278501]

[●] Evaluating data 3 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 2
Activation function: sigmoid
h00: 0.31864626621097447
h01: 0.36354745971843366

[●] Building layer 1
[✓] Successfully built layer 1
Number of neurons: 4
Activation function: sigmoid
h10: 0.5898752435296561
h11: 0.8216095372737501
h12: 0.7543651777362295
h13: 0.34919894670366747

[✓] Successfully evaluated data 3
Input: [0.2, -1.3, -1.0]
Output: [0.58987524 0.82160954 0.75436518 0.34919895]

[●] Evaluating data 4 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 2
Activation function: sigmoid
h00: 0.8234647252208833
```

```
h01: 0.5324543063873187

[●] Building layer 1
[✓] Successfully built layer 1
Number of neurons: 4
Activation function: sigmoid
h10: 0.6722003953978934
h11: 0.8166043910016146
h12: 0.5902025844263002
h13: 0.5087098836277426

[✓] Successfully evaluated data 4
Input: [-0.9, -0.7, -1.2]
Output: [0.6722004  0.81660439 0.59020258 0.50870988]

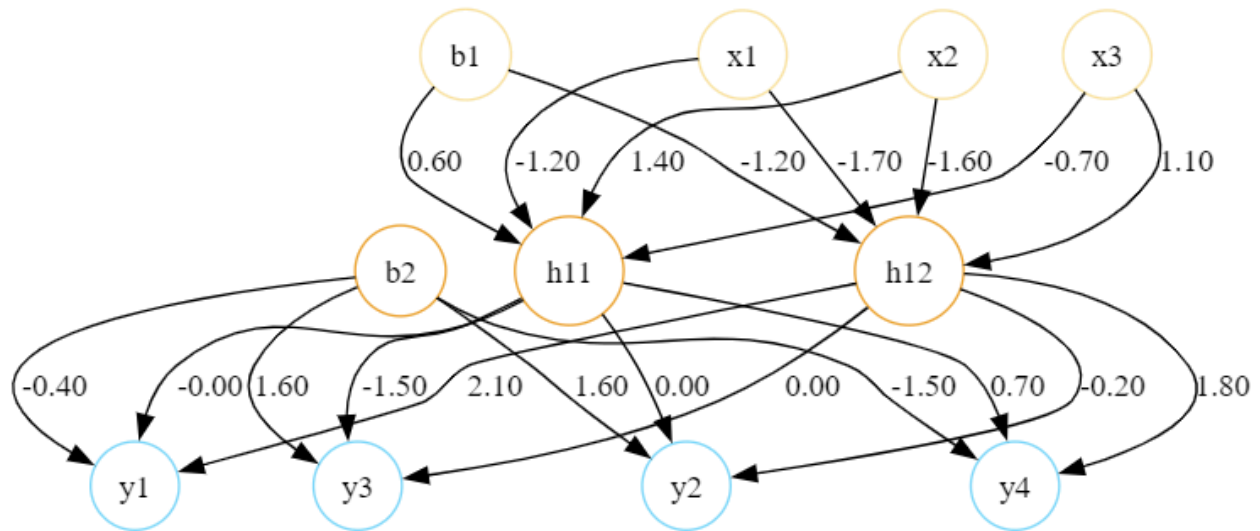
[●] Evaluating data 5 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 2
Activation function: sigmoid
h00: 0.5299640517645717
h01: 0.13943387296165005

[●] Building layer 1
[✓] Successfully built layer 1
Number of neurons: 4
Activation function: sigmoid
h10: 0.47322841097431845
h11: 0.8280846566130694
h12: 0.6910545248579614
h13: 0.2935832341653264

[✓] Successfully evaluated data 5
Input: [0.4, 0.1, 0.2]
Output: [0.47322841 0.82808466 0.69105452 0.29358323]

[●] Asserting output =====
Test status: PASS
Test result: 5/5
```



## 6. multilayer\_softmax.json

```
[✓] Successfully loaded model from 'models/multilayer_softmax.json'
```

```
[✓] Successfully initialized model
```

```
Number of features: 4
```

```
Number of layers: 4
```

```
Number of data: 1
```

```
[●] Evaluating data 1 =====
```

```
[●] Building layer 0
```

```
[✓] Successfully built layer 0
```

```
Number of neurons: 4
```

```
Activation function: relu
```

```
h00: 0.64
```

```
h01: 0.0
```

```
h02: 1.5300000000000002
```

```
h03: 0.0
```

```
[●] Building layer 1
```

```
[✓] Successfully built layer 1
```

```
Number of neurons: 4
```

```
Activation function: relu
```

```
h10: 2.45
```

```
h11: 0.0
```

```
h12: 2.3560000000000003
```

```
h13: 0.0
```

```
[●] Building layer 2
```

```
[✓] Successfully built layer 2
```

```
Number of neurons: 4
```

```
Activation function: relu
```

```

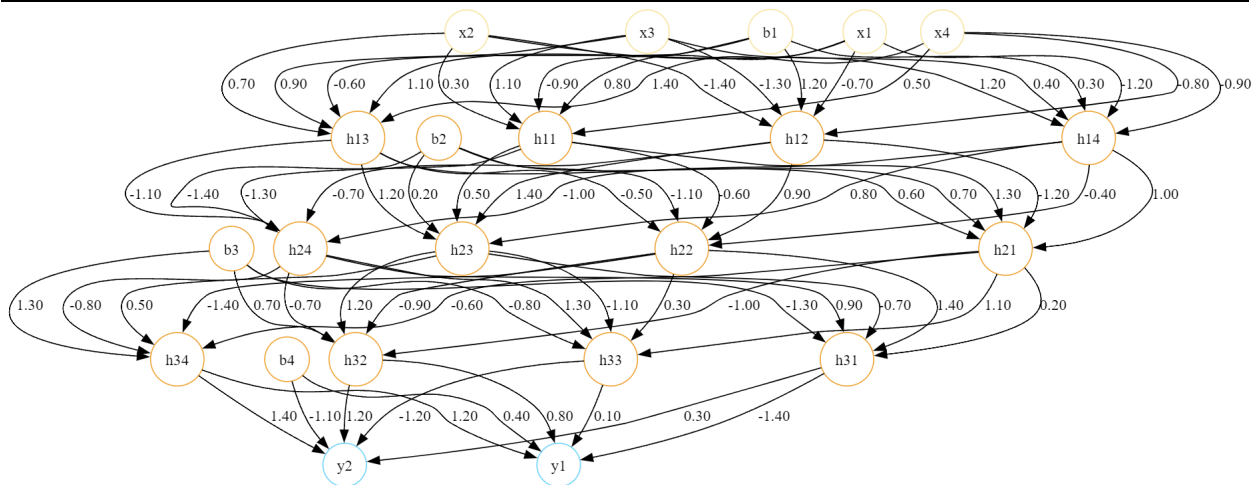
h20: 0.0
h21: 1.0772000000000002
h22: 0.0
h23: 1.0080000000000002

[●] Building layer 3
[✓] Successfully built layer 3
Number of neurons: 2
Activation function: softmax
h30: 0.7042293996883686
h31: 0.29577060031163127

[✓] Successfully evaluated data 1
Input: [0.1, -0.8, 1, 1.2]
Output: [0.7042294 0.2957706]

[●] Asserting output =====
Test status: PASS
Test result: 1/1

```



## 7. relu2.json

```

[✓] Successfully loaded model from 'models/relu2.json'

[✓] Successfully initialized model
Number of features: 2
Number of layers: 1
Number of data: 1

[●] Evaluating data 1 =====

[●] Building layer 0
[✓] Successfully built layer 0
Number of neurons: 3

```



```

Activation function: relu
h00: 0.30999999999999999
h01: 0.0
h02: 0.37500000000000001

```

```

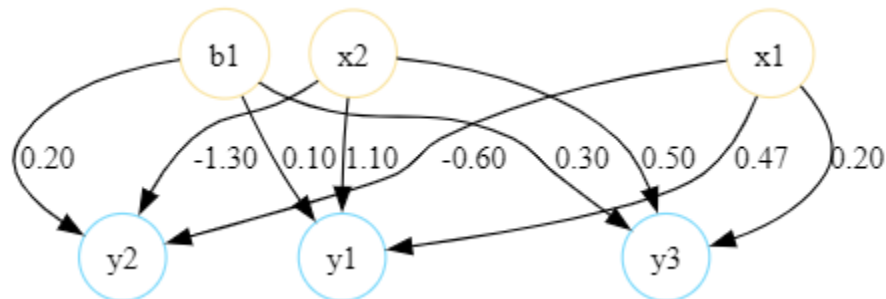
[✓] Successfully evaluated data 1
Input: [1.5, -0.45]
Output: [0.31 0. 0.375]

```

```

[●] Asserting output =====
Test status: PASS
Test result: 1/1

```



### III. Perbandingan dengan hasil perhitungan manual

#### 1. relu.json

relu	B1	X1	X2	W01	W02	W03	W11	W12	W13	W21	W22	W23	Sum H1	Sum H2	Sum H3	Actv H1	Actv H2	Actv H3
1	1	-1	0.5	0.1	0.2	0.3	0.4	-0.5	0.6	0.7	0.8	-0.9	0.05	1.1	-0.75	0.05	1.1	0

Berdasarkan perhitungan manual pada testcase relu.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (output layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: **0.05**
- Neuron 2: **1.1**
- Neuron 3: **0**

Hasil perhitungan berdasarkan kode pada [relu.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file relu.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

## 2. multilayer.json

relu	B1	X1	X2	X3	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4
1	1	-1	0.5	0.8	0.1	0.2	0.3	-1.2	-0.5	0.6	0.7	0.9	0.0	1	-1.1	-1	1.3	1.4	1.5	0.1	2.09	1.22	0.25	-2.12	2.09	1.22	0.25	0
relu	B2	W01	W02	W03	W11	W12	W13	W21	W22	W23	W31	W32	W33	W41	W42	W43	Sum H1	Sum H2	Sum H3	Actv H1	Actv H2	Actv H3						
1	1	0.1	0.1	0.3	-0.4	0.5	0.6	0.7	0.4	-0.9	0.2	0.3	0.4	-0.1	0.2	0.1	0.168	1.708	0.556	0.168	1.708	0.556						
relu	B3	W01	W02	W11	W12	W21	W22	W31	W32	Sum H1	Sum H2	Actv H1	Actv H2															
1	1	0.1	0.2	-0.3	0.4	0.6	0.1	0.1	-0.4	1.13	0.2156	1.13	0.2156															
sigmoid	B4	W01	W11	W21	Sum H1	Actv H1																						
1	1	0.1	-0.2	0.3	-0.09132	0.484674																						

Berdasarkan perhitungan manual pada testcase multilayer.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 2.09
- Neuron 2: 1.22
- Neuron 3: 0.25
- Neuron 4: 0

Kemudian hasil perhitungan manual untuk setiap neuron pada layer kedua (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 0.168
- Neuron 2: 1.708
- Neuron 3: 0.556

Kemudian hasil perhitungan manual untuk setiap neuron pada layer ketiga (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 0.13
- Neuron 2: 0.2156

Kemudian hasil perhitungan manual untuk setiap neuron pada layer keempat (output layer) dengan fungsi aktivasi sigmoid:

- Data 1:

- Neuron 1: **0.4846748018**

Hasil perhitungan berdasarkan kode pada [multilayer.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file multilayer.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

### 3. linear.json

linear	B1	X1	W01	W11	sum H1	Actv H1
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

Berdasarkan perhitungan manual pada testcase linear.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (output layer) dengan fungsi aktivasi linear:

- Data 1:

- Neuron 1: **-11**

- Data 2:

- Neuron 1: **-8**

- Data 3:

- Neuron 1: **-5**

- Data 4:

- Neuron 1: **-2**

- Data 5:

- Neuron 1: **1**

- Data 6:

- Neuron 1: **4**

- Data 7:

- Neuron 1: **7**

- Data 8:

- Neuron 1: **10**

- Data 9:

- Neuron 1: **13**

- Data 10:

- Neuron 1: **16**

Hasil perhitungan berdasarkan kode pada [linear.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file

linear.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

#### 4. softmax.json

softmax	B1	X1	X2	X3	X4	X5	X6	X7	X8	W01	W02	W03	W11	W12	W13	W21	W22	W23	W31	W32	W33	W41	W42	W43	W51	W52	W53	W61	W62	W63	W71	W72	W73	W81	W82	W83	Sum H1	Sum H2	Sum H3	Exp H1	Exp H2	Exp H3	Actv H1	Actv H2	Actv H3
1	1	-1	1	2.8	1.8	-0.45	0.24	0.15	0.2	0.1	0.9	-0.1	-0.2	0.9	0.2	0.3	-0.7	0.5	0.4	0.6	-0.4	0.5	0.5	0.5	-0.6	0.4	0.6	-0.7	-0.3	0.7	0.5	0.2	-0.6	0.9	-0.1	0	3.022	1.736	-0.442	20.5323	1.60300	0.042748	0.794395	0.211885	0.923925

Berdasarkan perhitungan manual pada testcase softmax.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (output layer) dengan fungsi aktivasi softmax:

- Data 1:

- Neuron 1: **0.7643906087**
- Neuron 2: **0.2116806829**
- Neuron 3: **0.0239287084**

Hasil perhitungan berdasarkan kode pada [softmax.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file softmax.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

#### 5. sigmoid.json

sigmoid	B1	X1	X2	X3	W01	W02	W11	W12	W21	W22	W31	W32	Sum H1	Sum H2	Actv H1	Actv H2
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	0.986074	0.021041
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	0.923437	0.800592
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	0.318646	0.363547
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	0.823464	0.532454
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	0.529964	0.139433

sigmoid	B2	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4
1	1	-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	-0.355813	1.595791	0.120888	-0.771873	0.411973	0.831429	0.530185	0.316073
2	1	-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	1.281243	1.439881	0.214843	0.587472	0.782661	0.808436	0.553505	0.642785
3	1	-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.363449	1.527290	1.122030	-0.622562	0.589875	0.821609	0.754365	0.349198
4	1	-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	0.718154	1.493509	0.364802	0.034843	0.672200	0.816604	0.590202	0.508709
5	1	-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	-0.107188	1.572113	0.805053	-0.878044	0.473228	0.828084	0.691054	0.293583

Berdasarkan perhitungan manual pada testcase sigmoid.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (hidden layer) dengan fungsi aktivasi sigmoid:

- Data 1:

- Neuron 1: 0.9860743597
- Neuron 2: 0.02104134702

- Data 2:

- Neuron 1: 0.9234378026
- Neuron 2: 0.8005922432

- Data 3:

- Neuron 1: 0.3186462662
- Neuron 2: 0.3635474597

- Data 4:

- Neuron 1: 0.8234647252
  - Neuron 2: 0.5324543064
- Data 5:
- Neuron 1: 0.5299640518
  - Neuron 2: 0.139433873

Kemudian hasil perhitungan manual untuk setiap neuron pada layer kedua (output layer) dengan fungsi aktivasi sigmoid:

- Data 1:
- Neuron 1: 0.4119734556
  - Neuron 2: 0.8314293994
  - Neuron 3: 0.5301853633
  - Neuron 4: 0.3160739649
- Data 2:
- Neuron 1: 0.7826614091
  - Neuron 2: 0.8084363083
  - Neuron 3: 0.5535051761
  - Neuron 4: 0.6427850098
- Data 3:
- Neuron 1: 0.5898752435
  - Neuron 2: 0.8216095373
  - Neuron 3: 0.7543651777
  - Neuron 4: 0.3491989467
- Data 4:
- Neuron 1: 0.6722003954
  - Neuron 2: 0.816604391
  - Neuron 3: 0.5902025844
  - Neuron 4: 0.5087098836
- Data 5:
- Neuron 1: **0.473228411**
  - Neuron 2: **0.8280846566**
  - Neuron 3: **0.6910545249**
  - Neuron 4: **0.2935832342**

Hasil perhitungan berdasarkan kode pada [sigmoid.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file sigmoid.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

## 6. multilayer\_softmax.json

relu	B1	X1	X2	X3	X4	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	W41	W42	W43	W44	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4	
	1	1	0.1	-0.8	1	1.2	-0.9	1.2	-0.6	0.3	0.8	-0.7	1.1	-1.2	0.3	-1.4	0.7	1.2	1.1	-1.3	0.9	0.4	0.5	-0.8	1.4	-0.9	0.64	-0.01	1.53	-1.46	0.64	0	1.53	0
relu	B2	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	W41	W42	W43	W44	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4					
	1	0.7	-1.1	0.2	-1.4	1.3	-0.6	0.5	-1.3	-1.2	0.9	1.4	-0.7	0.6	-0.5	1.2	-1.1	1	-0.4	0.8	-1	2.45	-2.249	2.356	-3.915	2.45	0	2.356	0					
relu	B2	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	W41	W42	W43	W44	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4					
	1	-1.3	0.7	-0.6	1.3	0.2	-1	1.1	-0.6	1.4	-0.9	0.3	-1.4	-0.7	1.2	-1.1	0.5	0.9	-0.7	1.3	-0.8	-2.4592	1.0772	-0.6966	1.008	0	1.0772	0	1.008					
softmax	B2	W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	W41	W42	W43	W44	Sum H1	Sum H2	Sum H3	Sum H4	Actv H1	Actv H2	Actv H3	Actv H4					
	1	0.4	-1.1	-1.4	0.3	0.8	1.2	0.1	-1.2	1.2	1.4	2.47136	1.60384	11.839534	4.972098	0.704229	0.295770																	

Berdasarkan perhitungan manual pada testcase multilayer\_softmax.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 0.64
- Neuron 2: 0
- Neuron 3: 1.53
- Neuron 4: 0

Kemudian hasil perhitungan manual untuk setiap neuron pada layer kedua (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 2.45
- Neuron 2: 0
- Neuron 3: 2.356
- Neuron 4: 0

Kemudian hasil perhitungan manual untuk setiap neuron pada layer ketiga (hidden layer) dengan fungsi aktivasi relu:

- Data 1:

- Neuron 1: 0
- Neuron 2: 1.0772
- Neuron 3: 0
- Neuron 4: 1.008

Kemudian hasil perhitungan manual untuk setiap neuron pada layer keempat (output layer) dengan fungsi aktivasi softmax:

- Data 1:

- Neuron 1: **0.7042293997**
- Neuron 2: **0.2957706003**

Hasil perhitungan berdasarkan kode pada [multilayer\\_softmax.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada

file multilayer\_softmax.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

## 7. relu2.json

relu	B1	X1	X2	W01	W02	W03	W11	W12	W13	W21	W22	W23	Sum H1	Sum H2	Sum H3	Actv H1	Actv H2	Actv H3
1	1	1.5	-0.45	0.1	0.2	0.3	0.47	-0.6	0.2	1.1	-1.3	0.5	0.31	-0.115	0.375	0.31	0	0.375

Berdasarkan perhitungan manual pada testcase relu2.json, didapat nilai aktivasi untuk setiap neuron pada layer pertama (output layer) dengan fungsi aktivasi relu:

- Data 1:

- IV. Neuron 1: **0.31**
- V. Neuron 2: **0**
- VI. Neuron 3: **0.375**

Hasil perhitungan berdasarkan kode pada [relu2.json](#) sudah sesuai dengan hasil perhitungan manual, begitu pula dengan expected output yang terdapat pada file relu2.json. Hal ini menandakan bahwa hasil perhitungan berdasarkan kode maupun hasil perhitungan secara manual sudah benar.

## VII. Pembagian tugas

- 13521001 Angger Ilham Amanullah: Kode FFNN, Laporan
- 13521005 Kelvin Rayhan Alkarim : Kode FFNN, Laporan
- 13521019 Ditra Rizqa Amadia : Kode FFNN, Laporan
- 13521021 Bernardus Willson : Kode FFNN, Laporan

## VIII. Lampiran

- [Github](#)
- [Perhitungan Excel](#)