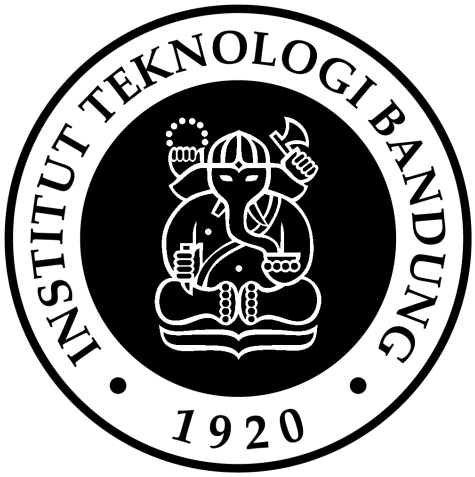
LAPORAN TUGAS BESAR A

“Implementasi Forward Propagation untuk Feed Forward Neural Network”

Dibuat untuk Memenuhi Tugas Mata Kuliah IF3270 Pembelajaran Mesin



Disusun Oleh:

13520130	Nelsen Putra
13521117	Maggie Zeta RS
13521133	Cetta Reswara Parahita
13521161	Ferindya Aulia Berlianty

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

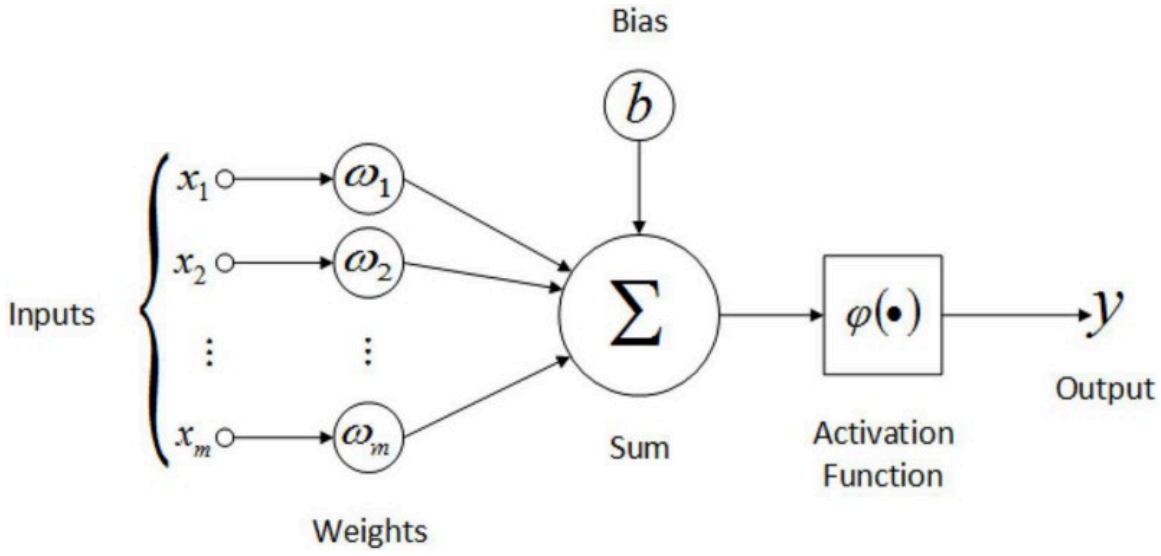
DAFTAR ISI

DASAR TEORI

A. *Feed Forward Neural Network*

Feed Forward Neural Network atau FFNN merupakan jaringan saraf tiruan yang paling sederhana. Arsitektur FFNN terdiri dari tiga jenis lapisan utama: lapisan input, satu atau lebih lapisan tersembunyi, dan lapisan *output*. Informasi dalam FFNN bergerak dari input, melalui lapisan tersembunyi, kemudian ke *output* tanpa siklus atau *loop* kembali ke lapisan sebelumnya, sehingga disebut "feed forward".

Dalam setiap neuron pada FFNN, dilakukan pengelolaan input seperti pada ilustrasi berikut. Nilai input akan dihitung berdasarkan pembobotan yang dimiliki masing-masing input. Selanjutnya dilakukan penambahan nilai bias dan hasil tersebut diolah melalui fungsi aktivasi menjadi sebuah output.



Secara umum struktur FFNN terdiri dari 3 lapisan:

1. Input *Layer*
   Lapisan ini berfungsi menerima input kemudian meneruskan ke lapisan selanjutnya.
2. *Hidden Layer*
   Lapisan ini berfungsi sebagai tempat di mana terjadinya pemrosesan aktual melalui neuron-neuron yang terhubung dengan bobot tertentu. Fungsi aktivasi diterapkan pada *output* dari setiap neuron. Lapisan ini tidak terlihat dari luar dan inilah yang memungkinkan FFNN mempelajari representasi abstrak.
3. *Output Layer*
   Lapisan ini berfungsi menghasilkan *output* akhir jaringan. Jumlah neuron di lapisan ini biasanya sesuai dengan jumlah kelas dalam tugas klasifikasi. Fungsi

aktivasi diterapkan pada *output* setiap neuron dan menentukan apakah neuron tersebut harus "diaktifkan" atau tidak. Beberapa fungsi aktivasi yang umum meliputi:

1. Linear : tidak mengubah input (*output*-nya adalah input linear).
2. Relu : memberi *output* 0 untuk input negatif dan *output* linear untuk input positif.
3. Sigmoid : mengubah *output* menjadi rentang antara 0 dan 1.
4. Softmax : digunakan di lapisan *output* untuk klasifikasi multikelas.

B. *Forward Propagation*

Forward Propagation adalah proses menghitung *output* jaringan dari input yang diberikan. Pada setiap lapisan, *output* dari semua neuron dihitung menggunakan bobot, bias, dan fungsi aktivasi. Proses ini berlangsung dari lapisan input hingga lapisan *output*.
Proses *Forward Propagation* :

1. Perkalian Bobot dan Input
   Setiap input dikalikan dengan bobot terkait dan bias ditambahkan untuk setiap neuron.
2. Aktivasi
   Output dari langkah sebelumnya diumpankan ke fungsi aktivasi neuron.
3. Propagasi ke Lapisan Berikutnya
   Output dari fungsi aktivasi menjadi input untuk lapisan berikutnya dan proses ini berulang hingga lapisan *output*.

PENJELASAN IMPLEMENTASI

Dalam mengimplementasikan forward pass pada algoritma FFNN, dibuat struktur repositori sebagai berikut:

```
├── README.md
├── main.ipynb
├── src
│   ├── __pycache__
│   ├── activation.py
│   ├── ffnn.py
│   ├── hidden_layer.py
│   ├── input_json.py
│   ├── layer.py
│   └── node.py
└── test
    ├── linear.json
    ├── multilayer.json
    ├── multilayer_softmax.json
    ├── relu.json
    ├── relu_1.json
    ├── sigmoid.json
    └── softmax.json
```

Folder ‘src’ berisi file-file kode dalam bahasa python yang masing-masing file berisi kelas-kelas modul sesuai dengan namanya untuk menangani struktur kelas yang diperlukan berikut fungsi-fungsi yang sering digunakan dalam mengakses tiap struktur kelas yang ada. Folder ‘test’ berisi test case dalam bahasa json sejumlah 7 dengan struktur:

```
{
	"case": {
		"model": {
			"input_size": 2,
			"layers": [ array of
					{
						"number_of_neurons": total neuron,
						"activation_function": fungsi aktivasi
					}
				]
		},
		"input": [array of array input dengan input size yang sudah
	ditetapkan],
		"weights": [array of weights array for all layers]
	},
	"expect": {
		"output": [array of expected output],
		"max_sse": threshold of sse yang diterima
		}
}
```

Masukan dalam json ini kemudian akan dikelola oleh library **json** yang diimplementasikan pada input_json.py. Terdapat dua fungsi utama yakni file_name(path) dan

open_json(file). Fungsi file_name(path) dibuat untuk melakukan validasi terhadap masukan nama file yang diberikan sedangkan open_json(file) adalah fungsi untuk mengambil data yang tersedia dalam file .json dan memberikan luaran berupa dictionary **case** dan dictionary **expect.**

Selanjutnya dari data yang dimiliki, case akan dimasukkan ke kelas FFNN untuk dikelola berdasarkan isi dari kelas FFNN. Berikut adalah cuplikan dari kode di ffnn.py:

```
class FFNN:
    '''Kelas untuk membuat model ffnn
    '''
    def __init__(self, case):
        self.input_size = case.get("model").get("input_size")
        self.nlayers = len(case.get("model").get("layers"))
        self.layers = HiddenLayer(case.get("model").get("layers"), case.get("weights"))
```

FFNN memiliki tiga atribut utama yakni (1) input_size berupa batasan size masukan yang dapat diterima oleh kelas model, (2) nlayers yakni jumlah layer pada model, dan (3) layers yang berisi hidden_layer dan output_layer. Selanjutnya diimplementasikan beberapa fungsi untuk melakukan kalkulasi dalam FFNN sebagai berikut:

```
def model_activator(self, input):
    model_copy = copy.deepcopy(self)
    model_copy.layers.activate_hidden_layers(input)
    return model_copy

def predict(self, input):
    output = []
    for i in range(len(input)):
        model_output = self.model_activator(input[i])
        output.append(model_output)
        self.layers.restart_hidden_layer()
    return output

def output_value(model):
    output_val = []
    for i in range(len(model)):
        output_val.append(model[i].layers.layers[-1].value)
    return output_val

def f_sse(output, expect):
    sse = []
    for i in range(len(output)):
        sse.append(0)
        for j in range(len(output[i])):
            sse[i] += (output[i][j] - expect[i][j]) ** 2
    return sse
```

1. Fungsi **model_activator()** merupakan fungsi yang mengaktifkan fungsi aktivasi pada layer-layer dibawahnya dan menyimpan model yang telah diaktivasi sebagai luaran.
2. Fungsi **predict()** merupakan fungsi yang melakukan prediksi pada sejumlah input yang dimasukkan dalam program. Input dalam berupa satuan maupun batch.
3. Fungsi **output_value()** merupakan fungsi yang melakukan ekstraksi array output value dari model yang sudah diprediksi.
4. Fungsi **f_sse()** merupakan fungsi yang menghitung *sum squared error* untuk mengetahui akurasi model setelah dibandingkan dengan ekspektasi luaran.

Kelas FFNN kemudian memanggil kelas HiddenLayer. Secara kasar kelas HiddenLayer merupakan kelas yang menyatukan berbagai layer yang tidak terlihat oleh pengguna dan layer paling akhirnya akan diekstraksi melalui kelas FFNN sebagai *output value*. HiddenLayer terdiri dari beberapa attribut antara lain n_layers yakni jumlah *hidden layer*, n_output yakni jumlah output, layer_sizes yang berupa array yang berisi ukuran atau jumlah neuron atau node pada setiap layer yang ada di dalam hidden layer, dan yang terakhir layers yang merupakan array berisi object Layer yang merupakan hidden layer pada model tersebut. Berikut adalah strukturnya dalam kode.

```
class HiddenLayer:
    def __init__(self, layers, weights):
        self.n_layers = len(layers)
        self.n_output = layers[self.n_layers-1].get("number_of_neurons")

        self.layer_sizes=np.zeros(self.n_layers)
        # Update layer size
        for i in range(self.n_layers):
            self.layer_sizes[i] = layers[i].get("number_of_neurons")

        self.layers = []
        # Update layers
        for i in range(self.n_layers):
            new_layers = Layer(weights[i], i+1, layers[i].get("activation_function"), self.layer_sizes[i])
            self.layers.append(new_layers)
```

Fungsi yang diterapkan di hidden layer cukup sederhana yakni **activate_hidden_layers()** yang akan mengaktivasi layer-layer di bawahnya secara rekursif dengan melakukan *passing* value yang telah didapatkan di layer sebelumnya. Sedangkan layer **restart_hidden_layer()** melakukan sebaliknya, yakni mengembalikan semua value dan net attribut menjadi 0 pada layer-layer di bawah hidden layer.

```
    def activate_hidden_layers(self, input):
        for i in range(self.n_layers):
            if (i == 0):
                self.layers[i].activate_layer(input)
            else:
                self.layers[i].activate_layer(self.layers[i-1].value)
        #return self.layers[-1]

    def restart_hidden_layer(self):
        for i in range(self.n_layers):
            self.layers[i].restart_layer()
```

Selanjutnya di kelas Layer dilakukan manajemen terhadap nodes atau neuron yang merupakan bagian dari layer tersebut. Attribut yang ada pada kelas tersebut antara lain nilai bias, weight yang telah di transpose, jumlah neuron, activation function dan namanya, dan beberapa array untuk penyimpanan data kolektif yakni value, net sum, dan node.

```
class Layer:
    def __init__(self, weight, name, activation, n_neuron):
        activation_function = {
            "linear" : linear,
            "relu" : relu,
            "sigmoid" : sigmoid,
            "softmax" : softmax,
            "None" : None
        }
        self.bias = 1
        self.weight = np.transpose(weight)
        self.name = name
        self.n_neuron = int(n_neuron)
        self.activation_function = activation_function[activation]
        self.activation_function_name = activation
        self.value = []
        self.net = []
        self.node = []
        self.generate()

    def generate(self):
        for i in range(self.n_neuron):
            self.node.append(Node(1, self.weight[i], self.activation_function, f'{self.name},{i+1}'))

    def activate_layer(self, input):
        if self.activation_function == softmax:
            for i in range(self.n_neuron):
                self.net.append(self.node[i].calculate_net(input))
            for i in range(self.n_neuron):
                self.node[i].update_net(self.net[i])
                self.value.append(self.node[i].activate_neuron(self.net))
        else:
            for i in range(self.n_neuron):
                self.net.append(self.node[i].calculate_net(input))
                self.value.append(self.node[i].activate_neuron())

    def restart_layer(self):
        self.value = []
        self.net = []
        for i in range(len(self.node)):
            self.node[i].restart_neuron()
```

Pada kelas Layer ini, dibuat beberapa fungsi antara lain:

1. **generate()** untuk membangkitkan neuron atau node anakan.
2. **activate_layer()** untuk melakukan aktivasi fungsi neuron dan node anakan. Dibedakan antara algoritma untuk fungsi aktivasi softmax dan lainnya karena fungsi softmax memerlukan data kolektif jumlah nett dari layer untuk didistribusi ke semua node sedangkan yang lainnya independen dapat menggunakan data pribadi di kelas Node.
3. **restart_layer()** untuk melakukan restorasi semua atribut pada layer agar kembali 0 atau kosong, lalu memanggil fungsi restart pada semua nodenya.

```
class Node:
    def __init__(self, bias, weight, activation_function, name):
        self.bias = bias
        self.weight = weight
        self.activation_function = activation_function
        self.name = name
        self.net = 0
        self.value = 0

    def calculate_net(self, input):
        net_array = np.concatenate(([1], input),)
        self.net = np.dot(self.weight,net_array)
        return self.net

    def activate_neuron(self, sum = None):
        if (self.activation_function != softmax):
            self.value = self.activation_function(self.net)
        else:
            self.value = self.activation_function(self.net, sum)
        return self.value

    def restart_neuron(self):
        self.net = 0
        self.value = 0

    def update_net(self, input):
        self.net = input
```

Lapisan paling akhir adalah kelas Node. Pada lapisan ini dilakukan manajemen terhadap kalkulasi utama seperti perhitungan jumlah nett dari *weight* dan array input dan juga aktivasi. Untuk mendukung itu dibuat juga fungi restart dan update. Fungsi-fungsi aktivasi dapat diakses melalui file **activation.py** yang berisi fungsi aktivasi dan turunannya yang akan dimanfaatkan pada tahap backward pass. Sebagai berikut.

```python
def linear(net, derivative=False):
    if derivative:
        return 1
    else:
        return net

def relu(net, derivative=False):
    if derivative:
        return 1 if net > 0 else 0
    else:
        return max(0, net)

def sigmoid(net, derivative=False):
    fsigmoid = (1 / (1 + np.exp(-net)))
    if derivative:
        return fsigmoid * (1 - fsigmoid)
    else:
        return fsigmoid

def softmax(net, numOfNet, derivative=False):
    sigma = 0
    for i in range (len(numOfNet)):
        sigma += np.exp(numOfNet[i])
    fsoftmax = np.exp(net) / sigma
    if derivative:
        return fsoftmax * (1 - fsoftmax)
    else:
        return fsoftmax
```

Selanjutnya modul-modul ini diimpor pada **main.ipynb** dan dilakukan uji-uji yang diperlukan. Untuk mempermudah pengujian dibuat beberapa fungsi tamabahan yakni:

1. **forward_propagation_ffnn()** untuk melakukan memanggil modul-modul yang berguna untuk rangkaian forward propagation dan mengembalikan model kosong, input, model yang sudah diaktivasi, dan perhitungan sse

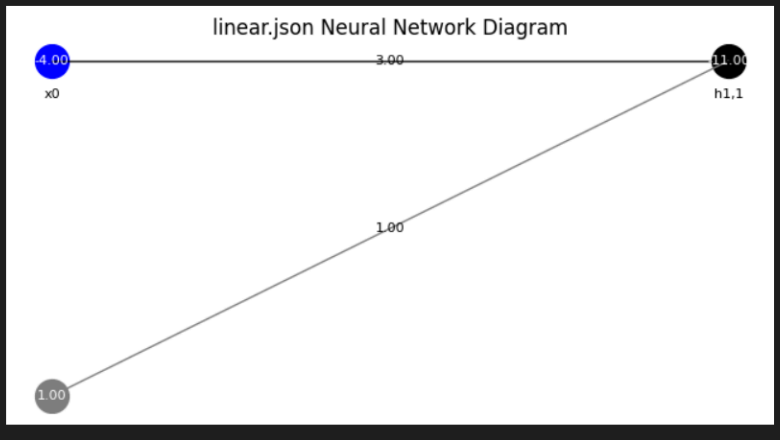
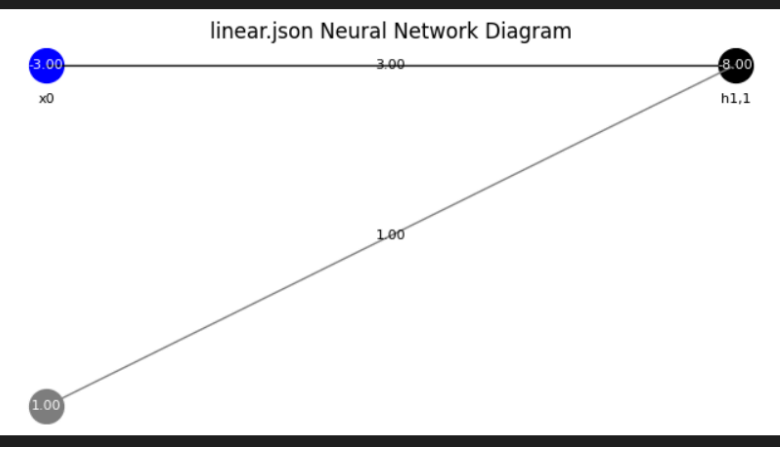
```python
def forward_propagation_ffnn(input_file_name):
    checked_file_name = file_name(input_file_name)
    case, expect = open_json(checked_file_name)
    model = FFNN(case)
    input = case.get("input")
    output = model.predict(input)
    output_val = FFNN.output_value(output)
    sse = FFNN.f_sse(output_val, expect.get("output"))
    return model, input, output, sse
```

✓ 0.0s Python

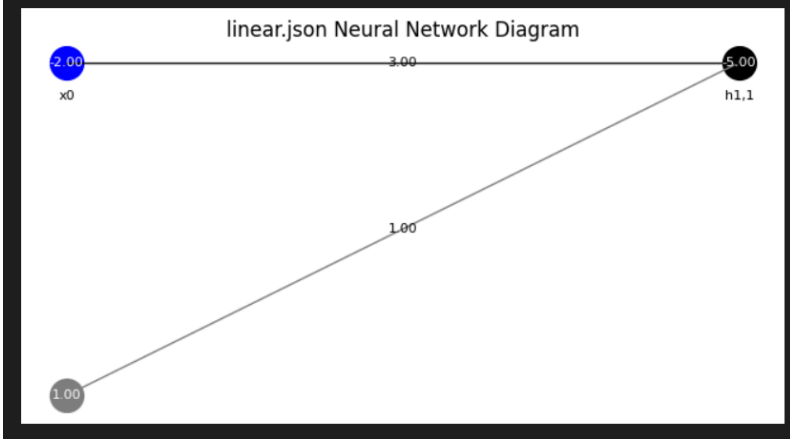
2. **visualize_model()** untuk melakukan visualisasi dengan bantuan matplotlib dalam menggambarkan hubungan neuron-neuron beserta value dan weight dari masing-masing neuron dan hubungannya.
3. **print_process()** untuk melakukan pencetakan nilai-nilai yang akan ditampilkan seperti nilai output dan sse dari tiap batch

HASIL PENGUJIAN

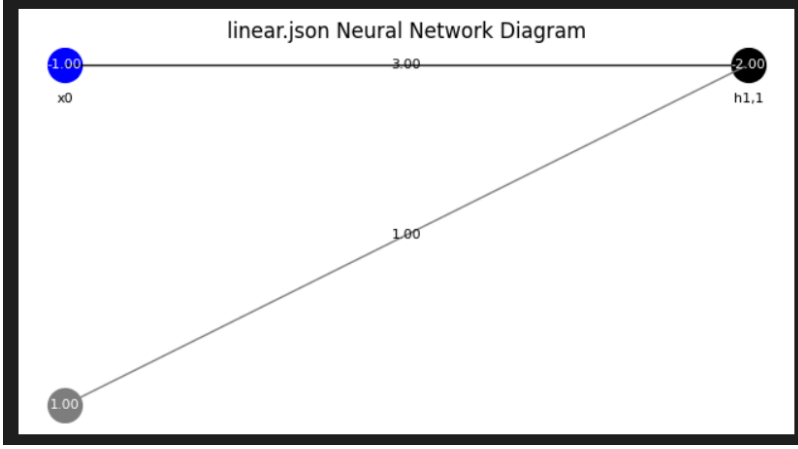
Berikut ini merupakan hasil pengujian dari setiap test case yang menampilkan nilai ekspektasi, SSE, serta hasil *Neural Network Diagram* dari masing-masing test case.

Test case 1: linear.json	Output 0: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]] SSE 0: 0.0  Output 1: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]] SSE 1: 0.0 

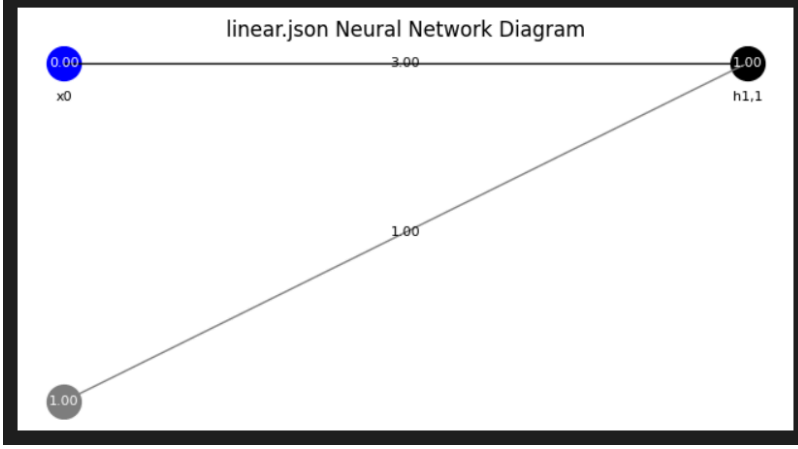
```
Output 2: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 2: 0.0
```



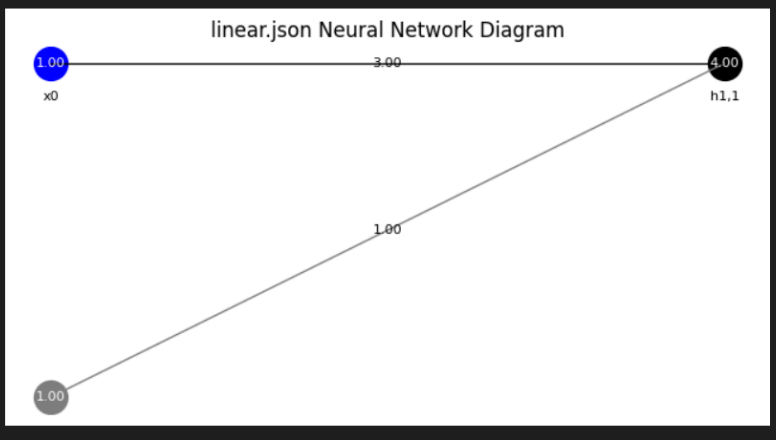

```
Output 3: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 3: 0.0
```



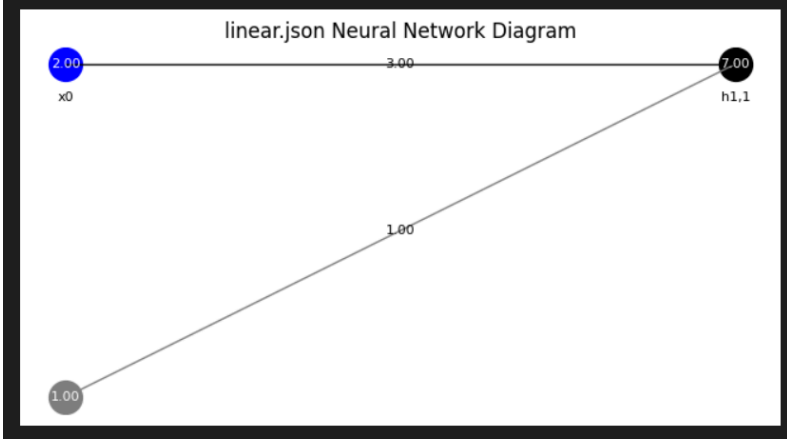

```
Output 4: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 4: 0.0
```



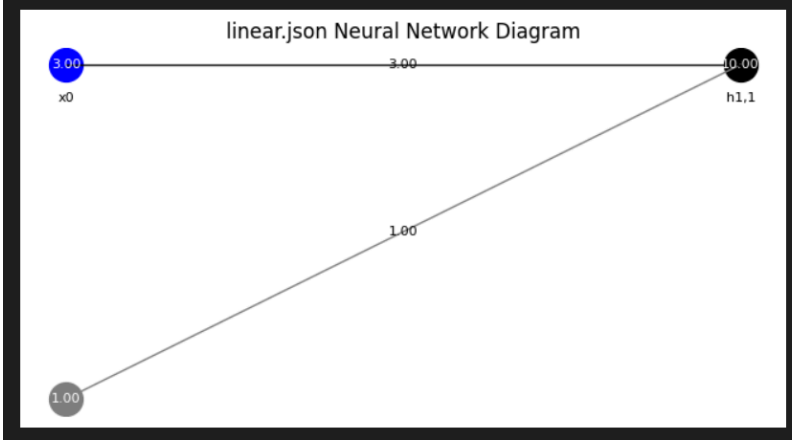
```
Output 5: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 5: 0.0
```

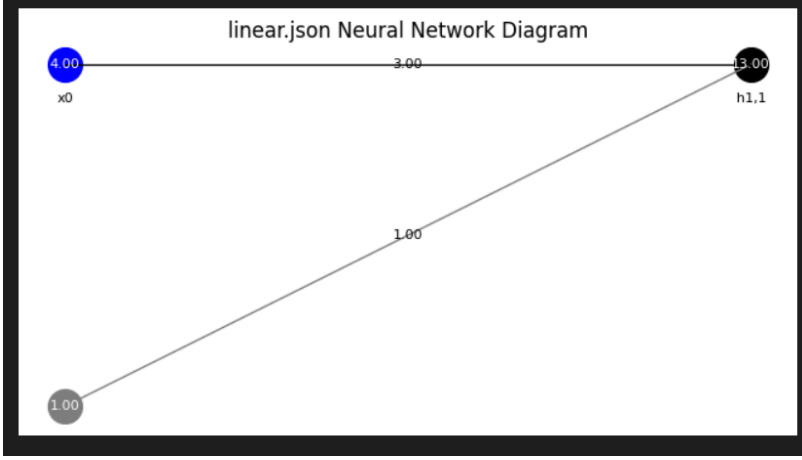
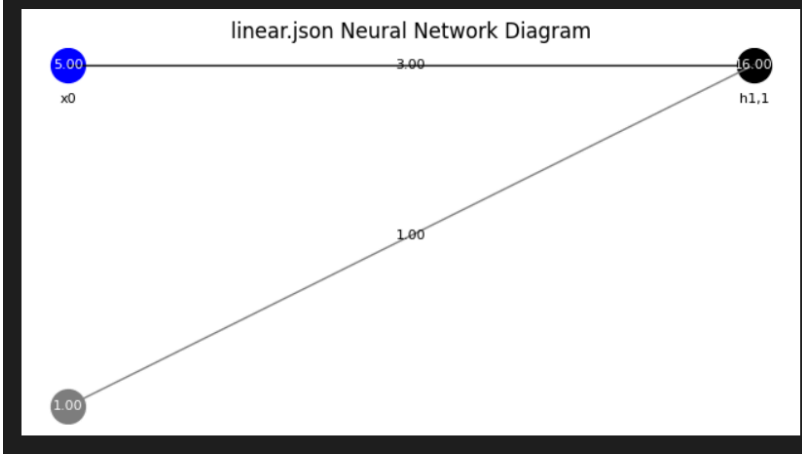


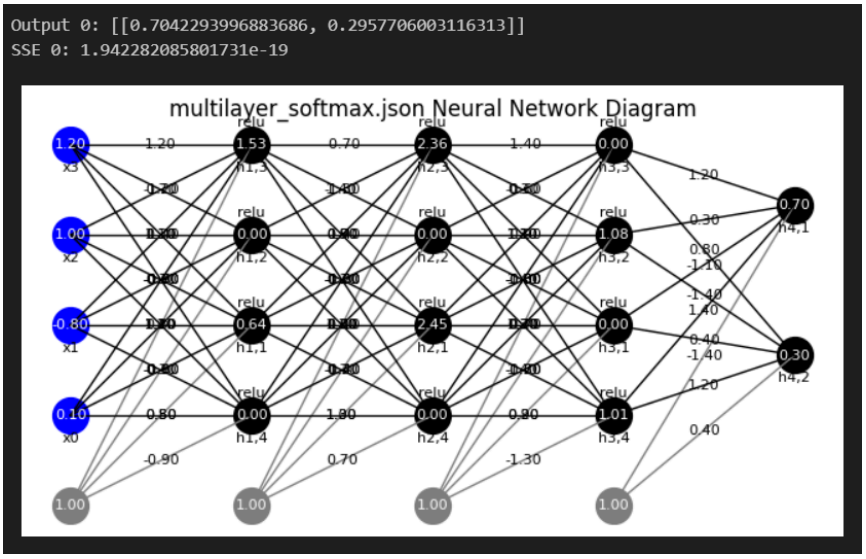
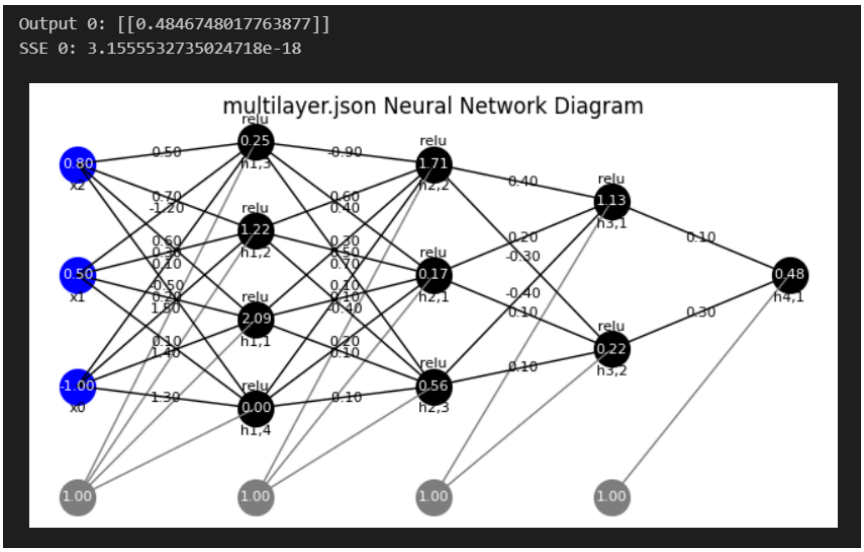

```
Output 6: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 6: 0.0
```

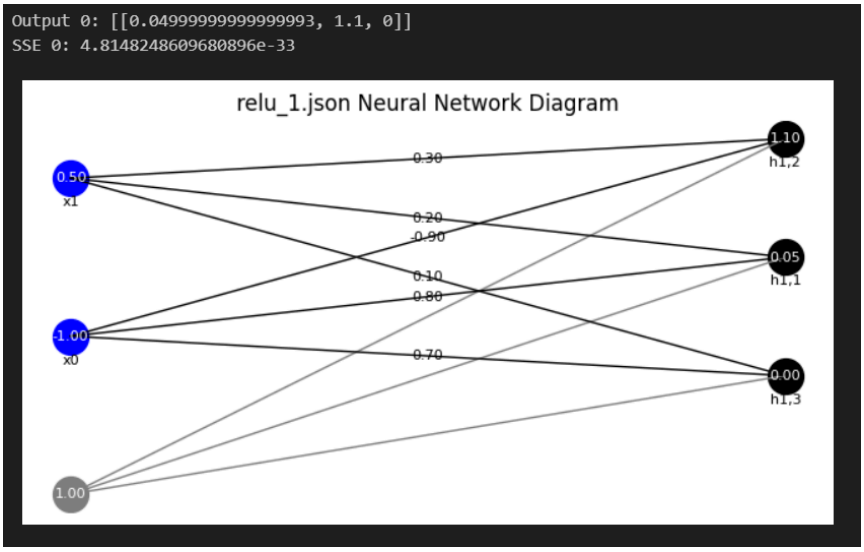
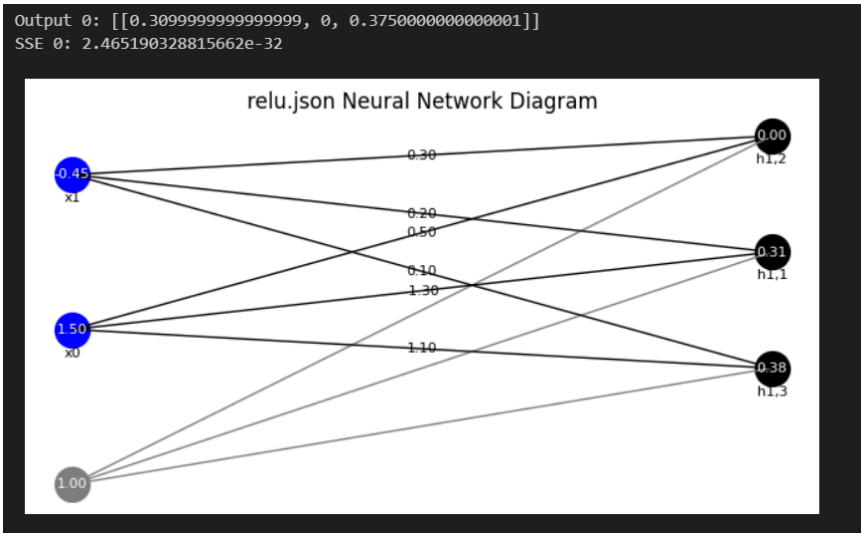


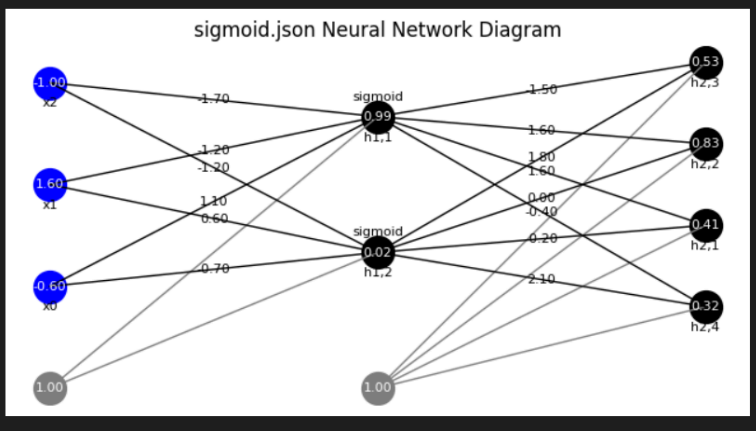
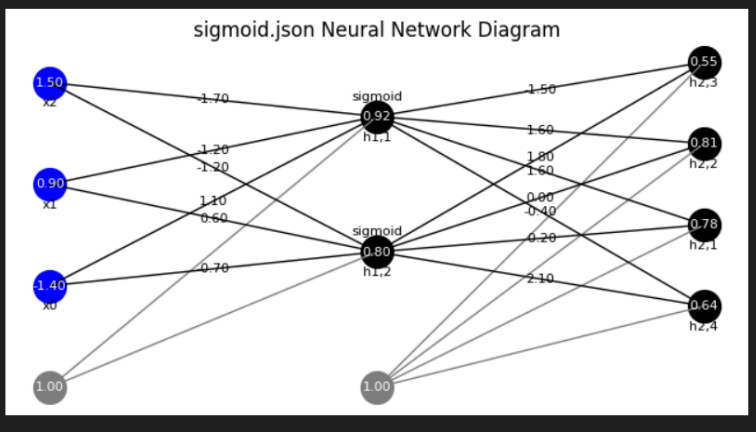

```
Output 7: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]]
SSE 7: 0.0
```



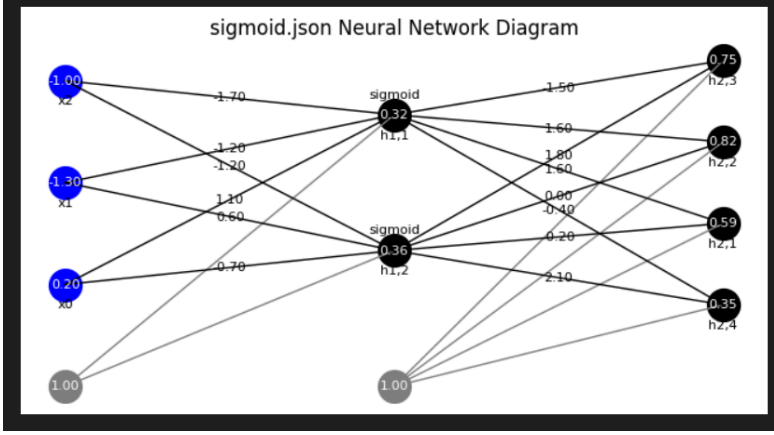
	Output 8: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]] SSE 8: 0.0  Output 9: [[-11.0], [-8.0], [-5.0], [-2.0], [1.0], [4.0], [7.0], [10.0], [13.0], [16.0]] SSE 9: 0.0  Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case linear.json keseluruhan memiliki SSE bernilai 0. Hal ini menunjukkan model mampu dengan sempurna menyesuaikan pola data dan tidak ada kesalahan prediksi sama sekali.

Test case 2: multilayer_softmax.json	 Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case multilayer_softmax.json memiliki nilai SSE yang sangat kecil atau mendekati 0 yaitu 1.942282085801731e-19. Nilai SSE yang mendekati nol menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.
Test case 3: multilayer.json	 Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case multilayer.json memiliki nilai SSE yang sangat kecil yaitu 3.1555532735024718e-18. Nilai SSE yang mendekati

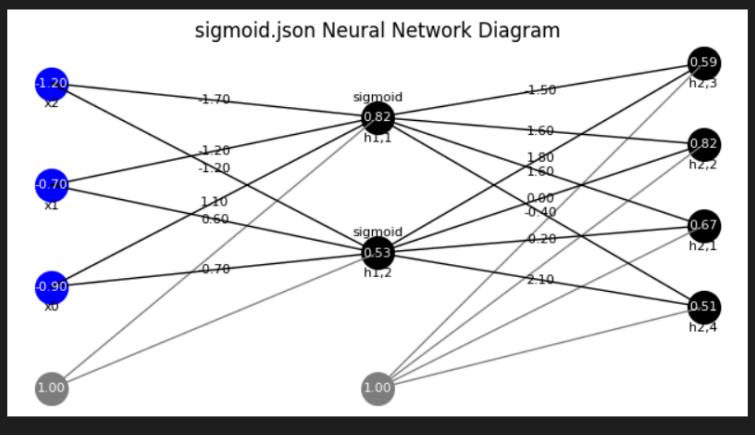
	nol menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.
Test case 4: relu_1.json	 Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case relu_1.json memiliki nilai SSE yang sangat kecil yaitu 4.8148248609680896e-33. Nilai SSE yang mendekati nol menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.
Test case 5: relu.json	 Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case relu.json memiliki nilai SSE yang sangat kecil

	yaitu 2.465190328815662e-32. Nilai SSE yang mendekati nol menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.
Test case 6: sigmoid.json	Output 0: [[0.41197345561004917, 0.8314293993694081, 0.5301853632739774, 0.31607396490909706], SSE 0: 5.448747611215099e-17  Output 1: [[0.41197345561004917, 0.8314293993694081, 0.5301853632739774, 0.31607396490909706], SSE 1: 1.8886184977662707e-17 

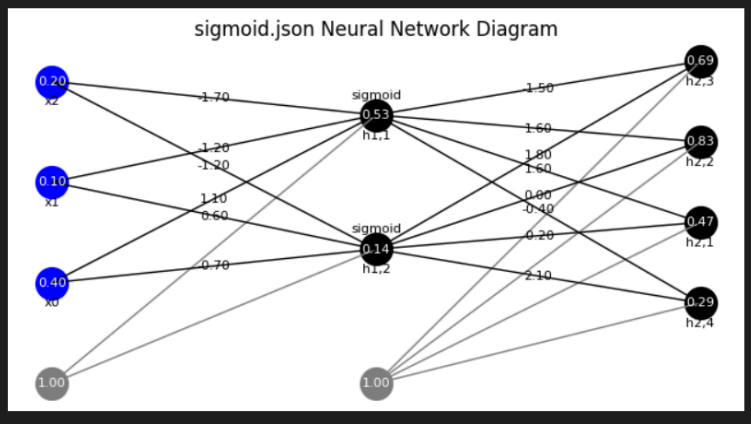
```
Output 2: [[0.41197345561004917, 0.8314293993694081, 0.5301853632739774, 0.31607396490909706],
SSE 2: 3.588137593505069e-17
```



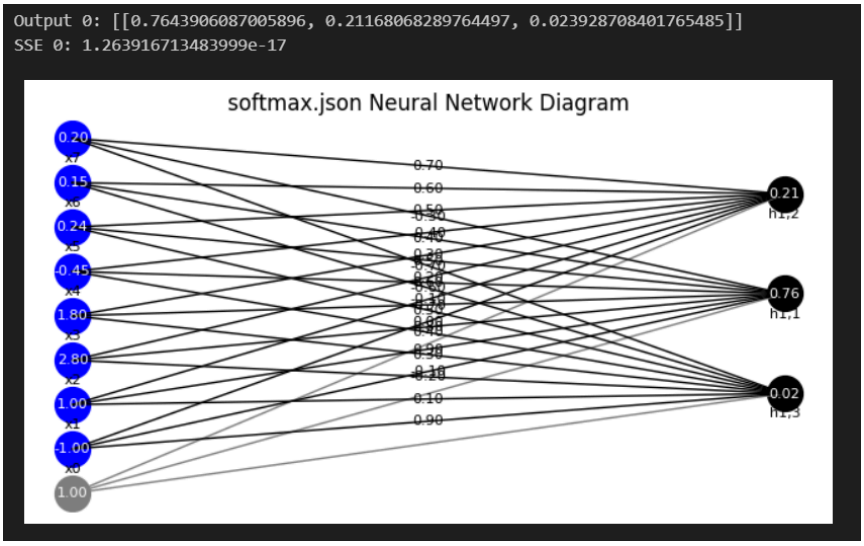

```
Output 3: [[0.41197345561004917, 0.8314293993694081, 0.5301853632739774, 0.31607396490909706],
SSE 3: 5.4935267400256686e-17
```




```
Output 4: [[0.41197345561004917, 0.8314293993694081, 0.5301853632739774, 0.31607396490909706],
SSE 4: 5.3370328317207175e-17
```



Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case sigmoid.json memiliki nilai SSE yang beragam.

	Untuk nilai ekspektasi 0, SSE bernilai 5.448747611215099e-17. Untuk nilai ekspektasi 1, SSE bernilai 1.8886184977662707e-17. Untuk nilai ekspektasi 2, SSE bernilai 3.588137593505069e-17. Untuk nilai ekspektasi 3, SSE bernilai 5.4935267400256686e-17. Untuk nilai ekspektasi 4, SSE bernilai 5.3370328317207175e-17. Dari keseluruhan nilai SSE yang didapatkan, semuanya bernilai sangat kecil atau mendekati 0. Hal ini menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.
Test case 7: softmax.json	 Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case softmax.json memiliki nilai SSE yang sangat kecil yaitu 1.263916713483999e-17. Nilai SSE yang mendekati nol menunjukkan bahwa perbedaan antara nilai prediksi dan nilai observasi sangat kecil atau bahkan tidak ada sama sekali.

PERBANDINGAN DENGAN PERHITUNGAN MANUAL

A. Perhitungan Manual

Perhitungan manual dilakukan sebagai pembanding yang dapat memperlihatkan akurasi dari program implementasi *forward propagation* untuk *Feed Forward Neural Network* (FFNN). Perhitungan manual dilakukan pada sebuah *excel/spreadsheet* dengan membentuk tabel penyelesaian kasus FFNN menggunakan *forward propagation*. Ke dalam *cell* yang terdapat pada kolom tiap tabel dimasukkan nilai-nilai input yang sesuai seperti nilai bias b = 1, nilai input x, nilai weight w, serta nilai target y sebagai *expected output*. Kemudian, untuk masing-masing fungsi aktivasi atau jenis pengujian yang di antaranya terdiri atas ReLU, *multilayer*, dan linear, akan dimasukkan rumus perhitungan yang sesuai untuk menghasilkan output atau keluaran dari persoalan FFNN tersebut. Pada kolom tabel yang terakhir, akan dihitung *error node output* yang dihitung berdasarkan nilai target y (*expected output),* nilai output o hasil dari perhitungan, serta rumus fungsi *error* masing-masing pengujian. Secara lengkap, perhitungan manual pada ketiga kasus uji dilakukan pada TubesA_13520130_Perhitungan Manual.xlsx.

B. Uji Kasus ReLU from Deliverable Docs

Berikut ini merupakan perbandingan hasil uji kasus ReLU dengan menggunakan program FFNN dan perhitungan manual.

<table>
<tr><td>Program FFNN</td><td>

```
Layer 1

Node  1,1
Weight:  [0.1 0.4 0.7]
Value:  0.04999999999999993

Node  1,2
Weight:  [ 0.2 -0.5  0.8]
Value:  1.1

Node  1,3
Weight:  [ 0.3  0.6 -0.9]
Value:  0

{'output': [[0.05, 1.1, 0.0]], 'max_sse': 1e-06}
```

</td></tr>
<tr><td>Perhitungan Manual</td><td>

Bias1	X1	X2	TARGET Y1	TARGET Y2	TARGET Y3	W01	W02	W03	W11	W12	W13	W21	W22	W23
1	-1	0,5	0,05	1,1	0	0,1	0,2	0,3	0,4	-0,5	0,6	0,7	0,8	-0,9

Sum O1	Sum O2	Sum O3	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)	(Target Y1 - Output1)	(Target Y2 - Output2)	(Target Y3 - Output3)	Error Node Output
0,05	1,1	-0,75	0,05	1,10	0,00	0,00	0,00	0,00	2,41E-33

</td></tr>
<tr><td>Keterangan</td><td>Hasil uji kasus ReLU from Deliverable Docs dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara default pada excel sehingga O1 bernilai bulat 0,05, namun telah dibuktikan sama dengan mengurangi nilai target Y1 dengan O1 dan mendapatkan hasil 0.</td></tr>
</table>

C. Uji Kasus ReLU

Berikut ini merupakan perbandingan hasil uji kasus linear dengan menggunakan program FFNN dan perhitungan manual.

<table>
<tr><td>Program FFNN</td><td>

```
Layer 1

Node  1,1
Weight:  [0.1  0.47 1.1 ]
Value:  0.3099999999999999

Node  1,2
Weight:  [ 0.2 -0.6 -1.3]
Value:  0

Node  1,3
Weight:  [0.3 0.2 0.5]
Value:  0.3750000000000001



{'output': [[0.31, 0, 0.375]], 'max_sse': 1e-06}
```

</td></tr>
<tr><td>Perhitungan Manual</td><td>

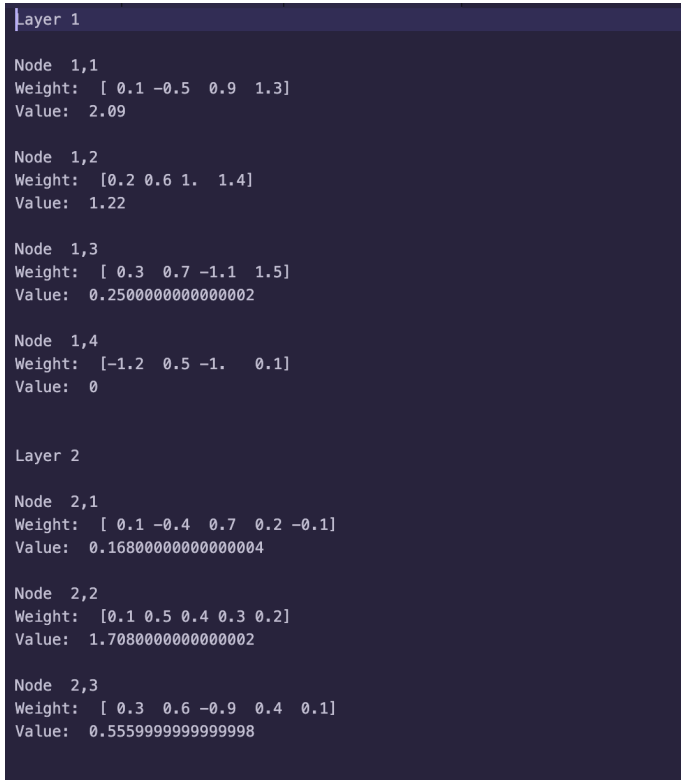
Bias1	X1	X2	TARGET Y1	TARGET Y2	TARGET Y3	W01	W02	W03	W11	W12	W13	W21	W22	W23
1	1.5	-0.45	0.31	0	0.375	0.1	0.2	0.3	0.47	-0.6	0.2	1.1	-1.3	0.5

Sum O1	Sum O2	Sum O3	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)	(Target Y1 - Output1)	(Target Y2 - Output2)	(Target Y3 - Output3)	Error Node Output
0.31	-0.115	0.375	0.31	0.00	0.38	0.00	0.00	0.00	0

</td></tr>
<tr><td>Keterangan</td><td>Hasil uji kasus ReLU dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara default pada excel sehingga O3 bernilai bulat 0,38, namun telah dibuktikan sama dengan mengurangi nilai target Y1 dengan O1 dan mendapatkan hasil 0.</td></tr>
</table>

D. Uji Kasus Multilayer

Berikut ini merupakan perbandingan hasil uji kasus multilayer dengan menggunakan program FFNN dan perhitungan manual.

Program FFNN	

<table>
<tr>
<td></td>
<td>

```
Layer 3

Node  3,1
Weight:  [ 0.1 -0.3  0.6  0.1]
Value:  1.1300000000000003

Node  3,2
Weight:  [ 0.2  0.4  0.1 -0.4]
Value:  0.21560000000000012


Layer 4

Node  4,1
Weight:  [ 0.1 -0.2  0.3]
Value:  0.4846748017763877




{'output': [[0.4846748]], 'max_sse': 1e-06}
```

</td>
</tr>
<tr>
<td>Perhitungan Manual</td>
<td>

Bias1	X1	X2	X3	TARGET Y1
1	-1	0,5	0,8	0,4846748

Layer 1

W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	W31	W32	W33	W34	Sum H11	Sum H12	Sum H13	Sum H14	Bias 1	Output H1 (Activation)	Output H2 (Activation)	Output H3 (Activation)	Output H4 (Activation)
0,1	0,2	0,3	-1,2	-0,5	0,6	0,7	0,5	0,9	1	-1,1	-1	1,3	1,4	1,5	0,1	2,09	1,22	0,25	-2,12	1	2,09	1,22	0,25	0,00

Layer 2

W01	W02	W03	W11	W12	W13	W21	W22	W23	W31	W32	W33	W41	W42	W43	Sum H21	Sum H22	Sum H23	Bias 2	Output H1 (Activation)	Output H2 (Activation)	Output H3 (Activation)
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	1	0,17	1,71	0,56

Layer 3

W01	W02	W11	W12	W21	W22	W31	W32	Sum H31	Sum H32	Bias 3	Output H1 (Activation)	Output H2 (Activation)
0,1	0,2	-0,3	0,4	0,6	0,1	0,1	-0,4	1,13	0,2156	1	1,13	0,22

Layer Output

W01	W11	W21	Sum O1	Output 1 (Activation)	(Target Y1 - Output1)	Error Node Output
0,1	-0,2	0,3	-0,06132	0,48	0,00	1,5778E-18

</td>
</tr>
<tr>
<td>Keterangan</td>
<td>Hasil uji kasus multilayer dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara default pada excel sehingga O1 bernilai bulat 0,48, namun telah dibuktikan sama dengan</td>
</tr>
</table>

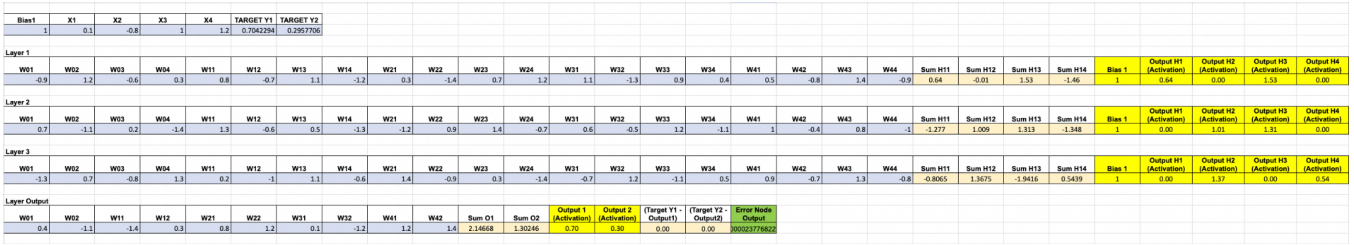
	mengurangi nilai target Y1 dengan O1 dan mendapatkan hasil 0.

E. Uji Kasus Multilayer Softmax

Berikut ini merupakan perbandingan hasil uji kasus linear dengan menggunakan program FFNN dan perhitungan manual

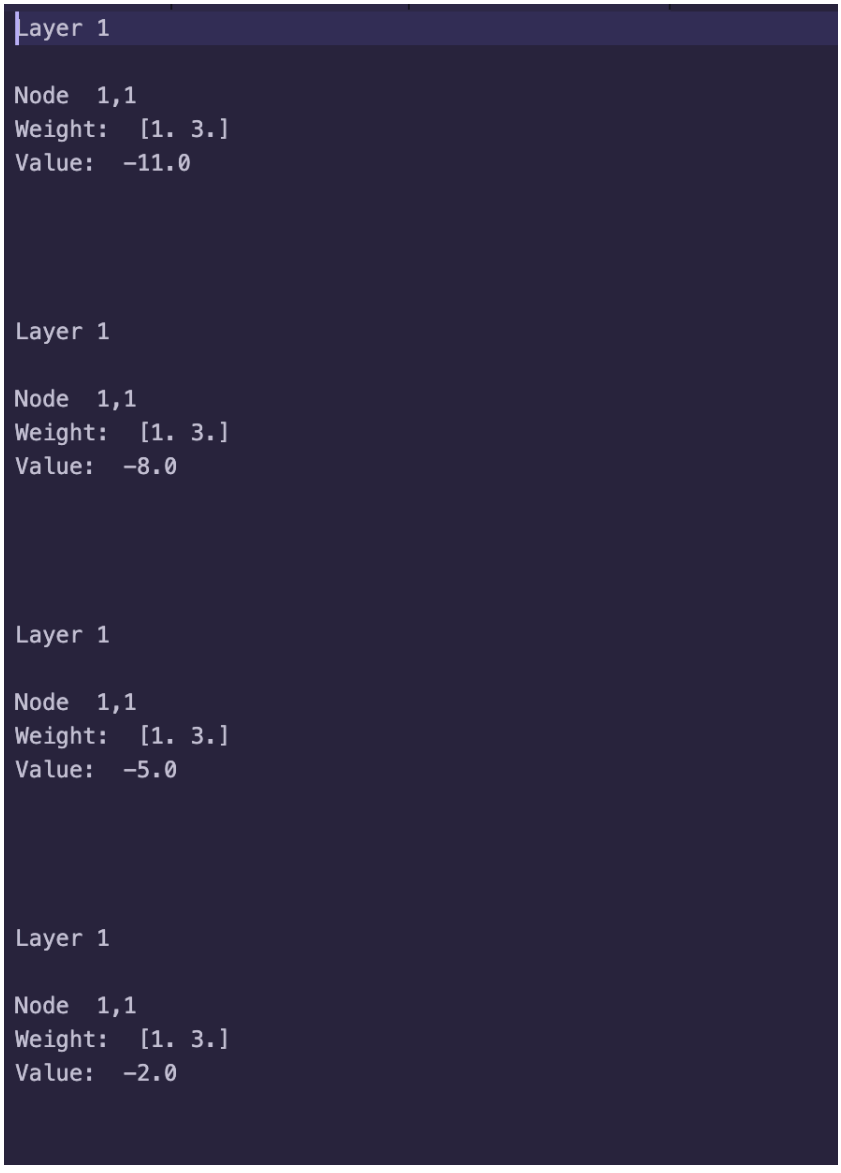
Program FFNN	Layer 1 Node 1,1 Weight: [-0.9 0.8 0.3 1.1 0.5] Value: 0.64 Node 1,2 Weight: [1.2 -0.7 -1.4 -1.3 -0.8] Value: 0 Node 1,3 Weight: [-0.6 1.1 0.7 0.9 1.4] Value: 1.5300000000000002 Node 1,4 Weight: [0.3 -1.2 1.2 0.4 -0.9] Value: 0 Layer 2 Node 2,1 Weight: [0.7 1.3 -1.2 0.6 1.] Value: 2.45 Node 2,2 Weight: [-1.1 -0.6 0.9 -0.5 -0.4] Value: 0 Node 2,3 Weight: [0.2 0.5 1.4 1.2 0.8] Value: 2.3560000000000003 Node 2,4 Weight: [-1.4 -1.3 -0.7 -1.1 -1.] Value: 0

```
Layer 3

Node  3,1
Weight:  [-1.3  0.2  1.4 -0.7  0.9]
Value:  0

Node  3,2
Weight:  [ 0.7 -1.  -0.9  1.2 -0.7]
Value:  1.0772000000000002

Node  3,3
Weight:  [-0.8  1.1  0.3 -1.1  1.3]
Value:  0

Node  3,4
Weight:  [ 1.3 -0.6 -1.4  0.5 -0.8]
Value:  1.0080000000000002


Layer 4

Node  4,1
Weight:  [ 0.4 -1.4  0.8  0.1  1.2]
Value:  0.7042293996883686

Node  4,2
Weight:  [-1.1  0.3  1.2 -1.2  1.4]
Value:  0.2957706003116313




{'output': [[0.7042294, 0.2957706]], 'max_sse': 1e-06}
```

Perhitungan Manual	 Bias1 | X1 | X2 | X3 | X4 | TARGET Y1 | TARGET Y2 1 | 0.1 | -0.8 | 1 | 1.2 | 0.7042294 | 0.2957706 Layer 1 W01 | W02 | W03 | W04 | W11 | W12 | W13 | W14 | W21 | W22 | W23 | W24 | W31 | W32 | W33 | W34 | W41 | W42 | W43 | W44 | Sum H11 | Sum H12 | Sum H13 | Sum H14 | Bias 1 | Output H1 (Activation) | Output H2 (Activation) | Output H3 (Activation) | Output H4 (Activation) -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 | 1 | 0.64 | 0.00 | 1.53 | 0.00 Layer 2 W01 | W02 | W03 | W04 | W11 | W12 | W13 | W14 | W21 | W22 | W23 | W24 | W31 | W32 | W33 | W34 | W41 | W42 | W43 | W44 | Sum H11 | Sum H12 | Sum H13 | Sum H14 | Bias 1 | Output H1 (Activation) | Output H2 (Activation) | Output H3 (Activation) | Output H4 (Activation) 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 | -1.277 | 1.009 | 1.313 | -1.348 | 1 | 0.00 | 1.01 | 1.31 | 0.00 Layer 3 W01 | W02 | W03 | W04 | W11 | W12 | W13 | W14 | W21 | W22 | W23 | W24 | W31 | W32 | W33 | W34 | W41 | W42 | W43 | W44 | Sum H11 | Sum H12 | Sum H13 | Sum H14 | Bias 1 | Output H1 (Activation) | Output H2 (Activation) | Output H3 (Activation) | Output H4 (Activation) -1.3 | 0.7 | -0.8 | 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 | -0.8065 | 1.3675 | -1.9416 | 0.5439 | 1 | 0.00 | 1.37 | 0.00 | 0.54 Layer Output W01 | W02 | W11 | W12 | W21 | W22 | W31 | W32 | W41 | W42 | Sum O1 | Sum O2 | Output 1 (Activation) | Output 2 (Activation) | (Target Y1 - Output1) | (Target Y2 - Output2) | Error Node Output 0.4 | -1.1 | -1.4 | 0.3 | 0.8 | 1.2 | 0.1 | -1.2 | 1.2 | 1.4 | 2.14668 | 1.30246 | 0.70 | 0.30 | 0.00 | 0.00 | 0.000023776622
Keterangan	Hasil uji kasus *multilayer softmax* dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara *default* pada *excel* sehingga O1 bernilai bulat 0,7 dan O2 bernilai bulat 0,3, namun telah dibuktikan sama dengan mengurangi nilai target Y1 dan Y2 dengan O1 dan O2 berturut-turut, dan mendapatkan hasil 0.

F. Uji Kasus Linear

Berikut ini merupakan perbandingan hasil uji kasus linear dengan menggunakan program FFNN dan perhitungan manual.

Program FFNN	

```
Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  1.0



Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  4.0



Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  7.0



Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  10.0
```

```
Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  13.0



Layer 1

Node  1,1
Weight:  [1. 3.]
Value:  16.0



{'output': [[-11], [-8], [-5], [-2], [1], [4], [7], [10], [13], [16]], 'max_sse': 1e-12}
```

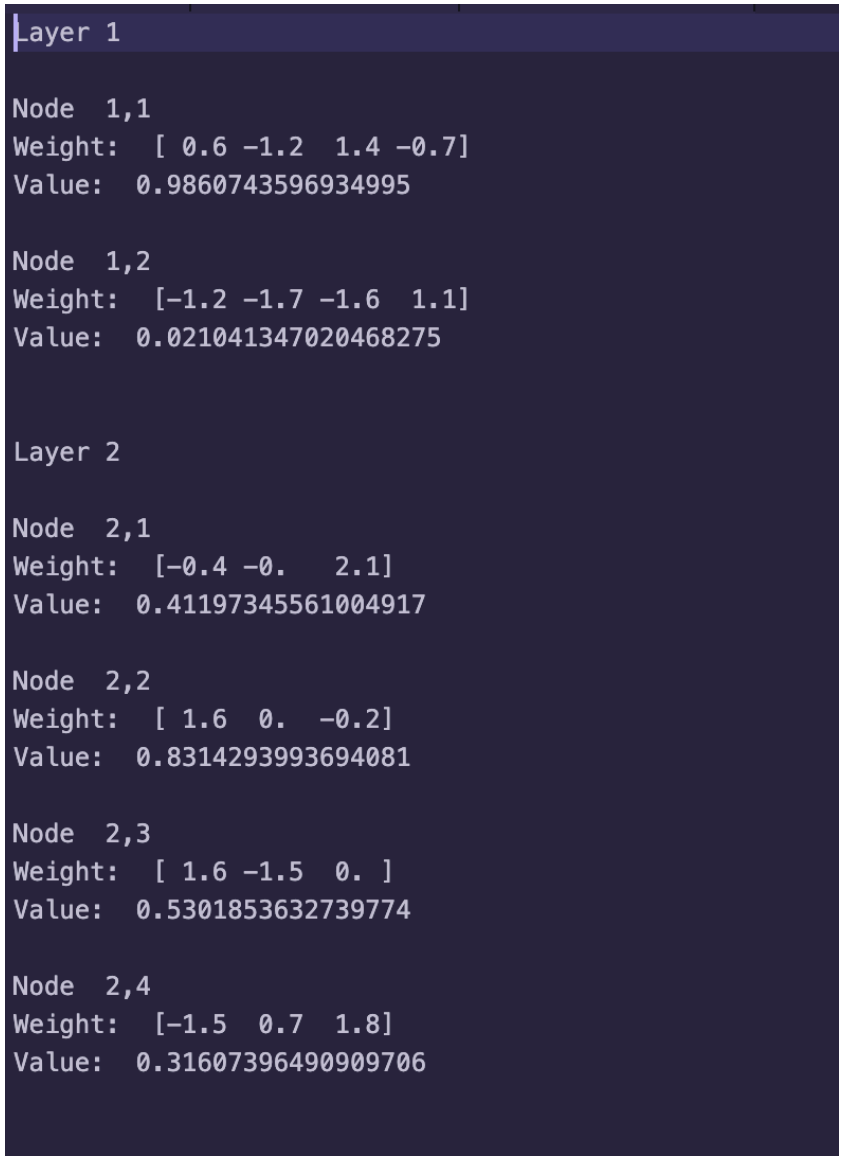
Perhitungan Manual	

Bias1	X1	TARGET Y1	W01	W11	Sum O1	Output O1 (Activation)	(Target Y1 - Output1)	Error Node Output
1	-4	-11	1	3	-11	-11,00	0,00	0
1	-3	-8	1	3	-8	-8,00	0,00	0
1	-2	-5	1	3	-5	-5,00	0,00	0
1	-1	-2	1	3	-2	-2,00	0,00	0
1	0	1	1	3	1	1,00	0,00	0
1	1	4	1	3	4	4,00	0,00	0
1	2	7	1	3	7	7,00	0,00	0
1	3	10	1	3	10	10,00	0,00	0
1	4	13	1	3	13	13,00	0,00	0
1	5	16	1	3	16	16,00	0,00	0

Keterangan	Hasil uji kasus linear dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang sama.

G. Uji Kasus Sigmoid

Berikut ini merupakan perbandingan hasil uji kasus linear dengan menggunakan program FFNN dan perhitungan manual.

Program FFNN	

	Input 1 Layer 1 Node 1,1 Weight: [0.6 -1.2 1.4 -0.7] Value: 0.9234378026481879 Node 1,2 Weight: [-1.2 -1.7 -1.6 1.1] Value: 0.8005922431513315 Layer 2 Node 2,1 Weight: [-0.4 -0. 2.1] Value: 0.7826614091150284 Node 2,2 Weight: [1.6 0. -0.2] Value: 0.8084363083194981 Node 2,3 Weight: [1.6 -1.5 0.] Value: 0.5535051760955713 Node 2,4 Weight: [-1.5 0.7 1.8] Value: 0.6427850098146376 Input 2

```
Layer 1

Node  1,1
Weight:  [ 0.6 -1.2  1.4 -0.7]
Value:  0.31864626621097447

Node  1,2
Weight:  [-1.2 -1.7 -1.6  1.1]
Value:  0.36354745971843366


Layer 2

Node  2,1
Weight:  [-0.4 -0.   2.1]
Value:  0.5898752435296561

Node  2,2
Weight:  [ 1.6  0.  -0.2]
Value:  0.8216095372737501

Node  2,3
Weight:  [ 1.6 -1.5  0. ]
Value:  0.7543651777362295

Node  2,4
Weight:  [-1.5  0.7  1.8]
Value:  0.34919894670366747
```

Input 3

```
Layer 1

Node  1,1
Weight:  [ 0.6 -1.2  1.4 -0.7]
Value:  0.8234647252208833

Node  1,2
Weight:  [-1.2 -1.7 -1.6  1.1]
Value:  0.5324543063873187


Layer 2

Node  2,1
Weight:  [-0.4 -0.   2.1]
Value:  0.6722003953978934

Node  2,2
Weight:  [ 1.6  0.  -0.2]
Value:  0.8166043910016146

Node  2,3
Weight:  [ 1.6 -1.5  0. ]
Value:  0.5902025844263002

Node  2,4
Weight:  [-1.5  0.7  1.8]
Value:  0.5087098836277426
```

	Input 4 Layer 1 Node 1,1 Weight: [0.6 -1.2 1.4 -0.7] Value: 0.5299640517645717 Node 1,2 Weight: [-1.2 -1.7 -1.6 1.1] Value: 0.13943387296165005 Layer 2 Node 2,1 Weight: [-0.4 -0. 2.1] Value: 0.47322841097431845 Node 2,2 Weight: [1.6 0. -0.2] Value: 0.8280846566130694 Node 2,3 Weight: [1.6 -1.5 0.] Value: 0.6910545248579614 Node 2,4 Weight: [-1.5 0.7 1.8] Value: 0.2935832341653264 Input 5

Perhitungan Manual	

Bias0	X1	X2	X3	TARGET Y1	TARGET Y2	TARGET Y3	TARGET Y4	W01	W02	W11	W12	W21	W22	W31	W32	Sum H11	Sum H12	Bias 1	Output H1 (Activation)	Output H2 (Activation)
1	-0.6	1.6	-1	0.4119734	0.8314294	0.5301853	0.3160739	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	4.26	-3.84	1	0.99	0.02
1	-1.4	0.9	1.5	0.7826614	0.8084363	0.5535051	0.6427850	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	2.49	1.39	1	0.92	0.80
1	0.2	-1.3	-1	0.5898752	0.8216095	0.7543651	0.3491989	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	-0.76	-0.56	1	0.32	0.36
1	-0.9	-0.7	-1.2	0.6722004	0.8166043	0.5902002	0.5087098	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	1.54	0.13	1	0.82	0.53
1	0.4	0.1	0.2	0.4732284	0.8280846	0.6910545	0.2935832	0.6	-1.2	-1.2	-1.7	1.4	-1.6	-0.7	1.1	0.12	-1.82	1	0.53	0.14

W01	W02	W03	W04	W11	W12	W13	W14	W21	W22	W23	W24	Sum H21	Sum H22	Sum H23	Sum H24	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)	Output O4 (Activation)
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	3558131	7595791	7	2088846	0.41	0.83	0.53	0.32
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	2812437	4398815	5	2148432	0.78	0.81	0.55	0.64
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	3634496	6527290	5	1220306	0.59	0.82	0.75	0.35
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	7181540	4493509	1	3648029	0.67	0.82	0.59	0.51
-0.4	1.6	1.6	-1.5	0	0	-1.5	0.7	2.1	-0.2	0	1.8	1071888	6572113	2	3050539	0.47	0.83	0.69	0.29

(Target Y1 - Output1)	(Target Y2 - Output2)	(Target Y3 - Output3)	(Target Y4 - Output4)	Error Node Output
0.00	0.00	0.00	0.00	0
0.00	0.00	0.00	0.00	0
0.00	0.00	0.00	0.00	0
0.00	0.00	0.00	0.00	0
0.00	0.00	0.00	0.00	0

Keterangan	Hasil uji kasus sigmoid dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara *default* pada *excel* sehingga O1, O2, O3, dan O4 bernilai bulat dengan dua digit di belakang koma, namun telah dibuktikan sama dengan mengurangi nilai target Y1, Y2, Y3, dan Y4 dengan O1, O2, O3, dan O4 secara berturut-turut, dan mendapatkan hasil 0.

H. Uji Kasus Softmax

Berikut ini merupakan perbandingan hasil uji kasus linear dengan menggunakan program FFNN dan perhitungan manual.

<table>
<tr><td>Program FFNN</td><td>

```
Layer 1

Node  1,1
Weight:  [ 0.1 -0.2  0.3  0.4  0.5 -0.6 -0.7  0.8  0.9]
Value:  0.7643906087005896

Node  1,2
Weight:  [ 0.9  0.8 -0.7  0.6  0.5  0.4 -0.3  0.2 -0.1]
Value:  0.21168068289764497

Node  1,3
Weight:  [-0.1  0.2  0.3 -0.4  0.5  0.6  0.7 -0.8  0. ]
Value:  0.023928708401765485



{'output': [[0.76439061, 0.21168068, 0.02392871]], 'max_sse': 1e-05}
```

</td></tr>
<tr><td>Perhitungan Manual</td><td>

Bias1	X1	X2	X3	X4	X5	X6	X7	X8	TARGET Y1	TARGET Y2	TARGET Y3	W01	W02	W03	W11	W12	W13	W21	W22	W23
1	-1	1	2.8	1.8	-0.45	0.24	0.15	0.2	0.764390	0.211680	0.023928	0.1	0.9	-0.1	-0.2	0.8	0.2	0.3	-0.7	0.3

W31	W32	W33	W41	W42	W43	W51	W52	W53	W61	W62	W63	W71	W72	W73	W81	W82	W83	Sum O1	Sum O2	Sum O3
0.4	0.6	-0.4	0.5	0.5	0.5	-0.6	0.4	0.6	-0.7	-0.3	0.7	0.8	0.2	-0.8	0.9	-0.1	0	3.022	1.738	-0.442

Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)	(Target Y1 - Output1)	(Target Y2 - Output2)	(Target Y3 - Output3)	Error Node Output
0.76	0.21	0.02	0.00	0.00	0.00	0

</td></tr>
<tr><td>Keterangan</td><td>Hasil uji kasus softmax dengan menggunakan program FFNN dengan perhitungan manual memberikan hasil yang relatif sama. Pada perhitungan manual dilakukan pembulatan yang telah diatur secara default pada excel sehingga O1 bernilai bulat 0,76, O2 bernilai bulat 0,21 dan O3 bernilai bulat 0,02, namun telah</td></tr>
</table>

	dibuktikan sama dengan mengurangi nilai target Y1, Y2, dan Y3 dengan O1, O2, dan O3 berturut-turut, dan mendapatkan hasil 0.

I. Analisis Perbandingan

Dari berbagai hasil kasus uji di atas dapat dilakukan analisis perbandingan antara pengujian kasus menggunakan program FFNN dengan perhitungan manual. Di samping itu terdapat pula beberapa aspek lain yang dapat dipertimbangkan selain melihat perbandingan hasil di antara keduanya. Analisis perbandingan tersebut dapat dilihat sebagai berikut.

1. Ketepatan Hasil (Accuracy)

   Program buatan mengimplementasikan *forward propagation* menggunakan perhitungan otomatis yang terprogram pada komputer sehingga dapat menghasilkan hasil dengan cepat dan akurat. Perhitungan manual memerlukan perhatian ekstra terhadap detail dan mungkin rentan terhadap kesalahan manusia. Misalnya, pada penggunaan *excel,* mungkin saja terjadi kesalahan dalam memasukkan rumus perhitungan, atau pada perhitungan mandiri tanpa *excel*, kesalahan menghitung sangat rentan terjadi. Akan tetapi, jika perhitungan manual dilakukan dengan benar dan teliti, seharusnya tetap dapat memberikan hasil yang akurat sesuai dengan hasil program.

2. Waktu Eksekusi

   Meskipun tidak dilakukan penghitungan waktu eksekusi pada penyelesaian persoalan FFNN pada kedua cara, dapat diasumsikan bahwa program buatan cenderung jauh lebih cepat dalam melakukan *forward propagation*, terutama untuk jaringan yang besar. Sedangkan perhitungan manual mungkin memerlukan waktu yang lebih lama, terutama jika jaringannya kompleks atau memiliki banyak neuron karena perlu merumuskan tabel penyelesaian yang lebih banyak jika menggunakan *excel* atau perlu melibatkan perhitungan yang kompleks dan panjang jika menghitung mandiri tanpa bantuan *excel*.

3. Kemudahan Penggunaan

   Program buatan biasanya lebih mudah digunakan karena memungkinkan untuk otomatisasi dan pengulangan, serta kemampuan untuk mengotomatisasi proses seperti tuning hiperparameter dan analisis hasil. Perhitungan manual membutuhkan pemahaman yang kuat tentang matematika di balik *forward propagation*, termasuk mengetahui rumus dan langkah penyelesaiannya, serta memerlukan waktu dan perhatian ekstra.

4. Kemampuan Pemecahan Masalah

   Program buatan dapat menghasilkan hasil yang cepat dan efisien, tetapi tidak selalu memberikan wawasan yang mendalam tentang proses yang sebenarnya terjadi di dalamnya. Di samping itu, perhitungan manual dapat membantu memahami inti dari algoritma *forward propagation* dengan lebih baik, karena memerlukan langkah-langkah yang lebih jelas dan detail dalam pengerjaannya.

5. Skalabilitas

   Program buatan lebih mudah untuk disesuaikan dengan jaringan yang lebih besar atau lebih kompleks, sedangkan perhitungan manual mungkin menjadi tidak praktis atau sangat rumit ketika jaringan tumbuh dalam skala yang lebih besar.

Dalam praktiknya, biasanya digunakan kombinasi dari kedua metode tersebut. Program buatan umumnya digunakan untuk pelatihan dan pengujian model secara efisien, sementara perhitungan manual digunakan untuk memeriksa hasil, memvalidasi pemahaman, atau melakukan analisis teoritis seperti yang telah dilakukan pada tugas besar ini.

REPOSITORI & PEMBAGIAN TUGAS

https://github.com/ferindya/TubesA_13520130.git

NIM	Nama Anggota	Peran
13520130	Nelsen Putra	• Melakukan uji kasus • Membuat analisis perbandingan hasil uji kasus program FFNN dengan perhitungan manual
13521117	Maggie Zeta RS	• Melakukan uji kasus • Membuat analisis model untuk FFNN
13521133	Cetta Reswara Parahita	• Melakukan implementasi pembuatan kelas FFNN dan Hidden Layer • Optimasi kelas Layer dan Node, • Debugging activation function, dan • Membuat visualisasi pada main.ipynb
13521161	Ferindya Aulia Berlianty	• Melakukan implementasi pembuatan kelas aktivasi • Melakukan implementasi pembuatan kelas Layer dan Node

DAFTAR REFERENSI

https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-feed-forward-network-in-deep-learning/
https://cdn-edunex.itb.ac.id/38024-Machine-Learning-Parallel-Class/69139-Minggu-06/36287-Feed-Forward-Neural-Network/1645167603203_IF3270_Materi05_Seg03_ANN-FFNN.pdf