Implementasi Forward Propagation Untuk Feed Forward Neural Network

Tugas Kecil I IF3270 Machine Learning

Disusun oleh

Rexy Gamaliel Rumahorbo	13519010
Louis Riemenn	13519016
Benidictus Galih Mahar putra	13519159
Kevin Katsura Dani Sitanggang	13519216

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

DAFTAR ISI

I. PENJELASAN IMPLEMENTASI

Berikut ini penjelasan implementasi dari program berdasarkan kelas-kelas dan fungsi yang dibangun.

A. File Konfigurasi/Model (JSON)

File konfigurasi untuk model Neural Network yang kami buat merupakan file dalam format JSON. Dalam file tersebut, terdapat informasi mengenai struktur dari model tersebut seperti kedalaman (banyak layer), banyak input, learning rate, dan weight/bobot dari setiap layer (kecuali layer paling bawah, yakni layer input). Setiap layer berisi data berupa lokasi layernya, nama layernya, banyak neuron pada layer tersebut, dan metode aktivasi yang digunakan pada layer tersebut. Data bobot berisikan bobot dari layer bawah terhadap layer tersebut. Berikut ini struktur dari file model yang diimplementasikan.

```
{
    "Network_Depth": data_integer,
    "Learning_Rate": data_float,
    "Num_Of_Input": data_integer,
    "Layers": [
        {
            "depth": data_integer,
            "name": data_string,
            "width": data_integer,
            "activation": data_string
        },...,
        {
            "depth": data_integer,
            "name": data_string,
            "width": data_integer,
            "activation": data_string
        }
    ],
    "Weights": [
        {
            "depth_origin": data_integer,
            "values": [
                [V1, V2, V3],...,
                [V1, V2, V3]
            ]
        },
        {
            "depth_origin": [V1, V2, V3],
            "values": [[V1, V2, V3]]
        }
    ],
    "Input": [
```

```
        [X1, Y1],..,
        [X1, Y1]
    ]
}
```

B. Implementasi Kelas Layer

Kelas *layer* merepresentasikan suatu *layer* pada *neural network.* Setiap *layer* memiliki beberapa *fields* yang menjadi sumber dari informasi layer tersebut. Berikut ini *fields* yang ada di kelas ini.

Tabel 1.2.1. *Fields* pada Kelas *Layer*

Field	**Penjelasan**
width	*width* berfungsi untuk menyimpan informasi dari lebar pada *layer* terkait. Lebar yang dimaksud ini adalah jumlah *node* pada *layer* tersebut.
activation	*activation* berfungsi untuk menyimpan referens ke fungsi aktivasi dari *layer* terkait. Dengan memanggil *field activation* ini, maka fungsi aktivasi dari *layer* akan mulai melakukan perhitungan. Dari informasi ini, dapat dilihat bahwa satu *layer* walaupun *node* berbeda hanya memiliki satu fungsi aktivasi.
nodes	*nodes* berfungsi sebagai wadah untuk menyimpan nilai awal/hasil perhitungan dari *layer* sebelumnya. *Nodes* ini merepresentasikan nilai masukan (X) yang akan diolah dan dioperasikan dengan matriks bobot.

Selain menyimpan *fields,* kelas *layer* memiliki beberapa fungsi yang menjadi perilaku/aktivitas dari setiap *layer.* Fungsi-fungsi yang dikandung ini berupa fungsi aktivasi yang memungkinkan menjadi informasi fungsi aktivasi dari *layer.* Selain fungsi aktivasi, terdapat fungsi *preprocessing* untuk mengolah data sebelum dimasukkan ke fungsi aktivasi. Berikut ini fungsi yang ada di kelas *layer.*

Tabel 1.2.2 Fungsi pada Kelas *Layer*

Fungsi	**Penjelasan**

`set_nodes(self, val=0)`	fungsi ini bertugas untuk menginisialisasi nilai setiap *nodes* pada *layer.* Jumlah elemen pada list ditetapkan sebesar jumlah data masukan dan nilai setiap *node* dijadikan 0 terlebih dahulu atau sesuai dengan masukan dari argumen val.
`activate(self)`	fungsi ini bertugas untuk menjalankan fungsi aktivasi pada *layer* tersebut. Dengan demikian, didapatkan nilai *nodes* terbaru dari hasil perhitungan berdasarkan fungsi aktivasi yang terdefinisi pada *layer* tersebut.
`sigmoid(self)`	fungsi ini bertugas untuk mengubah nilai dari atribut *nodes* dari hasil perkalian *nodes* dengan matriks bobot menjadi *list* dengan rentang nilai di antara 0 - 1 berdasarkan perhitungan *sigmoid*.
`linear(self)`	fungsi ini bertugas untuk menerima nilai dari atribut *nodes* berupa hasil perkalian dengan matriks bobot. Akan tetapi, dikarenakan fungsi ini dijalankan pada program *neural network* karena setiap *layer* ini, maka fungsi *linear* pada kelas *layer* ini tidak merubah nilai apapun.
`relu(self)`	fungsi ini bertugas untuk mengubah nilai dari atribut *nodes* dari hasil perkalian *nodes* dengan matriks bobot jika nilai yang dihasilkan lebih besar dari 0. Akan tetapi, jika nilai yang dihasilkan adalah negatif, maka akan menetapkan nilai *nodes* menjadi 0.
`softmax(self)`	fungsi ini bertugas untuk mengubah nilai dari atribut *nodes* dari hasil perkalian *nodes* dengan matriks bobot menjadi *list* berdasarkan perhitungan *softmax.*

C. Implementasi Kelas Neural Network

Kelas *NeuralNetwork* merepresentasikan algoritma yang digunakan dalam menghitung nilai *nodes* berdasarkan setiap data masukan dengan cara mengolah daftar *layer* yang ada dengan urutan yang telah terdefinisi. Kelas *NeuralNetwork* ini juga

memiliki beberapa *fields* yang menjadi sumber dari informasi layer tersebut. Berikut ini *fields* yang ada di kelas ini.

Tabel 1.3.1. *Fields* pada Kelas *NeuralNetwork*

Fields	**Penjelasan**
depth	*depth* berfungsi untuk menyimpan informasi kedalaman dari *neural network.* Informasi kedalaman ini akan didapatkan dari jumlah *layer* yang terlibat (*input layer* + *hidden layer* + *output layer*) dikurang satu.
n_input	*n_input* berfungsi untuk menyimpan informasi jumlah data masukan yang ada dalam bentuk integer. Data masukan yang dimaksud ini adalah data-data dimana setiap data ini memiliki nilai masukan sebanyak besar dimensi dari *neural network.*
_layers	*_layers* berfungsi untuk menyimpan daftar *layer* yang akan digunakan pada *neural network* tersebut. Setiap *layer* didapatkan dari hasil instansiasi kelas *layer* yang telah berisi informasi dari *layer* tersebut. Dikarenakan menyimpan banyak *layer,* maka tipe data dari atribut *_layers* ini adalah *array* dimana setiap *layer* disimpan terurut berdasarkan urutan *layer* nya (dari *input layer* ke *output layer*).
_weights	*_weights* berfungsi untuk menyimpan matriks bobot untuk setiap *layer.* Dikarenakan menyimpan banyak matriks bobot, maka tipe datanya adalah *array.* Setiap bobot disimpan secara berurutan mulai dari matriks bobot untuk mengolah nilai dari *input layer* ke *layer selanjutnya* sampai dengan *output layer.*

Selain menyimpan *fields,* kelas *NeuralNetwork* memiliki beberapa fungsi yang menjadi perilaku/aktivitas dari setiap *NeuralNetwork.* Fungsi-fungsi yang disimpan ini berupa penginisiasian nilai *input layer* pada struktur *Neural Network* dan proses

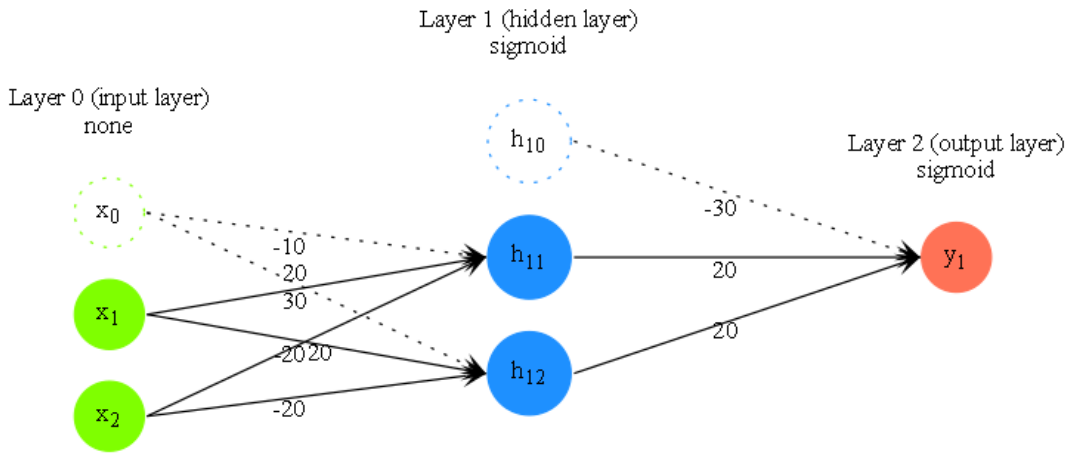
dilakukannya penghitungan prediksi nilai kelas dari model dan *input* yang diberikan. Berikut ini fungsi yang ada di kelas *NeuralNetwork.*

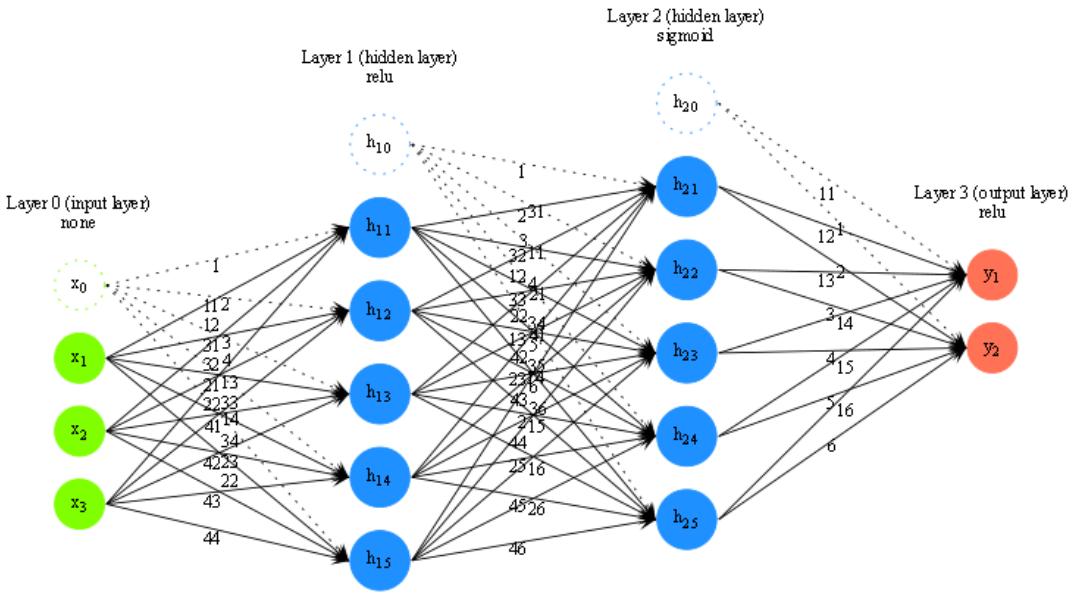
Tabel 1.3.2. Fungsi pada Kelas *NeuralNetwork*

Fungsi	**Penjelasan**
`init_input_layer(self, input_instance)`	Fungsi ini digunakan untuk menginisiasi *input layer* dengan menyimpan setiap nilai ke dalam sebuah *list* yang elemennya direpresentasikan sebagai *node layer* dari model (.json) yang diberikan.
`feed_forward(self, input_group)`	Fungsi ini adalah proses dilakukannya algoritma *Feed Forward Neural Network*. Fungsi ini digunakan untuk menghitung perkalian nilai vektor *weight* dengan nilai setiap *node* pada *layer* yang kemudian dihitung nilai fungsi aktivasi untuk disimpan dalam *node layer* selanjutnya dan akan dilakukan berulang hingga mencapai perhitungan pada *output layer*. Fungsi ini akan mengeluarkan hasil nilai *output node layer* sebanyak jumlah data yang disediakan.

D. Implementasi Visualisasi

Visualisasi *neural network* dilakukan dengan menggunakan *library* Graphviz. Untuk menggunakan Graphviz perlu terlebih dahulu dilakukan instalasi aplikasi Graphviz dengan file installer berikut. Pastikan untuk mencentang "Add to PATH" saat menginstall Graphviz agar modul dapat dieksekusi dengan command `dot` dari *command line*.





Model NeuralNetwork dikonversi ke dalam *string* sesuai sintaks dari dot. Visualisasi yang dilakukan mencakup setiap node yang dimiliki *layer* (baik *node* sesungguhnya dan *node* bias) yang tersusun berdasarkan *layer*, dan *weight* dari setiap *edge* yang menghubungkan suatu *node* dengan *node* lain. Pada setiap *layer* ditampilkan juga fungsi aktivasi apa yang digunakan.

Tabel 1.3.3. Fungsi untuk Visualisasi *NeuralNetwork*

Fungsi	**Penjelasan**
`generate_txt(neural_network)`	Fungsi ini menerima `neural_network`, objek `NeuralNetwork` dan menghasilkan *string* representasi dari model *Neural Network* tersebut sesuai sintaks dot yang digunakan oleh Graphviz
`build_format(result_txt)`	Fungsi ini menerima *string* representasi model *Neural Network* yang dihasilkan fungsi `generate_txt` di atas untuk kemudian dihasilkan visualisasinya dalam format gambar

E. Cara kerja program

Dalam program ini, ada beberapa kelas yang dibuat, ada kelas bernama ‘layer’, kelas ini akan menyimpan data-data yang berkaitan dengan sebuah layer yaitu jumlah neuron pada layer tersebut, metode aktivasi pada layer tersebut, dan nilai setiap neuron pada layer tersebut. Lalu ada pula kelas bernama ‘neural network’ yang bertugas untuk menyimpan layer pada network, kedalaman network, bobot setiap layer dan juga menjalankan feed forward pada network tersebut. Untuk FFNN sendiri, akan dijalankan dengan method dari kelas neural network.

Pertama-tama akan dibuat objek dari kelas neural network yang akan menyimpan data-data seperti yang disebutkan sebelumnya (hampir keseluruhan data ada pada kelas ini, karena kelas ini juga akan menyimpan si layer), kemudian barulah dilakukan FFNN.

Untuk FFNN akan dilakukan looping sebanyak instance input yang tersedia sehingga program dapat menghasilkan hasil FFNN untuk semua instance input yang ada. Untuk setiap loop, akan diambil instance input pada urutan loop tersebut. Instance input pertama pada urutan loop pertama dan seterusnya. Instance input selanjutnya disimpan dan akan digunakan sebagai masukan untuk layer di atasnya (input adalah layer paling bawah). Kemudian dilakukan looping untuk setiap layer yang ada (kecuali layer terbawah, yaitu layer input). Looping dilakukan dari layer bawah ke layer atas. Untuk setiap layer ini, akan dilakukan perkalian matrik dari bobot dari layer tersebut (bobot di sini selalu berarti bobot dari neuron yang ada pada layer dibawahnya yang akan digunakan pada layer tersebut) dengan masukan yang telah disimpan sebelumnya dan nilainya disimpan pada layer tersebut (anggap saja namanya nilai node). Nilai ini adalah nilai sum dari setiap neuron pada layer tersebut. Kemudian nilai ini pun akan diubah kembali/ di update dengan nilai aktivasi dengan cara melakukan aktivasi pada nilai tersebut sesuai metode aktivasi yang ada pada layer tersebut. Jadi pada akhirnya nilai node ini akan menyimpan nilai setiap neuron yang telah diaktivasi pada sebuah layer). Kemudian di akhir looping (loop untuk setiap layer) akan dilakukan pengecekan layer, jika layer tersebut adalah layer terakhir (layer output), maka nilai node akan dikembalikan dan menjadi nilai masukan yang baru. Namun jika tidak, maka nilai masukan yang baru akan berisi ‘1’ (sebagai bias) dan juga nilai node. hal ini akan dilakukan terus hingga loop berakhir (loop untuk setiap layer). Saat loop tersebut berakhir, nilai masukan tadi akan disimpan dan dimasukkan kedalam daftar hasil (nilai tersebut merupakan hasil FFNN dari sebuah instance input untuk network tersebut) dan juga akan ditampilkan. Begitu seterusnya untuk setiap instance input yang ada.

II. HASIL PENGUJIAN

Berikut ini hasil pengujian dari hasil penerapan fungsi aktivasi sigmoid.

1. Input 1 Instance

```
##############################
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [4.5397868702434395e-05, 0.9999999999999065]
Output Layer = y : [4.543910487654591e-05]
Hasil data 1: [0, 0] -> [0]
```

2. Input Sejumlah Instance

```
##############################
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [4.5397868702434395e-05, 0.9999999999999065]
Output Layer = y : [4.543910487654591e-05]
Hasil data 1: [0, 0] -> [0]

Input Layer : [x1, x2] = [0, 1]
Hidden Layer = [h1, h2] : [0.9999546021312976, 0.9999546021312976]
Output Layer = y : [0.999954519621495]
Hasil data 2: [0, 1] -> [1]

Input Layer : [x1, x2] = [1, 0]
Hidden Layer = [h1, h2] : [0.9999546021312976, 0.9999546021312976]
Output Layer = y : [0.999954519621495]
Hasil data 3: [1, 0] -> [1]

Input Layer : [x1, x2] = [1, 1]
Hidden Layer = [h1, h2] : [0.9999999999999065, 4.5397868702434395e-05]
Output Layer = y : [4.543910487654583e-05]
Hasil data 4: [1, 1] -> [0]
```

Berikut ini hasil pengujian dari hasil penerapan fungsi aktivasi relu & linear.

1. Input 1 Instance

```
##############################
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [0, 0]
Output Layer = y : [0]
Hasil data 1: [0, 0] -> [0]
```

2. Input Sejumlah Instance

```
##############################
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [0, 0]
Output Layer = y : [0]
Hasil data 1: [0, 0] -> [0]

Input Layer : [x1, x2] = [0, 1]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 2: [0, 1] -> [1]

Input Layer : [x1, x2] = [1, 0]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 3: [1, 0] -> [1]

Input Layer : [x1, x2] = [1, 1]
Hidden Layer = [h1, h2] : [2, 1]
Output Layer = y : [0]
Hasil data 4: [1, 1] -> [0]
```

Untuk perbandingan hasil program dengan perhitungan manual dapat dilihat pada bab 3 Perbandingan Hasil Program dan Manual.

III. PERBANDINGAN HASIL PROGRAM DAN MANUAL

A. XOR Sigmoid Model

$net_i = w_i^T \ x \ = \sum_j w_{i,j} x_j$, di mana:

- $i = \{0, 1, ..., m - 1\}$, m adalah banyak *node* pada *layer* $k + 1$
- $j = \{0, 1, ..., n\}$, n adalah banyak *node* pada *layer* k
- $w_i^T = [w_{i,0}, w_{i,1}, ..., w_{i,n}]^T$, w adalah matriks *weight* yang menghubungkan *layer* k dan *layer* $k + 1$
- x_j adalah *node* ke-j pada layer k, dengan $x_0 = 1$

$\sigma(net) \ = \ \frac{1}{1 + e^{-net}}$

Keterangan:

1. **Instance 1**

$$net_{h1} = \sum_{j=0}^{2} w_{0,j} x_j = x_0 \times w_{0,0} \ + \ x_1 \times w_{0,1} + \ x_2 \times w_{0,2}$$

$$net_{h1} = 1 \times (-10) + 0 \times 20 + 0 \times 20$$

$$net_{h1} = \ -10$$

$$\sigma(net_{h1}) = \sigma(-10)$$

$$\sigma(net_{h1}) = \frac{1}{1 + e^{-(-10)}} = 0.00004539786870243442 \approx 0$$

$$h1 \ = \ \sigma(net_{h1}) \ = \ 0 \ [karena \ \sigma(net_{h1}) \ mendekati \ 0]$$

$$net_{h2} = \sum_{j=0}^{2} w_{1,j} x_j = x_0 \times w_{1,0} \ + \ x_1 \times w_{1,1} + \ x_2 \times w_{1,2}$$

$$net_{h2} = 1 \times (30) + 0 \times (-20) + 0 \times (-20)$$

$$net_{h2} = 30$$

$$\sigma(net_{h2}) = \sigma(30)$$

$$\sigma(net_{h2}) = \frac{1}{1 + e^{-(30)}} = 0.9999999999999065 \approx 1$$

$$h2 \ = \ \sigma(net_{h2}) \ = 1 \ [karena \ \sigma(net_{h2}) \ mendekati \ 1]$$

$$net_y = \sum_{j=0}^{2} w_{0,j} x_j = h_0 \times w_{0,0} \ + \ h_1 \times w_{0,1} + \ h_2 \times w_{0,2}$$

$$net_y = 1 \times (-30) + 0 \times 20 + 1 \times 20$$

$net_y = -10$

$\sigma(net_y) = \sigma(-10)$

$\sigma(net_y) = \frac{1}{1+e^{-(-10)}} = 0.00004539786870243442$

$y = \sigma(net_y) = 0\ [karena\ \sigma(net_y)\ mendekati\ 0]$

2. Instance 2

$net_{h1} = \sum_{j=0}^{2} w_{0,j}x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$

$net_{h1} = 1 \times (-10) + 0 \times 20 + 1 \times 20$

$net_{h1} = 10$

$\sigma(net_{h1}) = \sigma(10)$

$\sigma(net_{h1}) = \frac{1}{1+e^{-(10)}} = 0.9999546021313 \approx 1$

$h1 = \sigma(net_{h1}) = 1\ [karena\ \sigma(net1)\ mendekati\ 1]$

$net_{h2} = \sum_{j=0}^{2} w_{1,j}x_j = x_0 \times w_{1,0} + x_1 \times w_{1,1} + x_2 \times w_{1,2}$

$net_{h2} = 1 \times (30) + 0 \times (-20) + 1 \times (-20)$

$net_{h2} = 10$

$\sigma(net_{h2}) = \sigma(10)$

$\sigma(net_{h2}) = \frac{1}{1+e^{-(10)}} = 0.9999546021313 \approx 1$

$h2 = \sigma(net_{h2}) = 1\ [karena\ \sigma(net1)\ mendekati\ 1]$

$net_y = \sum_{j=0}^{2} w_{0,j}x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$

$net_y = 1 * (-30) + 1 * 20 + 1 * 20$

$net_y = 10$

$\sigma(net_y) = \sigma(10)$

$\sigma(net_y) = \frac{1}{1+e^{-(10)}} = 0.9999546021313$

$y = \sigma(net_y) = 1\ [karena\ \sigma(net_y)\ mendekati\ 1]$

3. **Instance 3**

$$net_{h1} = \sum_{j=0}^{2} w_{0,j} x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$$
$$net_{h1} = 1 \times (-10) + 1 \times 20 + 0 \times 20$$
$$net_{h1} = 10$$
$$\sigma(net_{h1}) = \sigma(10)$$
$$\sigma(net_{h1}) = \frac{1}{1 + e^{-(10)}} = 0.9999546021313 \approx 1$$
$$h1 = \sigma(net_{h1}) = 1 [karena\ \sigma(net_{h1})\ mendekati\ 1]$$

$$net_{h2} = \sum_{j=0}^{2} w_{1,j} x_j = x_0 \times w_{1,0} + x_1 \times w_{1,1} + x_2 \times w_{1,2}$$
$$net_{h2} = 1 \times (30) + 1 \times (-20) + 0 \times (-20)$$
$$net_{h2} = 10$$
$$\sigma(net_{h2}) = \sigma(10)$$
$$\sigma(net_{h2}) = \frac{1}{1 + e^{-(10)}} = 0.9999546021313 \approx 1$$
$$h2 = \sigma(net_{h2}) = 1 [karena\ \sigma(net_{h2})\ mendekati\ 1]$$

$$net_{y} = \sum_{j=0}^{2} w_{0,j} x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$$
$$net_{y} = 1 * (-30) + 1 * 20 + 1 * 20$$
$$net_{y} = 10$$
$$\sigma(net_{y}) = \sigma(10)$$
$$\sigma(net_{y}) = \frac{1}{1 + e^{-(10)}} = 0.9999546021313 \approx 1$$
$$y = \sigma(net_{y}) = 1 [karena\ \sigma(net_{y})\ mendekati\ 1]$$

4. **Instance 4**

$$net_{h1} = \sum_{j=0}^{2} w_{0,j} x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$$
$$net_{h1} = 1 \times (-10) + 1 \times 20 + 1 \times 20$$
$$net_{h1} = 30$$
$$\sigma(net_{h1}) = \sigma(30)$$
$$\sigma(net_{h1}) = \frac{1}{1 + e^{-(30)}} = 0.99999999999991 \approx 1$$
$$h1 = \sigma(net_{h1}) = 1 [karena\ \sigma(net_{h1})\ mendekati\ 1]$$

$$net_{h2} = \sum_{j=0}^{2} w_{1,j} x_j = x_0 \times w_{1,0} + x_1 \times w_{1,1} + x_2 \times w_{1,2}$$

$$net_{h2} = 1 \times (30) + 1 \times (-20) + 1 \times (-20)$$

$$net_{h2} = -10$$

$$\sigma(net_{h2}) = \sigma(-10)$$

$$\sigma(net_{h2}) = \frac{1}{1 + e^{-(-10)}} = 0.00004539786870243442 \approx 0$$

$$h2 = \sigma(net_{h2}) = 0 \; [karena \; \sigma(net_{h2}) \; mendekati \; 0]$$

$$net_{y} = \sum_{j=0}^{2} w_{0,j} x_j = x_0 \times w_{0,0} + x_1 \times w_{0,1} + x_2 \times w_{0,2}$$

$$net_{y} = 1 \times (-30) + 1 \times 20 + 0 \times 20$$

$$net_{y} = -10$$

$$\sigma(net_{y}) = \sigma(-10)$$

$$\sigma(net_{y}) = \frac{1}{1 + e^{-(-10)}} = 0.00004539786870243442 \approx 0$$

$$y = \sigma(net_{y}) = 0 \; [karena \; \sigma(net_{y}) \; mendekati \; 0]$$

Tabel 3.1.1. Perhitungan Manual

id	x0	x1	x2	f	$\sum h1$	h1	$\sum h2$	h2	$\sum y$	y
1	1	0	0	0	-10	0	30	1	-10	0
2	1	0	1	1	10	1	10	1	10	1
3	1	1	0	1	10	1	10	1	10	1
4	1	1	1	0	30	1	-10	0	-10	0

Tabel 3.1.2. Hasil Program

```
##########
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [0, -1]
Output Layer = y : [0]
Hasil data 1: [0, 0] -> [0]

Input Layer : [x1, x2] = [0, 1]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 2: [0, 1] -> [1]

Input Layer : [x1, x2] = [1, 0]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 3: [1, 0] -> [1]

Input Layer : [x1, x2] = [1, 1]
Hidden Layer = [h1, h2] : [2, 1]
Output Layer = y : [0]
Hasil data 4: [1, 1] -> [0]
```

B. XOR ReLu and Linear

$h = ReLU(XW_{xh} + c) = max\{0, XW + C\}$, di mana:

- $X = [x1, x2, \ldots, xn]$, x adalah nilai masukan.
- W_{xh} merupakan matrik bobot 2 dimensi dimana x adalah baris dan y adalah kolom.
- c adalah konstanta

$\sigma(net) = \frac{1}{1 + e^{-net}}$

Keterangan:
Berikut ini matriks bobot yang digunakan.

- W_{xh}= $[[1, 1], [1, 1]]$
- W_{hy}= $[1, -2]$

1. Instance 1

$h = max\{0, XW + C\}\ dengan\ X = [0, 0]\ dan\ C = [0, -1]$
Dengan menggunakan nilai X = [0,0] maka

$$h = max\{0, [0, 0] * [[1, 1], [1, 1]] + [0, -1]\}$$
$$h = max\{0, [0, 0] + [0, -1]\}$$
$$h = max\{0, [0, -1]\}$$
$$h = [0, 0]$$

$h = max\{0, hW + b\}\ dengan\ h\ didapatkan\ dari\ nilai\ layer\ sebelumnya\ dan\ b = 0.$
Dengan menggunakan nilai h = [0,0] maka

$$y = max\{0, [0, 0] * [[1, -2]]\}$$
$$y = max\{0, 0\}$$
$$y = 0$$

---

2. Instance 2

$h = max\{0, XW + C\}\ dengan\ X = [0, 1]\ dan\ C = [0, -1]$
Dengan menggunakan nilai X = [0,1] maka

$$h = max\{0, [0, 1] * [[1, 1], [1, 1]] + [0, -1]\}$$
$$h = max\{0, [1, 1] + [0, -1]\}$$
$$h = max\{0, [1, 0]\}$$
$$h = [1, 0]$$

$h = max\{0, hW + b\}\ dengan\ h\ didapatkan\ dari\ nilai\ layer\ sebelumnya\ dan\ b = 0.$
Dengan menggunakan nilai h = [1,0] maka

$$y = max\{0, [1, 0] * [[1, -2]]\}$$
$$y = max\{0, 1\}$$
$$y = 1$$

3. **Instance 3**

$h = max\{0, XW + C\}\ dengan\ X = [1, 0]\ dan\ C = [0, -1]$

Dengan menggunakan nilai X = [1,0] maka

$h = max\{0\ ,\ [1, 0]\ *\ [[1, 1], [1, 1]]\ +\ [0, -1]\}$

$h = max\{0\ ,\ [1, 1]\ +\ [0, -1]\}$

$h = max\{0,\ [1,\ 0]\}$

$h = [1, 0]$

$h = max\{0, hW + b\}\ dengan\ h\ didapatkan\ dari\ nilai\ layer\ sebelumnya\ dan\ b = 0.$

Dengan menggunakan nilai h = [1,0] maka

$y = max\{0, [1, 0]\ *\ [[1, -2]]\}$

$y = max\{0, 1\}$

$y = 1$

---

4. **Instance 4**

$h = max\{0, XW + C\}\ dengan\ X = [1, 1]\ dan\ C = [0, -1]$

Dengan menggunakan nilai X = [1,1] maka

$h = max\{0, [1, 1]\ *\ [[1, 1], [1, 1]]\ +\ [0, -1]\}$

$h = max\{0, [2, 2]\ +\ [0, -1]\}$

$h = max\{0, [2, 1]\}$

$h = [2, 1]$

$h = max\{0, hW + b\}\ dengan\ h\ didapatkan\ dari\ nilai\ layer\ sebelumnya\ dan\ b = 0.$

Dengan menggunakan nilai h = [2,1] maka

$y = max\{0, [2, 1]\ *\ [[1, -2]]\}$

$y = max\{0, 0\}$

$y = 0$

Tabel 3.2.1. Perhitungan Manual

id	x0	x1	x2	f	h	y
1	1	0	0	0	[0,0]	0
2	1	0	1	1	[1,0]	1
3	1	1	0	1	[1,0]	1
4	1	1	1	0	[2,1]	0

Tabel 3.2.2. Hasil Program

```
################################
Batch 1:
Input Layer : [x1, x2] = [0, 0]
Hidden Layer = [h1, h2] : [0, 0]
Output Layer = y : [0]
Hasil data 1: [0, 0] -> [0]

Input Layer : [x1, x2] = [0, 1]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 2: [0, 1] -> [1]

Input Layer : [x1, x2] = [1, 0]
Hidden Layer = [h1, h2] : [1, 0]
Output Layer = y : [1]
Hasil data 3: [1, 0] -> [1]

Input Layer : [x1, x2] = [1, 1]
Hidden Layer = [h1, h2] : [2, 1]
Output Layer = y : [0]
Hasil data 4: [1, 1] -> [0]
```

IV. PEMBAGIAN TUGAS

Tugas	NIM
Kode (Layer, Neural Network), Laporan	13519010
Kode (Layer, Neural Network), Laporan	13519016
Kode (Layer, Neural Network), Laporan	13519159
Kode (Layer, Neural Network), Visualisasi, Laporan	13519216