# IF3270 - Pembelajaran Mesin
# Bagian A: Implementasi Forward Propagation untuk Feed Forward Neural Network


Disusun oleh:

| | |
|---|---|
| Bryan Rinaldo | 13519103 |
| Naufal Alexander Suryasumirat | 13519135 |
| Naufal Yahya Kurnianto | 13519141 |
| Maximillian Lukman | 13519153 |

**PROGRAM STUDI TEKNIK INFORMATIKA**
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**
**INSTITUT TEKNOLOGI BANDUNG**

**2021**

# 1. Implementasi

## 1.1. Implementasi Kelas Layer, FFNN, dan fungsi aktivasi

Implementasi kelas `Layer` dilakukan dengan mendefinisikan atribut yang dimiliki oleh sebuah Layer. Di dalam sebuah Layer, terdapat jumlah neuron, *weight matrix* koneksi dari *input* ke *neuron* yang terdapat di dalam Layer, dan fungsi aktivasi yang digunakan oleh Layer tersebut. Kelas Layer melakukan kalkulasi berupa *matrix dot product* antara input *matrix* yang diberikan kepada Layer tersebut dan hasil *transpose* terhadap *weight matrix* yang dimilikinya. Selain itu, kelas Layer memiliki sebuah fungsi untuk mendapatkan informasi mengenai Layer tersebut untuk keperluan menampilkan model.

```
class Layer:
    # n_neuron: number of neuron, weights: weight matrix,
    # activation: activation function
    def __init__(self, n_neuron: int, weights: np.array, activation: str):
        self.n_neuron = n_neuron # visualization purposes
        self.weights = weights
        self.activation = activation
        self.act_function = activation_functions[activation]

    def calculate(self, in_matrix: np.array):
        return self.act_function(np.dot(self.weights.transpose(), in_matrix))

    def get_structure(self):
        # mengembalikan informasi untuk keperluan menampilkan model
```

Implementasi kelas FFNN atau *Feedforward Neural Network* dilakukan dengan mendefinisikan atribut yang dimiliki oleh sebuah FFNN. Di dalam sebuah FFNN, terdapat sejumlah *hidden layers* yang merupakan instansi dari kelas Layer. Selain itu, terdapat juga *input layer* yang merupakan input berupa matrix data XOR yang dapat berjumlah satu atau lebih dari satu data/*batch* sejumlah *instances*, dan *output layer* yang juga merupakan instansi dari kelas Layer. Kelas FFNN memiliki fungsi untuk melakukan *feed forward* dengan memanggil fungsi *calculate* yang dimiliki oleh setiap Layer-nya untuk *input* yang diberikan, dan melanjutkan *output* tersebut ke Layer selanjutnya hingga *output layer* dan mendapatkan hasil prediksi. Kelas FFNN memiliki fungsi yang dapat memprediksi kelas dari input data secara *batch* menggunakana atau satu per satu menggunakan fungsi *predict* yang dimilikinya. Selain itu, kelas FFNN memiliki sebuah fungsi untuk mendapatkan informasi mengenai Layer tersebut untuk keperluan menampilkan model.

```
class FFNN:
    def __init__(self,  hidden_layers: list, input_layer = None, threshold = 0.5):
        self.hidden_layers = hidden_layers
        self.output_layer = hidden_layers[-1]
        self.input_layer = input_layer
```

```
        self.threshold = threshold

    def feed_forward(self):
        if (isinstance(self.input_layer, type(None))): return None
        if len(self.input_layer.shape) == 1: return self.forward(self.input_layer)
        else:
            outputs = []
            for data in self.input_layer: outputs.append(self.forward(data))
            if (self.output_layer.activation == 'softmax'): return outputs
            return np.array(outputs).flatten()

    def forward(self, input):
        output = input
        for i in range(0, len(self.hidden_layers)):
            output = self.hidden_layers[i].calculate(np.append(output, 1))
        if (self.output_layer.activation == 'softmax'): return output
        return int(output > self.threshold)

    def attach_hidden_layer(self, hidden_layer: Layer):
        self.hidden_layers.append(hidden_layer)

    def predict(self, input_layer: np.array): # input_layer without bias
        self.input_layer = input_layer
        return self.feed_forward()

    def get_structure(self):
        # mengembalikan informasi untuk keperluan menampilkan model
```

Implementasi fungsi aktivasi dilakukan dengan mendefinisikan fungsi *lambda* yang sesuai dengan tiap fungsi aktivasi sebagai berikut. Selanjutnya, setiap fungsi aktivasi kecuali fungsi aktivasi *softmax* dilakukan *np.vectorize* agar fungsi dapat digunakan untuk suatu *np.array* secara langsung. Terdapat juga *dictionary* yang dapat digunakan untuk memanggil fungsi dengan mudah.

```
# Activation functions
linear = lambda x: x
linear = np.vectorize(linear)

sigmoid = lambda x: 1 / (1 + math.exp(-x))
sigmoid = np.vectorize(sigmoid)

relu = lambda x: max(0, x)
relu = np.vectorize(relu)

softmax = lambda x: np.exp(x) / np.exp(x).sum()

activation_functions = {
    'linear': linear,
    'sigmoid': sigmoid,
    'relu': relu,
```

```
    'softmax': softmax
}
```

### 1.2. Implementasi Input/Output File

Implementasi fungsi _input dimulai dengan menerima dua buah parameter yaitu nama file testcase yang akan digunakan serta dengan variabel “with_input” yang menandakan apakah file yang dimasukkan termasuk data input atau tidak. Format untuk file yang dimasukkan adalah .txt di mana isinya terdiri dari berapa banyak jumlah layer pada suatu model yang akan dimasukkan ke dalam variabel nLayer, berapa banyak jumlah neuron pada setiap layer yang akan dimasukkan ke dalam array n_layer_neurons, dan terakhir bias dan bobot serta fungsi aktivasi dari setiap layer yang akan dimasukkan ke dalam dictionary struct_model. Kemudian untuk setiap layer yang ada pada model yang dimasukkan, akan dibuat menjadi class bernama Layer yang kemudian akan dijadikan parameter bagi class FFNN yang akan di-*return*.

```
def _input(filename: str, with_input = False) -> tuple((FFNN, np.array, np.array)):
  f = open(filename, "r")
  f = f.readlines()
  f = [line.strip() for line in f]

  nLayer = int(f[0])
  f = f[1:]
  n_layer_neurons = []
  struct_model = {}

  for i in range(nLayer-1):
    struct_model[i] = {}
    n_layer_neurons.append(int(f[0]))
    struct_model[i]["b"] = [float(b) for b in f[1].split()]
    struct_model[i]["w"] = [[float(w) for w in weights.split()] for weights in
f[2:(2 + int(f[0]))]]
    struct_model[i]["f"] = f[2 + int(f[0])]
    f = f[2 + int(f[0]) + 1:]

  n_layer_neurons.append(int(f[0]))

  if (with_input):
    n_input = int(f[1])
    f = f[2:]
    input_data = []
    for i in range(n_input):
      input = [int(x) for x in (f[i].split())]
      input_data.append(input)

    f = f[n_input:]
    validation_data = []
    for i in range(n_input):
      result = [int(y) for y in (f[i].split())]
      validation_data.append(result)
```

```
  model_layers = []
  for i in range (nLayer-1):
    weight = struct_model[i]["w"]
    weight.append(struct_model[i]["b"])
    layer = Layer(n_layer_neurons[i+1], np.array(weight),
struct_model[i]["f"].lower())
    model_layers.append(layer)

  if (with_input):
    return FFNN(model_layers, np.array(input_data)), input_data, validation_data
  else:
    return FFNN(model_layers)
```

Di bawah ini merupakan beberapa contoh file input yang kami gunakan untuk menguji fungsi FFNN:

| **sigmoid_sigmoid.txt** | **relu_linear.txt** |
|---|---|
| 3<br>2<br>-10 30<br>20 -20<br>20 -20<br>sigmoid<br>2<br>-30<br>20<br>20<br>sigmoid<br>1 | 3<br>2<br>0 -1<br>1 1<br>1 1<br>relu<br>2<br>0<br>1<br>-2<br>linear<br>1 |

### 1.3. Implementasi Menampilkan Model

Implementasi fungsi showModel memiliki satu buah parameter yaitu FFNN. Untuk menampilkan model dari sebuah FFNN, pengguna hanya perlu memasukkan model FFNN yang telah dibuat kedalam fungsi showModel ini. Fungsi showModel akan menampilkan input yang dimasukkan, hidden layer, output layer, dan predicted result. Pada hidden layer dan output layer sendiri, akan ditampilkan data input, weight, dan bias. Pada saat dijalankan, fungsi akan membuat variabel baru untuk menyimpan model.get_structure. Pada variabel tersebut sudah terdapat beberapa data yang diperlukan untuk ditampilkan yang bisa didaptkan dari class FFNN. Fungsi akan mencari apakah input yang diberikan satu atau lebih dari satu. Setelah itu fungsi akan masuk ke loop untuk menampilkan setiap data input.

```
def showModel(model: FFNN): #masukan berupa FFNN
  initLayers = model.get_structure()

  countInput = len(initLayers[0])
  countLayer = len(initLayers[1])
  if(initLayers[0].ndim == 1):
    countInput = 1
  else:
    countInput = len(initLayers[0])
  for i in range(0, countInput):
    if(countInput == 1):
      inputLayer = initLayers[0]
    else:
      inputLayer = initLayers[0][i]
    output = inputLayer
    print("==============Data %d==============\n"%(i+1))
    print("-------------Input------------")
    print("Input Layer: ", inputLayer)
    print("------------------------------")
    for j in range(0, countLayer):
      weight = initLayers[1][j][1]
      bias = initLayers[1][j][2]
      activation = initLayers[1][j][3]
      combinedArr = initLayers[1][j][4]

      if(j==(countLayer-1)):
        print("------ Output Layer ------" )
        print("Input: ", output)
        print("H%d Weight: " %(j+1), weight)
        print("H%d Bias: " %(j+1), bias)
        print('\n')
        output = activation_functions[activation](np.dot(combinedArr.transpose(),
np.append(output, 1)))
        print("------------------------------")
        print("Predicted Result: ", model.forward(inputLayer))
      else:
        print("--- Hidden Layer %d ---" %(j+1))
        print("Input: ", output)
        print("H%d Weight: " %(j+1), weight)
```

```
        print("H%d Bias: " %(j+1), bias)
        print('\n')
        output = activation_functions[activation](np.dot(combinedArr.transpose(),
np.append(output, 1)))
        print("------------------------------")

showModel(ffnn_sigmoid)
```

## 2. Hasil Pengujian

Pada Pengujian Model FFNN yang kami bentuk, berikut merupakan bentuk input untuk diuji.

Input 1 Instance

```
# XOR Single Dataset
input1 = np.array([1,1])
input2 = np.array([1,0])
input3 = np.array([0,0])
input4 = np.array([0,1])
```

Input batch sejumlah instances

```
# XOR Multi/Batch Dataset
input = np.array([
    [1, 1],
    [1, 0],
    [0, 0],
    [0, 1]
])
```

Selain membuat inputnya, kami membentuk 4 tipe model untuk memprediksi input-input tersebut. Berikut merupakan hasil pengujian model-model yang kami bentuk.

### 2.1. Model 1

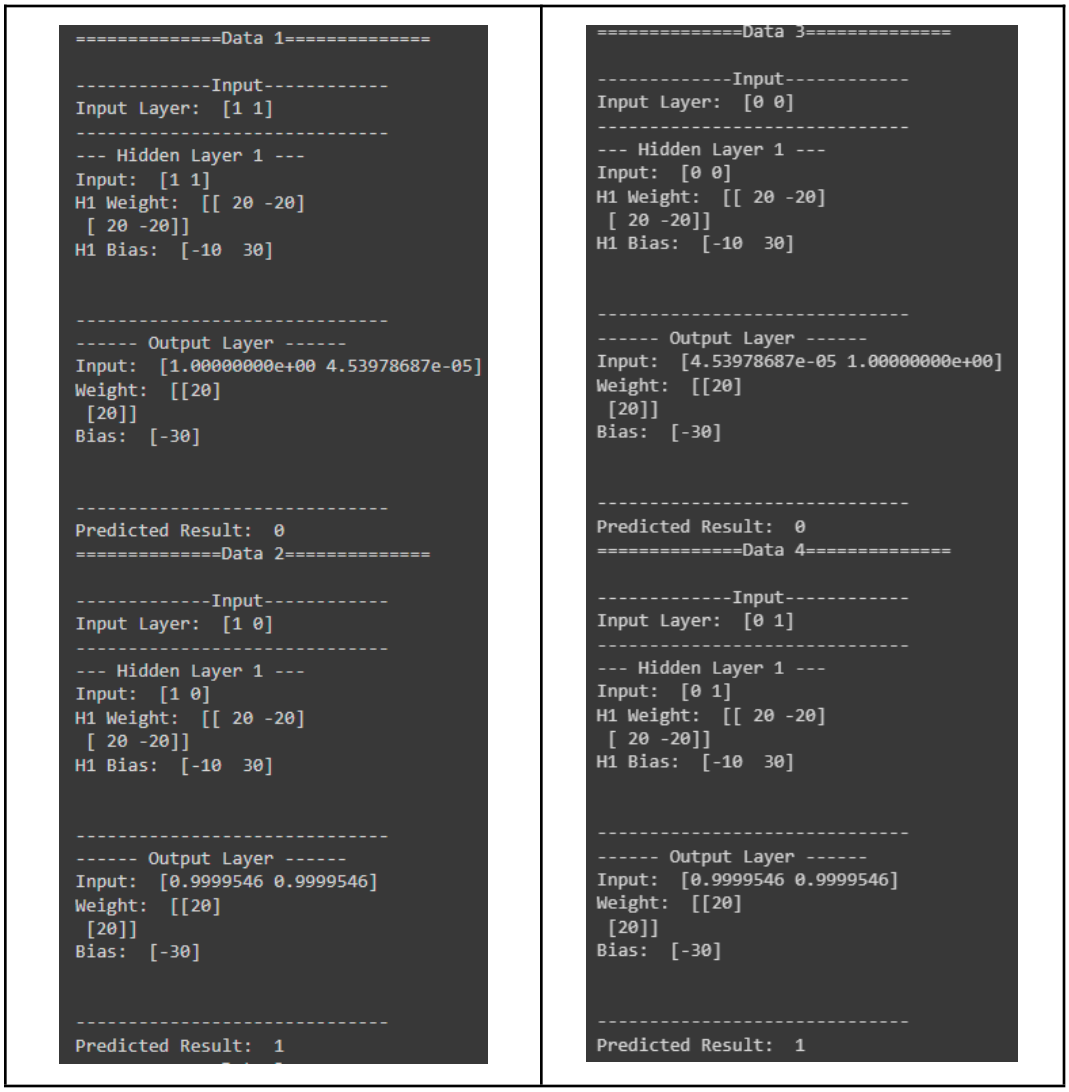
Model 1 menggunakan fungsi aktivasi sigmoid untuk kedua layer yang dimilikinya seperti yang dapat dilihat pada struktur file masukkan di atas. Berikut tangkapan layar bagi masing-masing pengujian menggunakan input 1 instance dan batch.

#### 2.1.1. Input 1 Instance

```
==============Data 1==============

-------------Input------------
Input Layer:  [0 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 0]
H1 Weight:  [[ 20 -20]
 [ 20 -20]]
H1 Bias:  [-10  30]


------------------------------
------ Output Layer ------
Input:  [4.53978687e-05 1.00000000e+00]
Weight:  [[20]
 [20]]
Bias:  [-30]


------------------------------
Predicted Result:  0
```

Dapat terlihat pada gambar di atas bahwa model 2 memprediksi dengan tepat yaitu kelas 0 untuk masukkan berupa 0 dan 0 dari dataset XOR. Sehingga untuk input 1 *instance* seperti yang ditampilkan di atas, model 1 memiliki akurasi sebesar 100%.

2.1.2. Input batch sejumlah instances




Dapat terlihat pada gambar di atas bahwa model 1 memprediksi dengan tepat untuk seluruh dataset yang diberikan dengan jumlah 4 buat data dari dataset XOR. Table prediksi model 1 dapat dilihat sebagai berikut.

| x1 | x2 | y | prediksi_model 1 |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

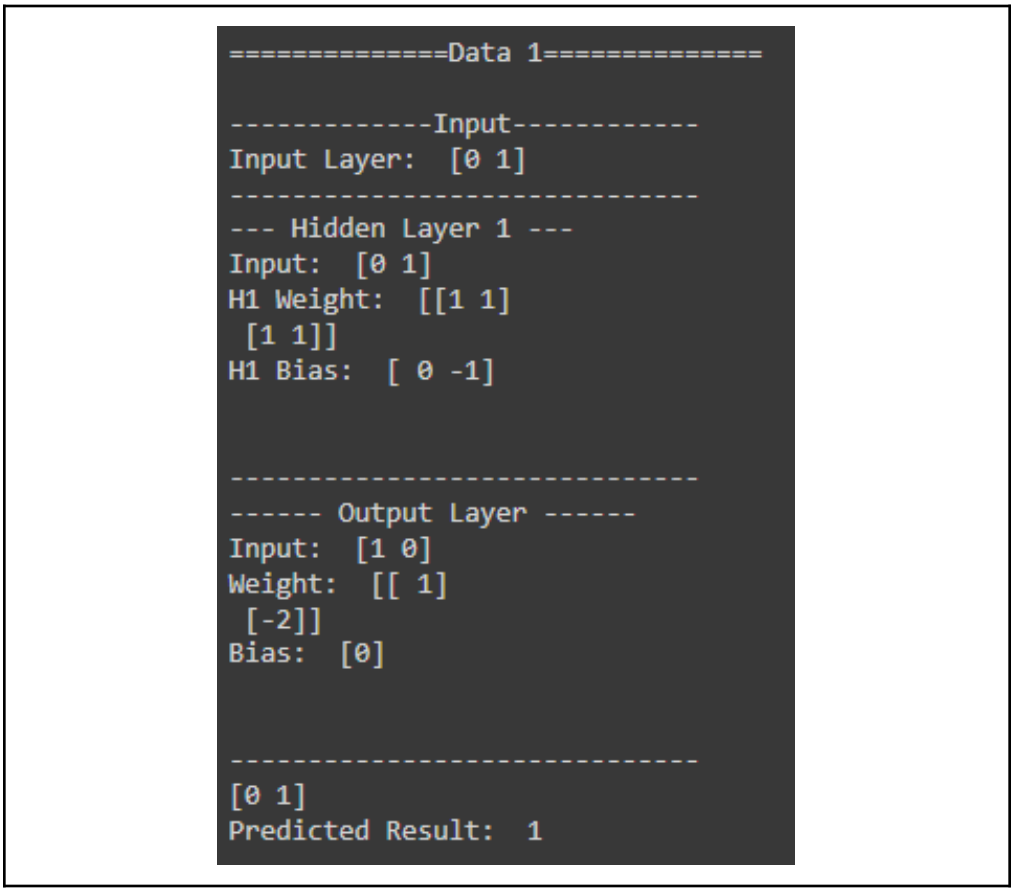
| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 1 |

Dapat terlihat pada tabel di atas bahwa model 1 memprediksi dengan tepat untuk seluruh input batch yang diberikan. Sehingga dapat disimpulkan bahwa akurasi dari model 1 adalah 100% untuk dataset XOR.

## 2.2.Model 2

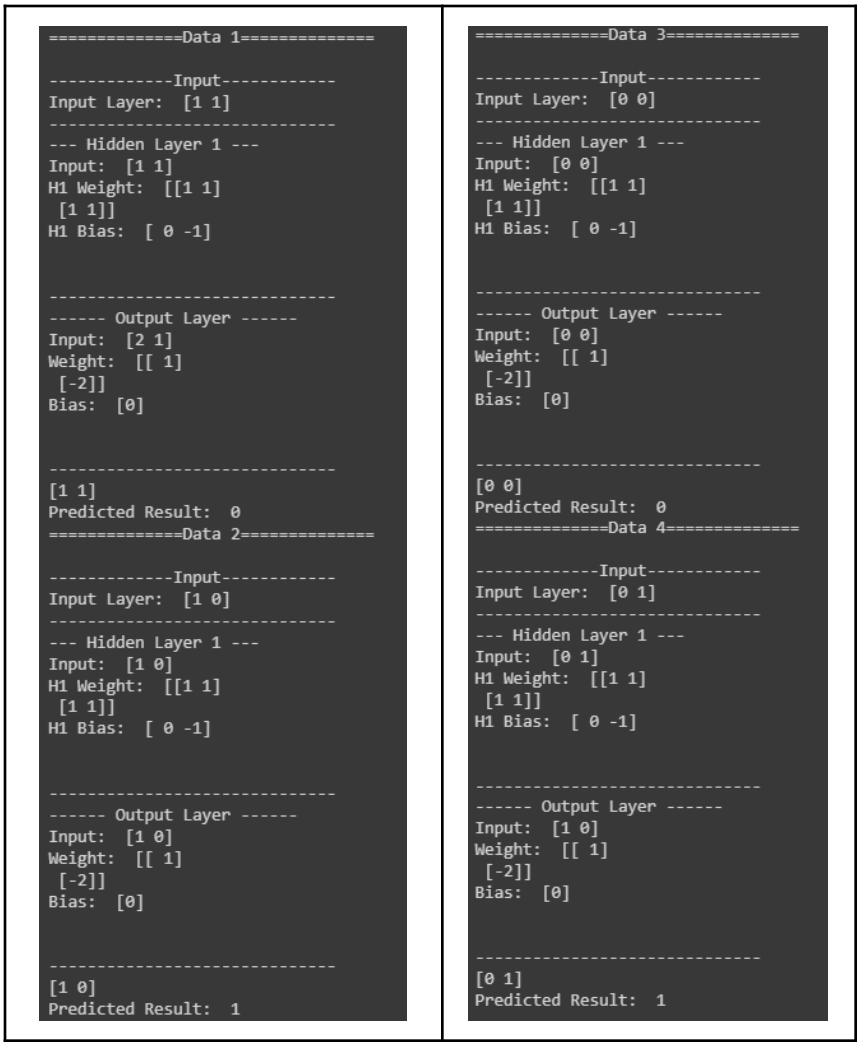
Model 2 menggunakan fungsi aktivasi ReLU untuk layer pertama dan fungsi aktivasi linear untuk layer kedua seperti yang dapat dilihat pada kode di atas. Berikut tangkapan layar bagi masing-masing pengujian menggunakan input 1 instance dan input batch.

### 2.2.1. Input 1 Instance




Dapat terlihat pada gambar di atas bahwa model 2 memprediksi dengan tepat yaitu kelas 1 untuk masukkan berupa 0 dan 1 dari dataset XOR. Sehingga untuk input 1 *instance* seperti yang ditampilkan di atas, model 2 memiliki akurasi sebesar 100%.

2.2.2. Input batch sejumlah instances




Dapat terlihat pada gambar di atas bahwa model 2 memprediksi dengan tepat untuk seluruh dataset yang diberikan dengan jumlah 4 buah data dari dataset XOR. Tabel prediksi model 3 dapat dilihat sebagai berikut.

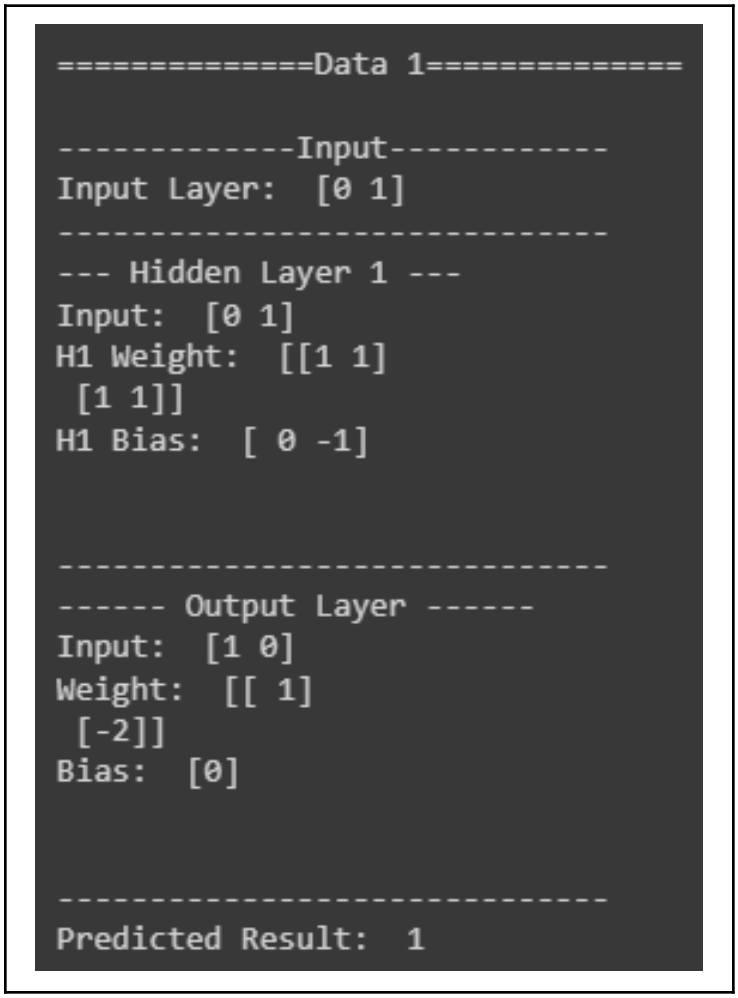
| x1 | x2 | y | prediksi_model 3 |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |

Dapat terlihat pada tabel di atas bahwa model 2 memprediksi dengan tepat untuk seluruh input batch tersebut. Sehingga dapat disimpulkan bahwa akurasi dari model 2 adalah 100% untuk dataset XOR.

### 2.3.Model 3

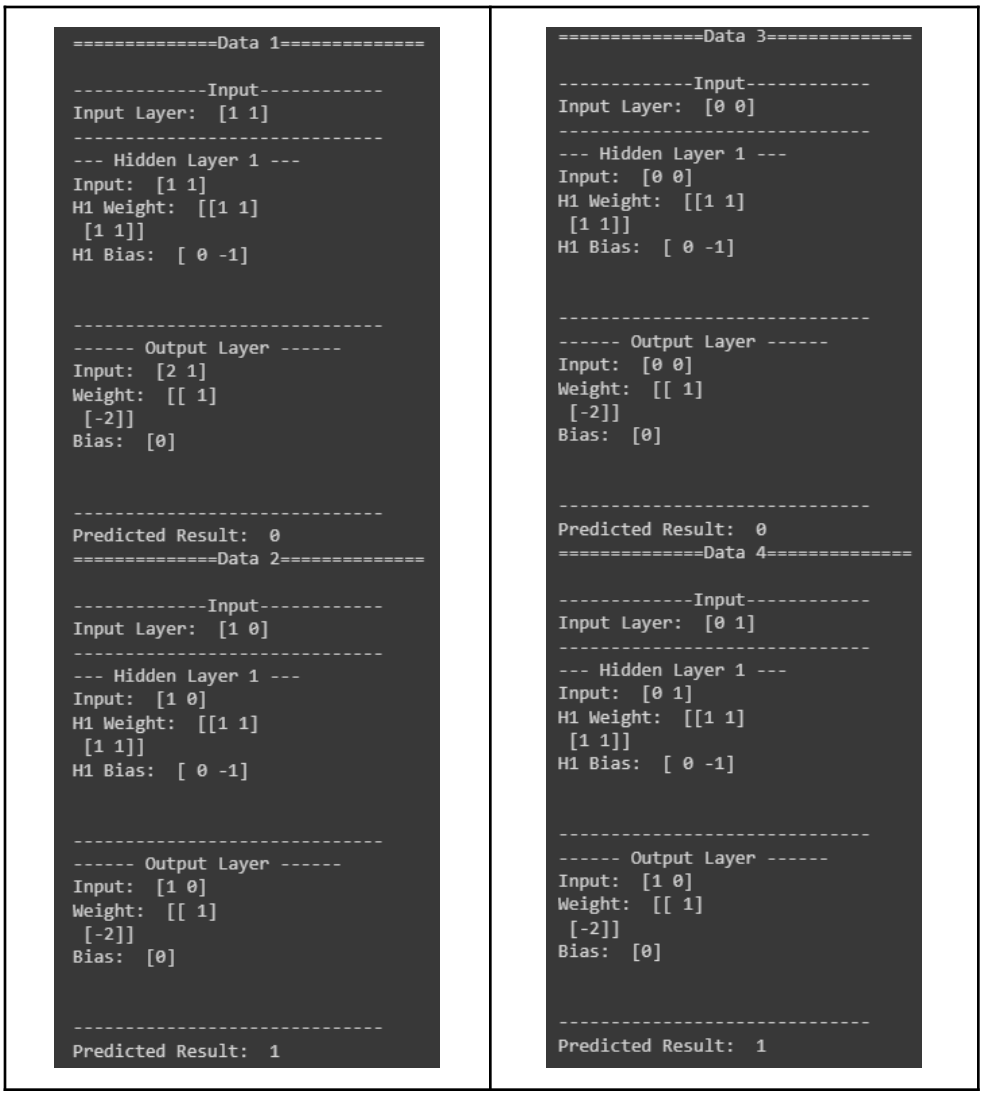
Model 3 menggunakan fungsi aktivasi ReLU untuk kedua layer seperti yang dapat dilihat pada file masukan di atas. Berikut tangkapan layar bagi masing-masing pengujian menggunakan input 1 instance dan batch.

2.3.1. Input 1 Instance




Dapat terlihat pada gambar di atas bahwa model 3 memprediksi dengan tepat yaitu kelas 1 untuk masukkan berupa 0 dan 1 dari dataset XOR. Sehingga untuk input 1 *instance* seperti yang ditampilkan di atas, model 3 memiliki akurasi sebesar 100%.

2.3.2. Input batch sejumlah instances




Dapat terlihat pada gambar di atas bahwa model 3 memprediksi dengan tepat untuk seluruh dataset yang diberikan dengan jumlah 4 buah data dari dataset XOR. Tabel prediksi model 3 dapat dilihat sebagai berikut.

| x1 | x2 | y | prediksi_model 3 |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |

Dapat terlihat pada tabel di  atas bahwa model 3 memprediksi dengan tepat untuk seluruh input batch tersebut. Sehingga dapat disimpulkan bahwa akurasi dari model 3 adalah 100% untuk dataset XOR.

## 2.4.Model 4

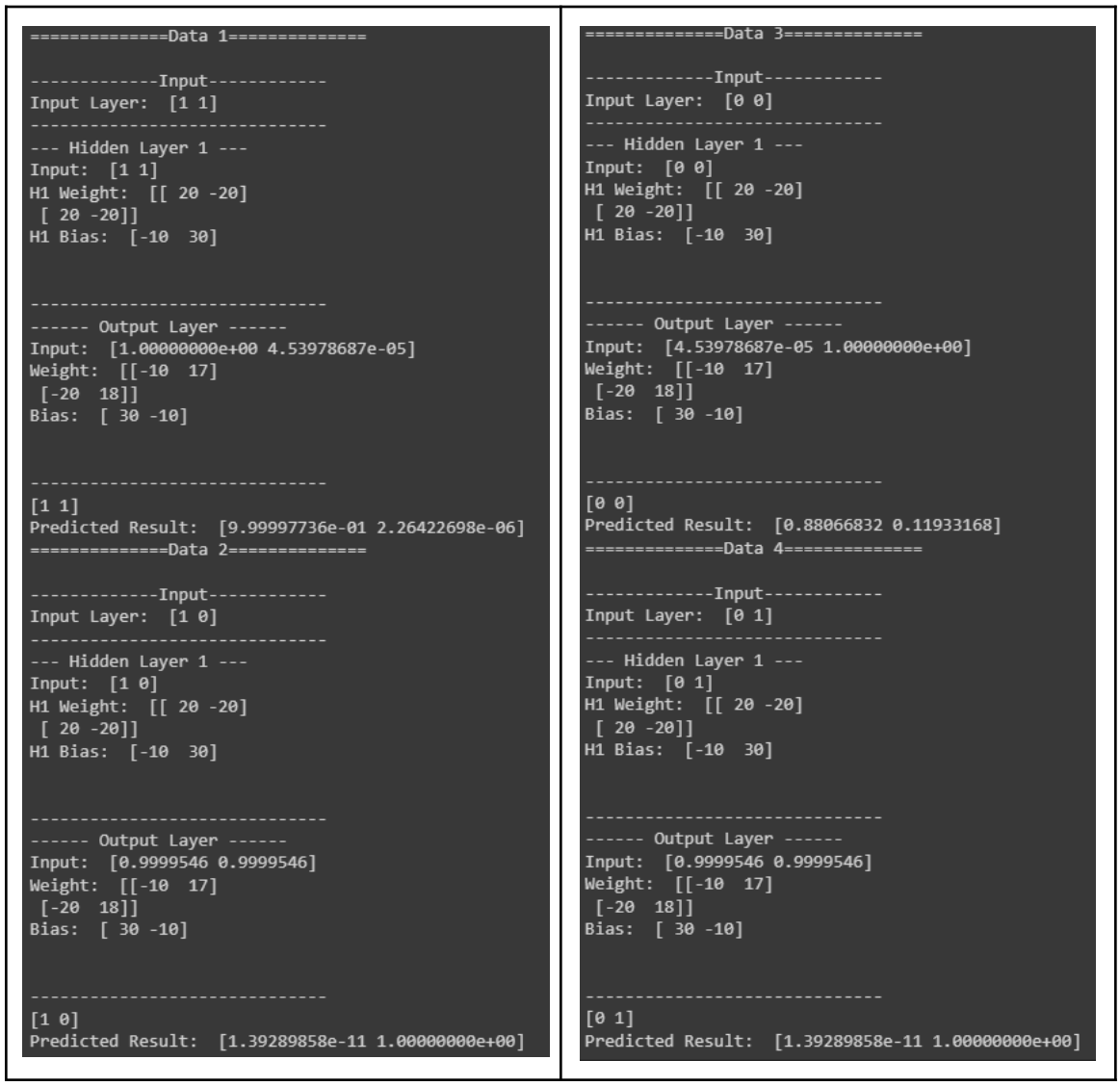
Model 4 menggunakan fungsi aktivasi Sigmoid layer pertama dan Softmax untuk layer kedua seperti yang dapat dilihat pada kode di atas. Berikut tangkapan layar bagi masing-masing pengujian menggunakan input 1 instance dan batch.

### 2.4.1. Input 1 Instance

```
==============Data 1==============

-------------Input------------
Input Layer:  [1 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 1]
H1 Weight:  [[ 20 -20]
 [ 20 -20]]
H1 Bias:  [-10  30]



------------------------------
------ Output Layer ------
Input:  [1.00000000e+00 4.53978687e-05]
Weight:  [[-10  17]
 [-20  18]]
Bias:  [ 30 -10]



------------------------------
[1 1]
Predicted Result:  [9.99997736e-01 2.26422698e-06]
```

Pada model 4 yang menggunakan softmax, index prediksi dengan nilai terbesar merupakan hasil kelas yang diprediksikan oleh model. Dapat dilihat pada prediksi data 1 bahwa kelas hasil prediksi untuk input [1, 1] adalah 0 karena elemen pada index 0 memiliki probabilitas yang paling besar. Sehingga untuk input 1 *instance* seperti yang ditampilkan di atas, model 4 memiliki akurasi sebesar 100%.

2.4.2. Input batch sejumlah instances




Pada model 4 yang menggunakan softmax, index prediksi dengan nilai terbesar merupakan hasil kelas yang diprediksikan oleh model. Dapat dilihat pada prediksi data 1 bahwa kelas hasil prediksi untuk input [1, 1] adalah 0 karena elemen pada index 0 memiliki probabilitas yang paling besar. Dapat terlihat juga di bawah ini, akurasi model 4 adalah 100%.

| x1 | x2 | y | prediksi_model 4 |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |

## 3. Perbandingan dengan Hasil Perhitungan Manual

### 3.1. Perhitungan Manual Model 1 (Sigmoid - Sigmoid)

| xbias | x1 | x2 | Σh1 | h1 | Σh2 | h2 | Σy | y |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | -10 | 0.0000453978687 | 30 | 1 | -9.999092043 | 0.00004543910488 |
| 1 | 0 | 1 | 10 | 0.9999546021 | 10 | 0.9999546021 | 9.998184085 | 0.9999545196 |
| 1 | 1 | 0 | 10 | 0.9999546021 | 10 | 0.9999546021 | 9.998184085 | 0.9999545196 |
| 1 | 1 | 1 | 30 | 1 | -10 | 0.0000453978687 | -9.999092043 | 0.00004543910488 |

σ(net) = 1/(1+e^(-net))
h1 = σ(-10*xbias+20*x1+20*x2)
h2 = σ(30*xbias-20*x1-20*x2)
y = σ(-30*xbias+20*h1+20*h2)

Melalui perhitungan manual, diperoleh hasil FFNN pada layer y adalah 0 untuk x1 = x2 dan 1 untuk x1 =/= x2. Hal ini benar dan hasil pengujian dengan model yang telah kami buat juga menunjukkan hasil yang sama yaitu hasil yang benar.

### 3.2. Perhitungan Manual Model 2 (ReLU - Linear)

| x1 | x2 | Σh1 | h1 | Σh2 | h2 | y |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | -1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 2 | 2 | 1 | 1 | 0 |

h1 = ReLU(0+x1+x2) = MAX(0, [0+x1+x2])
h2 = ReLU(-1+x1+x2) = MAX(0, [-1+x1+x2])
y = 0+h1-2*h2; Matriks yang dipakai adalah [1 -2]

Melalui perhitungan manual, diperoleh hasil FFNN pada layer y adalah 0 untuk x1 = x2 dan 1 untuk x1 =/= x2. Hal ini benar dan hasil pengujian dengan model yang telah kami buat juga menunjukkan hasil yang sama yaitu hasil yang benar.

## 4. Pembagian Tugas

| NIM | Nama | Pembagian Tugas |
|---|---|---|
| 13519103 | Bryan Rinaldo | ● Implementasi: menampilkan struktur model<br>● Laporan: Implementasi menampilkan model, hasil pengujian |
| 13519135 | Naufal Alexander Suryasumirat | ● Implementasi: kelas dan fungsi aktivasi<br>● Laporan: Implementasi kelas dan fungsi aktivasi, hasil pengujian |
| 13519141 | Naufal Yahya Kurnianto | ● Implementasi: kelas dan fungsi aktivasi<br>● Laporan: Perbandingan dengan perhitungan manual, hasil pengujian |
| 13519153 | Maximillian Lukman | ● Implementasi: input/output<br>● Laporan: Implementasi input/output, hasil pengujian |