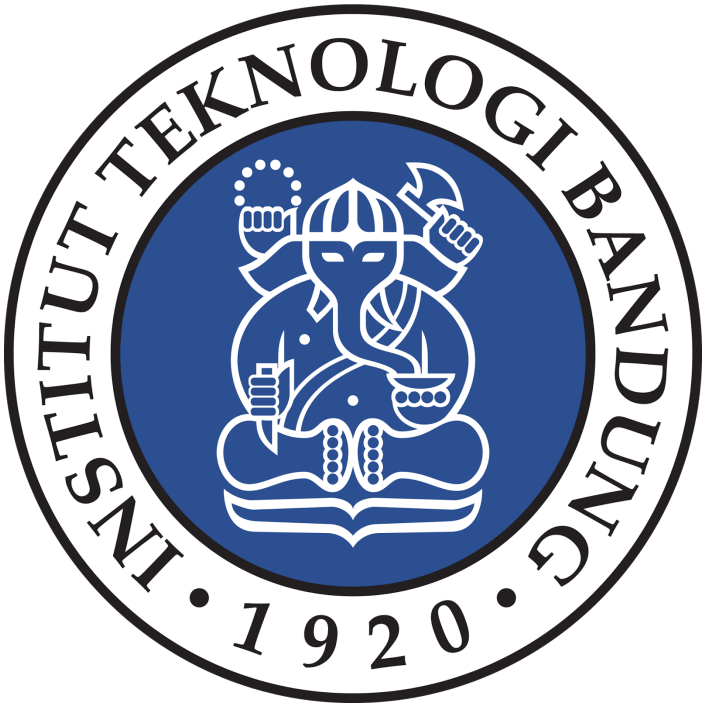
Laporan Tugas Besar IF3270 Pembelajaran Mesin
Bagian A: Implementasi Forward Propagation untuk Feed Forward Neural Network



Disusun oleh:

Jevant Jedidia	13520133
Vincent Christian	13520136
David Karel	13520154
Willy Wilsen	13520160

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Bab I
Implementasi

Program

Pada program model FFNN blok 1, terdapat fungsi-fungsi aktivasi yang diperlukan yaitu linear, reLU, sigmoid, dan softmax. Setiap fungsi aktivasi ini akan menerima parameter berupa nilai net dan mengembalikan output yang dihitung berdasarkan fungsi aktivasi tersebut. Pada blok ini juga didefinisikan fungsi CountSSE untuk menghitung error dari ekspektasi output.

Pada program model FFNN blok 2, dilakukan pembacaan file model FFNN dalam format .json yang akan dibangun. Kemudian, model tersebut akan dituangkan ke dalam variabel-variabel yang diperlukan untuk melakukan Forward Propagation. Variabel-variabel ini berupa inp, activation, neuron, weight, eoutput, dan sse. Setelah itu, pengguna akan memasukkan input berupa jumlah instance dan nilai dari setiap instance tersebut, kemudian dimasukkan pada variabel inp bertipe array 2D untuk diproses selanjutnya.

Pada program model FFNN blok 3, dilakukan Forward Propagation pada input instance yang telah dilakukan. Untuk setiap layer, akan dihitung nilai net pada setiap neuron dari layer tersebut. Nilai net ini dihitung berdasarkan fungsi aktivasi yang dimasukkan pada layer tersebut. Proses ini dilakukan hingga didapatkan output untuk model FFNN yang telah diinput. Di akhir, dilakukan klasifikasi untuk menentukan kelas dari output tersebut.

Pada program model FFNN blok 4, dilakukan penggambaran model FFNN yang telah diinput. Yang ditampilkan adalah ANN dalam bentuk *compact*, bobot dari sebuah neuron ke neuron lainnya, serta fungsi aktivasi dari sebuah neuron ke neuron lainnya. Fungsi aktivasi dari sebuah neuron ke neuron lainnya ditulis dalam format layerX -> layerY = fungsiAktivasi sedangkan bobot dituliskan dalam format neuronX -> neuronY = bobot. Sebuah neuron dituliskan dalam format layer(neuron) sehingga untuk neuron pertama dari hidden layer pertama, formatnya adalah h1(1).

Pada program model FFNN blok 5, dibuat sebuah visualisasi model FFNN berbentuk graph apabila model yang dibuat pada blok 5 kurang jelas. Arah dari graph adalah dari atas ke bawah.

Teks Input

Penjelasan berikut dikutip dari README yang diberikan asisten:

```
Ada dua bagian: `case` dan `expect`.

Isi dari `case`:

* `model.input_size` (sudah jelas)

* `model.layers`, terdiri dari daftar objek beratribut berikut:

    * `number_of_neurons` (sudah jelas)
```

        * `activation_function` (nilai valid: `linear`, `relu`, `sigmoid`, `softmax`)
* `input`:
    * Dimensi 1 menandakan vektor ke-i. Ukurannya _*arbitrary*_.
        * Dimensi 2 menandakan isi suatu vektor. Ukurannya sesuai dengan `model.input_size`
* `weights`:
    * Dimensi 1 bersesuaian dengan _*layer*_ di `layers`.
    * Dimensi 2 berukuran banyak neuron pada lapisan sebelumnya + 1.
        * Khusus untuk _*layer*_ pertama: `model.input_size` + 1
        * Baris pertama adalah bias.
    * Dimensi 3 berukuran banyak _*neuron*_ pada lapisan yang bersesuaian.

Isi dari `expect`:
* `output`:
    * Ukuran dimensi 1 sesuai dengan ukuran dimensi 1 dari `input`.
    * Ukuran dimensi 2 sesuai dengan banyak _*neuron*_ pada lapisan terakhir.
* `max_sse`, nilai _*sum of squares error*_ yang dapat ditoleransi.

Bab II
Pengujian

1 Instance

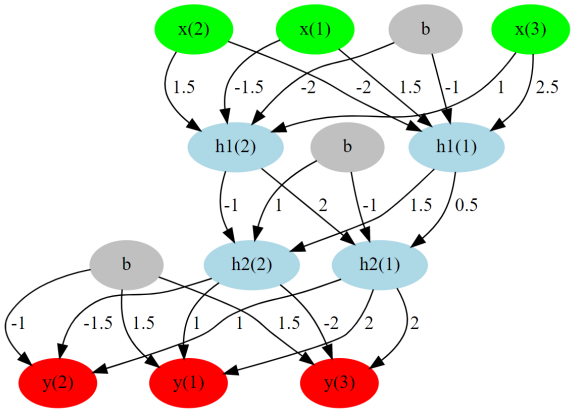
Akan digunakan **model.json** untuk dilakukan pengujian.

Model

```
x(3) --> h1(2) --> h2(2) --> y(3)

Weight:
        Input layer:
        * b -> h1(1) = -1
        * b -> h1(2) = -2
        * x(1) -> h1(1) = 1.5
        * x(1) -> h1(2) = -1.5
        * x(2) -> h1(1) = -2
        * x(2) -> h1(2) = 1.5
        * x(3) -> h1(1) = 2.5
        * x(3) -> h1(2) = 1

        Hidden Layer:
        h1
        * b -> h2(1) = -1
        * b -> h2(2) = 1
        * h1(1) -> h2(1) = 0.5
        * h1(1) -> h2(2) = 1.5
        * h1(2) -> h2(1) = 2
        * h1(2) -> h2(2) = -1
        h2
        * b -> y(1) = 1.5
        * b -> y(2) = -1
        * b -> y(3) = 1.5
        * h2(1) -> y(1) = 2
        * h2(1) -> y(2) = 1
        * h2(1) -> y(3) = 2
        * h2(2) -> y(1) = 1
        * h2(2) -> y(2) = -1.5
        * h2(2) -> y(3) = -2
```



```
Activation function:
x --> h1 = linear
h1 --> h2 = sigmoid
h2 --> y = softmax
```

Input

<1,0,1>

Output

```
Instance: [1.0, 0.0, 1.0]
Output: [0.9465151777162716, 0.006313173227643057, 0.04717164905608533]
Expected output: [0.946482, 0.006316, 0.0472008]
sse: 1.958529030069099e-09
max_sse: 1e-06
True
```

Batch instances

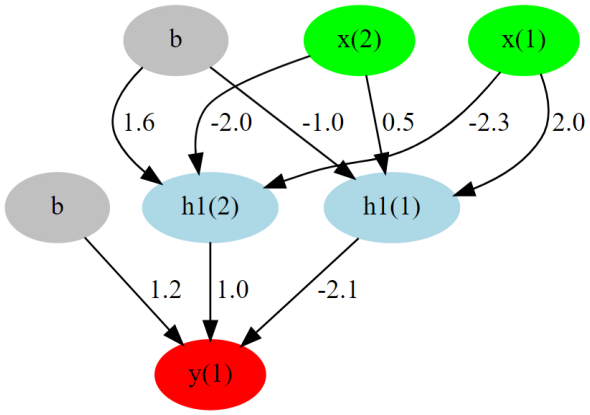
Akan digunakan **model2.json** untuk dilakukan pengujian.

Model

```
x(2) --> h1(2) --> y(1)

Weight:
        Input layer:
        * b -> h1(1) = -1.0
        * b -> h1(2) = 1.6
        * x(1) -> h1(1) = 2.0
        * x(1) -> h1(2) = -2.3
        * x(2) -> h1(1) = 0.5
        * x(2) -> h1(2) = -2.0

        Hidden Layer:
        h1
        * b -> y(1) = 1.2
        * h1(1) -> y(1) = -2.1
        * h1(2) -> y(1) = 1.0
```



Activation function: **x --> h1 = softmax** **h1 --> y = sigmoid**	

Input

[<-2,1>, <3,2>]

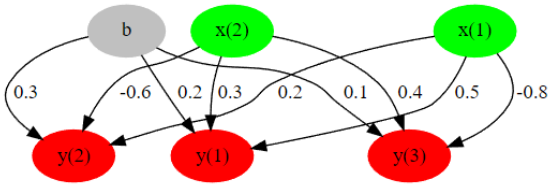
Output

```
Instance: [-2.0, 1.0]
Output: [0.9002031346932379]
Expected output: [0.900232]
sse: 8.332059344736148e-10
max_sse: 1e-06
True

Instance: [3.0, 2.0]
Output: [0.2890506417422056]
Expected output: [0.28905]
sse: 4.118330585172976e-13
max_sse: 1e-06
True
```

Test Case linear.json

Model:

x(2) --> y(3) Weight: Input layer: * b -> y(1) = 0.2 * b -> y(2) = 0.3 * b -> y(3) = 0.1 * x(1) -> y(1) = 0.5 * x(1) -> y(2) = 0.2 * x(1) -> y(3) = -0.8 * x(2) -> y(1) = 0.3 * x(2) -> y(2) = -0.6 * x(2) -> y(3) = 0.4 Activation function: x --> y = linear	

Input:
<3.0,1.0>
Hasil:

```
Instance: [3.0, 1.0]
Output: [2.0, 0.30000000000000016, -1.9000000000000004]
Expected output: [2.0, 0.3, -1.9]
sse: 2.2494861750442915e-31
max_sse: 1e-06
True
```

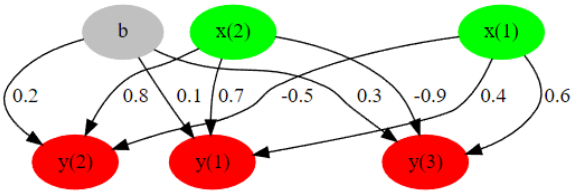
Test Case relu.json

Model:

```
x(2) --> y(3)

Weight:
        Input layer:
        * b -> y(1) = 0.1
        * b -> y(2) = 0.2
        * b -> y(3) = 0.3
        * x(1) -> y(1) = 0.4
        * x(1) -> y(2) = -0.5
        * x(1) -> y(3) = 0.6
        * x(2) -> y(1) = 0.7
        * x(2) -> y(2) = 0.8
        * x(2) -> y(3) = -0.9

Activation function:
x --> y = relu
```



Input:
<-1.0,0.5>
Hasil:

```
Instance: [-1.0, 0.5]
Output: [0.04999999999999993, 1.1, 0]
Expected output: [0.05, 1.1, 0.0]
sse: 4.8148248609680896e-33
max_sse: 1e-06
True
```

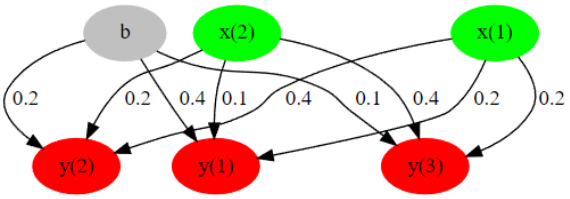
Test Case sigmoid.json

Model:

```
x(2) --> y(3)

Weight:
        Input layer:
        * b -> y(1) = 0.4
        * b -> y(2) = 0.2
        * b -> y(3) = 0.1
        * x(1) -> y(1) = 0.2
        * x(1) -> y(2) = 0.4
        * x(1) -> y(3) = 0.2
        * x(2) -> y(1) = 0.1
        * x(2) -> y(2) = 0.2
        * x(2) -> y(3) = 0.4

Activation function:
x --> y = sigmoid
```



Input:
<0.2,0.4>
Hasil:

```
Instance: [0.2, 0.4]
Output: [0.617747874769249, 0.5890404340586651, 0.574442516811659]
Expected output: [0.617747, 0.58904, 0.574442]
sse: 1.2207224545374987e-12
max_sse: 1e-06
True
```

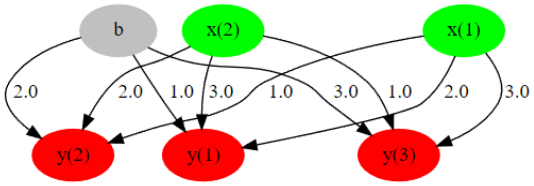
Test Case softmax.json

Model:

```
x(2) --> y(3)

Weight:
        Input layer:
        * b -> y(1) = 1.0
        * b -> y(2) = 2.0
        * b -> y(3) = 3.0
        * x(1) -> y(1) = 2.0
        * x(1) -> y(2) = 1.0
        * x(1) -> y(3) = 3.0
        * x(2) -> y(1) = 3.0
        * x(2) -> y(2) = 2.0
        * x(2) -> y(3) = 1.0

Activation function:
x --> y = softmax
```



Input:
<1.0,2.0>
Hasil:

```
Instance: [1.0, 2.0]
Output: [0.6652409557748219, 0.09003057317038045, 0.24472847105479764]
Expected output: [0.665241, 0.090031, 0.244728]
sse: 4.0603201288236806e-13
max_sse: 1e-06
True
```

Test Case multilayer.json

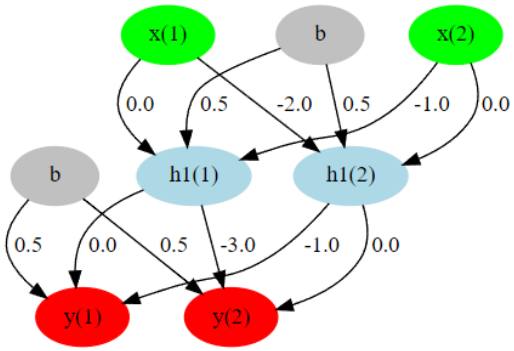
Model:

```
x(2) --> h1(2) --> y(2)

Weight:
        Input layer:
        * b -> h1(1) = 0.5
        * b -> h1(2) = 0.5
        * x(1) -> h1(1) = 0.0
        * x(1) -> h1(2) = -2.0
        * x(2) -> h1(1) = -1.0
        * x(2) -> h1(2) = 0.0

        Hidden Layer:
        h1
        * b -> y(1) = 0.5
        * b -> y(2) = 0.5
        * h1(1) -> y(1) = 0.0
        * h1(1) -> y(2) = -3.0
        * h1(2) -> y(1) = -1.0
        * h1(2) -> y(2) = 0.0

Activation function:
x --> h1 = linear
h1 --> y = relu
```



Input:
<1.0,0.0>,
<0.0,0.1>,
<0.0,0.0>

Hasil:

```
Instance: [1.0, 0.0]
Output: [2.0, 0]
Expected output: [2.0, 0.0]
sse: 0.0
max_sse: 1e-06
True

Instance: [0.0, 1.0]
Output: [0, 2.0]
Expected output: [0.0, 2.0]
sse: 0.0
max_sse: 1e-06
True

Instance: [0.0, 0.0]
Output: [0, 0]
Expected output: [0.0, 0.0]
sse: 0.0
max_sse: 1e-06
True
```

Bab III
Perhitungan Manual

Perhitungan untuk 1 instance

Instance = <1,0,1>

Hidden layer 1

h11 = (-1 * 1) + (1,5 * 1) + (-2 * 0)+(2,5 * 1)
= -1 + 1.5 + 0 + 2.5
= 3

h12 = (-20 * 1) + (-15 * 1) + (15 * 0) + (10 * 1)
= -2 -1.5 + 0 + 1
= -2.5

Hidden Layer 2

h21 = σ((-1 * 1) + (0.5 * 3) + (2 * -2,5))
= σ(-4.5)
= 1/(1+e^(-4.5))
= 0.01098694263

h22 = σ((1 * 1) + (1.5 * 3) + (-1 * -2))
= σ(7.5)
= (1/1+e^-7.5)
= 0.9994472214

Output

net y1 = (1.5 * 1) + (2 * 0.01098694263) + (1 * 0.9994472214)
= 2.521421107

net y2 = (-1 * 1) + (1 * 0.01098694263) + (-1.5 * 0.9994472214)
= -2.488183889

net y3 = (1.5 * 1) + (2 * 0.01098694263) + (-2 * 0.9994472214)
= -0.4769205575

y1 = e^(2.521421107)/((e^ -2.488183889)+(e^-0.4769205575)+(e^2.521421107)
= 0.9464827967

y2 = e^(-2.488183889)/((e^ -2.488183889)+(e^-0.4769205575)+(e^2.521421107)
= 0.006316389724

y3 = e^(e^-0.4769205575)/((e^ -2.488183889)+(e^-0.4769205575)+(e^2.521421107)
= 0.04720081362

Perhitungan Batch Instance

a. Instance = <-2,1>

Hidden layer 1

net h11= (-2.0 * 1) + (2.0 * -2) + (0.5 * 1)
= -5.5
net h12= (1.6 * 1) + (-2.3 * -2) + (-2 * 1)
= 4.2
h11 = e^(-5.5)/(e^(4.2)+e^(-5.5))
= 6.12797396 * 10^(-5)
h12 = e^(4.2)/(e^(4.2)+e^(-5.5))
= 0.9999387203

Output layer

y1 = σ((1.2 * 1)+(-2.1 * 1.66558065 * 10^(-4))+ (1 * 0.9998334419))
= σ(2.199810033)
= 1/(1+e^-2.19948367)
=0.90023245048

b. Instance = <3,2>

Hidden layer 1

net h11= (-2.0 * 1) + (2.0 * 3) + (0.5 * 2)
= 5
net h12= (1.6 * 1) + (-2.3 * 3) + (-2 * 2)
= -9,3
h11 = e^(5)/e^(-9.3)+e^(5)
= 0.9999993
h12 = e^(-9.3)/e^(-9.3)+e^(5)
= 6.16011247 * 10^-7

Output layer

y1 = σ((1.2 * 1)+(-2.1 * 0.999999384)+ (1 * 6.16011247 * 10^-7))
= σ(-0.8999980904)
= 1/(1+e^0.8999980904)
= 0.2890508898

Perhitungan Test Case linear.json

Instance = <3,1>

Output layer

y1 = (1*0.2) + (3*0.5) + (1* 0.3)

= 2
y2 = (1*0.3) + (3*0.2) + (1* -0.6)
= 0.3
y3 = (1*0.1) + (3*-0.8) + (1* 0.4)
= -1.9

Perhitungan Test Case relu.json

Instance = <-1,0.5>
Output layer
net y1 = (1*0.1) + (-1*0.4) + (0.5* 0.7)
= -0.05
net y2 = (1*0.2) + (-1*-0.5) + (0.5* 0.8)
= 1.1
net y3 = (1*0.3) + (-1*0.6) + (0.5* -0.9)
= -0.75
y1 = -0.05
y2 = 1.1
y3 = 0

Perhitungan Test Case sigmoid.json

Instance = <0.2,0.4>
Output layer
net y1 = (1*0.4) + (0.2*0.2) + (0.4* 0.1)
= 0.48
net y2 = (1*0.2) + (0.2*0.4) + (0.4* 0.2)
= 0.36
net y3 = (1*0.1) + (0.2*0.2) + (0.4* 0.4)
= 0.3
y1 = 1/(1+e^0.48) = 0.6177478748
y2 = 1/(1+e^0.36) = 0.5890404341
y3 = 1/(1+e^0.3) = 0.5744425168

Perhitungan Test Case softmax.json

Instance = <1,2>
Output layer
net y1 = (1*1) + (1*2) + (2* 3)
= 9
net y2 = (1*2) + (1*1) + (2* 2)

= 7

net y3 = (1*3) + (1*3) + (3* 1)

= 8

y1 = e^9/(e^9+e^8+e^7) = 0.6652409558

y2 = e^7/(e^9+e^8+e^7) = 0.09003057317

y3 = e^8/(e^9+e^8+e^7) = 0.2447284711

Perhitungan Test Case Linear.json

a. Instance = <1,0>

Hidden layer 1

h11= (1*0.5) + (1*0) + (0* -1)

= 0.5

h12 = (1*0.5) + (1*-2) + (0* 0)

= -1.5

Output layer

net y1 = (1*0.5) + (0.5*0) + (-1.5* -1)

= 2.0

net y2 = (1*0.5) + (0.5*-3) + (-1.5* 0)

= -1.0

y1 = 2.0

y2 = 0.0

b. Instance = <0,1>

Hidden layer 1

h11= (1*0.5) + (0*0) + (1* -1)

= -0.5

h12 = (1*0.5) + (0*-2) + (1* 0)

= 0.5

Output layer

net y1 = (1*0.5) + (-0.5*0) + (0.5* -1)

= 0.0

net y2 = (1*0.5) + (-0.5*-3) + (0.5* 0)

= 2.0

y1 = 0.0

y2 = 2.0

c. Instance = < 0,0>

Hidden layer 1

h11= (1*0.5) + (0*0) + (0* -1)

= 0.5

h12 = (1*0.5) + (0*-2) + (0* 0)
= 0.5

Output layer

net y1 = (1*0.5) + (0.5*0) + (0.5* -1)
= 0.0

net y2 = (1*0.5) + (0.5*-3) + (0.5* 0)
= -1.0

y1 = 0.0

y2 = 0.0

Bab IV
Pembagian Tugas

NIM	Nama	Tugas
13520133	Jevant Jedidia	Program FFNN, Laporan Implementasi Program, Visualisasi FFNN non-Graf
13520136	Vincent Christian	Visualisasi Graf, Perhitungan Manual
13520154	David Karel	Program FFNN, Perhitungan Manual
13520160	Willy Wilsen	Program FFNN, Laporan Implementasi Program