

Laporan Tugas Besar A IF3270 Pembelajaran Mesin

Implementasi Forward Propagation untuk Feed Forward Neural Network



Disusun oleh:

K01 13520010 - Ken Kalang Al Qalyubi

K01 13520036 - I Gede Arya Raditya Parameswara

K02 13520061 - Gibran Darmawan

K03 13520119 - Marchotridyo

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha No. 10, Bandung 40132

2023

Implementasi Program

Program ini mengimplementasikan *forward propagation* untuk Feed Forward Neural Network (FFNN). Secara singkat, Forward propagation adalah proses pengiriman data input melalui jaringan saraf tiruan (neural network) feedforward (FFNN) dari lapisan input ke lapisan output. Setiap neuron pada lapisan input menerima nilai input, kemudian menghitung nilai keluarannya dengan menggunakan fungsi aktivasi. Nilai keluaran ini kemudian diteruskan sebagai input ke setiap neuron pada lapisan berikutnya, dan proses penghitungan nilai keluaran diulang sampai mencapai lapisan output.

Pada program ini, model FFNN dibuat dan disimpan dalam sebuah *file* tipe JSON. *File* tersebut berisikan layer input, weight setiap hidden layer, fungsi aktivasi, beserta layer output. Pada setiap layer juga diberikan informasi jumlah neuronnya. Isi dari file tersebut kemudian dibaca oleh program untuk membentuk kelas pada program. Kelas-kelas tersebut adalah Layer, FileUtility, Neuron, dan ANNGraph. Kelas Layer adalah kelas yang menyerupai *layer* pada FFNN dan memiliki atribut neuron, tipe layer, dan juga fungsi aktivasi yang berada pada layer tersebut. Kelas Neuron merupakan kelas yang merepresentasikan *neuron* pada FFNN dan memiliki atribut *layer* yang merupakan *layer* yang ditempati oleh *neuron*, *weight* yang merupakan bobot neuron tersebut, dan bias *neuron* tersebut. Kelas *neuron* dapat diaktifkan sesuai dengan jenis fungsi aktivasi yang ditentukan untuk *neuron* tersebut. Menjalankan kode utama forward propagation disimpan dalam kelas ANNGraph. Kelas ini memiliki atribut *file_path* untuk menyimpan alamat file, *layers* yang berisi layer pada model FFNN, dan atribut untuk membangun graph ANN. Terdapat fungsi-fungsi untuk membangun *graph* ANN, *predict* untuk menghasilkan output dari input yang diberikan pada *file* JSON, mengaktivasi neuron, menghitung *net* neuron, dan menampilkan hasil dari *predict* menggunakan fungsi *print_details*.

Langkah-langkah program berjalan secara umum adalah memanggil konstruktor kelas ANNGraph, konstruktor tersebut memanggil method *build_ann_graph*. Method tersebut mengambil file JSON model ANN untuk membangun layer-layer serta isinya untuk membangun Model ANN. Setelah itu, program memanggil method *draw_ann_graph* untuk menampilkan model yang dijalankan. Untuk mengerjakan *test case* sesuai dengan input JSON yang diberikan, method *solve* perlu dipanggil.

Hasil Eksekusi

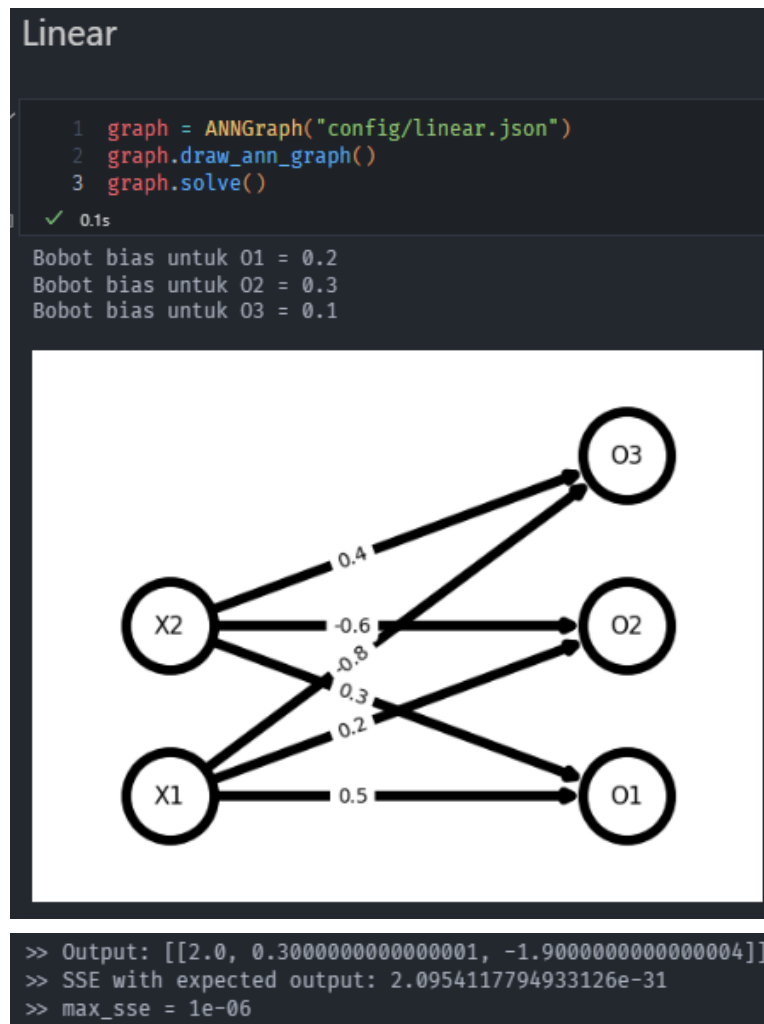
Hasil eksekusi dilakukan terhadap *test case* yang sudah diberikan oleh asisten.

Kasus 1: Satu layer, linear

Kasus yang diberikan

```
{
  "case": {
    "model": {
      "input_size": 2,
      "layers": [
        {
          "number_of_neurons": 3,
          "activation_function": "linear"
        }
      ]
    },
    "input": [
      3.0, 1.0
    ],
    "weights": [
      [
        0.2, 0.3, 0.1,
        0.5, 0.2, -0.8,
        0.3, -0.6, 0.4
      ]
    ]
  },
  "expect": {
    "output": [
      2.0, 0.3, -1.9
    ],
    "max_sse": 0.000001
  }
}
```

Hasil eksekusi program



Nilai $SSE < max_sse$, perhitungan kami berhasil memenuhi *test case*.

Perbandingan dengan perhitungan manual

Apabila dihitung secara manual:

- Net pada O1 = $(0.2)(1) + (0.5)(3) + (0.3)(1) = 2$
- Net pada O2 = $(0.3)(1) + (0.2)(3) + (-0.6)(1) = 0.3$
- Net pada O3 = $(0.1)(1) + (-0.8)(3) + (0.4)(1) = -1.9$

Karena fungsi aktivasinya linear, $value = net$.

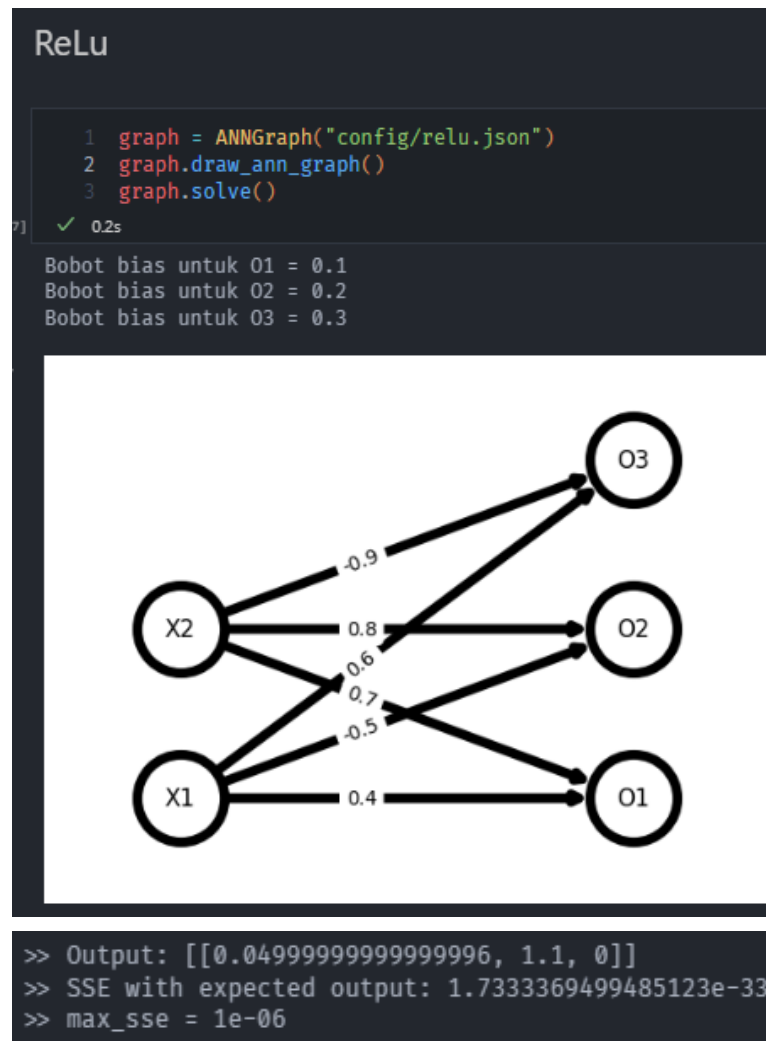
Perhatikan bahwa output hasil perhitungan manual, $[[2, 0.3, -1.9]]$ sangat dekat dengan hasil perhitungan kami yaitu $[[2.0, 0.30000000000000001, -1.9000000000000004]]$. Dengan nilai SSE di sekitar $2e-31$, hasil perhitungan kami berhasil memenuhi *test case*.

Kasus 2: Satu layer, ReLu

Kasus yang digunakan

```
{
  "case": {
    "model": {
      "input_size": 2,
      "layers": [
        {
          "number_of_neurons": 3,
          "activation_function": "relu"
        }
      ]
    },
    "input": [[-1.0, 0.5]],
    "weights": [
      [
        [0.1, 0.2, 0.3],
        [0.4, -0.5, 0.6],
        [0.7, 0.8, -0.9]
      ]
    ]
  },
  "expect": {
    "output": [[0.05, 1.1, 0.0]],
    "max_sse": 0.000001
  }
}
```

Hasil eksekusi program



Nilai $SSE < max_sse$, perhitungan kami berhasil memenuhi *test case*.

Perbandingan dengan perhitungan manual

Apabila dihitung secara manual:

- Net pada O1 = $(0.1)(1) + (0.4)(-1) + (0.7)(0.5) = 0.05$
 - $ReLU(0.05) = 0.05$
- Net pada O2 = $(0.2)(1) + (-0.5)(-1) + (0.8)(0.5) = 1.1$
 - $ReLU(1.1) = 1.1$
- Net pada O3 = $(0.3)(1) + (0.8)(-1) + (-0.9)(0.5) = -0.95$
 - $ReLU(-0.95) = 0$

Perhatikan bahwa output hasil perhitungan manual, $[[0.05, 1.1, 0]]$ sangat dekat dengan hasil perhitungan kami yaitu $[[0.049999999999999996, 1.1, 0]]$. Dengan nilai SSE di sekitar $1.73e-33$, hasil perhitungan kami berhasil memenuhi *test case*.

Kasus 3: Satu layer, Sigmoid

Kasus yang digunakan

```
{
  "case": {
    "model": {
      "input_size": 2,
      "layers": [
        {
          "number_of_neurons": 3,
          "activation_function": "sigmoid"
        }
      ]
    },
    "input": [[0.2, 0.4]],
    "weights": [
      [
        [0.4, 0.2, 0.1],
        [0.2, 0.4, 0.2],
        [0.1, 0.2, 0.4]
      ]
    ]
  },
  "expect": {
    "output": [[0.617747, 0.589040, 0.574442]],
    "max_sse": 0.000001
  }
}
```

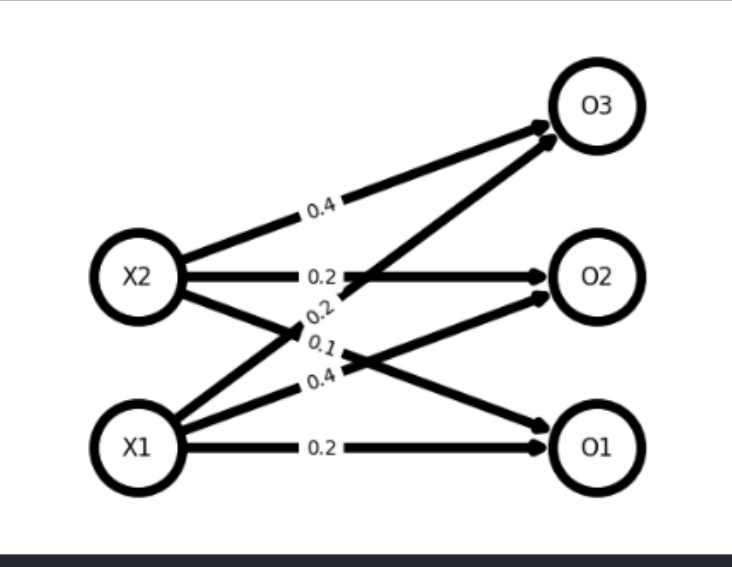
Hasil eksekusi program

```
Sigmoid

1 graph = ANNGraph("config/sigmoid.json")
2 graph.draw_ann_graph()
3 graph.solve()

✓ 0.1s

Bobot bias untuk O1 = 0.4
Bobot bias untuk O2 = 0.2
Bobot bias untuk O3 = 0.1
```



```
>> Output: [[0.617747874769249, 0.5890404340586651, 0.574442516811659]]
>> SSE with expected output: 1.2207224545374987e-12
>> max_sse = 1e-06
```

The diagram shows a neural network with two input nodes, X1 and X2, and three output nodes, O1, O2, and O3. The connections and their weights are as follows: X1 to O1 (0.2), X1 to O2 (0.4), X1 to O3 (0.1), X2 to O1 (0.2), X2 to O2 (0.2), and X2 to O3 (0.4).

Nilai SSE < max_sse, perhitungan kami berhasil memenuhi *test case*.

Perbandingan dengan perhitungan manual

Apabila dihitung secara manual:

- Net pada O1 = $(0.4)(1) + (0.2)(0.2) + (0.1)(0.4) = 0.48$
 - $\text{sigmoid}(0.48) = 0.617747$
- Net pada O2 = $(0.2)(1) + (0.4)(0.2) + (0.2)(0.4) = 0.36$
 - $\text{sigmoid}(0.36) = 0.589040$
- Net pada O3 = $(0.1)(1) + (0.2)(0.2) + (0.4)(0.4) = 0.3$
 - $\text{sigmoid}(0.3) = 0.0574442$

Perhatikan bahwa output hasil perhitungan manual, $[[0.617747, 0.589040, 0.574442]]$ sangat dekat dengan hasil perhitungan kami yaitu $[[0.617747874769249, 0.5890404340586651, 0.574442516811659]]$. Dengan nilai SSE di sekitar $1.22e-12$, hasil perhitungan kami berhasil memenuhi *test case*.

Kasus 4: Satu layer, Softmax

Kasus yang digunakan

```
{
  "case": {
    "model": {
      "input_size": 2,
      "layers": [
        {
          "number_of_neurons": 3,
          "activation_function": "softmax"
        }
      ]
    },
    "input": [[1.0, 2.0]],
    "weights": [
      [
        [1.0, 2.0, 3.0],
        [2.0, 1.0, 3.0],
        [3.0, 2.0, 1.0]
      ]
    ]
  },
  "expect": {
    "output": [[0.665241, 0.090031, 0.244728]],
    "max_sse": 0.000001
  }
}
```

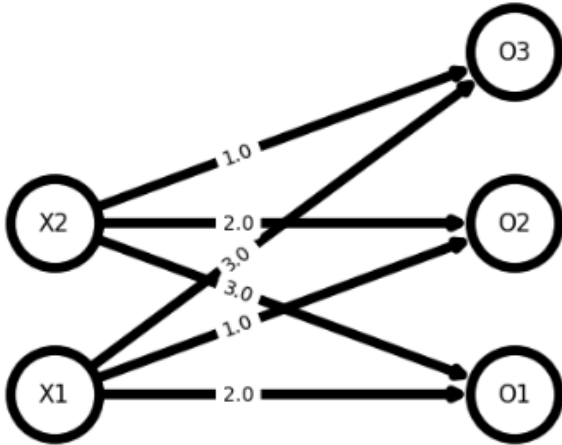
Hasil eksekusi program

```
Softmax

1 graph = ANNGraph("config/softmax.json")
2 graph.draw_ann_graph()
3 graph.solve()

✓ 0.2s

Bobot bias untuk O1 = 1.0
Bobot bias untuk O2 = 2.0
Bobot bias untuk O3 = 3.0
```



```
>> Output: [[0.6652409557748219, 0.09003057317038045, 0.24472847105479764]]
>> SSE with expected output: 4.0603201288236806e-13
>> max_sse = 1e-06
```

Nilai SSE < max_sse, perhitungan kami berhasil memenuhi *test case*.

Perbandingan dengan perhitungan manual

Apabila dihitung secara manual:

- Net pada O1 = $(1)(1) + (2)(1) + (3)(2) = 9$
- Net pada O2 = $(2)(1) + (1)(1) + (2)(2) = 7$
- Net pada O3 = $(3)(1) + (3)(1) + (1)(2) = 8$

Lalu, perhatikan bahwa $\sum(e^{\text{net}}) = e^9 + e^7 + e^8$.

- Pada O1, hasil akhirnya adalah $e^9 / \sum(e^{\text{net}}) = 0.665241$
- Pada O2, hasil akhirnya adalah $e^7 / \sum(e^{\text{net}}) = 0.090031$
- Pada O3, hasil akhirnya adalah $e^8 / \sum(e^{\text{net}}) = 0.244728$

Perhatikan bahwa output hasil perhitungan manual, $[[0.665241, 0.090031, 0.244728]]$ sangat dekat dengan hasil perhitungan kami yaitu $[[0.6652409557748219, 0.09003057317038045, 0.24472847105479764]]$. Dengan nilai SSE di sekitar $4.06e-13$, hasil perhitungan kami berhasil memenuhi *test case*.

Kasus 5: Dua layer, Linear dan ReLU, batch input

Kasus yang digunakan

```
{
  "case": {
    "model": {
      "input_size": 2,
      "layers": [
        {
          "number_of_neurons": 2,
          "activation_function": "linear"
        },
        {
          "number_of_neurons": 2,
          "activation_function": "relu"
        }
      ]
    },
    "input": [
      [1.0, 0.0],
      [0.0, 1.0],
      [0.0, 0.0]
    ],
    "weights": [
      [
        [0.5, 0.5],
        [0.0, -2.0],
        [-1.0, 0.0]
      ],
      [
        [0.5, 0.5],
        [0.0, -3.0],
        [-1.0, 0.0]
      ]
    ],
    "expect": {
      "output": [
        [2.0, 0.0],
        [0.0, 2.0],
        [0.0, 0.0]
      ],
      "max_sse": 0.000001
    }
  }
}
```

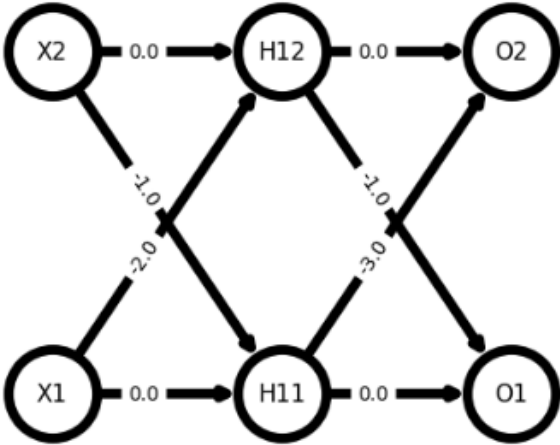
Hasil eksekusi program

```
Multilayer

1 graph = ANNGraph("config/multilayer.json")
2 graph.draw_ann_graph()
3 graph.solve()

✓ 0.1s

Bobot bias untuk H11 = 0.5
Bobot bias untuk H12 = 0.5
Bobot bias untuk O1 = 0.5
Bobot bias untuk O2 = 0.5
```



```
>> Output: [[2.0, 0], [0, 2.0], [0, 0]]
>> SSE with expected output: 0.0
>> max_sse = 1e-06
```

Nilai $SSE < max_sse$, perhitungan kami berhasil memenuhi *test case*.

Perbandingan dengan perhitungan manual

Perhitungan di layer hidden pertama (linear)

Matriks input:

$$x = \begin{bmatrix} 1 & 1.0 & 0.0 \\ 1 & 0.0 & 1.0 \\ 1 & 0.0 & 0.0 \end{bmatrix}$$

Matriks bobot:

$$w = \begin{bmatrix} 0.5 & 0.5 \\ 0.0 & -2.0 \\ -1.0 & 0.0 \end{bmatrix}$$

Matriks xw :

$$xw = \begin{bmatrix} \frac{1}{2} & -\frac{3}{2} \\ -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Karena linear, diteruskan sebagai input di layer kedua.

Perhitungan di layer output

Matriks input sama dengan xw di layer sebelumnya, hanya saja ditambahkan bias di kolom pertama:

$$x = \begin{bmatrix} 1 & \frac{1}{2} & -\frac{3}{2} \\ 1 & -\frac{1}{2} & \frac{1}{2} \\ 1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Matriks bobot:

$$w = \begin{bmatrix} 0.5 & 0.5 \\ 0.0 & -3.0 \\ -1.0 & 0.0 \end{bmatrix}$$

Matriks xw :

$$xw = \begin{bmatrix} 2 & -1 \\ 0 & 2 \\ 0 & -1 \end{bmatrix}$$

Berikan fungsi ReLU ke setiap anggota xw :

$$ReLU(xw) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}$$

Baris pertama dari $ReLU(xw)$, $[2, 0]$, merupakan output dari input pertama, $[1, 0]$. Begitu pula dengan baris kedua dan baris ketiga. Sehingga, didapatkan output dari pertama adalah "output": $[[2.0, 0.0], [0.0, 2.0], [0.0, 0.0]]$. Hasil perhitungan ini sama persis dengan output yang program kami hasilkan, yaitu $[[2.0, 0], [0, 2.0], [0, 0]]$. Dengan $SSE = 0$, hasil perhitungan kami berhasil memenuhi *test caase*.

Pembagian Tugas

NIM	Nama	Bagian Tugas
13520010	Ken Kalang Al Qalyubi	Inisialisasi Repository, Fungsi Utilitas
13520036	I Gede Arya Raditya Parameswara	Insialisasi Repository, ANN Graph, Model Config, Predict
13520061	Gibran Darmawan	Inisialisasi laporan
1320119	Marchotridiyo	Refactor kelas neuron dan penambahan kelas layer dari inisialisasi, <i>fix</i> fungsi <i>softmax</i> , visualisasi gambar ANN, pencocokan format input/output program dengan <i>test case</i> asisten, analisis SSE

Tautan Penting

[PPT Machine Learning Teknik Informatika ITB Feed Forward Neural](#)