Implementasi *Forward Propagation* untuk *Feed Forward Neural Network*

Tugas Besar 1 Bagian A IF3270 Pembelajaran Mesin

Kelas 2 dan 3

Primanda Adyatma Hafiz	13520022
Vincent Prasetiya Atmadja	13520099
Gerald Abraham Sianturi	13520138
Daffa Romyz Aufa	13520162

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

TEKNIK INFORMATIKA

2023

Penjelasan Implementasi

No	Bagian	Penjelasan
1	**Fungsi aktivasi**	Terdapat empat fungsi aktivasi yang diimplementasikan, yakni fungsi aktivasi linear, ReLU, sigmoid, dan softmax. Keempat fungsi aktivasi disimpan dalam *dictionary* agar fungsi aktivasi dapat dipanggil dengan *string identifier* yang ditentukan
2	**Kelas**	Secara umum, *design pattern* yang digunakan *composite*, dimana model ANN disusun dari beberapa *layer*, dan tiap layer tersusun atas beberapa *node*
	`Node`	Digunakan sebagai representasi neuron pada ANN, menyimpan properti - `id`, untuk identifikasi `Node` - `layerId`, untuk menyimpan *id* dari `Layer` yang berkorespondensi dengan `Node` - `weight`, - `bias`, - `activFunctionType`, Terdapat metode untuk menghitung *output* berdasarkan masukan dari *layer* sebelumnya.
	`Layer`	Digunakan sebagai representasi *layer* pada ANN, menyimpan properti - `id`, untuk identifikasi `Layer` - `nodeAmount` untuk menyimpan jumlah `Node` - `layerType` untuk menyimpan apakah `Layer` merupakan *input layer*, *hidden layer*, atau output layer, - `nodes` untuk menyimpan kumpulan `Node` pada layer. Terdapat metode untuk menambahkan *node* baru pada `nodes`
	`ANNModel`	Digunakan sebagai representasi model ANN, menyimpan properti `layers` untuk menyimpan kumpulan `Layer`yang menyusun model ANN. Terdapat metode untuk menghasilkan nilai dari *output layer*.
3	*File loader*	Digunakan untuk melakukan *setup* model ANN dengan struktur file .json yang telah terdefinisi sebelumnya.
4	Fungsi representasi model (gambar graf dan teks)	Memanfaatkan *library* eksternal `networkx` dan

		`matplotlib` untuk memvisualisasikan ANN dalam bentuk graf. Untuk merepresentasikan data dalam bentuk teks juga dapat digunakan metode `describe` pada kelas `ANNModel`

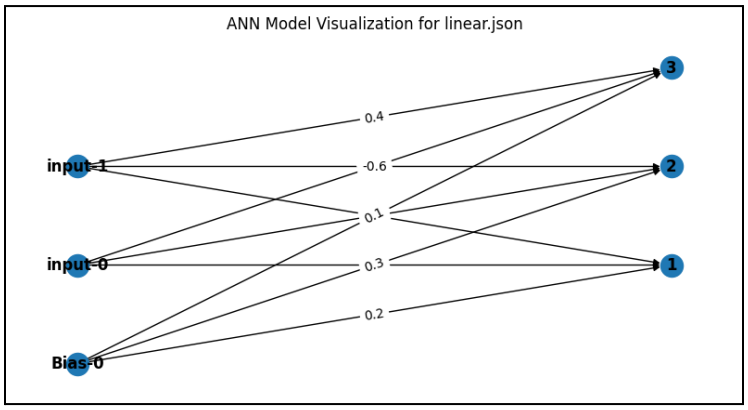
Hasil pengujian

1. linear.json

1. Model pada `linear.json`


```
{
    "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
    }
}
```


2. Representasi model
    a. Dalam bentuk graf



    b. Dalam bentuk teks

```
>>> annModel.describe()
Layer with id 1:
      Amount of node -> 3
      Layer type -> LayerType.Output
Node with id 1:
      Layer id -> 0
      Weight -> [0.5, 0.3]
      Bias -> 0.2
      Activation function -> linear
Node with id 2:
      Layer id -> 0
      Weight -> [0.2, -0.6]
      Bias -> 0.3
      Activation function -> linear
Node with id 3:
      Layer id -> 0
      Weight -> [-0.8, 0.4]
      Bias -> 0.1
      Activation function -> linear
```

3. Nilai dari *output layer*

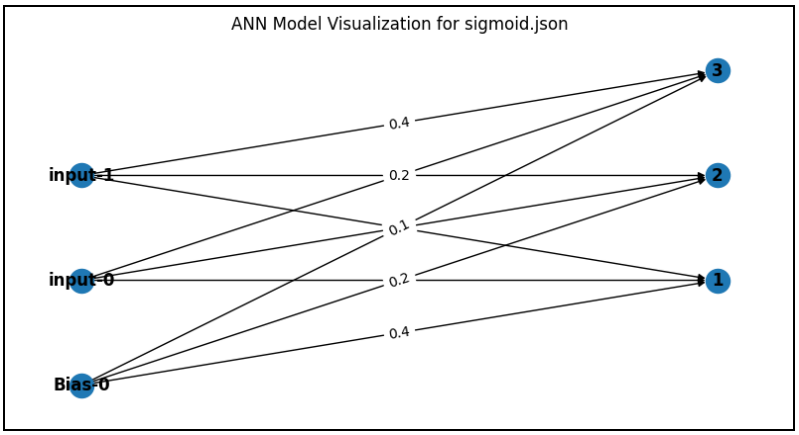
```
>>> annModel.predictMultiple(listInput)
[[2.0, 0.3, -1.9]]
```

2. sigmoid.json

1. Model pada `sigmoid.json`

```
{
    "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
    }
}
```

2. Representasi model
    a. Dalam bentuk graf



b. Dalam bentuk teks

```
Layer with id 1:
      Amount of node -> 3
      Layer type -> LayerType.Output
Node with id 1:
      Layer id -> 0
      Weight -> [0.2, 0.1]
      Bias -> 0.4
      Activation function -> sigmoid
Node with id 2:
      Layer id -> 0
      Weight -> [0.4, 0.2]
      Bias -> 0.2
      Activation function -> sigmoid
Node with id 3:
      Layer id -> 0
      Weight -> [0.2, 0.4]
      Bias -> 0.1
      Activation function -> sigmoid
```

3. Nilai dari *output layer*

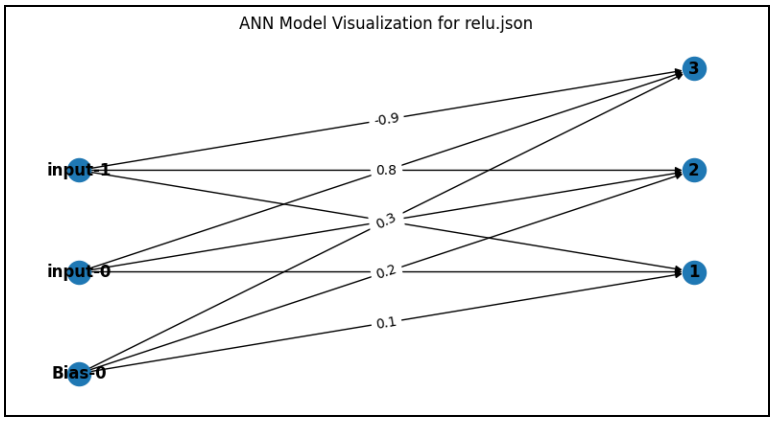
```
>>> annModel.predictMultiple(listInput)
[[0.62, 0.59, 0.57]]
```

3. relu.json

1. Model pada `relu.json`

```
{
    "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
    }
}
```

2. Representasi model
    a. Dalam bentuk graf



    b. Dalam bentuk teks

```
Layer with id 1:
      Amount of node -> 3
      Layer type -> LayerType.Output
Node with id 1:
      Layer id -> 0
      Weight -> [0.4, 0.7]
      Bias -> 0.1
      Activation function -> relu
Node with id 2:
      Layer id -> 0
      Weight -> [-0.5, 0.8]
      Bias -> 0.2
      Activation function -> relu
Node with id 3:
      Layer id -> 0
      Weight -> [0.6, -0.9]
      Bias -> 0.3
      Activation function -> relu
```

3. Nilai dari *output layer*

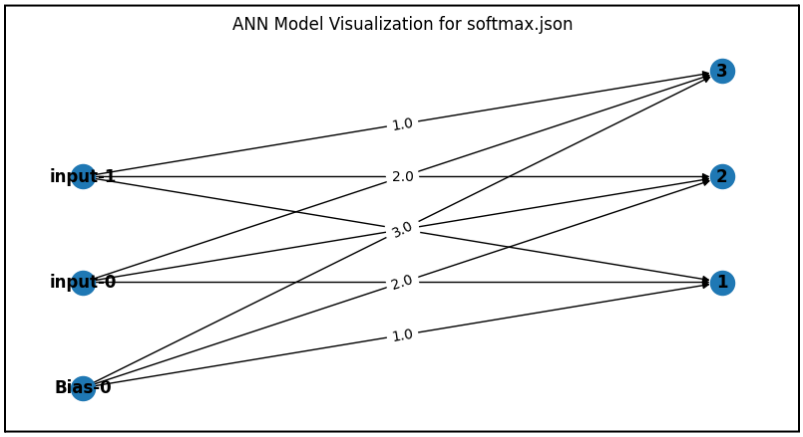
```
>>> annModel.predictMultiple(listInput)
[[0.05, 1.1, 0]]
```

4. softmax.json

1. Model pada `softmax.json`


```
{
    "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
    }
}
```


2. Representasi model
    a. Dalam bentuk graf



    b. Dalam bentuk teks

```
>>> annModel.describe()
Layer with id 1:
      Amount of node -> 3
      Layer type -> LayerType.Output
Node with id 1:
      Layer id -> 0
      Weight -> [2.0, 3.0]
      Bias -> 1.0
      Activation function -> softmax
Node with id 2:
      Layer id -> 0
      Weight -> [1.0, 2.0]
      Bias -> 2.0
      Activation function -> softmax
Node with id 3:
      Layer id -> 0
      Weight -> [3.0, 1.0]
      Bias -> 3.0
      Activation function -> softmax
```

3. Nilai dari *output layer*

```
>>> annModel.predictMultiple(listInput)
[[0.6652409103932707, 0.09003035136819242, 0.244728738238537]]
```

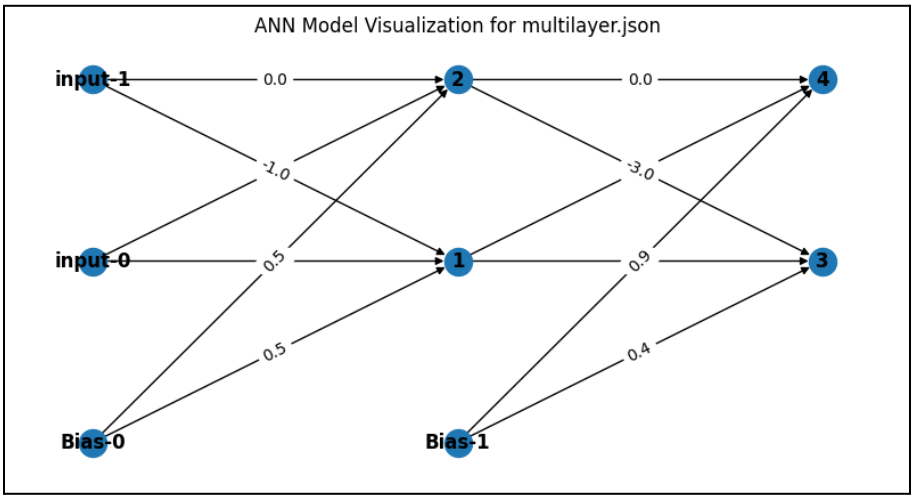
5. multilayer.json

1. Model pada `multilayer.json`

```
{
      "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]
            ],
            [
                  [0.4, 0.9],
                  [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
      }
}
```

2. Representasi model

a. Dalam bentuk graf



b. Dalam bentuk teks

```
>>> annModel.describe()
Layer with id 1:
      Amount of node -> 2
      Layer type -> LayerType.Hidden
Node with id 1:
      Layer id -> 0
      Weight -> [0.0, -1.0]
      Bias -> 0.5
      Activation function -> linear
Node with id 2:
      Layer id -> 0
      Weight -> [-2.0, 0.0]
      Bias -> 0.5
      Activation function -> linear


Layer with id 2:
      Amount of node -> 2
      Layer type -> LayerType.Output
Node with id 3:
      Layer id -> 1
      Weight -> [0.0, -1.0]
      Bias -> 0.4
      Activation function -> relu
Node with id 4:
      Layer id -> 1
      Weight -> [-3.0, 0.0]
      Bias -> 0.9
      Activation function -> relu
```

3. Nilai dari *output layer*

```
>>> annModel.predictMultiple(listInput)
[[1.9, 0], [0, 2.4], [0, 0]]
```

Perbandingan dengan perhitungan manual

1. linear.json

Input = [3.0, 1.0]

Layer 1 (Linear)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.2	0.2	2	2	4
2	3.0	0.5	1.5			
3	1.0	0.3	0.3			
1 (bias)	1.0	0.3	0.3	0.3	0.3	5
2	3.0	0.2	0.6			
3	1.0	-0.6	-0.6			
1 (bias)	1.0	0.1	0.1	-1.9	-1.9	6
2	3.0	-0.8	-2.4			
3	1.0	0.4	0.4			

2. sigmoid.json

Input = [0.2, 0.4]

Layer 1 (Sigmoid)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.4	0.4	0.48	0.617747	4
2	0.2	0.2	0.04			
3	0.4	0.1	0.04			
1 (bias)	1.0	0.2	0.2	0.36	0.589040	5
2	0.2	0.4	0.08			

3	0.4	0.2	0.08			
1 (bias)	1.0	0.1	0.1	0.3	0.574442	6
2	0.2	0.2	0.04			
3	0.4	0.4	0.16			

3. relu.json

Input = [-1.0, 0.5]

Layer 1 (Relu)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.1	0.1	0.05	0.05	4
2	-1.0	0.4	-0.4			
3	0.5	0.7	0.35			
1 (bias)	1.0	0.2	0.2	1.1	1.1	5
2	-1.0	-0.5	0.5			
3	0.5	0.8	0.4			
1 (bias)	1.0	0.3	0.3	-0.75	0	6
2	-1.0	0.6	-0.6			
3	0.5	-0.9	-0.45			

4. softmax.json

Input = [1.0, 2.0]

Layer 1 (Softmax)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	1.0	1.0	9.0	8103.08 / 12180.67 =	4

2	1.0	2.0	2.0		0.665241	
3	2.0	3.0	6.0			
1 (bias)	1.0	2.0	2.0	7.0	1096.63 / 12180.67 = 0.090031	5
2	1.0	1.0	1.0			
3	2.0	2.0	4.0			
1 (bias)	1.0	3.0	3.0	8.0	2980.95 / 12180.67 = 0.244728	6
2	1.0	3.0	3.0			
3	2.0	1.0	2.0			

5. multilayer.json

Input = [1.0, 0.0]

Layer 1 (Linear)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.5	0.5	0.5	0.5	5
2	1.0	0.0	0.0			
3	0.0	-1.0	0.0			
1 (bias)	1.0	0.5	0.5	-1.5	-1.5	6
2	1.0	-2.0	-2.0			
3	0.0	0.0	0.0			

Layer 2 (Relu)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
4 (bias)	1.0	0.5	0.5	2.0	2.0	7
5	0.5	0.0	0.0			
6	-1.5	-1.0	1.5			
4 (bias)	1.0	0.5	0.5	-1.0	0.0	8

5	0.5	-3.0	-1.5			
6	-1.5	0.0	0.0			

Input = [0.0, 1.0]

Layer 1 (Linear)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.5	0.5	-0.5	-0.5	5
2	0.0	0.0	0.0			
3	1.0	-1.0	-1.0			
1 (bias)	1.0	0.5	0.5	0.5	0.5	6
2	0.0	-2.0	0.0			
3	1.0	0.0	0.0			

Layer 2 (Relu)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
4 (bias)	1.0	0.5	0.5	0.0	0.0	7
5	-0.5	0.0	0.0			
6	0.5	-1.0	-0.5			
4 (bias)	1.0	0.5	0.5	2.0	2.0	8
5	-0.5	-3.0	1.5			
6	0.5	0.0	0.0			

Input = [0.0, 0.0]

Layer 1 (Linear)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
1 (bias)	1.0	0.5	0.5	0.5	0.5	5

2	0.0	0.0	0.0			
3	0.0	-1.0	0.0			
1 (bias)	1.0	0.5	0.5	0.5	0.5	6
2	0.0	-2.0	0.0			
3	0.0	0.0	0.0			

Layer 2 (Relu)

Node Input	Nilai Input	Weight	Input x Weight	Y	Nilai Output	Node output
4 (bias)	1.0	0.5	0.5	0.0	0.0	7
5	0.5	0.0	0.0			
6	0.5	-1.0	-0.5			
4 (bias)	1.0	0.5	0.5	-1.0	0.0	8
5	0.5	-3.0	-1.5			
6	0.5	0.0	0.0			

Pembagian tugas

NIM	Nama	Bagian pengerjaan
13520022	Primanda Adyatma Hafiz	Fungsi aktivasi
13520099	Vincent Prasetiya Atmadja	Kelas yang digunakan
13520138	Gerald Abraham Sianturi	*File loader*
13520162	Daffa Romyz Aufa	Visualisasi graf dan teks