Implementasi Forward Propagation untuk Feed Forward Neural Network

Dibuat sebagai Salah Satu Komponen Pengumpulan Milestone 1 Tugas Besar Pembelajaran Mesin IF3170

Oleh:

Henry Anand Septian Radityo	13521004
Matthew Mahendra	13521007
Hidayatullah Wildan Ghaly Buchary	13521015
Ahmad Nadil	13521024

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

Daftar Isi

Daftar Tabel

BAB 1

Implementasi

1.1 Implementasi Layer

Setiap layer pada model diimplementasikan sebagai sebuah kelas dengan jumlah neuron tertentu dan fungsi aktivasi tertentu. Nilai bobot setiap neuron dan nilai bias disimpan di dalam kelas ini sebagai atribut. Fungsi-fungsi aktivasi berupa ReLU, Sigmoid, Linear, dan Softmax diwujudkan sebagai *class method*. Terdapat method kakas untuk mengembalikan objek instansiasi kelas sebagai sebuah string yang memberikan informasi atribut kelas.

Perhitungan input dilakukan pada method forward, dengan melakukan perkalian antara vektor input dengan bobot setiap neuron ditambah dengan bias yang telah didefinisikan. Hasil perhitungan tersebut kemudian dijadikan sebagai parameter method calculate untuk menentukan penggunaan fungsi aktivasi mana yang harus digunakan.

Kode untuk kelas Layer dapat dilihat pada listing 1.1

```
class Layer:
    '''
    Layer class with different methods of activation

    Attributes
    ----------
    activation: str
        The activation type. Must be either 'linear', 'sigmoid', 'relu
            ', or 'softmax'

    neuron: int
        Number of neurons on the layer. Must at least be 1

    weights: np.array
        Weights of each neuron corresponding to the previous layers

    bias: np.array
        Bias of each neuron. Element of array must match the number of
            neurons available
    '''
    LINEAR = 'linear'
    SIGMOID = 'sigmoid'
    RELU = 'relu'
    SOFTMAX = 'softmax'

    def __init__(self, activation: str, neuron: int, weights: np.array,
        bias: np.array):
        '''
        Class constructor
```

```
        Parameters
        ----------
        activation: str
        The activation type. Must be either 'linear', 'sigmoid', 'relu
            ', or 'softmax'

        neuron: int
            Number of neurons on the layer. Must at least be 1

        weights: np.array
            Weights of each neuron corresponding to the previous layers

        bias: np.array
            Bias of each neuron. Element of array must match the number
                of neurons available
        '''
        self.activation = activation
        self.neuron = neuron
        self.weights = weights
        self.bias = bias

    def __str__(self):
        return f"Activation Function: {self.activation}; Neuron: {self.
            neuron}; weights: {np.array_str(self.weights)}; bias: {np.
            array_str(self.bias)}"

    def linear(self, x):
        '''
        Linear activation function

        Parameters
        ----------
        x: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Activated inputs
        '''
        return x

    def relu(self, x):
        '''
        ReLU activation function

        Parameters
        ----------
        x: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Activated inputs
        '''
        return np.maximum(0, x)
```

```
    def sigmoid(self, x):
        '''
        Sigmoid activation function

        Parameters
        ----------
        x: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Activated inputs
        '''
        return 1/(1+np.exp(-x))

    def softmax(self, x):
        '''
        Softmax activation function

        Parameters
        ----------
        x: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Activated inputs
        '''
        exp = np.exp(x-np.max(x))
        exps = np.sum(exp)
        return exp / exps

    def calculate(self, x):
        '''
        Calculation of each input with the corresponding activation
           method

        Parameters
        ----------
        x: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Activated inputs
        '''
        if self.activation == Layer.LINEAR:
            return self.linear(x)
        elif self.activation == Layer.SIGMOID:
            return self.sigmoid(x)
        elif self.activation == Layer.RELU:
            return self.relu(x)
        elif self.activation == Layer.SOFTMAX:
            return self.softmax(x)
```

```
    def forward(self, input: np.array):
        '''
        Forward API for the neural network class

        Parameters
        ----------
        input: np.array
            Array of inputs

        Returns
        -------
        x: np.array
            Calculated and activated inputs of net
        '''
        return self.calculate(np.dot(input, self.weights) + self.bias)
```

Listing 1.1: Layer Class

1.2 Implementasi Feed Forward Neural Network

Model FFNN diimplementasikan sebagai sebuah kelas yang akan menyimpan layer-layer yang didefinisikan, hasil, dan input masukan. Konstruksi model dapat dilakukan secara manual ataupun melalui sebuah file json. Model dapat disimpan ke file .joblib menggunakan kakas joblib, untuk dapat digunakan kembali. Data input, jika konstruksi model dilakukan secara manual, disimpan melalui metode fit.

Method forward digunakan untuk melakukan iterasi terhadap seluruh layer yang ada dan kemudian memanggil method forward pada masing-masing layer. Input yang digunakan di awal adalah input pengguna atau input json. Hasil dari input tersebut kemudian akan digunakan pada iterasi layer berikutnya hingga ke layer yang paling terakhir. Terdapat method summarize untuk memberikan informasi layer secara teks maupun secara visualisasi. Terdapat juga method SSE untuk menentukan apakah sebuah uji kasus berhasil memiliki error di bawah error maksimum atau tidak.

Kode program yang digunakan dapat dilihat pada listing 1.2

```
class FFNN:
    '''
    An implementation of Feed Forward Neural Network

    Attributes
    ----------
    layer: list[Layer]
        The layers of the neural network

    results: np.array
        The results of the final output layer
    '''
    def __init__(self, layers: list[Layer] = None):
        '''
        Class Constructor

        Parameters
        ----------
```

```
        layer: list[Layer]
            The layers of the neural network
        '''
        self.layers = layers
        self.results = None
        self.input = None
        self.filename = None

    def load_model(self, filename: str):
        '''
        Function to load a model given a filename

        Parameters
        ----------
        filename: str
            Filename of the json file. Must be located in a models path
                relative to this file
        '''
        try:
            self.filename = filename
            file = open(f'models/{filename}.json', 'r')
            model = json.load(file)
            file.close()
            json_layer = model['case']['model']['layers']
            json_weights = model['case']['weights']
            json_input = model['case']['input']
            layers = []
            # Create the layers from the layer
            for i in range(len(json_layer)):
                layer = Layer(activation=json_layer[i]['
                    activation_function'],
                              neuron=json_layer[i]['number_of_neurons'
                                  ],
                              weights=np.array(json_weights[i][1:]),
                              bias=np.array(json_weights[i][0]))
                layers.append(layer)

            self.layers = layers
            self.input = np.array(json_input)

        except FileNotFoundError:
            print("Model not found")
            exit()

    def save_model(self, filename:str):
        '''
        Function to output a model to file

        Parameters
        ----------
        filename: str
            Model filename save
        '''
        joblib.dump(self, f'res/{filename}.joblib')

    def fit(self, input: np.array):
        '''
        Fit the inputs
```

```
        '''
        self.input = input

    def add_layer(self, layer: Layer):
        '''
        Function to add layers to the neural network

        Parameters
        ----------
        layer: Layer
            The layer to be added

        Returns
        -------
        None
        '''
        self.layers.append(layer)

    def forward(self):
        '''
        Function to forward the inputs

        Parameters
        ----------
        None

        Returns
        -------
        results: np.array
            Output of the final output layer
        '''

        self.results = self.input
        for layer in self.layers:
            self.results = layer.forward(self.results)
        return self.results

    def summarize(self, verbose: int = 1):
        '''
        Function to summarize the model's layers

        Parameters
        ----------
        verbose: int
            Verbose of information. 1 for text, 2 for visualization

        Returns
        -------
        None
        '''

        if(verbose == 1):
            print("Model␣Network")
            for i, layer in enumerate(self.layers):
                print(f"{i+1}.␣{layer}")
        elif(verbose == 2):
            # graph = Graph(self)
            # graph.draw_neural_net()
```

```
total_nodes = sum([len(layer.weights) for layer in self.
    layers]) + len(self.results)
fig_width = max(12, total_nodes / 3)
fig_height = max(6, total_nodes / 6)

fig, ax = plt.subplots(figsize=(fig_width, fig_height))

layer_sizes = []
for layer in self.layers:
    layer_sizes.append(len(layer.weights))
layer_sizes.append(len(self.layers[len(self.layers)-1].
    weights[0]))

weights = []
biases = []

for layer in self.layers:
    weights.append(layer.weights)
    biases.append(layer.bias)

n_layers = len(layer_sizes)
v_spacing = (1. / max(layer_sizes)) * .8
h_spacing = 1.0 / (n_layers - 1)
node_size = 2000
input_color = 'gold'
hidden_color= 'gray'
output_color = 'salmon'
bias_color = 'lightgreen'
edge_color = 'black'
layers = []
proporsi = 0.75

G = nx.DiGraph()
node_colors={}

for i in range(n_layers):
    layer = []
    if i < n_layers - 1:
        layer.append('b'+str(i))
    for j in range(layer_sizes[i]):
        if i == 0:
            layer.append('x'+str(j+1))
        elif i < n_layers-1:
            layer.append('h'+str(i)+str(j+1))
        else:
            layer.append('o'+str(j))
    layers.append(layer)

for i in range(n_layers):
    for j in range(len(layers[i])):
        if "x" in layers[i][j]:
            node_color = input_color
        elif "h" in layers[i][j]:
            node_color = hidden_color
        elif "o" in layers[i][j]:
            node_color = output_color
        elif "b" in layers[i][j]:
            node_color = bias_color
```

```
                G.add_node(layers[i][j], pos=(h_spacing*i,
                    v_spacing*(len(layers[i])-j)-0.1))
                node_colors[layers[i][j]] = node_color

        for i in range(n_layers-1):
            for j in range(len(layers[i])):
                target_start_index = 1 if i+1 < n_layers-1 else 0
                for k in range(target_start_index, len(layers[i+1])
                    ):
                    G.add_edge(layers[i][j], layers[i+1][k], color=
                        edge_color)
                    x = [G.nodes[layers[i][j]]['pos'][0], G.nodes[
                        layers[i+1][k]]['pos'][0]]
                    y = [G.nodes[layers[i][j]]['pos'][1], G.nodes[
                        layers[i+1][k]]['pos'][1]]

                    label_x = (proporsi * x[0]) + ((1-proporsi) * x
                        [1])
                    label_y = (proporsi * y[0]) + ((1-proporsi) * y
                        [1])

                    if (j == 0):
                        weight_label = biases[i][k-1] if i <
                            n_layers - 2 else biases[i][k]
                        ax.text(label_x, label_y, str(weight_label)
                            , color='red', ha='center', va='center')
                    else:
                        weight_label = weights[i][max(j-1, 0)][k-
                            target_start_index] if j > 0 else str(
                            weights[i][j][k-1])
                        ax.text(label_x, label_y, str(weight_label)
                            , color='red', ha='center', va='center')

        pos = nx.get_node_attributes(G, 'pos')
        edges = G.edges()
        colors = [G[u][v]['color'] for u,v in edges]

        nx.draw(G, pos, ax=ax, node_size=node_size, node_color=[
            node_colors[node] for node in G.nodes()], edge_color=
            colors, with_labels=True, font_size=8)

def sse(self, filename: str):
    '''
    Sum Squared Error method to determine if a test case succeeds
        or not

    Parameters
    ----------
    filename: str
        Name of the file to be used as a testcase

    Returns
    -------
    verdict: bool
        Verdict of the testcase
    '''
    if(self.results is not None):
```

```
            file = open(f'models/{filename}.json', 'r')
            model_json = json.load(file)
            file.close()

            sse_max = model_json['expect']['max_sse']
            output = np.array(model_json['expect']['output'])
            sse = np.sum(np.square(output.flatten()-self.results.
               flatten()))
            print(f'SSE: {sse}')
            return sse <= sse_max
        else:
            raise Exception("Please run the forward function first!")
```

Listing 1.2: FFNN Class

1.3 Contoh Penggunaan Secara Manual

Berikut adalah contoh penggunaan model dengan instansiasi secara manual. Model yang dibangun adalah model dengan 2 layer. Layer 1 memiliki 3 neuron dengan fungsi aktivasi ReLU dan bobotnya. Layer 2 memiliki 1 neuron dengan fungsi aktivasi linear.

```
model = FFNN(
    [
        Layer('relu', 3, np.array([[0.4, -0.5, 0.6], [0.7,0.8,-0.9]]),
           np.array([0.1,0.2,0.3])),
        Layer('linear', 1, np.array([[0.1], [0.02], [-0.01]]), np.array
           ([-0.2]))
    ]
)

model.fit(np.array([-1.0, 0.5]))

model.forward()
print(model.results)
model.summarize(1)
```

Listing 1.3: Manual Instantiating

1.4 Contoh Penggunaan dengan Uji Kasus

Berikut contoh penggunaan dari uji kasus yang diberikan pada file relu.json. Digunakan pula method sse untuk penentuan kelulusan test case berupa true or false. Nama file yang digunakan hanya nama sebelum ekstensi .json dan file diletkkan di dalam folder models relatif terhadap tempat menjalankan program.

```
model = FFNN()
model.load_model('relu')
model.forward()
model.sse('relu')
```

Listing 1.4: Load Model

BAB 2

Hasil dan Pengujian

2.1 Test Case ReLU - 1

Pengujian pada test case dengan fungsi aktvasi ReLU memberikan hasil sebagai berikut

- Output layer terakhir: [[0.31 0. 0.375]]
- SSE: $3.0814879110195774e-33 \sim 0$
- Verdict: True

2.1.1 Perhitungan Manual ReLU - 1

Didefinisikan matriks-matriks yang dibutuhkan sebagai berikut

$$I = \begin{bmatrix} 1.5 & -0.45 \end{bmatrix}$$
$$W_{xh_1} = \begin{bmatrix} 0.47 & -0.6 & 0.2 \\ 1.1 & -1.3 & 0.5 \end{bmatrix}$$
$$b = \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}$$

Menghitung output dari 1 layer saja dengan perhitungan sebagai berikut,

$$\begin{aligned} O_L &= ReLU(IW_{xh_1} + b) \\ &= \begin{bmatrix} MAX(0, 0.31) & MAX(0, -0.115) & MAX(0, 0.375) \end{bmatrix} \\ &= \begin{bmatrix} 0.31 & 0 & 0.375 \end{bmatrix} \end{aligned}$$

Didapatkan SSE 0 untuk expected output yang diberikan pada test case. Hasil perhitungan manual dengan pengujian test case dengan program mendapatkan hasil yang sama. Artinya, perhitungan pada program sudah benar.

2.2 Test Case ReLU-2

- Output layer terakhir: [[0.05 1.1 0.]]
- SSE: $1.7333369499485123e-33 \sim 0$
- Verdict: True

2.2.1 Perhitungan Manual ReLU-2

Didefinisikan matriks-matriks yang dibutuhkan sebagai berikut

$$I = \begin{bmatrix} -1.0 & 0.5 \end{bmatrix}$$
$$W_{xh_1} = \begin{bmatrix} 0.4 & -0.5 & 0.6 \\ 0.7 & 0.8 & -0.9 \end{bmatrix}$$
$$b = \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}$$

Menghitung output dari 1 layer saja dengan perhitungan sebagai berikut,

$$\begin{aligned} O_L &= ReLU(IW_{xh_1} + b) \\ &= \begin{bmatrix} MAX(0, 0.05) & MAX(0, 1.1) & MAX(0, -0.75) \end{bmatrix} \\ &= \begin{bmatrix} 0.05 & 1.1 & 0.0 \end{bmatrix} \end{aligned}$$

Didapatkan SSE 0 untuk expected output yang diberikan pada test case. Hasil perhitungan manual dengan pengujian test case dengan program mendapatkan hasil yang sama. Artinya, perhitungan pada program sudah benar.

2.3 Test Case Multilayer

Pengujian pada test case untuk model multilayer memberikan hasil sebagai berikut

- Output Layer terakhir: [[0.4846748]]
- SSE: $3.1555534707211278e^{-18} \sim 0$
- Verdict: True

2.3.1 Perhitungan Manual Multilayer

Didefinisikan matriks-matriks yang dibutuhkan sebagai berikut

$$
\begin{aligned}
I &= \begin{bmatrix} -1.0 & 0.5 & 0.8 \end{bmatrix} \\
W_{xh_1} &= \begin{bmatrix} -0.5 & 0.6 & 0.7 & 0.5 \\ 0.9 & 1.0 & -1.1 & -1.0 \\ 1.3 & 1.4 & 1.5 & 0.1 \end{bmatrix} \\
b_1 &= \begin{bmatrix} 0.1 & 0.2 & 0.3 & -1.2 \end{bmatrix} \\
W_{xh_2} &= \begin{bmatrix} -0.4 & 0.5 & 0.6 \\ 0.7 & 0.4 & -0.9 \\ 0.2 & 0.3 & 0.4 \\ -0.1 & 0.2 & 0.1 \end{bmatrix} \\
b_2 &= \begin{bmatrix} 0.1 & 0.1 & 0.3 \end{bmatrix} \\
W_{xh_3} &= \begin{bmatrix} -0.3 & 0.4 \\ 0.6 & 0.1 \\ 0.1 & -0.4 \end{bmatrix} \\
b_3 &= \begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \\
W_{xh_4} &= \begin{bmatrix} -0.2 \\ -0.3 \end{bmatrix} \\
b_4 &= \begin{bmatrix} 0.1 \end{bmatrix}
\end{aligned}
$$

Secara komposisi, dilakukan perhitungan sebagai berikut,

$$
\begin{aligned}
O_L &= Sigmoid(ReLU(ReLU(ReLU(IW_{xh_1} + b_1)W_{xh_2} + b_2)W_{xh_3} + b_3)W_{xh_4} + b_4) \\
&= Sigmoid(ReLU(ReLU(\begin{bmatrix} 2.09 & 1.22 & 0.25 & 0 \end{bmatrix} W_{xh_2} + b_2)W_{xh_3} + b_3)W_{xh_4} + b_4) \\
&= Sigmoid(ReLU(\begin{bmatrix} 0.168 & 1.708 & 0.556 \end{bmatrix} W_{xh_3} + b_3)W_{xh_4} + b_4) \\
&= Sigmoid(\begin{bmatrix} 1.13 & 0.2156 \end{bmatrix} W_{xh_4} + b_4) \\
&= \begin{bmatrix} 0.4846748 \end{bmatrix}
\end{aligned}
$$

Didapatkan SSE 0 untuk expected output yang diberikan pada test case. Hasil perhitungan manual dengan pengujian test case dengan program mendapatkan hasil yang sama. Artinya, perhitungan pada program sudah benar.

2.4 Test Case Linear

Pengujian pada test case untuk model linear memberikan hasil sebagai berikut

- Output Layer terakhir: [[-11], [-8],[-5],[-2],[1],[4],[7],[10],[13],[16]]
- SSE: 0.0
- Verdict: True

2.4.1 Perhitungan Manual Linear

Didefinisikan matriks-matriks yang dibutuhkan sebagai berikut

$$I = \begin{bmatrix} -4 \\ -3 \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$
$$W_{xh_1} = \begin{bmatrix} 3 \end{bmatrix}$$
$$b = \begin{bmatrix} 1 \end{bmatrix}$$

Menghitung output dari 1 layer saja dengan perhitungan sebagai berikut,

$$O_L = Linear(IW_{xh_1} + b)$$
$$= \begin{bmatrix} -11 \\ -8 \\ -5 \\ -2 \\ 1 \\ 4 \\ 7 \\ 10 \\ 13 \\ 16 \end{bmatrix}$$

Didapatkan SSE 0 untuk expected output yang diberikan pada test case. Hasil perhitungan manual dengan pengujian test case dengan program mendapatkan hasil yang sama. Artinya, perhitungan pada program sudah benar.

2.5 Test Case Softmax

Pengujian pada test case untuk model softmax memberikan hasil sebagai berikut

- Output Layer terakhir: [[0.76439061 0.21168068 0.02392871]]
- SSE: $1.2639167390097123e-17 \sim 0.0$
- Verdict: True

2.5.1 Perhitungan Manual Softmax

Didefinisikan matriks-matriks yang dibutuhkan sebagai berikut

$$I = \begin{bmatrix} -1 & 1 & 2.8 & 1.8 & -0.45 & 0.24 & 0.15 & 0.2 \end{bmatrix}$$

$$W_{xh_1} = \begin{bmatrix} -0.2 & 0.8 & 0.2 \\ 0.3 & -0.7 & 0.3 \\ 0.4 & 0.6 & -0.4 \\ 0.5 & 0.5 & 0.5 \\ -0.6 & 0.4 & 0.6 \\ -0.7 & -0.3 & 0.7 \\ 0.8 & 0.2 & -0.8 \\ 0.9 & -0.1 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0.1 & 0.9 & -0.1 \end{bmatrix}$$

Menghitung output dari 1 layer saja dengan perhitungan sebagai berikut,

$$\begin{aligned} O_L &= Softmax(IW_{xh_1} + b) \\ &= \begin{bmatrix} \frac{e^{(3.022)}}{e^{(3.022)} + e^{(1.738)} + e^{(-0.442)}} & \frac{e^{(1.738)}}{e^{(3.022)} + e^{(1.738)} + e^{(-0.442)}} & \frac{e^{(-0.442)}}{e^{(3.022)} + e^{(1.738)} + e^{(-0.442)}} \end{bmatrix} \\ &= \begin{bmatrix} 0.7643906087 & 0.2116806829 & 0.0239287084 \end{bmatrix} \end{aligned}$$

Didapatkan SSE 0 untuk expected output yang diberikan pada test case. Hasil perhitungan manual dengan pengujian test case dengan program mendapatkan hasil yang sama. Artinya, perhitungan pada program sudah benar.

2.6 Test Case Sigmoid

- Output Layer terakhir: [[0.41197346 0.8314294 0.53018536 0.31607396] [0.78266141 0.80843631 0.55350518 0.64278501] [0.58987524 0.82160954 0.75436518 0.34919895] [0.6722004 0.81660439 0.59020258 0.50870988] [0.47322841 0.82808466 0.69105452 0.29358323]]
- SSE: $2.1756063348585549e-16 \sim 0.0$
- Verdict: True

2.7 Test Case Multilayer Softmax

- Output Layer terakhir: [[0.7042294 0.2957706]]
- SSE: $1.9422827777624138e-19 \sim 0.0$
- Verdict: True

Daftar Pustaka

[1] T. P. IF3270, "Perkuliahan pembelajaran mesin," 2024.

Lampiran A

Pembagian Kerja

NIM	Nama	Pekerjaan
13521004	Henry Anand Septian Radityo	Visualisasi Model
13521007	Matthew Mahendra	Implementasi FFNN dan Layer
13521015	Hidayatullah Wildan Ghaly Buchary	Visualisasi Model
13521024	Ahmad Nadil	Implementasi FFNN dan Layer

Tabel A.1: Pembagian Kerja Kelompok