

**LAPORAN TUGAS BESAR  
IF3270 PEMBELAJARAN MESIN**

**ARTIFICIAL NEURAL NETWORK  
BAGIAN B:  
IMPLEMENTASI MINI-BATCH GRADIENT DESCENT**



Oleh:

Eugene Yap Jin Quan	13521074
Michael Utama	13521137
Johann Christian Kandani	13521138
Dewana Gustavus Haraka Otang	13521173

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

**2024**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>I: SPESIFIKASI</b>	<b>3</b>
<b>II: IMPLEMENTASI</b>	<b>5</b>
Deskripsi Implementasi	5
<b>III: PENGUJIAN</b>	<b>10</b>
Hasil Pengujian Algoritma	10
Pengujian Penyimpanan Model	14
Perbandingan Algoritma Dengan Penggunaan Library sklearn dan Keras	15
<b>IV: PEMBAGIAN TUGAS</b>	<b>22</b>
<b>REFERENSI</b>	<b>23</b>
<b>LAMPIRAN</b>	<b>23</b>

## I: SPESIFIKASI

Pada Tugas Besar Bagian B, kami diinstruksikan untuk mengimplementasi algoritma *backpropagation* dari modul FFNN yang telah diimplementasikan pada Tugas Besar Bagian A. Berikut adalah detail spesifikasi implementasi algoritma *backpropagation*.

1. Implementasi algoritma *backpropagation* dapat melakukan *update weight* pada saat *training* menggunakan *mini-batch*, yang diatur melalui parameter *batch\_size*.
2. Implementasi algoritma *backpropagation* menggunakan fungsi aktivasi berupa ReLU, sigmoid, linear, dan *softmax*, dengan fungsi turunan dari masing-masing fungsi aktivasi adalah sebagai berikut.

Nama Fungsi Aktivasi	Turunan
ReLU	$\frac{d}{dx}ReLU(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Sigmoid	$\frac{d}{dx}\sigma(x) = \sigma(x) \times (1 - \sigma(x))$
Linear	$\frac{d}{dx}x = 1$
Softmax*	$\frac{\partial E_d}{\partial net(x)} = \begin{cases} p_j, & j \neq \text{target} \\ -(1 - p_j), & j = \text{target} \end{cases}$

*Turunan fungsi aktivasi*

3. Implementasi algoritma *backpropagation* menghitung *loss* berdasarkan persamaan-persamaan berikut.

Fungsi Aktivasi	Loss
ReLU, sigmoid, dan linear	$E = \frac{1}{2} \sum_{k \in \text{output}} (t_k - o_k)^2$
Softmax	$E = -\log(p_k), k=\text{target}$

*Perhitungan loss untuk masing-masing fungsi aktivasi*

4. Update weight pada gradient descent dilakukan dengan aturan rantai.

- a. Persamaan untuk update weight

$$\Delta w = -\text{gradient} \times \text{learning rate}$$

$$w_{\text{new}} = w_{\text{old}} + \Delta w$$

- b. Persamaan gradient pada hidden layer berbeda dengan output layer

- 1) Hidden layer

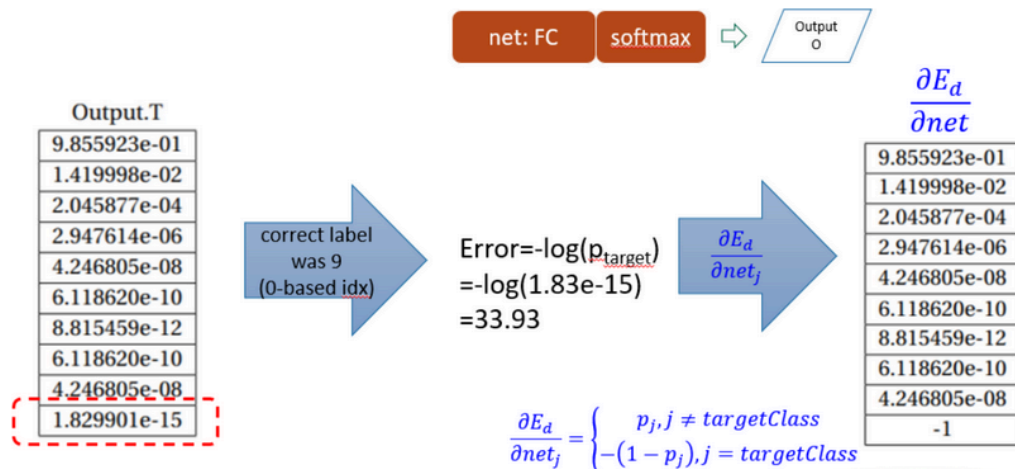
$$\frac{dE}{dw} = \frac{dE}{dnet} \frac{dnet}{dw}$$

- 2) Output layer

$$\frac{dE}{dw} = \frac{dE}{dOut} \frac{dOut}{dnet} \frac{dnet}{dw}$$

Pengecualian pada **softmax** karena langsung berbentuk  $dE/dnet$  sehingga pada softmax  $dE/dw = dE/dnet * dnet/dw$

*Perhitungan untuk proses update weight*



- Kondisi berhenti *training* adalah ketika nilai *error kumulatif*  $\leq$  *threshold* atau iterasi maksimum tercapai. Error threshold dan iterasi maksimum merupakan parameter yang dapat diubah.
- Parameter yang wajib ada pada implementasi algoritma adalah struktur jaringan (jumlah layer, jumlah neuron setiap layer, fungsi aktivasi setiap layer), *learning\_rate*, *error\_threshold*, *max\_iter*, dan *batch\_size*. Parameter lain dapat didefinisikan sesuai kebutuhan.

Batasan dari implementasi adalah satu layer memiliki fungsi aktivasi yang sama; layer yang berbeda dapat memiliki fungsi aktivasi yang berbeda.

## II: IMPLEMENTASI

### Deskripsi Implementasi

Berikut adalah detail model FFNN-*backpropagation* yang telah diimplementasikan.

- Model Neural Network diinisialisasi dengan memasukkan bobot awal dan daftar fungsi aktivasi untuk setiap layer. Model ini memiliki fungsi FFNN untuk melakukan pelatihan dengan menggunakan algoritma *forward propagation/feed-forward*. Model ini melanjutkan implementasi model pada Tugas Besar Bagian A.

```
1 # input      : 1d array
2 # weight     : list of 2d array
3 # activation  : 1d array
4
5
6 class NeuralNetwork:
7     def __init__(self, weights: list[np.ndarray], activation_functions: list[str]):
8         self.weights = weights
9         self.activation_functions = activation_functions
10        self.stopped_by = ""
11        self.values = []
12
13    def ffnn(self, input: np.ndarray) -> np.ndarray:
14        self.values = []
15        next = input
16        for weight, activation in zip(self.weights, self.activation_functions):
17            current = np.insert(next, 0, 1., axis=0)
18            next = weight.transpose().dot(current)
19            match activation:
20                case "linear":
21                    next = Linear(next)
22                case "relu":
23                    next = ReLU(next)
24                case "sigmoid":
25                    next = Sigmoid(next)
26                case "softmax":
27                    next = Softmax(next)
28
29            self.values += [current]
30            self.values += [next]
31
32        return next
33
```

#### *Implementasi FFNN*

- Implementasi FFNN dilengkapi dengan algoritma *backpropagation* yang dipanggil melalui fungsi *train*. *Backpropagation* digunakan untuk menghitung nilai  $\Delta w$  pada setiap bobot. Proses perhitungan  $\Delta w$  dilakukan secara mundur dari *layer* output menuju *layer* input. Keluaran fungsi ini adalah daftar  $\Delta w$  yang digunakan dalam proses koreksi bobot.

- Fungsi *train* pada model memanggil pelatihan FFNN dan proses *backpropagation* secara iteratif. Pada setiap iterasi dilakukan koreksi menggunakan setiap *batch* baris data. Untuk setiap baris data, model melakukan proses pelatihan menggunakan error prediksi FFNN terhadap atribut baris dan total nilai  $\Delta w$  pada *batch*. Pada akhir batch dilakukan modifikasi terhadap bobot model menggunakan  $\Delta w$  yang tercatat pada *batch* tersebut. Iterasi pelatihan berhenti apabila ditemukan error di bawah *threshold* atau tercapai iterasi maksimum.

```

1 class NeuralNetwork:
2
3
4     # Call after forward propagation
5     # target: single target
6     # return delta matrix for single instance
7     def __backward_propagation(self, target: np.ndarray):
8         # weight change by this operation
9         delta_w: list[np.ndarray] = [np.zeros(x.shape) for x in self.weights]
10
11         # calculate output error term
12         error_term = count_error_term(self.values[-1], target, self.activation_functions[-1])
13
14         delta_w[-1] = error_term[np.newaxis,:] * self.values[-2][:, np.newaxis]
15
16         # Loop from output to input
17         for i in range(len(self.weights)-2, -1, -1):
18             # select derivative func
19             if self.activation_functions[i] == 'linear':
20                 derivative_fun = linear_derivative
21             elif self.activation_functions[i] == 'relu':
22                 derivative_fun = ReLU_derivative
23             elif self.activation_functions[i] == 'sigmoid':
24                 derivative_fun = Sigmoid_derivative
25             else:
26                 raise Exception(f"No such activation function {self.activation_functions[i]}")
27
28             # calculate hidden layer error term
29             next_error_term = derivative_fun(self.values[i+1][1:]) * self.weights[i+1].dot(error_term[1:])
30             # update delta_w
31             delta_w[i] = next_error_term[np.newaxis,:] * self.values[i][:, np.newaxis]
32
33             error_term = next_error_term
34
35         return delta_w
36
37     def train(self, learning_parameters: LearningParameters, input: list[np.ndarray], target: list[np.ndarray]):
38         for iter in range(learning_parameters.max_iteration):
39             no_in_batch = 0
40             delta_w = [np.zeros(x.shape) for x in self.weights]
41
42             total_err = 0
43
44             for x, y in zip(input, target):
45                 output = self.ffnn(x)
46                 total_err += Log_error(y, output) if self.activation_functions[-1] == 'softmax' else Sum_squared_error(y, output)
47                 delta_delta_w = self.__backward_propagation(y)
48
49                 for i in range(len(delta_w)):
50                     delta_w[i] += delta_delta_w[i]
51
52             no_in_batch += 1
53             if no_in_batch == learning_parameters.batch_size:
54                 for i in range(len(self.weights)):
55                     self.weights[i] += learning_parameters.learning_rate * delta_w[i]
56                 delta_w = [np.zeros(x.shape) for x in self.weights]
57                 no_in_batch = 0
58
59             if no_in_batch > 0:
60                 for i in range(len(self.weights)):
61                     self.weights[i] += learning_parameters.learning_rate * delta_w[i]
62
63             average_err = total_err / len(input)
64
65             if average_err <= learning_parameters.error_threshold:
66                 self.stopped_by = "error_threshold"
67                 self.after_train()
68                 return
69
70             self.stopped_by = "max_iteration"
71             self.after_train()
72             return
73
74         # Cleanup so temporary values not saved
75         def after_train(self):
76             self.values = []
77

```

### Implementasi Algoritma Backpropagation

- Apabila iterasi *train* terhenti akibat *threshold* atau akibat tercapainya jumlah iterasi maksimum, nilai-nilai output pada setiap *layer* pada model dihapus untuk mencegah proses penyimpanan model.
- Fungsi *train* menerima parameter berupa *learning\_rate*, *batch\_size*, *max\_iteration*, dan *error\_threshold*.

```

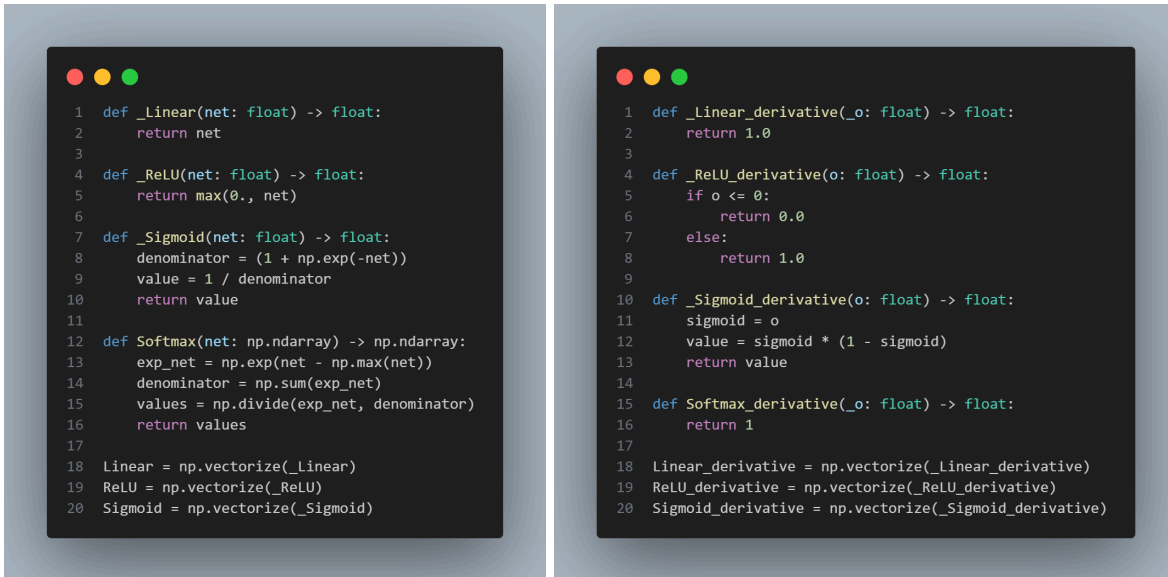
1  class Layer:
2      number_of_neurons: int
3      activation_function: str
4
5  class Model:
6      input_size: int
7      layers: list[Layer]
8
9  class LearningParameters:
10     learning_rate: float
11     batch_size: int
12     max_iteration: int
13     error_threshold: float
14
15  class Case:
16     model: Model
17     input: list[list[float]]
18     initial_weights: list[list[list[float]]]
19     target: list[list[float]]
20     learning_parameters: LearningParameters
21
22  class Expect:
23     stopped_by: str
24     final_weights: list[list[list[float]]]
25
26  class InputData:
27     case: Case
28     expect: Expect

```

*Tipe data yang digunakan dalam pelatihan model*

- Model Neural Network yang telah dibuat, baik dilatih menggunakan FFNN atau *backward propagation* dapat disimpan ke dalam sebuah *file*. *File* ini kemudian dapat dibaca kembali sebagai model Neural Network yang terlatih. Penyimpanan dan pembacaan dilakukan menggunakan modul *pickle*.

- Fungsi aktivasi, error, beserta turunan terhadap keduanya diimplementasikan sebagai fungsi yang dapat dipanggil oleh model.



```

1 def _Linear(net: float) -> float:
2     return net
3
4 def _ReLU(net: float) -> float:
5     return max(0., net)
6
7 def _Sigmoid(net: float) -> float:
8     denominator = (1 + np.exp(-net))
9     value = 1 / denominator
10    return value
11
12 def Softmax(net: np.ndarray) -> np.ndarray:
13     exp_net = np.exp(net - np.max(net))
14     denominator = np.sum(exp_net)
15     values = np.divide(exp_net, denominator)
16     return values
17
18 Linear = np.vectorize(_Linear)
19 ReLU = np.vectorize(_ReLU)
20 Sigmoid = np.vectorize(_Sigmoid)

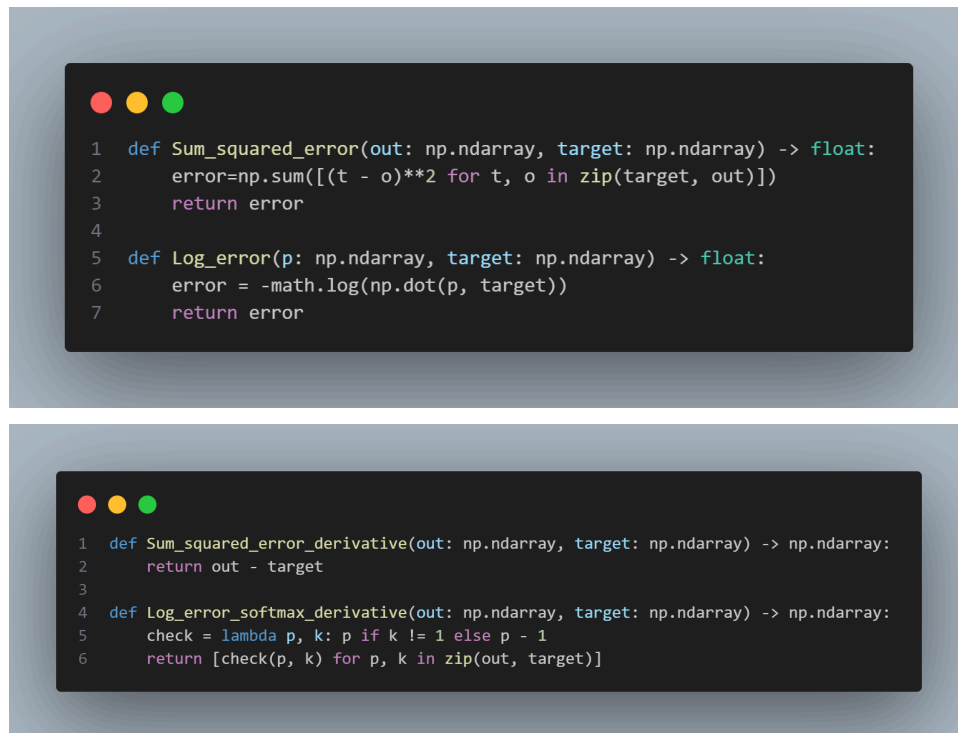
```

```

1 def _Linear_derivative(_o: float) -> float:
2     return 1.0
3
4 def _ReLU_derivative(o: float) -> float:
5     if o <= 0:
6         return 0.0
7     else:
8         return 1.0
9
10 def _Sigmoid_derivative(o: float) -> float:
11     sigmoid = o
12     value = sigmoid * (1 - sigmoid)
13     return value
14
15 def Softmax_derivative(_o: float) -> float:
16     return 1
17
18 Linear_derivative = np.vectorize(_Linear_derivative)
19 ReLU_derivative = np.vectorize(_ReLU_derivative)
20 Sigmoid_derivative = np.vectorize(_Sigmoid_derivative)

```

*Implementasi fungsi aktivasi dan turunan fungsi aktivasi*



```

1 def Sum_squared_error(out: np.ndarray, target: np.ndarray) -> float:
2     error=np.sum([(t - o)**2 for t, o in zip(target, out)])
3     return error
4
5 def Log_error(p: np.ndarray, target: np.ndarray) -> float:
6     error = -math.log(np.dot(p, target))
7     return error

```

```

1 def Sum_squared_error_derivative(out: np.ndarray, target: np.ndarray) -> np.ndarray:
2     return out - target
3
4 def Log_error_softmax_derivative(out: np.ndarray, target: np.ndarray) -> np.ndarray:
5     check = lambda p, k: p if k != 1 else p - 1
6     return [check(p, k) for p, k in zip(out, target)]

```

*Implementasi fungsi error dan turunan fungsi error*



```

1 def calculate_activation_derivative(out: np.ndarray, activation: str) -> float:
2     match activation:
3         case "linear":
4             return Linear_derivative(out)
5         case "relu":
6             return ReLU_derivative(out)
7         case "sigmoid":
8             return Sigmoid_derivative(out)
9         case "softmax":
10            return Softmax_derivative(out)
11
12 def calculate_error_derivative(out: np.ndarray, target: np.ndarray, activation: str) -> np.ndarray:
13     match activation:
14         case "linear":
15             return Sum_squared_error_derivative(out, target)
16         case "relu":
17             return Sum_squared_error_derivative(out, target)
18         case "sigmoid":
19             return Sum_squared_error_derivative(out, target)
20         case "softmax":
21             return Log_error_softmax_derivative(out, target)
22
23 def count_error_term(o: np.ndarray, t: np.ndarray, type: str) -> np.ndarray:
24     if type == 'softmax':
25         return t - o
26     else:
27         if type == 'linear':
28             fun = Linear_derivative
29         elif type == 'relu':
30             fun = ReLU_derivative
31         elif type == 'sigmoid':
32             fun = Sigmoid_derivative
33         else:
34             raise Exception("No such method")
35     return fun(o) * (t - o)
36

```

*Fungsi wrapper untuk turunan fungsi aktivasi dan error*

- Contoh pemanggilan proses pelatihan model adalah sebagai berikut.

```

1 test_cases = [
2     "tcB/linear.json",
3     "tcB/linear_small_lr.json",
4     "tcB/linear_two_iteration.json",
5     "tcB/mlp.json",
6     "tcB/relu_b.json",
7     "tcB/sigmoid.json",
8     "tcB/softmax.json",
9     "tcB/softmax_two_layer.json",
10 ]
11
12 for test_case in test_cases:
13     data = read_json(test_case)
14     model = input_data_to_model(data)
15
16     model.train(
17         learning_parameters=data.case.learning_parameters,
18         input=data.case.input,
19         target=data.case.target,
20     )
21
22 # output formatting
23 expected_final_weights = [np.array(x) for x in data.expect.final_weights]
24
25 print("testcase:", test_case)
26 print("expected result:")
27 print("stopped by:", data.expect.stopped_by)
28 print("final weights:")
29 print(expected_final_weights[0])
30 print()
31 print("training result:")
32 print("stopped by:", model.stopped_by)
33 print("final weights:")
34 print(model.weights[0])
35 print()
36

```

*Contoh pelatihan model NeuralNetwork*

### III: PENGUJIAN

#### Hasil Pengujian Algoritma

Pengujian algoritma *backpropagation* dan pelatihan model dilakukan terhadap delapan kasus uji. Pemanggilan proses pelatihan untuk kedelapan kasus uji tersebut adalah sebagai berikut.

```
1 test_cases = [  
2     "tcB/linear.json",  
3     "tcB/linear_small_lr.json",  
4     "tcB/linear_two_iteration.json",  
5     "tcB/mlp.json",  
6     "tcB/relu_b.json",  
7     "tcB/sgmold.json",  
8     "tcB/softmax.json",  
9     "tcB/softmax_two_layer.json",  
10 ]  
11  
12 for test_case in test_cases:  
13     data = read_json(test_case)  
14     model = input_data_to_model(data)  
15  
16     model.train(  
17         learning_parameters=data.case.learning_parameters,  
18         input=data.case.input,  
19         target=data.case.target,  
20     )  
21  
22     # output formatting  
23     expected_final_weights = [np.array(x) for x in data.expect.final_weights]  
24  
25     print("testcase:", test_case)  
26     print("expected result:")  
27     print("stopped by:", data.expect.stopped_by)  
28     print("final weights:")  
29     print(expected_final_weights[0])  
30     print()  
31     print("training result:")  
32     print("stopped by:", model.stopped_by)  
33     print("final weights:")  
34     print(model.weights[0])  
35     print()
```

*Pengujian algoritma pelatihan model*

Hasil pengujian kedelapan kasus uji adalah sebagai berikut.

```
testcase: tcB/linear.json  
expected result:  
stopped by: max_iteration  
final weights:  
[[ 0.22  0.36  0.11]  
 [ 0.64  0.3  -0.89]  
 [ 0.28 -0.7   0.37]]  
  
training result:  
stopped by: max_iteration  
final weights:  
[[ 0.22  0.36  0.11]
```

```
[ 0.64  0.3  -0.89]
[ 0.28 -0.7   0.37]]
```

**testcase: tcB/linear\_small\_lr.json**

expected result:

stopped by: max\_iteration

final weights:

```
[[ 0.1008  0.3006  0.1991]
 [ 0.402   0.201  -0.7019]
 [ 0.101  -0.799   0.4987]]
```

training result:

stopped by: max\_iteration

final weights:

```
[[ 0.1012  0.3006  0.1991]
 [ 0.4024  0.201  -0.7019]
 [ 0.1018 -0.799   0.4987]]
```

**testcase: tcB/linear\_two\_iteration.json**

expected result:

stopped by: max\_iteration

final weights:

```
[[ 0.166  0.338  0.153]
 [ 0.502  0.226 -0.789]
 [ 0.214 -0.718  0.427]]
```

training result:

stopped by: max\_iteration

final weights:

```
[[ 0.166  0.338  0.153]
 [ 0.502  0.226 -0.789]
 [ 0.214 -0.718  0.427]]
```

**testcase: tcB/mlp.json**

expected result:

stopped by: max\_iteration

final weights:

```
[[ 0.08592  0.32276 ]
 [-0.33872  0.46172 ]
 [ 0.449984 0.440072]]
```

```
training result:
stopped by: max_iteration
final weights:
[[ 0.08592  0.32276 ]
 [-0.33872  0.46172 ]
 [ 0.449984 0.440072]]
```

```
testcase: tcB/relu_b.json
expected result:
stopped by: max_iteration
final weights:
[[-0.211  0.105  0.885 ]
 [ 0.3033 0.5285  0.3005]
 [-0.489 -0.905  0.291 ]]
```

```
training result:
stopped by: max_iteration
final weights:
[[-0.211  0.105  0.885 ]
 [ 0.3033 0.5285  0.3005]
 [-0.489 -0.905  0.291 ]]
```

```
testcase: tcB/sigmoid.json
expected result:
stopped by: max_iteration
final weights:
[[0.2329 0.0601]
 [0.1288 0.6484]
 [0.8376 0.2315]]
```

```
training result:
stopped by: max_iteration
final weights:
[[0.23291176 0.06015346]
 [0.12884088 0.64849474]
 [0.837615 0.23158199]]
```

```
testcase: tcB/softmax.json
expected result:
stopped by: max_iteration
final weights:
```

```
[ [ 0.12674605  0.9149538 -0.14169985]
  [-0.33551647  0.67700488  0.45851159]
  [ 0.48314436 -0.85241216  0.2692678 ]
  [ 0.3400255   0.57237542 -0.31240092]
  [ 0.31397716  0.46349737  0.72252547]
  [-0.69652442  0.4789189   0.61760552]
  [-0.50884515 -0.36354141  0.57238656]
  [ 0.41891295  0.26354517 -0.48245812]
  [ 0.90374164 -0.01759501 -0.08614663]]
```

training result:

stopped by: max\_iteration

final weights:

```
[ [ 0.12674605  0.9149538 -0.14169985]
  [-0.33551647  0.67700488  0.45851159]
  [ 0.48314436 -0.85241216  0.2692678 ]
  [ 0.3400255   0.57237542 -0.31240092]
  [ 0.31397716  0.46349737  0.72252547]
  [-0.69652442  0.4789189   0.61760552]
  [-0.50884515 -0.36354141  0.57238656]
  [ 0.41891295  0.26354517 -0.48245812]
  [ 0.90374164 -0.01759501 -0.08614663]]
```

**testcase: tcB/softmax\_two\_layer.json**

expected result:

stopped by: error\_threshold

final weights:

```
[ [-0.28730211 -0.28822282 -0.70597451  0.42094471]
  [-0.5790794  -1.1836444  -1.34287961  0.69575311]
  [-0.41434377  1.51314676 -0.97649086 -1.3043465 ]]
```

training result:

stopped by: error\_threshold

final weights:

```
[ [-0.28730211 -0.28822282 -0.70597451  0.42094471]
  [-0.5790794  -1.1836444  -1.34287961  0.69575311]
  [-0.41434377  1.51314676 -0.97649086 -1.3043465 ]]
```

Berdasarkan hasil di atas, dapat dilihat bahwa kondisi berhenti pada pengujian dan harapan kasus uji adalah sama untuk setiap kasus uji. Hasil pengujian juga menunjukkan bahwa bobot akhir hasil pelatihan sama dengan nilai harapan pada setiap kasus uji. Akan tetapi, terdapat dua dari delapan kasus uji yang mengalami perbedaan presisi *floating point* pada pencetakan nilai bobot akhir pelatihan, yakni kasus uji *sigmoid.json* dan kasus uji *linear\_small\_lr.json*.

## Pengujian Penyimpanan Model

Penyimpanan dan pembacaan model dari *file* dilakukan menggunakan *library pickle*. Pengujian dilakukan dengan menyimpan model hasil pelatihan menggunakan data *softmax\_two\_layer.json*. Kode pengujian yang digunakan adalah sebagai berikut.

```
1 data: InputData = read_json("tcB/softmax_two_layer.json")
2 model = input_data_to_model(data)
3 model.train(
4     learning_parameters=data.case.learning_parameters,
5     input=data.case.input,
6     target=data.case.target,
7 )
8
9 with open("model_save.pkl", "wb") as file:
10     pickle.dump(model, file)
11
12 # =====
13
14 with open("model_save.pkl", "rb") as file:
15     loaded_model: NeuralNetwork = pickle.load(file)
16
17 # output formatting
18 expected_final_weights = [np.array(x) for x in data.expect.final_weights]
19
20 ffnn_output = [loaded_model.ffnn(input) for input in data.case.input]
21 ffnn_output = [[round(x, 4) for x in output] for output in ffnn_output]
22
23 print("expected result:")
24 print("stopped by:", data.expect.stopped_by)
25 print("model output:")
26 print(*data.case.target, sep="\n")
27 print("final weights:")
28 print(expected_final_weights[0])
29 print()
30 print("model result:")
31 print("stopped by:", loaded_model.stopped_by)
32 print("model output:")
33 print(*ffnn_output, sep="\n")
34 print("final weights:")
35 print(loaded_model.weights[0])
```

*Penyimpanan dan pembacaan model dari file*

Hasil pengujian yang diperoleh adalah sebagai berikut. Dapat diperhatikan bahwa model yang diperoleh melalui proses penyimpanan dan pembacaan *file* mampu menghasilkan output yang diharapkan pada kasus uji.

```

expected result:
stopped by: error_threshold
model output:
[0, 1]
[1, 0]
[0, 1]
[1, 0]
[1, 0]
[0, 1]
[1, 0]
[0, 1]
final weights:
[[-0.28730211 -0.28822282 -0.70597451  0.42094471]
 [-0.5790794  -1.1836444  -1.34287961  0.69575311]
 [-0.41434377  1.51314676 -0.97649086 -1.3043465  ]]

model result:
stopped by: error_threshold
model output:
[0.0304, 0.9696]
[0.9998, 0.0002]
[0.0048, 0.9952]
[1.0, 0.0]
[0.9938, 0.0062]
[0.0017, 0.9983]
[0.9965, 0.0035]
[0.0304, 0.9696]
final weights:
[[-0.28730211 -0.28822282 -0.70597451  0.42094471]
 [-0.5790794  -1.1836444  -1.34287961  0.69575311]
 [-0.41434377  1.51314676 -0.97649086 -1.3043465  ]]

```

## Perbandingan Algoritma Dengan Penggunaan Library sklearn dan Keras

Untuk menguji performa prediksi dari model, kinerja model yang telah diimplementasikan dibandingkan dengan performa model Neural Network dari *library sklearn (MLPClassifier)* dan *Keras (Sequential)*. Performa model dari ketiga implementasi diuji menggunakan kasus uji *iris.csv*. Skema pelatihan masing-masing model adalah sebagai berikut.

- Pembagian data *train* dan *test* adalah 75% - 25% (diperoleh dari parameter *default*). Pembagian dilakukan dengan menggunakan fungsi *train\_test\_split* dari *sklearn* dengan parameter *train\_test\_split(attributes, target, shuffle=True, random\_state=1)*.
- *learning rate* = 0.1
- *max\_iteration* = 420
- *batch\_size* = 10

- *error\_threshold* = 0.01
- Susunan *hidden layer* adalah 3 *layer* dengan 10 neuron untuk masing-masing *layer*.
- Fungsi aktivasi untuk *hidden layer* dan output secara berturut-turut adalah *linear*, *relu*, *sigmoid*, *softmax*.
- Berikut beberapa pengecualian parameter akibat batasan pada *library sklearn* dan *Keras*
  - Model *sklearn* mengimplementasikan fungsi aktivasi *sigmoid* untuk setiap *hidden layer*.
  - Model *sklearn* dan *Keras* tidak mengimplementasikan *error\_threshold*

Berikut adalah kode pemanggilan proses pelatihan pada masing-masing model.

```

1 X_train, X_test, y_train, y_test = train_test_split(attributes, target, shuffle=True, random_state=1)
2
3 print("Train data count:", len(X_train))
4 print("Test data count:", len(X_test))
5
6 # test MLP performance
7 learning_rate = 0.1
8
9 clf = MLPClassifier(hidden_layer_sizes=(10, 10, 10,),
10                     activation="logistic",
11                     max_iter=420,
12                     batch_size=10,
13                     solver="adam",
14                     learning_rate="constant",
15                     learning_rate_init=learning_rate).fit(X_train, y_train)
16 print("Prediction probabilities:", clf.predict_proba(X_test[:1]))
17 print('Test accuracy:', clf.score(X_test, y_test))

```

*Kode pengujian performa model MLPClassifier sklearn*



```

1 x_train, x_test, y_train, y_test = train_test_split(attributes, target, shuffle=True, random_state=1)
2 y_train = to_categorical(y_train, num_classes=number_classes)
3 y_test = to_categorical(y_test, num_classes=number_classes)
4
5 print("Train data count:", len(x_train))
6 print("Test data count:", len(x_test))
7
8 input_size = len(attributes[0])
9
10 model = Sequential()
11 model.add(Dense(10, activation='linear', input_shape=(input_size,)))
12 model.add(Dense(10, activation='relu'))
13 model.add(Dense(10, activation='sigmoid'))
14 model.add(Dense(number_classes, activation='softmax'))
15
16 model.summary()

```

```

1 # Learning parameters
2 learning_rate = 0.1
3 error_threshold = 0.1
4 batch_size = 10
5 max_iteration = 420
6
7 model.compile(loss="categorical_crossentropy",
8               optimizer=SGD(learning_rate=learning_rate),
9               metrics = ['accuracy'])
10 model.fit(x_train, y_train, batch_size=batch_size, epochs=max_iteration)
11 test_loss, test_acc = model.evaluate(x_test, y_test)
12 print('Test accuracy:', test_acc)

```

*Kode pengujian performa model Sequential Keras*

```

1 x_train, x_test, y_train, y_test = train_test_split(attributes, target, shuffle=True, random_state=1)
2 y_train = to_categorical(y_train, num_classes=number_classes)
3 y_test = to_categorical(y_test, num_classes=number_classes)
4
5 print("Train data count:", len(x_train))
6 print("Test data count:", len(x_test))
7
8 input_size = len(attributes[0])
9
10 # Layers
11 input_layer = Layer()
12 input_layer.number_of_neurons = 10
13 input_layer.activation_function = "linear"
14
15 hidden_layer_1 = Layer()
16 hidden_layer_1.number_of_neurons = 10
17 hidden_layer_1.activation_function = "relu"
18
19 hidden_layer_2 = Layer()
20 hidden_layer_2.number_of_neurons = 10
21 hidden_layer_2.activation_function = "sigmoid"
22
23 output_layer = Layer()
24 output_layer.number_of_neurons = number_classes
25 output_layer.activation_function = "softmax"
26
27 layers = [input_layer, hidden_layer_1, hidden_layer_2, output_layer]
28
29 # Initial weights
30 initial_weights: list[list[list[float]]] = []
31 for i, layer in enumerate(layers):
32     previous_number_of_neurons = input_size if i == 0 else layers[i-1].number_of_neurons
33     weight_matrix = np.random.rand(previous_number_of_neurons + 1, layer.number_of_neurons) - 0.5
34     initial_weights.append(weight_matrix)
35
36 # Model
37 model = Model()
38 model.input_size = input_size
39 model.layers = layers
40
41 # Learning parameters
42 learning_parameters = LearningParameters()
43 learning_parameters.learning_rate = 0.01
44 learning_parameters.batch_size = 10
45 learning_parameters.max_iteration = 420
46 learning_parameters.error_threshold = 0.01
47
48 # Wrapping case
49 model_case = Case()
50 model_case.model = model
51 model_case.input = x_train
52 model_case.initial_weights = initial_weights
53 model_case.target = y_train
54 model_case.learning_parameters = learning_parameters
55
56 # Model neural network
57 input_data = InputData()
58 input_data.case = model_case
59 neural_network = input_data_to_model(input_data)
60 print("Training model...")
61 neural_network.train(learning_parameters, x_train, y_train)
62
63 # Testing model
64 correct_count = 0
65 print("Testing model...")
66 for x, y in zip(x_test, y_test):
67     result = neural_network.ffnn(x)
68     output = np.argmax(result)
69     target = np.argmax(y)
70
71     if output == target:
72         correct_count += 1
73
74 accuracy = correct_count / len(x_test)
75 print("Test accuracy:", accuracy)

```

*Kode pengujian performa model NeuralNetwork*

Berikut adalah hasil pengujian performa untuk masing-masing implementasi model.

<b><i>sklearn MLPClassifier</i></b>																				
Train data count: 112 Test data count: 38 Prediction probabilities: [[0.99273101 0.0052438 0.00202519]] <b>Test accuracy: 0.9473684210526315</b>																				
<b><i>Keras Sequential</i></b>																				
Train data count: 112 Test data count: 38 Model: "sequential_1"																				
<table> <tr> <th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr> <tr> <td colspan="3">=====</td></tr> <tr> <td>dense_4 (Dense)</td><td>(None, 10)</td><td>50</td></tr> <tr> <td>dense_5 (Dense)</td><td>(None, 10)</td><td>110</td></tr> <tr> <td>dense_6 (Dense)</td><td>(None, 10)</td><td>110</td></tr> <tr> <td>dense_7 (Dense)</td><td>(None, 3)</td><td>33</td></tr> </table>			Layer (type)	Output Shape	Param #	=====			dense_4 (Dense)	(None, 10)	50	dense_5 (Dense)	(None, 10)	110	dense_6 (Dense)	(None, 10)	110	dense_7 (Dense)	(None, 3)	33
Layer (type)	Output Shape	Param #																		
=====																				
dense_4 (Dense)	(None, 10)	50																		
dense_5 (Dense)	(None, 10)	110																		
dense_6 (Dense)	(None, 10)	110																		
dense_7 (Dense)	(None, 3)	33																		
=====																				
Total params: 303 (1.18 KB) Trainable params: 303 (1.18 KB) Non-trainable params: 0 (0.00 Byte)																				
<table> <tr> <td colspan="3">=====</td></tr> <tr> <td colspan="3">Epoch 1/420</td></tr> <tr> <td colspan="3">12/12 [=====] - 1s 3ms/step - loss:</td></tr> <tr> <td colspan="3">1.0790 - accuracy: 0.5089</td></tr> </table>			=====			Epoch 1/420			12/12 [=====] - 1s 3ms/step - loss:			1.0790 - accuracy: 0.5089								
=====																				
Epoch 1/420																				
12/12 [=====] - 1s 3ms/step - loss:																				
1.0790 - accuracy: 0.5089																				

```

Epoch 2/420
12/12 [=====] - 0s 3ms/step - loss:
0.9309 - accuracy: 0.5982
Epoch 3/420
12/12 [=====] - 0s 6ms/step - loss:
0.7873 - accuracy: 0.6786
Epoch 4/420
12/12 [=====] - 0s 4ms/step - loss:
0.6743 - accuracy: 0.6964
Epoch 5/420
12/12 [=====] - 0s 4ms/step - loss:
0.5935 - accuracy: 0.7411
Epoch 6/420
12/12 [=====] - 0s 4ms/step - loss:
0.5286 - accuracy: 0.6875
Epoch 7/420
12/12 [=====] - 0s 5ms/step - loss:
0.5065 - accuracy: 0.7500
Epoch 8/420
12/12 [=====] - 0s 5ms/step - loss:
0.4949 - accuracy: 0.7232
Epoch 9/420
12/12 [=====] - 0s 4ms/step - loss:
0.4801 - accuracy: 0.7857
Epoch 10/420
12/12 [=====] - 0s 4ms/step - loss:
0.4795 - accuracy: 0.8214

(400 baris dilewatkan).....

Epoch 410/420
12/12 [=====] - 0s 13ms/step - loss:
0.1166 - accuracy: 0.9643
Epoch 411/420
12/12 [=====] - 0s 9ms/step - loss:
0.0958 - accuracy: 0.9732
Epoch 412/420
12/12 [=====] - 0s 9ms/step - loss:
0.1040 - accuracy: 0.9643
Epoch 413/420
12/12 [=====] - 0s 6ms/step - loss:
0.0943 - accuracy: 0.9732
Epoch 414/420
12/12 [=====] - 0s 7ms/step - loss:
0.1658 - accuracy: 0.9286
Epoch 415/420

```

```

12/12 [=====] - 0s 7ms/step - loss:
0.1118 - accuracy: 0.9643
Epoch 416/420
12/12 [=====] - 0s 6ms/step - loss:
0.0947 - accuracy: 0.9732
Epoch 417/420
12/12 [=====] - 0s 8ms/step - loss:
0.0875 - accuracy: 0.9732
Epoch 418/420
12/12 [=====] - 0s 7ms/step - loss:
0.0920 - accuracy: 0.9732
Epoch 419/420
12/12 [=====] - 0s 9ms/step - loss:
0.1749 - accuracy: 0.9196
Epoch 420/420
12/12 [=====] - 0s 10ms/step - loss:
0.1335 - accuracy: 0.9464
2/2 [=====] - 0s 11ms/step - loss:
0.0539 - accuracy: 0.9737
Test accuracy: 0.9736841917037964

```

### ***Model NeuralNetwork***

```

Train data count: 112
Test data count: 38
Training model...
Testing model...
Test accuracy: 1.0

```

Berdasarkan hasil pengujian, performa ketiga model dalam memprediksi data *test* berada di atas nilai akurasi 0.9. Pada instansi pengujian di atas, nilai akurasi untuk model *sklearn*, *Keras*, dan *NeuralNetwork* secara berturut-turut adalah 0.9473684210526315, 0.9736841917037964, dan 1.0. Nilai akurasi ketiga model menunjukkan bahwa model memiliki kinerja dan akurasi yang sejenis.

Meskipun demikian, nilai akurasi untuk ketiga implementasi model ini tidak bernilai konstan. Pada beberapa pengujian, ditemukan bahwa nilai akurasi ketiga model dapat berubah-ubah ketika pengujian dijalankan berulang kali. Hal ini diduga akibat pemberian nilai bobot acak untuk masing-masing model pada tahap inisialisasi pelatihan model.

## IV: PEMBAGIAN TUGAS

Pembagian tugas pada pengerjaan Tugas Besar IF3270 Bagian B adalah sebagai berikut.

NIM	Nama	Tugas
13521074	Eugene Yap Jin Quan	sklearn, pengujian, analisis hasil dan perbandingan, input dataset iris
13521137	Michael Utama	backpropagation, fungsi train, model untuk dataset iris
13521138	Johann Christian Kandani	backpropagation, fungsi turunan, model untuk dataset iris
13521173	Dewana Gustavus Haraka Otang	keras, fungsi input json, pengujian dataset iris dengan model, contoh visualisasi dan penyimpanan model

## REFERENSI

<https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1>

[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html#sklearn.neural\\_network.MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier)


<https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/>

## LAMPIRAN

Dokumen Spesifikasi:

 IF3270 - Spesifikasi Tugas Besar

Kasus Pengujian:

 Test Case Bagian B

Kasus Pengujian (iris):

[iris.csv](#)