Implementasi Mini-batch Gradient Descent

Diajukan sebagai pemenuhan tugas II

Oleh

13521116 Juan Christopher Santoso

13521135 Nicholas Liem

13521139 Nathania Calista Djunaedi

13521162 Antonio Natthan Krishna

Dosen Pengampu : Fariska Zakhralativa Ruskanda, S.T.,M.T.

IF3270 - Pembelajaran Mesin

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Bab I

Penjelasan Implementasi

Pada bagian ini, dijelaskan seluruh *class* yang digunakan dalam pengimplementasian *mini-batch gradient descent*. Setiap *class* berkaitan satu sama lain mulai dari membaca file input berupa JSON hingga dikeluarkan sebuah output. Masing-masing *class* yang telah dibuat dan kegunaannya antara lain adalah:

- *JSON Model Parser*, berfungsi untuk melakukan *parsing* dan mengidentifikasi data yang ada pada input file JSON,
- *Artificial Neural Network*, berfungsi sebagai *class* utama yang menjalankan *mini-batch gradient descent*,
- *Forward Propagation,* berfungsi sebagai *class* yang mengatur pelaksanaan *forward propagation,*
- *Backward Propagation,* berfungsi sebagai *class* yang mengatur pelaksanaan *backward propagation,*
- *Activation Function,* berfungsi sebagai *class* yang mengandung fungsi aktivasi yang dibutuhkan, dan
- *Loss Function,* berfungsi sebagai *class* yang mengandung fungsi loss.

1.1. Implementasi JSON Model Parser

```
class JsonModelParser:
    def __init__(self, filepath):
        self.filepath = filepath
        self.data = self.load_json_file()
        self.parse_model_data()

    def printDetails(self):
        print("\tINPUT SIZE:",self.input_size)
        print("\tLAYERS:", self.layers)
        print("\tINPUT:", self.input)
        print("\tINITIAL WEIGHTS:", self.initial_weights)
        print("\tTARGET:", self.target)
        print("\tLEARNING RATE:", self.learning_rate)
        print("\tBATCH SIZE:", self.batch_size)
        print("\tMAX ITERATION:", self.max_iteration)
        print("\tERROR THRESHOLD:", self.error_threshold)

    def load_json_file(self):
```

```
        try:
            with open(self.filepath, 'r', encoding='utf-8') as
file:
                return json.load(file)
        except FileNotFoundError:
            print(f"The file {self.filepath} was not found")
            return None
        except json.JSONDecodeError:
            print(f"Error decoding JSON from the file
{self.filepath}")
            return None

    def parse_model_data(self):
        if self.data:
            self.case = self.data.get('case', {})
            self.model = self.case.get('model', {})
            self.input_size = self.model.get('input_size')

            raw_layers = self.model.get('layers', [])
            self.layers = [{'number_of_neurons':
layer.get('number_of_neurons'),
                            'activation_function':
layer.get('activation_function')}
                           for layer in raw_layers]

            self.input = self.case.get('input', [])
            self.initial_weights = self.case.get('initial_weights',
[])
            self.target = self.case.get('target', [])
            self.parameters = self.case.get('learning_parameters',
{})

            self.learning_rate =
self.parameters.get('learning_rate')
            self.batch_size = self.parameters.get('batch_size')
            self.max_iteration =
self.parameters.get('max_iteration')
            self.error_threshold =
self.parameters.get('error_threshold')
```

```
        self.expect = self.data.get('expect', {})
        self.stopped_by = self.expect.get('stopped_by', '')
        self.final_weights = self.expect.get('final_weights',
[])
```

Method	**Deskripsi**
init	Inisialisasi kelas dan menerima 2 parameter, yaitu *layers* dan *architecture* yang bertipe *class JSONModelParser*.
printDetails	Melakukan print setiap atribut pada kelas JsonModelParser
load_json_file	Melakukan loading dari json file dari filepath
parse_model_data	Melakukan parsing dari json file ke dalam bentuk struktur data di Python

1.2. Implementasi Artificial Neural Network

Class Artificial Neural Network adalah *class* utama yang digunakan pada pengimplementasian program ini. Dalam *class* ini, seluruh rangkaian pembelajaran mesin diimplementasikan, mencakup pemanggilan fungsi *forward propagation* dan *backward propagation* pada *class nya* masing-masing.

```
class ArtificialNeuralNetwork:
   def __init__(self, architecture = None):
     if architecture != None:
       self.layers = architecture.layers
       self.input_size = architecture.input_size
       self.learning_rate = architecture.learning_rate
       self.error_threshold = architecture.error_threshold
       self.max_iter = architecture.max_iteration
       self.batch_size = architecture.batch_size
       self.input_data = architecture.input
```

```
        self.target = architecture.target
        self.weights = architecture.initial_weights
        self.expected_stopped_by = architecture.stopped_by
        self.expect_weights = architecture.final_weights

    def predict(self):
        res = []
        for i in range (len(self.input_data)):
            output, _, _ =
ForwardPropagation.process([self.input_data[i]], self.layers,
self.weights)
            res.append(output)
        return res

    def train(self):
        temp_weight = copy.deepcopy(self.weights)

        for j in range (self.max_iter):
            # Set initial error_total. Reset every iteration
            error_total = 0
            minibatch = self.batch_size

            # batches_amount = math.ceil(len(self.input_data)/
minibatch)

            for i in range (len(self.input_data)):
                if (minibatch == 0):
                    self.weights = copy.deepcopy(temp_weight)
                    minibatch = self.batch_size
                # Forward Propagation
                output, _, neuron_out =
ForwardPropagation.process([self.input_data[i]], self.layers,
self.weights)
                # Calculate Loss
                error_total += LossFunction.calculate(output[0],
self.target[i], self.layers)
                # Backward Propagation
                temp_weight =
BackwardPropagation.process(temp_weight, output[0],
```

```
self.target[i], neuron_out, self.layers, self.learning_rate,
self.input_data[i])
                minibatch = minibatch - 1

            self.weights = copy.deepcopy(temp_weight)

            # Adjust error to mean value
            error_total /= len(self.input_data)

            if (error_total < self.error_threshold):
                break

        if (j == self.max_iter-1):
            self.stopped_by = "MAX ITERATION"
        elif (error_total < self.error_threshold):
            self.stopped_by = "ERROR THRESHOLD"
        else:
            self.stopped_by = "UNIDENTIFIED"

        print("\n")
        print("LAST ITERATION :", j)
        print("TOTAL ERROR VAL:", error_total)
        print("\n")

        print("EXCPECTED")
        print("STOPPED BY :", self.expected_stopped_by)
        for weight_group in self.expect_weights:
            print("[")
            for weight in weight_group:
                print("\t", weight)
            print("]")
        print("\n")

        print("RESULT")
        print("STOPPED BY :", self.stopped_by)
        for weight_group in self.weights:
            print("[")
            for weight in weight_group:
                print("\t", weight)
```

```python
        print("]")

def save_weights(self, file_path):
  data = {
      "case": {
          "model": {
              "input_size": self.input_size,
              "layers": self.layers
          },
          "input": self.input_data,
          "initial_weights": self.weights,
          "learning_parameters": {
              "learning_rate": self.learning_rate,
              "batch_size": self.batch_size,
              "max_iteration": self.max_iter,
              "error_threshold": self.error_threshold
          }
      }
  }
  with open(file_path, 'w') as json_file:
      json.dump(data, json_file, indent=4)
 def load_model(self, file_path):
  architecture = JsonModelParser(file_path)
  self.layers = architecture.layers
  self.input_size = architecture.input_size
  self.learning_rate = architecture.learning_rate
  self.error_threshold = architecture.error_threshold
  self.max_iter = architecture.max_iteration
  self.batch_size = architecture.batch_size
  self.weights = architecture.initial_weights
  self.expected_stopped_by = architecture.stopped_by
  self.expect_weights = architecture.final_weights
```

Class ArtificialNeuralNetwork memiliki beberapa *method* yang akan dijelaskan pada tabel di bawah ini

Method	**Deskripsi**

init	Melakukan inisialisasi kelas ArtificialNeuralNetwork
predict	Melakukan propagasi maju untuk input data yang diberikan
train	Melakukan training model
save_model	Menyimpan model
load_model	Meload model

1.3. Implementasi Forward Propagation

```
class ForwardPropagation:
   @staticmethod
   def process(input_data, layers, weights):
       activations = input_data
       neuron_net = []
       neuron_out = []

       for i in range(len(layers)):
           activations_with_bias = np.insert(activations, 0, 1,
axis=1)
           net_input = np.dot(activations_with_bias, weights[i])
           activation_mode = layers[i]['activation_function']
           activationFunc = ActivationFunction(activation_mode)
           activations = activationFunc.func(net_input)
           neuron_net.append(net_input)
           neuron_out.append(activations)

       return activations, neuron_net, neuron_out
```

Method	**Deskripsi**
process	Fungsi propagasi maju

1.4. Implementasi Backward Propagation

```
class BackwardPropagation:
   @staticmethod
   def process(weights, output, target, neuron_out, layers,
learning_rate, input_data):

       # Initiate variables
       delta = []
       delta_layer = []

       # Delta Output Layer
       activation_mode = layers[-1]['activation_function']
       activationFunc = ActivationFunction(activation_mode)

       # Handle Case for Softmax
       if (activation_mode == 'softmax'):
           for i in range (len(output)):

delta_layer.append(activationFunc.dfuncerr(output[i], target[i]))
       else:
           for i in range (len(output)):

delta_layer.append(activationFunc.dfuncerr(output[i], target[i]) *
activationFunc.dfunc(output[i]))
       delta.append(delta_layer)

       # Delta Hidden Layer
       for i in range(len(layers) - 2, -1, -1):
           activation_mode = layers[i]['activation_function']
           activationFunc = ActivationFunction(activation_mode)

           # Save previous delta layer
           # If 2nd last, then return the Output layer
           # Else, return the previous layer but exclude the Bias
val
```

```
            prev_delta_layer = delta_layer if (i ==
(len(layers)-2)) else delta_layer[1:]

            # Initiate new Delta Layer
            delta_layer = []
            layer_weight = weights[i+1]
            layer_output = neuron_out[i][0]

            # Iterate each neuron
            for j in range (len(layer_weight)):
                neuron_weight = layer_weight[j]
                sigma = np.dot(neuron_weight, prev_delta_layer)

                # Make sure hidden layers are not softmax
                assert activation_mode != "softmax", "Softmax
cannot be in hidden layers"
                delta_layer.append(sigma * activationFunc.dfunc(1
if j == 0 else layer_output[j-1]))
            # Append but push from front
            delta = [delta_layer[1:]] + delta

        # Update Weight
        # Iterate all the layers
        for i in range (len(layers)):
            if (i == 0):
                layer_input = input_data
            else :
                layer_input = neuron_out[i-1][0] # [0] because it
is a list inside a list
            layer_input = np.insert(layer_input, 0, 1)

            # Iterate all weights
            for j in range (len(weights[i])):
                # Iterate all weights
                for k in range (len(weights[i][j])):
                    # It is based on formula wji = wji + learn_rate
* dE/dnet * dnet/dw
                    weights[i][j][k] += learning_rate *
(delta[i][k] * layer_input[j])
```

```python
        return weights
```

Method	Deskripsi
process	Fungsi propagasi mundur

1.5. Implementasi Activation Function

```python
class ActivationFunction:
    def __init__(self, types='Sigmoid'):
        self.func = self.sigmoid
        self.dfunc = self.dsigmoid
        self.dfuncerr = self.dsum_square

        match types:
            case 'sigmoid':
                self.func = self.sigmoid
                self.dfunc = self.dsigmoid
                self.dfuncerr = self.dsum_square
            case 'linear':
                self.func = self.linear
                self.dfunc = self.dlinear
                self.dfuncerr = self.dsum_square
            case 'softmax':
                self.func = self.softmax
                # self.dfuncerr = self.derr_softmax
                self.dfuncerr = self.dsum_square
            case 'relu':
                self.func = self.relu
                self.dfunc = self.drelu
                self.dfuncerr = self.dsum_square

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
```

```python
def dsigmoid(self, x):
    sig = self.sigmoid(x)
    return sig * (1-sig)

def linear(self, x):
    return x

def dlinear(self, x):
    return 1

def softmax(self, x):
    expX = np.exp(x - np.max(x, axis=1, keepdims=True))
    return expX / np.sum(expX, axis=1, keepdims=True)

def relu(self, x):
    return np.maximum(0, x)

def drelu(self, x):
    return np.where(x > 0, 1, 0)

def dsum_square(self, output, target):
    return target - output

def derr_softmax(self, output, target):
    if (target != 1):
        return output
    else:
        return output - 1
```

Method	**Deskripsi**
init	Melakukan inisialisasi kelas dari ActivationFunction dengan menerima input tipe dari fungsi aktivasinya
sigmoid	Fungsi aktivasi sigmoid

dsigmoid	Fungsi turunan aktivasi sigmoid
linear	Fungsi aktivasi linear
dlinear	Fungsi turunan aktivasi linear
softmax	Fungsi aktivasi softmax
relu	Fungsi aktivasi relu
drelu	Fungsi turunan aktivasi relu
dsum_square	Fungsi turunan sum square
derr_softmax	Fungsi turunan softmax

1.6. Implementasi Loss Function

```
class LossFunction:
   @staticmethod
   def calculate(output, target, layers):
       activation_mode = layers[-1]['activation_function']
       if (activation_mode == "softmax"):
           return LossFunction.loss_softmax(output, target)
       else:
           return LossFunction.loss_rsl(output, target)

   @staticmethod
   def loss_rsl(output, target):
       err = 0
       for i in range (len(output)):
           err += (target[i] - output[i])**2
       return 0.5 * err


   @staticmethod
   def loss_softmax(output, target):
```

```
        # idx = np.argmax(target)
        # return -1* np.log10(output[idx])

        err = 0
        for i in range(len(output)):
            err += -1* target[i] * np.log(output[i])
        return err
```

Method	**Deskripsi**
calculate	Interface kalkulasi fungsi calculate untuk fungsi loss
loss_rsl	Fungsi loss RSL
loss_softmax	Fungsi loss softmax

Bab II
Hasil Pengujian

2.1. Hasil Pengujian terhadap File *linear.json*

```
FILENAME: linear.json
      INPUT SIZE: 2
      LAYERS: [{'number_of_neurons': 3, 'activation_function':
'linear'}]
      INPUT: [[3.0, 1.0], [1.0, 2.0]]
      INITIAL WEIGHTS: [[[0.1, 0.3, 0.2], [0.4, 0.2, -0.7], [0.1, -0.8,
0.5]]]
      TARGET: [[2.0, 0.3, -1.9], [1.3, -0.7, 0.1]]
      LEARNING RATE: 0.1
      BATCH SIZE: 2
      MAX ITERATION: 1
      ERROR THRESHOLD: 0.0

LAST ITERATION : 0
TOTAL ERROR VAL: 0.3325

EXCPECTED
STOPPED BY : max_iteration
[
       [0.22, 0.36, 0.11]
       [0.64, 0.3, -0.89]
       [0.28, -0.7, 0.37]
]

RESULT
STOPPED BY : MAX ITERATION
[
       [0.21999999999999997, 0.36, 0.10999999999999999]
       [0.64, 0.30000000000000004, -0.8900000000000001]
       [0.28, -0.7, 0.36999999999999994]
]

SSE (SUM SQUARED ERROR): 1.9451892438311082e-32
```

2.2. Hasil Pengujian terhadap File *linear_small_lr.json*

```
FILENAME: linear_small_lr.json
      INPUT SIZE: 2
```

```
      LAYERS: [{'number_of_neurons': 3, 'activation_function':
'linear'}]
      INPUT: [[3.0, 1.0], [1.0, 2.0]]
      INITIAL WEIGHTS: [[[0.1, 0.3, 0.2], [0.4, 0.2, -0.7], [0.1, -0.8,
0.5]]]
      TARGET: [[2.0, 0.3, -1.9], [1.3, -0.7, 0.1]]
      LEARNING RATE: 0.001
      BATCH SIZE: 2
      MAX ITERATION: 1
      ERROR THRESHOLD: 0.0

LAST ITERATION : 0
TOTAL ERROR VAL: 0.3325

EXCPECTED
STOPPED BY : max_iteration
[
       [0.1008, 0.3006, 0.1991]
       [0.402, 0.201, -0.7019]
       [0.101, -0.799, 0.4987]
]

RESULT
STOPPED BY : MAX ITERATION
[
       [0.10120000000000001, 0.3006, 0.1991]
       [0.40240000000000004, 0.201, -0.7018999999999999]
       [0.10180000000000002, -0.799, 0.4987]
]

SSE (SUM SQUARED ERROR): 9.600000000000327e-07
```

2.3. Hasil Pengujian terhadap File *linear_two_iteration.json*

```
FILENAME: linear_two_iteration.json
      INPUT SIZE: 2
      LAYERS: [{'number_of_neurons': 3, 'activation_function':
'linear'}]
      INPUT: [[3.0, 1.0], [1.0, 2.0]]
      INITIAL WEIGHTS: [[[0.1, 0.3, 0.2], [0.4, 0.2, -0.7], [0.1, -0.8,
0.5]]]
      TARGET: [[2.0, 0.3, -1.9], [1.3, -0.7, 0.1]]
      LEARNING RATE: 0.1
      BATCH SIZE: 2
      MAX ITERATION: 2
      ERROR THRESHOLD: 0.0

LAST ITERATION : 1
TOTAL ERROR VAL: 0.0909250000000001
```

```
EXCPECTED
STOPPED BY : max_iteration
[
        [0.166, 0.338, 0.153]
        [0.502, 0.226, -0.789]
        [0.214, -0.718, 0.427]
]

RESULT
STOPPED BY : MAX ITERATION
[
        [0.166, 0.33799999999999997, 0.15300000000000008]
        [0.502, 0.22599999999999995, -0.7889999999999999]
        [0.21400000000000008, -0.718, 0.42700000000000005]
]

SSE (SUM SQUARED ERROR): 3.543711097672514e-32
```

2.4. Hasil Pengujian terhadap File *mlp.json*

```
FILENAME: mlp.json
      INPUT SIZE: 2
      LAYERS: [{'number_of_neurons': 2, 'activation_function':
'linear'}, {'number_of_neurons': 2, 'activation_function': 'relu'}]
      INPUT: [[-1.0, 0.2], [0.2, -1.0]]
      INITIAL WEIGHTS: [[[0.1, 0.2], [-0.3, 0.5], [0.4, 0.5]], [[0.2,
0.1], [0.4, -0.5], [0.7, 0.8]]]
      TARGET: [[1.0, 0.1], [0.1, 1.0]]
      LEARNING RATE: 0.1
      BATCH SIZE: 2
      MAX ITERATION: 1
      ERROR THRESHOLD: 0.0

LAST ITERATION : 0
TOTAL ERROR VAL: 0.338476

EXCPECTED
STOPPED BY : max_iteration
[
        [0.08592, 0.32276]
        [-0.33872, 0.46172]
        [0.449984, 0.440072]
]
[
        [0.2748, 0.188]
        [0.435904, -0.53168]
        [0.68504, 0.7824]
]


RESULT
```

```
STOPPED BY : MAX ITERATION
[
        [0.08592000000000001, 0.32276000000000005]
        [-0.33871999999999997, 0.46171999999999996]
        [0.449984, 0.440072]
]
[
        [0.27480000000000004, 0.188]
        [0.435904, -0.53168]
        [0.68504, 0.7824000000000001]
]

SSE (SUM SQUARED ERROR): 2.484449628259534e-32
```

2.5. Hasil Pengujian terhadap File *relu_b.json*

```
FILENAME: relu_b.json
      INPUT SIZE: 2
      LAYERS: [{'number_of_neurons': 3, 'activation_function': 'relu'}]
      INPUT: [[1.0, 0.8], [-0.3, -1.0]]
      INITIAL WEIGHTS: [[[-0.2, 0.2, 1.0], [0.3, 0.5, 0.5], [-0.5,
-1.0, 0.5]]]
      TARGET: [[1.0, 0.1, 0.1], [0.1, 0.1, 1.0]]
      LEARNING RATE: 0.1
      BATCH SIZE: 2
      MAX ITERATION: 1
      ERROR THRESHOLD: 0.0

LAST ITERATION : 0
TOTAL ERROR VAL: 1.3967749999999999

EXCPECTED
STOPPED BY : max_iteration
[
        [-0.211, 0.105, 0.885]
        [0.3033, 0.5285, 0.3005]
        [-0.489, -0.905, 0.291]
]

RESULT
STOPPED BY : MAX ITERATION
[
        [-0.21100000000000002, 0.105, 0.885]
        [0.3033, 0.5285, 0.3005]
        [-0.489, -0.905, 0.291]
]

SSE (SUM SQUARED ERROR): 7.703719777548943e-34
```

2.6. Hasil Pengujian terhadap File *softmax.json*

```
FILENAME: softmax.json
      INPUT SIZE: 8
      LAYERS: [{'number_of_neurons': 3, 'activation_function':
'softmax'}]
      INPUT: [[-2.4, -2.78, -0.6, 0.37, 2.46, -0.92, 2.76, 2.62],
[-1.79, 1.65, -0.77, -1.03, 0.1, 2.12, -2.36, 1.25], [1.65, 2.34, 0.27,
2.34, 0.52, 1.37, 1.77, 0.62]]
      INITIAL WEIGHTS: [[[0.1, 0.9, -0.1], [-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, 0.0]]]
      TARGET: [[0, 1, 0], [1, 0, 0], [0, 0, 1]]
      LEARNING RATE: 0.01
      BATCH SIZE: 1
      MAX ITERATION: 10
      ERROR THRESHOLD: 0.05

LAST ITERATION : 9
TOTAL ERROR VAL: 0.8224087756463518

EXCPECTED
STOPPED BY : max_iteration
[
       [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]
]

RESULT
STOPPED BY : MAX ITERATION
[
       [0.1267460546396248, 0.9149537996911864, -0.14169985433081125]
       [-0.33551647297979087, 0.6770048848163939, 0.4585115881633968]
       [0.4831443627109018, -0.8524121579144623, 0.26926779520356103]
       [0.3400255028017777, 0.5723754169791629, -0.3124009197809405]
       [0.31397716312208085, 0.4634973686971105, 0.7225254681808086]
       [-0.6965244228677997, 0.4789189012582683, 0.617605521609532]
       [-0.5088451517488036, -0.3635414093560929, 0.572386561104897]
       [0.41891294634289283, 0.2635451692222726, -0.4824581155651655]
       [0.9037416396108255, -0.01759501074624284, -0.08614662886458259]
]

SSE (SUM SQUARED ERROR): 1.950457691387836e-16
```

2.7. Hasil Pengujian terhadap File *softmax_two_layer.json*

```
FILENAME: softmax_two_layer.json
      INPUT SIZE: 2
      LAYERS: [{'number_of_neurons': 4, 'activation_function': 'relu'},
{'number_of_neurons': 2, 'activation_function': 'softmax'}]
      INPUT: [[3.99, 2.96], [-0.71, 2.8], [-2.43, -0.2], [-1.9, 2.62],
[-2.58, 1.43], [-3.43, -0.25], [1.15, -2.3], [4.28, 3.45]]
      INITIAL WEIGHTS: [[[0.1, -0.1, 0.1, -0.1], [-0.1, 0.1, -0.1,
0.1], [0.1, 0.1, -0.1, -0.1]], [[0.12, -0.1], [-0.12, 0.1], [0.12,
-0.1], [-0.12, 0.1], [0.02, 0.0]]]
      TARGET: [[0, 1], [1, 0], [0, 1], [1, 0], [1, 0], [0, 1], [1, 0],
[0, 1]]
      LEARNING RATE: 0.1
      BATCH SIZE: 1
      MAX ITERATION: 200
      ERROR THRESHOLD: 0.01

LAST ITERATION : 106
TOTAL ERROR VAL: 0.009936242476110798

EXCPECTED
STOPPED BY : error_threshold
[
       [-0.28730211, -0.28822282, -0.70597451, 0.42094471]
       [-0.5790794, -1.1836444, -1.34287961, 0.69575311]
       [-0.41434377, 1.51314676, -0.97649086, -1.3043465]
]
[
       [-1.72078607, 1.74078607]
       [-0.50352956, 0.48352956]
       [1.25764816, -1.23764816]
       [-1.16998784, 1.14998784]
       [1.0907634, -1.0707634]
]

RESULT
STOPPED BY : ERROR THRESHOLD
[
       [-0.28730210942847473, -0.2882228247672504, -0.7059745091124837,
0.42094470920655774]
       [-0.5790793999252973, -1.1836444019669579, -1.3428796084622934,
0.6957531055595403]
       [-0.4143437695986788, 1.5131467608357143, -0.9764908601424801,
-1.3043464969311709]
]
[
       [-1.7207860719146808, 1.7407860719146813]
       [-0.5035295626116904, 0.4835295626116901]
       [1.257648155745211, -1.23764815574521]
       [-1.1699878437697828, 1.1499878437697821]
       [1.0907633975375397, -1.070763397537539]
]
```

```
SSE (SUM SQUARED ERROR): 1.584550421684593e-16
```

Bab III

Perbandingan dengan Penggunaan Library Tensorflow

3.1. Algoritma Library Tensorflow

Untuk membandingkan hasil yang kami dapatkan dengan algoritma kami, kami menggunakan library *tensorflow* untuk melakukan perhitungan. Alasan dari pemilihan *library* ini adalah karena *tensorflow* menyediakan fleksibilitas untuk mengatur *initial weights* Dengan menggunakan *library tensorflow,* kami dapat menggunakan beberapa fungsi bawaan yang dimiliki oleh *tensorflow,* misalnya fungsi untuk menghitung *loss* dari setiap *epoch,* membuat prediksi, dan masih banyak lagi.

Namun, hasil yang didapatkan dari penggunaan *library* ini memang berbeda dengan *expected weights* yang didefinisikan di dalam file json, khususnya dalam fungsi aktivasi *linear*. Dari eksplorasi yang kami sudah lakukan terkait *library* ini, dapat kami simpulkan bahwa terdapat perbedaan implementasi fungsi MeanSquaredError yang digunakan oleh *tensorflow* dengan *loss function* yang digunakan oleh kami. Implementasi *library tensorflow* dapat dilihat di bawah ini.

```
import tensorflow as tf
import numpy as np

class TensorFlowModel:
    def __init__(self, inputs, targets, learning_rate, batch_size,
initial_weights, layers) -> None:
        self.inputs = inputs
        self.targets = targets
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.input_size = len(inputs[0])
        self.initial_weights = initial_weights
        self.dataset =
tf.data.Dataset.from_tensor_slices((self.inputs, self.targets))
        self.dataset = self.dataset.batch(batch_size)
        self.model = tf.keras.Sequential()

        for i, layer in enumerate(layers):
            if i == 0:
                self.model.add(tf.keras.layers.Dense(
                    layer['number_of_neurons'],
```

```
                    activation=layer['activation_function'],
                    input_shape=(self.input_size,),
                    kernel_initializer=lambda shape, dtype:
tf.constant_initializer(initial_weights[i][1:])(shape,
dtype=dtype),
                    bias_initializer=lambda shape, dtype:
tf.constant_initializer(initial_weights[i][0])(shape, dtype=dtype)
                ))
            else:
                self.model.add(tf.keras.layers.Dense(
                    layer['number_of_neurons'],
                    activation=layer['activation_function'],
                    kernel_initializer=lambda shape, dtype:
tf.constant_initializer(initial_weights[i][1:])(shape,
dtype=dtype),
                    bias_initializer=lambda shape, dtype:
tf.constant_initializer(initial_weights[i][0])(shape, dtype=dtype)
                ))

        loss_function = tf.keras.losses.CategoricalCrossentropy()
if layers[-1]['activation_function'] == 'softmax' else
tf.keras.losses.MeanSquaredError()
        optimizer =
tf.keras.optimizers.SGD(learning_rate=self.learning_rate)
        self.model.compile(optimizer=optimizer,
loss=loss_function, metrics=['accuracy'])

    def fit_model(self, max_epochs, error_threshold):
        class ThresholdCallback(tf.keras.callbacks.Callback):
            def on_epoch_end(self, _, logs = None):
                if(logs.get("loss") < error_threshold):
                    self.model.stop_training = True
        thressholdCallback = ThresholdCallback()
        self.model.fit(self.dataset, epochs=max_epochs,
callbacks=[thressholdCallback])
    def predict(self):
        input_array = np.array(self.inputs)
        if input_array.ndim == 1:
            input_array = np.expand_dims(input_array, axis=0)
```

```
        results = self.model.predict(input_array)
        return results

    def show_prediction(self):
        total_weights = []
        total_biases = []
        print("============================= Result
=============================\n")
        if(self.model.stop_training):
            print(f"Stopped by : error_threshold\n")
        else:
            print("Stopped by : max_iteration\n")
        for i,layer in enumerate(self.model.layers):
            print(layer.get_weights())
            weights, biases = layer.get_weights()
            total_weights.append(weights)
            total_biases.append(biases)
            print(f"Layer-{i}")
            print(f"Weights : {weights}\n")
            print(f"Biases : {biases}\n")

print("==================================================================
=======")
        return total_weights, total_biases
```

Method	**Deskripsi**
init	Inisialisasi kelas dan menerima 7 parameter, yaitu *inputs, targets, learning_rate, batch_size, initial_weights,* dan *layers.*
fit_model	Algoritma yang digunakan untuk melatih model dengan menggunakan fungsi bawaan Tensorflow, yaitu fungsi *fit.* Menerima 2 parameter, yaitu *max_epochs* dan

	error_threshold.
predict	Algoritma yang digunakan untuk membuat prediksi dari *input* yang dimiliki. Proses *predict* ini menggunakan fungsi bawaan dari *tensorflow,* yaitu fungsi *predict.*
show_prediction	Fungsi yang digunakan untuk menampilkan *weight* yang sudah diperbaharui dan menampilkan apakah proses *train* berhenti karena mencapai iterasi maksimal atau karena *error* sudah lebih kecil dari *error_threshold.*

3.2. Hasil Pengujian terhadap File *linear.json*

```
============================= Result =============================

Stopped by : max_iteration

[array([[ 0.48      ,  0.23333333, -0.7633333 ],
       [ 0.16      , -0.7666667 ,  0.45666665]], dtype=float32),
array([0.14      , 0.32000002, 0.17      ], dtype=float32)]
Layer-0
Weights : [[ 0.48        0.23333333 -0.7633333 ]
 [ 0.16       -0.7666667   0.45666665]]

Biases : [0.14       0.32000002 0.17      ]

==================================================================
```

3.3. Hasil Pengujian terhadap File *linear_small_lr.json*

```
============================= Result =============================

Stopped by : max_iteration

[array([[ 0.40080002,  0.20033334, -0.70063335],
       [ 0.1006    , -0.7996667 ,  0.49956667]], dtype=float32),
array([0.1004    , 0.30020002, 0.1997    ], dtype=float32)]
Layer-0
Weights : [[ 0.40080002  0.20033334 -0.70063335]
```

```
 [ 0.1006     -0.7996667   0.49956667]]

Biases : [0.1004     0.30020002 0.1997    ]

==================================================================
```

3.4. Hasil Pengujian terhadap File *linear_two_iteration.json*

```
============================ Result ============================

Stopped by : max_iteration

[array([[ 0.518     ,  0.24733335, -0.79433334],
       [ 0.19266666, -0.74644446,  0.4341111 ]], dtype=float32),
array([0.16066667, 0.33088893, 0.15477778], dtype=float32)]
Layer-0
Weights : [[ 0.518       0.24733335 -0.79433334]
 [ 0.19266666 -0.74644446  0.4341111 ]]

Biases : [0.16066667 0.33088893 0.15477778]

=================================================================
```

3.5. Hasil Pengujian terhadap File *mlp.json*

```
============================ Result ============================

Stopped by : max_iteration

[array([[-0.31936002,  0.48086   ],
       [ 0.424992  ,  0.470036  ]], dtype=float32), array([0.09296
, 0.26138002], dtype=float32)]
Layer-0
Weights : [[-0.31936002  0.48086   ]
 [ 0.424992    0.470036  ]]

Biases : [0.09296    0.26138002]

[array([[ 0.417952  , -0.51584   ],
       [ 0.69251996,  0.79120004]], dtype=float32), array([0.2374,
0.144 ], dtype=float32)]
Layer-1
Weights : [[ 0.417952   -0.51584   ]
 [ 0.69251996  0.79120004]]

Biases : [0.2374 0.144 ]
```

```
========================================================================
```

3.6. Hasil Pengujian terhadap File *relu_b.json*

```
============================== Result ==============================

Stopped by : max_iteration

[array([[ 0.30110002,  0.5095    ,  0.4335    ],
       [-0.49633333, -0.9683333 ,  0.43033332]], dtype=float32),
array([-0.20366667,  0.16833334,  0.96166664], dtype=float32)]
Layer-0
Weights : [[ 0.30110002  0.5095      0.4335    ]
 [-0.49633333 -0.9683333   0.43033332]]

Biases : [-0.20366667  0.16833334  0.96166664]

====================================================================
```

3.7. Hasil Pengujian terhadap File *softmax.json*

```
============================== Result ==============================

Stopped by : max_iteration

[array([[-0.33551654,  0.67700493,  0.45851156],
       [ 0.4831443 , -0.8524122 ,  0.26926786],
...

Biases : [ 0.12674606  0.9149538  -0.14169984]

====================================================================
```

3.8. Hasil Pengujian terhadap File *softmax_two_layer.json*

```
============================== Result ==============================

Stopped by : error_threshold

[array([[-0.57907915, -1.1836435 , -1.3428789 ,  0.6957532 ],
       [-0.41434368,  1.513146  , -0.97649115, -1.3043467 ]],
```

```
      dtype=float32), array([-0.28730208, -0.2882229 , -0.7059747
,  0.42094487], dtype=float32)]
Layer-0
Weights : [[-0.57907915 -1.1836435  -1.3428789   0.6957532 ]
 [-0.41434368  1.513146   -0.97649115 -1.3043467 ]]

Biases : [-0.28730208 -0.2882229  -0.7059747   0.42094487]

[array([[-0.50352937,  0.48352933],
       [ 1.2576483 , -1.2376484 ],
       [-1.1699877 ,  1.1499876 ],
       [ 1.0907636 , -1.0707631 ]], dtype=float32),
array([-1.720785 ,  1.7407851], dtype=float32)]
Layer-1
Weights : [[-0.50352937  0.48352933]
 [ 1.2576483  -1.2376484 ]
 [-1.1699877   1.1499876 ]
 [ 1.0907636  -1.0707631 ]]

Biases : [-1.720785   1.7407851]

==================================================================
```

Bab IV
Pembagian Tugas Anggota Kelompok

No	Nama Anggota	NIM Anggota	Pembagian Tugas
1	Juan Christopher Santoso	13521116	Semua
2	Nicholas Liem	13521135	Semua
3	Nathania Calista Djunaedi	13521139	Semua
4	Antonio Natthan Krishna	13521162	Semua

Lampiran

Berikut adalah *link repository* pada GitHub yang digunakan dalam pembuatan program:

https://github.com/NicholasLiem/IF3270_BP_ANN