Tugas Besar IF 3230 Machine Learning

Bagian B: Implementasi Mini-batch Gradient Descent

Disusun oleh:

Febryan Arota Hia	13521120
Dhanika Novlisariyanti	13521132
Made Debby Almadea Putri	13521153
Kandida Edgina Gunawan	13521155

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

1. Dasar Teori

Backward propagation adalah algoritma dalam pelatihan neural networks dimana model jaringan saraf belajar dari kesalahan prediksi mereka dengan memperbarui bobot untuk mengurangi network error. Pada forward propagation, alur dimulai dari layer input hingga ke layer output. Pada backward propagation, alurnya dibalik sehingga dimulai mempropagasi kesalahan dari lapisan output hingga mencapai lapisan input melalui hidden layer.

Pada tugas besar ini, backward propagation akan diimplementasikan dengan melakukan update weight saat training secara mini–batch. Algoritma fungsi aktivasi yang diimplementasikan adalah ReLU, sigmoid, linear, dan softmax.

Berikut adalah turunan fungsi aktivasi yang digunakan

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

Berikut adalah fungsi loss yang digunakan

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

Update weight pada gradient descent dilakukan dengan aturan rantai.

a. Persamaan untuk update weight

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

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

b. Persamaan gradient pada hidden layer berbeda dengan output layer

1) Hidden layer

$$\frac{\mathrm{d}E}{\mathrm{d}w} = \frac{\mathrm{d}E}{\mathrm{d}net}\frac{\mathrm{d}net}{\mathrm{d}w}$$

2) Output layer

$$\frac{\mathrm{d}E}{\mathrm{d}w} = \frac{\mathrm{d}E}{\mathrm{d}Out}\frac{\mathrm{d}Out}{\mathrm{d}net}\frac{\mathrm{d}net}{\mathrm{d}w}$$

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

2. Penjelasan Implementasi

Pada implementasi tugas besar bagian b akan digunakan beberapa modul python yaitu modul numpy untuk operasi numerik, model json untuk membaca dan mengolah data json, modul pickle untuk load dan save model, modul csv untuk membaca data csv, modul pandas untuk data, serta modul keras untuk membandingkan hasil implementasi dengan keras library.

```
import numpy as np
import json
import pickle
import random
import csv
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.optimizers import SGD
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import StandardScaler
```

Kelas Layer digunakan untuk merepresentasikan lapisan/layer jaringan yang memiliki method __init__ (constructor), o_i, dan __str__.

__init__ digunakan untuk inisialisasi objek Layer dengan parameter yang diterima adalah jumlah neuron dalam lapisan (neuron), bobot (weight) dalam bentuk array numpy, bias (bias) dalam bentuk float, dan fungsi aktivasi (activation_func) dalam bentuk string. Method ini juga memeriksa apakah fungsi aktivasi yang diberikan valid, yaitu "linear", "relu", "sigmoid", atau "softmax". Jika tidak valid, maka akan menimbulkan pengecualian (Exception).

o_i digunakan untuk menghitung output dari lapisan berdasarkan input yang diberikan. Method ini menghitung nilai net, yaitu hasil dari perkalian dot antara input dengan bobot lapisan ditambah dengan bias. Selanjutnya, berdasarkan fungsi aktivasi yang telah ditentukan, method ini mengembalikan output sesuai dengan fungsi aktivasi tersebut

derivative digunakan untuk turunan fungsi dari fungsi aktivasi. **getOutputGradient** untuk menghitung hasil output gradien dan mengembalikan gradien beserta errornya. **getHiddenGradient** untuk menghitung hasil hidden gradien dan mengembalikan gradient hidden layer beserta errornya. **updateDeltaWeigth** digunakan untuk mengupdate bobot sementara. **updateWeight** digunakan untuk mengupdate bobot akhirnya.

```
class Layer:
  def __init__ (self, neuron: int, weight : np.array, bias: float,
activation_func: str):
      self.neuron = neuron
```

```
      self.weight= weight
      self.deltaWeight = np.array(np.zeros_like(weight))
      self.bias = bias
      self.inputLayer = None
      self.outputLayer = None
      self.activation_function = activation_func
      valid_function = ["linear", "relu", "sigmoid", "softmax"]
      if activation_func not in valid_function:
        raise Exception("Invalid function!")

  def o_i(self, input: np.array):
    self.inputLayer = input
    net = np.dot(input, self.weight[1:]) + self.bias
    match self.activation_function:
      case "relu":
        self.outputLayer = np.maximum(0, net)
      case "linear":
        self.outputLayer = net
      case "sigmoid":
        self.outputLayer = (1/(1+np.exp((-1)*net)))
      case "softmax":
        self.outputLayer =  np.exp(net - np.max(net)) /
np.sum(np.exp(net - np.max(net)))
    return self.outputLayer

  def derivative(self, output, target):
    match self.activation_function:
      case "relu":
        return (output > 0).astype(float)
      case "linear":
        return 1
      case "sigmoid":
        return (output * (1 - output))
      case "softmax":
        return output - target

  def getOutputGradient (self, target: np.array):
    bias_term = np.ones((self.inputLayer.shape[0], 1))
    inputs_with_bias = np.hstack([bias_term,
```

```
np.array(self.inputLayer)])

    derivative = self.derivative(self.outputLayer, target)
    if self.activation_function != "softmax":
      err = np.subtract(self.outputLayer, target) * derivative
    else:
      err = derivative

    gradient = np.dot(inputs_with_bias.T, err)
    return gradient, err

  def getHiddenGradient(self, err_term: np.array):
    bias_term = np.ones((self.inputLayer.shape[0], 1))
    inputs_with_bias = np.hstack([bias_term,
np.array(self.inputLayer)])

    derivative = self.derivative(self.outputLayer, err_term)
    if self.activation_function != "softmax":
      err = err_term * derivative
    else:
      err = derivative

    gradient = np.dot(inputs_with_bias.T, err)
    return gradient, err

  def updateDeltaWeight(self, learning_rate: float, gradien:
np.array):
      # print("==============UPDATE DELTA
WEIGHT====================")
      # print(f"Delta weight sekarang {self.deltaWeight}")
      # print(f"Gradien: {gradien}")

      self.deltaWeight -= (learning_rate * gradien)

      # print(f"Delta weight setelah update sekarang
{self.deltaWeight}")

  def updateWeight(self):
      # print("==============UPDATE WEIGHT======================")
```

```
        # print(f"Weight lama: {self.weight}")
        # print(f"Delta weight: {self.deltaWeight}")

        self.weight += self.deltaWeight
        self.bias = self.weight[0]

        # print(f"Update weight baru jadi: {self.weight}")

        self.deltaWeight = np.array(np.zeros_like(self.weight))
```

Kelas BackwardPropagation menerima parameter layers, learning rate, ukuran batch, maksimum iterasi, dan error threshold. Pada kelas ini didefinisikan beberapa method yaitu addLayer, forward, backward, fit, predict, export, import, dan compare. **addLayer** digunakan untuk menambahkan layer baru ke jaringan, sedangkan **forward** untuk melakukan proses *feedforward* untuk menghitung output berdasarkan input yang diberikan menggunakan **o_i**.

Proses utama dilakukan pada **fit** dengan iterasi selama jumlah iterasi kurang dari jumlah maksimum iterasi dan nilai error di atas threshold. Setiap iterasi data diproses dengan menggunakan **forward** untuk menghitung output dan dilanjutkan dengan **backward** untuk menghitung gradien secara *reverse.* Kemudian melakukan pengupdatean bobot menggunakan **updateDeltaWeight** dan **updateWeight** untuk mengupdate bobot terakhir pada batch terakhir. Jika iterasi mencapai jumlah maksimum iterasi atau error melebih threshold, proses akan diberhentikan. Setelah proses iterasi selesai, bobot akhir setiap layer disimpan dalam final_weights

Selain itu terdapat method **predict** untuk melakukan prediksi, **export model** untuk menyimpan model, **import model** untuk mengambil mode, dan **compare** untuk membandingkan hasil pelatihan dengan hasil yang diharapkan.

```
class BackwardPropagation:
    def __init__(self, _layers: list, learning_rate: float,
batch_size: int, max_iteration: int, error_threshold: float):
```

```
      self.layers = _layers
      self.output = None
      self.num_of_layers = len(_layers)
      self.num_of_output_neuron = None
      self.final_weights = None

      self.learning_rate = learning_rate
      self.batch_size = batch_size
      self.max_iteration = max_iteration
      self.error_threshold = error_threshold
      self.stopped_by = None

    def addLayer(self, layer: Layer):
      self.layers.append(layer)
      self.num_of_layers += 1
      self.num_of_output_neuron = layer.neuron


    def forward(self, _input1 : np.array):
      # print("\nInput =================\n")
      self.output = _input1
      for l in self.layers:
        # print(f"Input : {self.output}")
        self.output = np.array(l.o_i(self.output))
        # print(f"Output: {self.output}")


    def backward(self, _target1):
      err_term = None
      reversed_layers = reversed(self.layers)
      for index, layer in enumerate(reversed_layers):
          if(index == 0):
            # print(f"Getting Output Layer Gradient for:
{layer.activation_function}")

            gradient, err_term = layer.getOutputGradient(_target1)
          else:
            # print(f"Getting Hidden Layer Gradient for:
{layer.activation_function}")
```

```
            err_term = np.dot(err_term,
self.layers[len(self.layers) - index].weight.T)[:, 1:]
            gradient, err_term = layer.getHiddenGradient(err_term)

          layer.updateDeltaWeight(self.learning_rate, gradient)

    def fit(self, _inputs: np.array, _targets: np.array):
      iteration = 0
      curr_error = np.inf

      while(iteration < self.max_iteration and curr_error >
self.error_threshold):
        print("\n\n", iteration / self.max_iteration * 100, "%")
        print("======================= ITERATION", iteration,
"=======================")

        curr_error = 0
        for i in range(len(_inputs)):
          # print("=======================FORWARD
PROPAGATION===========================")
          self.forward(np.array([_inputs[i]]))

          # print("=======================BACKWARD
PROPAGATION===========================")
          self.backward(np.array([_targets[i]]))

          if(i == len(_inputs) - 1 or ((i + 1 - self.batch_size) %
self.batch_size == 0)):
            print("UPDATE WEIGHT")
            for layer in self.layers:
              print()
              layer.updateWeight()
              print(layer.weight)

          if(self.layers[-1].activation_function == "softmax"):
            curr_error += 0
            for j in range(len(_targets[i])):
                if _targets[i][j] == 1:
```

```
                  # print(f"prev error : {curr_error}")

                  curr_error += (-np.log(self.output[0][j]))

                  # print(f"curr error: {curr_error} ")
                  break
          else:
            for j in range(len(_targets[i])):
                # print(f"prev error : {curr_error}")

                curr_error += ((((self.output[0][j] -
_targets[0][j]))**2) /2)

                # print(f"curr error: {curr_error} ")

            curr_error /= len(_targets[i])

        curr_error /= len(_inputs)

        print(f"FINAL ERROR : {curr_error}")

        # for layer in self.layers:
        #   layer.deltaWeight =
np.array(np.zeros_like(layer.weight))

        iteration += 1

      if(iteration == self.max_iteration):
        self.stopped_by = "max_iteration"
      else:
        self.stopped_by = "error_threshold"

      self.final_weights = []
      for layer in self.layers:
        self.final_weights.append(layer.weight)

    def predict(self, _inputs: np.array):
      self.forward(_inputs)
      print(f"Final weights: {self.final_weights}")
```

```
      print(f"Final output: {self.output}")
      max_values = self.output.max(axis = 1)
      mask = (self.output.T == max_values).T
      result = mask.astype(int)
      return result

    def export_model(self, name='bp-model.pkl'):
      with open(name, 'wb') as file:
        pickle.dump(self, file)

    def import_model(self, name):
      with open(name, 'rb') as file:
        loaded_model = pickle.load(file)

      return loaded_model

    def compare(self, expected_stopped_by, expected_final_weights,
max_sse=(pow(10, -7))):
      sse = 0
      for i in range(len(self.final_weights)):
        for j in range(len(self.final_weights[i])):
          for k in range(len(self.final_weights[i][j])):
            sse += pow(self.final_weights[i][j][k] -
expected_final_weights[i][j][k], 2)

      return self.stopped_by == expected_stopped_by, sse <=
max_sse, sse
```

Lalu terdapat kelas **ModelConfig** untuk memudahkan konfigurasi dari sebuah model.

```
class ModelConfig:
  def __init__(self, _input):
    self.layers = _input["case"]["model"]['layers']
    self.input = np.array(_input["case"]["input"])
    self.initial_weights = _input["case"]["initial_weights"]
    self.target = _input["case"]["target"]
```

```
    # learning parameters
    self.learning_rate =
_input["case"]["learning_parameters"]["learning_rate"]
    self.batch_size =
_input["case"]["learning_parameters"]["batch_size"]
    self.max_iteration =
_input["case"]["learning_parameters"]["max_iteration"]
    self.error_threshold =
_input["case"]["learning_parameters"]["error_threshold"]

    # expected
    self.expected_stopped_by = _input["expect"]["stopped_by"]
    self.expected_final_weights =
_input["expect"]["final_weights"]
```

Kelas **IO** untuk digunakan untuk pembacaan file JSON serta menambahkan model.

```
class IO:
  def read_json(self, file: str):
    input = open(file, "r")
    input = json.load(input)

    config = ModelConfig(input)

    return config

  def read(self, file: str):
    config = self.read_json(file)
    model = BackwardPropagation([], config.learning_rate,
config.batch_size, config.max_iteration, config.error_threshold)
    for i in range(len(config.layers)):
      neuron = config.layers[i]['number_of_neurons']
      activation_func = config.layers[i]['activation_function']
      weight = np.array(config.initial_weights[i])
      # print(f"Weight : {weight}")
      bias = np.array(config.initial_weights[i][0])
      layer = Layer(neuron, weight, bias, activation_func)
      model.addLayer(layer)
```

```
    # model.fit(arr_input, config.target)

    return config, model
```

Untuk melihat hasil perhitungan bobot akhir, informasi pemberhentian serta perhitungan error dapat dilihat pada kode di bawah

```
def print_compare(config, model):

print("\n======================COMPARE======================")
    print("STOPPED BY:", model.stopped_by)
    print("EXPECTED STOPPED BY:", config.expected_stopped_by)

    print("FINAL WEIGHTS:")
    print(model.final_weights)
    print("EXPECTED FINAL WEIGHTS:")
    print(config.expected_final_weights)

    same_stopped_by, same_final_weights, sse =
model.compare(config.expected_stopped_by,
config.expected_final_weights)

    print("SSE:", sse)

    print("PASSED:", same_stopped_by and same_final_weights)
```

Untuk pengujian data iris.csv, akan dilakukan *preprocessing* pada kelas **CSVProcessing** dimana kelas tersebut berisi fileName, inputs, outputs, targetLabels, inputValidation, outputValidation serta removeHeader. Lalu pada fungsi **readFile** akan dilakukan pembacaan file csv serta mengisikan tiap baris data menjadi format untuk dilakukan backward propagation sesuai konfigurasi yang dibuat oleh kami.

```
class CSVProcessing:
  def __init__(self, fileName: str, removeHeader: bool):
    self.fileName = fileName #filename.csv
    self.removeHeader = removeHeader
```

```
    self.inputs = None
    self.outputs = None
    self.targetLabels = None
    self.inputValidation = None
    self.outputValidation = None

  def readFile(self):
    file = open(self.fileName)
    csvreader = csv.reader(file)
    rows = []
    rows2 = []
    for index, row in enumerate(csvreader):
      if(index == 0 and self.removeHeader):
        continue
      if((index >= 41 and index <= 50) or (index >= 91 and index
<= 100) or (index >= 141 and index <= 150)):
        rows2.append(row)
      else:
        rows.append(row)

    self.inputs = [[float(val) for val in row[:-1]] for row in
rows]
    self.inputs = [row[1:] for row in self.inputs]
    self.inputValidation = [[float(val) for val in row[:-1]] for
row in rows2]
    self.inputValidation = [row[1:] for row in
self.inputValidation]
    self.outputs = [row[-1] for row in rows]
    self.outputValidation = [row[-1] for row in rows2]

  def labelEncoding(self):
    self.targetLabels =
pd.Series(self.outputs).drop_duplicates().to_list()
    out = []
    for i in range(len(self.outputs)):
      temp = []
      for j in range (len(self.targetLabels)):
        index = self.targetLabels.index(self.outputs[i])
        if(j == index):
```

```
          temp.append(1)
        else:
          temp.append(0)
      out.append(temp)
    self.outputs = out
    out = []
    for i in range(len(self.outputValidation)):
      temp = []
      for j in range (len(self.targetLabels)):
        index = self.targetLabels.index(self.outputValidation[i])
        if(j == index):
          temp.append(1)
        else:
          temp.append(0)
      out.append(temp)
    self.outputValidation = out
```

Eksekusi Model Backpropagation dari scratch dapat dilihat pada kode dibawah

```
# Build JSON for for MLP

#TODO BUILD THIS MLP WITH THIS REQ:
# 1 input layer, 1 hidden layer (4 neuron) , 1 output layer
# max_iter = 10000
# batch_size = 64
# learning_rate = 0.001
# activation function for hidden layer : relu
# activation function for output layer : softmax
# error threshold = 0.0001
csvProcessing = CSVProcessing(tc_folder + "iris.csv", True)
csvProcessing.readFile()
csvProcessing.labelEncoding()
input_size = len(csvProcessing.inputs[0])
print(f"input size = {input_size}")
num_of_neurons = len(csvProcessing.targetLabels)
print(num_of_neurons)


initial_weights = []
hidden_weight = []
temp = []
for i in range(input_size + 1):
  for j in range(4):
    temp.append(random.uniform(0, 1))
  hidden_weight.append(temp)
  temp = []

initial_weights.append(hidden_weight)
hidden_weight = []
temp = []

for i in range(5):
  for j in range(num_of_neurons):
    temp.append(random.uniform(-1, 1))
  hidden_weight.append(temp)
```

```
  temp = []
initial_weights.append(hidden_weight)
print(initial_weights)


data = {
    "case":{
        "model":{
            "input_size": input_size,
            "layers":[
                {
                    "number_of_neurons": 4,
                    "activation_function" : "relu"
                },
                {
                    "number_of_neurons" : num_of_neurons,
                    "activation_function": "softmax"
                }
            ]
        },
        "input": csvProcessing.inputs,
        "initial_weights" : initial_weights,
        "target" :csvProcessing.outputs,
        "learning_parameters" : {
            "learning_rate" : 0.01,
            "batch_size" : 64,
            "max_iteration" : 1000,
            "error_threshold" : 0.01
        }
    },
    "expect": {
        "stopped_by" : "-",
        "final_weights" : [

        ]
    }
}
file = tc_folder + "test.json"
with open(file, "w") as json_file:
```

```
json.dump(data, json_file)
```

Dalam memproses data pada iris.csv di sklearn, kami menstandarisasi data dengan **standardscaler**. Standardisasi data adalah proses mengubah nilai pada suatu dataset ke dalam skala yang sama agar data lebih mudah dibandingkan, diproses, dan dianalisis. Standardscaler dalam scikit-learn akan mengubah fitur-fitur dalam dataset sehingga memiliki rata-rata nol dan simpangan baku satu. Pada kode dibawah, akan dilakukan **fit** terlebih dahulu yaitu standardscaler akan menghitung rata-rata dan simpangan baku dari setiap fitur pada dataset yang diberikan, lalu akan di **transform** dimana nilai setiap fitur dikurangi dengan rata-rata fitur tersebut dan hasilnya dibagi dengan simpangan baku fitur tersebut.

```
y_train = data_train[['Species']]
X_train = data_train.drop(['Species', 'Id'], axis=1)
y_val = data_test[['Species']]
X_val = data_test.drop(['Species', 'Id'], axis=1)

scaler = StandardScaler()
scaler.fit(X_train)
scaled_data = scaler.transform(X_train)
X_train_scaled = pd.DataFrame(scaled_data, index=X_train.index,
columns=X_train.columns)
scaled_data = scaler.transform(X_val)
X_val_scaled = pd.DataFrame(scaled_data, index=X_val.index,
columns=X_val.columns)

encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train.values.ravel())
y_val_encoded = encoder.transform(y_val.values.ravel())

print('y_train_encoded', y_train_encoded)
print('y_val_encoded', y_val_encoded)

num_classes = len(encoder.classes_)
y_train_categorical = to_categorical(y_train_encoded, num_classes)
y_val_categorical = to_categorical(y_val_encoded, num_classes)
```

Untuk eksekusi Sklearn MLP

```
print(f"Bias and weights layer 1: {initial_weights[0]}")
print(f"Bias and weights output layer: {initial_weights[1]}")
```

```
model = Sequential()
model.add(Dense(4, input_dim=X_train.shape[1], activation='relu',
                kernel_initializer='random_uniform',
bias_initializer='zeros'))
model.add(Dense(num_classes, activation='softmax',
                kernel_initializer='random_uniform',
bias_initializer='zeros'))
```

```
# Compilation of the model
optimizer = SGD(learning_rate=0.01)
model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

#Setting weight (same with the one we made from scratch)
weights_layer1 = np.array(initial_weights[0][1:])  #weight at
hidden layer
biases_layer1 = np.array(initial_weights[0][0])    #bias at hidden
layer
weights_layer2 = np.array(initial_weights[1][1:])  #weight at
output layer
biases_layer2 = np.array(initial_weights[1][0])       #bias at
output layer

print("weights_layer1\n", weights_layer1)
print("biases_layer1\n", biases_layer1)
print("weights_layer2\n", weights_layer2)
print("biases_layer2\n", biases_layer2)
model.layers[0].set_weights([weights_layer1, biases_layer1])
model.layers[1].set_weights([weights_layer2, biases_layer2])

model.fit(X_train, y_train_categorical,
batch_size=config.batch_size, epochs=1000, verbose=1)

loss, accuracy = model.evaluate(X_val, y_val_categorical,
verbose=0)
print(f"Accuracy Keras Model: {accuracy * 100} %")

for layer in model.layers:
    weights, biases = layer.get_weights()
    print("Weights:", weights)
    print("Biases:", biases)
```

3. Hasil pengujian

Untuk melakukan pengujian perlu diberikan akses kepada drive untuk mengakses folder test case lalu dapat dijalankan menggunakan

```
tc_folder = "/content/drive/MyDrive/IF3270 - Tugas Besar ML/Tubes
B/tc/"
io = IO()
config, model = io.read(tc_folder + "linear.json")
model.fit(config.input, config.target)
print_compare(config, model)
```

a. **Relu_b.json**

```
 0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[-0.211   0.105   0.885 ]
 [ 0.3033  0.5285  0.3005]
 [-0.489  -0.905   0.291 ]]
FINAL ERROR : 0.25048055555555554

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[-0.211 ,  0.105 ,  0.885 ],
       [ 0.3033,  0.5285,  0.3005],
       [-0.489 , -0.905 ,  0.291 ]])]
EXPECTED FINAL WEIGHTS:
[[[-0.211, 0.105, 0.885], [0.3033, 0.5285, 0.3005], [-0.489,
-0.905, 0.291]]]
SSE: 3.851859888774472e-33
PASSED: True
```

b. **Sigmoid.json**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[0.29297259 0.09580488]
 [0.19280794 0.60482098]
 [0.80367835 0.29308146]]
FINAL ERROR : 0.11223794618876645


 10.0 %
======================= ITERATION 1 =======================
UPDATE WEIGHT
```

```
[[0.28601527 0.09165692]
 [0.18563174 0.60964881]
 [0.80737589 0.28617965]]
FINAL ERROR : 0.11208327034025578


 20.0 %
======================= ITERATION 2 =======================
UPDATE WEIGHT

[[0.27912837 0.08755599]
 [0.17847183 0.61448329]
 [0.81109236 0.2792947 ]]
FINAL ERROR : 0.11193235769812637


 30.0 %
======================= ITERATION 3 =======================
UPDATE WEIGHT

[[0.27231218 0.08350195]
 [0.1713286  0.61932423]
 [0.81482749 0.27242675]]
FINAL ERROR : 0.1117852082751855


 40.0 %
======================= ITERATION 4 =======================
UPDATE WEIGHT

[[0.26556695 0.07949467]
 [0.16420247 0.62417143]
 [0.81858101 0.2655759 ]]
FINAL ERROR : 0.11164181977565399


 50.0 %
======================= ITERATION 5 =======================
UPDATE WEIGHT

[[0.25889291 0.07553397]
 [0.1570938  0.62902471]
 [0.82235265 0.25874227]]
FINAL ERROR : 0.11150218765153971


 60.0 %
======================= ITERATION 6 =======================
UPDATE WEIGHT

[[0.25229024 0.07161969]
 [0.15000297 0.63388387]
```

```
 [0.82614215 0.25192597]]
FINAL ERROR : 0.11136630516183006


 70.0 %
======================= ITERATION 7 =======================
UPDATE WEIGHT

[[0.24575908 0.06775164]
 [0.14293032 0.63874872]
 [0.82994922 0.2451271 ]]
FINAL ERROR : 0.11123416343422836


 80.0 %
======================= ITERATION 8 =======================
UPDATE WEIGHT

[[0.23929957 0.06392963]
 [0.13587618 0.64361907]
 [0.8337736  0.23834574]]
FINAL ERROR : 0.11110575152915961


 90.0 %
======================= ITERATION 9 =======================
UPDATE WEIGHT

[[0.23291176 0.06015346]
 [0.12884088 0.64849474]
 [0.837615   0.23158199]]
FINAL ERROR : 0.11098105650577514

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[0.23291176, 0.06015346],
       [0.12884088, 0.64849474],
       [0.837615  , 0.23158199]])]
EXPECTED FINAL WEIGHTS:
[[[0.2329, 0.0601], [0.1288, 0.6484], [0.8376, 0.2315]]]
SSE: 2.0589774528721306e-08
PASSED: True
```

c. **Linear.json**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[ 0.22  0.36  0.11]
```

```
 [ 0.64  0.3  -0.89]
 [ 0.28 -0.7   0.37]]
FINAL ERROR : 0.8022222222222221

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[ 0.22,  0.36,  0.11],
       [ 0.64,  0.3 , -0.89],
       [ 0.28, -0.7 ,  0.37]])]
EXPECTED FINAL WEIGHTS:
[[[0.22, 0.36, 0.11], [0.64, 0.3, -0.89], [0.28, -0.7,
0.37]]]
SSE: 1.9451892438311082e-32
PASSED: True
```

d. **Linear_two_iteration.json**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[ 0.22  0.36  0.11]
 [ 0.64  0.3  -0.89]
 [ 0.28 -0.7   0.37]]
FINAL ERROR : 0.8022222222222221


 50.0 %
======================= ITERATION 1 =======================
UPDATE WEIGHT

[[ 0.166  0.338  0.153]
 [ 0.502  0.226 -0.789]
 [ 0.214 -0.718  0.427]]
FINAL ERROR : 0.41558055555555556

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[ 0.166,  0.338,  0.153],
       [ 0.502,  0.226, -0.789],
       [ 0.214, -0.718,  0.427]])]
EXPECTED FINAL WEIGHTS:
[[[0.166, 0.338, 0.153], [0.502, 0.226, -0.789], [0.214,
-0.718, 0.427]]]
SSE: 1.6948183510607676e-32
PASSED: True
```

e. **Linear_small_lr.json**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[ 0.1012  0.3006  0.1991]
 [ 0.4024  0.201  -0.7019]
 [ 0.1018 -0.799   0.4987]]
FINAL ERROR : 0.8022222222222221

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[ 0.1012,  0.3006,  0.1991],
       [ 0.4024,  0.201 , -0.7019],
       [ 0.1018, -0.799 ,  0.4987]])]
EXPECTED FINAL WEIGHTS:
[[[0.1008, 0.3006, 0.1991], [0.402, 0.201, -0.7019], [0.101,
-0.799, 0.4987]]]
SSE: 9.600000000000107e-07
PASSED: False
```

f. **Softmax**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

[[ 9.24700795e-02  9.07552947e-01 -1.00023026e-01]
 [-1.81928191e-01  7.81872927e-01  2.00055263e-01]
 [ 3.20933179e-01 -7.20997192e-01  3.00064013e-01]
 [ 4.04517952e-01  5.95468232e-01 -3.99986184e-01]
 [ 4.97213929e-01  5.02794590e-01  4.99991480e-01]
 [-6.18523605e-01  4.18580249e-01  5.99943355e-01]
 [-6.93072473e-01 -3.06948711e-01  7.00021184e-01]
 [ 7.79217419e-01  2.20846133e-01 -8.00063553e-01]
 [ 8.80271608e-01 -8.02112791e-02 -6.03290704e-05]]
UPDATE WEIGHT

[[ 0.1024255   0.90754514 -0.10997064]
 [-0.19974839  0.7818869   0.2178615 ]
 [ 0.33735962 -0.72101007  0.28365045]
 [ 0.39685228  0.59547424 -0.39232652]
 [ 0.48695985  0.50280263  0.51023753]
 [-0.61752806  0.41857947  0.59894859]
 [-0.67196698 -0.30696526  0.67893224]
 [ 0.75572263  0.22086455 -0.77658718]
 [ 0.89271588 -0.08022103 -0.01249485]]
UPDATE WEIGHT
```

```
[[ 0.09782506  0.90467606 -0.10250111]
 [-0.20733912  0.7771529   0.23018622]
 [ 0.32659459 -0.72772373  0.30112914]
 [ 0.39561016  0.59469959 -0.39030975]
 [ 0.47619481  0.49608896  0.52771622]
 [-0.61992029  0.41708754  0.60283275]
 [-0.67826959 -0.3108959   0.68916549]
 [ 0.74757984  0.21578627 -0.76336611]
 [ 0.88986361 -0.08199987 -0.00786374]]
FINAL ERROR : 2.7316518286333373

...


80.0 %
======================= ITERATION 8 =======================
UPDATE WEIGHT

[[ 0.11510315  0.91664636 -0.13174951]
 [-0.30098268  0.68559872  0.41538396]
 [ 0.4603772  -0.84086845  0.28049125]
 [ 0.35364887  0.57405806 -0.32770692]
 [ 0.34230657  0.4701641   0.68752933]
 [-0.69245478  0.4768793   0.61557547]
 [-0.54115621 -0.35775879  0.598915  ]
 [ 0.47139289  0.26478417 -0.53617707]
 [ 0.8885214  -0.0194832  -0.06903819]]
UPDATE WEIGHT

[[ 0.12398801  0.91661448 -0.14060249]
 [-0.31688658  0.6856558   0.43123079]
 [ 0.47503722 -0.84092106  0.26588384]
 [ 0.34680752  0.57408261 -0.32089013]
 [ 0.33315517  0.47019694  0.69664789]
 [-0.69156629  0.47687612  0.61469018]
 [-0.52232031 -0.35782638  0.58014669]
 [ 0.45042462  0.26485942 -0.51528405]
 [ 0.89962747 -0.01952306 -0.08010441]]
UPDATE WEIGHT

[[ 0.12177184  0.91499852 -0.13677036]
 [-0.32054326  0.68298947  0.43755379]
 [ 0.46985138 -0.84470239  0.27485101]
 [ 0.34620916  0.5736463  -0.31985546]
 [ 0.32796933  0.46641561  0.70561506]
 [-0.6927187   0.47603582  0.61668288]
 [-0.52535646 -0.36004024  0.5853967 ]
 [ 0.446502    0.26199919 -0.50850119]
 [ 0.89825345 -0.02052495 -0.07772849]]
FINAL ERROR : 0.9543661079889602


 90.0 %
```

```
====================== ITERATION 9 ======================
UPDATE WEIGHT

[[ 0.12030864  0.91648879 -0.13679743]
 [-0.31703159  0.67941285  0.43761874]
 [ 0.47391906 -0.84884532  0.27492625]
 [ 0.34708707  0.57275214 -0.31983922]
 [ 0.32742794  0.46696701  0.70560505]
 [-0.69631816  0.47970186  0.6166163 ]
 [-0.52401032 -0.36141128  0.5854216 ]
 [ 0.44246358  0.26611231 -0.50857589]
 [ 0.89441987 -0.01662047 -0.0777994 ]]
UPDATE WEIGHT

[[ 0.12876987  0.91645252 -0.14522238]
 [-0.33217718  0.67947777  0.45269941]
 [ 0.48788008 -0.84890516  0.26102508]
 [ 0.34057193  0.57278007 -0.313352  ]
 [ 0.31871288  0.46700437  0.71428275]
 [-0.69547204  0.47969823  0.61577381]
 [-0.50607253 -0.36148817  0.5675607 ]
 [ 0.42249509  0.2661979  -0.48869299]
 [ 0.9049964  -0.01666581 -0.0883306 ]]
UPDATE WEIGHT

[[ 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]]
FINAL ERROR : 0.8224087756463518

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[ 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]])]
EXPECTED 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]]]
SSE: 1.9504576812662859e-16
PASSED: True
```

g. **Softmax_two_layer.json**

```
...

53.0 %
======================= ITERATION 106 =======================
UPDATE WEIGHT

[[-0.28729516 -0.28829239 -0.70513538  0.42017911]
 [-0.5790214  -1.18375843 -1.34179685  0.69487267]
 [-0.41325798  1.51025096 -0.97583149 -1.30258562]]

[[-1.71810891  1.73810891]
 [-0.50304062  0.48304062]
 [ 1.25595778 -1.23595778]
 [-1.16883532  1.14883532]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

[[-0.28729516 -0.28823949 -0.70513538  0.42017911]
 [-0.5790214  -1.18379599 -1.34179685  0.69487267]
 [-0.41325798  1.51039908 -0.97583149 -1.30258562]]

[[-1.71808768  1.73808768]
 [-0.50304062  0.48304062]
 [ 1.25605927 -1.23605927]
 [-1.16883532  1.14883532]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

[[-0.2868146  -0.28945403 -0.70400586  0.42017911]
 [-0.58018918 -1.18084466 -1.34454158  0.69487267]
 [-0.4133541   1.51064199 -0.97605739 -1.30258562]]

[[-1.71857503  1.73857503]
 [-0.5036266   0.4836266 ]
 [ 1.25494504 -1.23494504]
 [-1.17017583  1.15017583]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

[[-0.2868146  -0.28945083 -0.70400586  0.42017911]
 [-0.58018918 -1.18085073 -1.34454158  0.69487267]
 [-0.4133541   1.51065037 -0.97605739 -1.30258562]]
```

```
[[-1.71857375  1.73857375]
 [-0.5036266   0.4836266 ]
 [ 1.25495263 -1.23495263]
 [-1.17017583  1.15017583]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

[[-0.28747623 -0.28778214 -0.70556092  0.42017911]
 [-0.57848215 -1.18515595 -1.34052954  0.69487267]
 [-0.41430024  1.51303659 -0.97828112 -1.30258562]]

[[-1.71790356  1.73790356]
 [-0.50321177  0.48321177]
 [ 1.25824816 -1.23824816]
 [-1.16925825  1.14925825]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

[[-0.28730211 -0.28822282 -0.70515165  0.42017911]
 [-0.5790794  -1.1836444  -1.34193332  0.69487267]
 [-0.41434377  1.51314676 -0.97838344 -1.30258562]]

[[-1.71808008  1.73808008]
 [-0.50352956  0.48352956]
 [ 1.25764816 -1.23764816]
 [-1.16998852  1.14998852]
 [ 1.08926832 -1.06926832]]
UPDATE WEIGHT

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

[[-1.7177254   1.7377254 ]
 [-0.50352956  0.48352956]
 [ 1.25764816 -1.23764816]
 [-1.16998784  1.14998784]
 [ 1.0907634  -1.0707634 ]]
UPDATE WEIGHT

[[-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 ]]
FINAL ERROR : 0.009936242476110798

=======================COMPARE=======================
STOPPED BY: error_threshold
```

```
EXPECTED STOPPED BY: error_threshold
FINAL WEIGHTS:
[array([[-0.28730211, -0.28822282, -0.70597451,  0.42094471],
       [-0.5790794 , -1.1836444 , -1.34287961,  0.69575311],
       [-0.41434377,  1.51314676, -0.97649086, -1.3043465
]]), array([[-1.72078607,  1.74078607],
       [-0.50352956,  0.48352956],
       [ 1.25764816, -1.23764816],
       [-1.16998784,  1.14998784],
       [ 1.0907634 , -1.0707634 ]])]
EXPECTED 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]],
[[-1.72078607, 1.74078607], [-0.50352956, 0.48352956],
[1.25764816, -1.23764816], [-1.16998784, 1.14998784],
[1.0907634, -1.0707634]]]
SSE: 1.584550445891344e-16
PASSED: True
```

h. **Mlp.json**

```
0.0 %
======================= ITERATION 0 =======================
UPDATE WEIGHT

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

[[ 0.2748    0.188   ]
 [ 0.435904 -0.53168 ]
 [ 0.68504   0.7824  ]]
FINAL ERROR : 0.160644

=======================COMPARE=======================
STOPPED BY: max_iteration
EXPECTED STOPPED BY: max_iteration
FINAL WEIGHTS:
[array([[ 0.08592 ,  0.32276 ],
       [-0.33872 ,  0.46172 ],
       [ 0.449984,  0.440072]]), array([[ 0.2748  ,  0.188
],
       [ 0.435904, -0.53168 ],
       [ 0.68504 ,  0.7824  ]])]
EXPECTED FINAL WEIGHTS:
[[[0.08592, 0.32276], [-0.33872, 0.46172], [0.449984,
0.440072]], [[0.2748, 0.188], [0.435904, -0.53168], [0.68504,
0.7824]]]
SSE: 2.1570415377137042e-32
PASSED: True
```

4. Perbandingan dengan penggunaan library Keras

Dengan menggunakan kelas yang dibuat oleh kami, hasil prediksinya sebagai berikut

```
Final weights: [array([[ 2.34803198e+00,  3.31818845e-03,
-2.22067821e-01,
        -1.06822373e-01],
       [ 3.09854006e-01,  1.10063142e+00,  8.83051397e-01,
        -6.12178746e-01],
       [-3.94741025e-01,  3.26006575e-01,  1.76901627e-01,
        -9.83636762e-01],
       [ 4.17980683e-01,  7.55054157e-01,  1.13559407e+00,
         3.68518636e+00],
       [ 3.14009906e-01,  9.54684022e-01,  8.25708011e-01,
         3.02920077e+00]]), array([[ 3.47351397, -1.22846376,
-2.89450051],
       [-2.93050777,  2.61830779,  0.5478958 ],
       [-1.21281619, -0.1677414 ,  0.62438257],
       [-0.5009981 , -0.9350211 , -0.18407121],
       [-0.01028354, -1.31285612,  2.46490323]])]
Final output: [[1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.29022782e-07 6.20025027e-08 2.66482021e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [7.69447361e-07 1.25752172e-08 1.90842879e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.04967068e-06 9.52818289e-09 1.80077527e-09]
 [1.57265642e-09 3.18076364e-06 6.07464229e-09]
 [1.71520383e-09 2.94349926e-06 5.97689464e-09]
 [1.60000895e-09 3.13213611e-06 6.05509034e-09]
 [3.99005048e-09 1.38440601e-06 5.10414897e-09]
 [1.97539493e-09 2.59453995e-06 5.82113461e-09]
 [5.32964223e-09 1.06890051e-06 4.83524457e-09]
 [3.32149350e-09 1.63088570e-06 5.28218961e-09]
 [1.70651785e-09 2.95688161e-06 5.98257064e-09]
 [7.53664112e-09 7.84311858e-07 4.53194189e-09]
 [2.64764229e-09 1.99713187e-06 5.51093102e-09]
 [7.33671471e-13 4.87532190e-10 2.44887655e-01]
 [1.57445185e-12 7.02762100e-09 1.47222660e-03]
 [1.71468743e-10 4.59802046e-08 2.91904736e-04]
 [5.24546731e-13 3.55109329e-10 2.39605306e-01]
 [5.68584325e-13 1.73212666e-10 5.02660597e-01]
 [2.10076924e-12 4.14281078e-09 7.23641988e-03]
 [2.64558870e-11 9.51566536e-08 3.32680028e-04]
 [1.48852726e-11 2.82230412e-08 1.47734483e-04]
 [1.61621780e-11 1.41892537e-09 3.32516832e-03]
 [3.28096271e-10 1.02162856e-07 8.79757161e-06]]
y_pred: [[1 0 0]
 [1 0 0]
```

```
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [1 0 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]]
accuracy: 100.0 %
```

Dengan menggunakan model Keras

```
Accuracy Keras Model: 100.0 %
Weights: [[ 0.5220537   0.06257433  1.1460109  -0.25308472]
 [ 0.3773468  -0.00195586  1.7266299  -0.13540508]
 [ 1.0012404   1.3606733  -0.3789903   1.2328194 ]
 [ 0.24429248  1.0870359  -0.15793945  1.4911242 ]]
Biases: [ 0.5939091  -0.1759873   0.991088   -0.17164637]
Weights: [[-0.6132058   0.76396257  0.08493862]
 [-1.2331151  -0.3438353   0.8207773 ]
 [ 0.7767202  -0.5056506  -1.8911585 ]
 [-0.09089827 -0.43311146  1.6657734 ]]
Biases: [-0.78794897  0.59585655 -0.45735636]
```

Pada hasil di atas dapat dilihat bahwa akurasi pada keras dapat mencapai 100%. Kedua model menghasilkan matriks bobot akhir yang berbeda tetapi keduanya dapat memisahkan kelas pada dataset dengan sempurna dilihat dari hasil akurasinya. Hal ini terjadi karena dengan program yang dibuat oleh kami memiliki pembelajaran yang berbeda dari model Keras. Meskipun begitu, hasil untuk memisahkan kelas-kelas dalam dataset berhasil dan mencapai akurasi 100% karena model yang dibuat oleh kami memiliki parameter yang dapat dikendalikan oleh kami dan model keras dapat menemukan parameter yang optimal untuk mencapai akurasi 100%.

5. Pembagian tugas setiap anggota kelompok

NIM	Nama	Tugas
13521120	Febryan Arota Hia	Laporan
13521132	Dhanika Novlisariyanti	Laporan
13521153	Made Debby Almadea Putri	Fungsi softmax pada output layer, fungsi hidden layer, pemrosesan dan perhitungan dataset iris
13521155	Kandida Edgina Gunawan	Fungsi relu, linear, dan sigmoid output layer, Kelas Layer, Kelas BackwardPropagation, pemrosesan dan perhitungan dataset iris