TubesB_13519096

March 26, 2022

1 Tugas Besar B - IF3270 Pembelajaran Mesin

Authors:

1. 13519096 Girvin Junod
2. 13519116 Jeane Mikha Erwansyah
3. 13519131 Hera Shafira
4. 13519188 Jeremia Axel

1.1 Install Libraries

[ ]:
```
!pip install icecream
!pip install networkx
!pip install pandas
!pip install numpy
!pip install matplotlib
```

1.2 Load libraries

[2]:
```
import pandas as pd
import os, subprocess, sys
import json, math, typing, copy
import numpy as np, networkx as nx, matplotlib as plt
from icecream import ic
```

1.3 Enums

[3]:
```
class ActivationFunction:
        SIGMOID = "sigmoid"
        RELU = "relu"
        SOFTMAX = "softmax"
        LINEAR = "linear"
```

1.4 Utility Functions

[4]:
```
class Utils:
        @staticmethod
        def get_activation_func(activation_func: str):
                if activation_func == 'sigmoid':
                  return ActivationFunction.SIGMOID
                elif activation_func == 'linear':
                  return ActivationFunction.LINEAR
                elif activation_func == 'relu':
                  return ActivationFunction.RELU
                elif activation_func == 'softmax':
                  return ActivationFunction.SOFTMAX
        @staticmethod
        def install(package):
                subprocess.check_call([sys.executable, "-m", "pip", "install",␣
 ↪''.join(package)])
```

[5]:
```
class Activations:
  @staticmethod
  def sigmoid(x):
    return 1/(1+np.exp(-x))
  @staticmethod
  def relu(x):
    return np.maximum(0, x)
  @staticmethod
  def linear(x):
    return x
  @staticmethod
  def softmax(x):
    e_x = np.exp(x-np.max(x))
    return e_x/e_x.sum(axis=1).reshape(-1,1)
  @staticmethod
  def d_sigmoid(x):
    return Activations.sigmoid(x) * (1 - Activations.sigmoid(x))
  @staticmethod
  def d_linear(x):
    return np.ones(x.shape)
  @staticmethod
  def d_relu(x):
    return (x>=0).astype(int)
  @staticmethod
  def d_softmax(x, y):
    # x itu de_dnet
    x[np.arange(y.flatten().shape[0]), y.flatten()] = -(1-x[np.arange(y.
 ↪flatten().shape[0]), y.flatten()])
    return x*-1
```

1.5 Layer Class

[6]:
```
class Layer:
        def __init__(self, prev_nodes: int, num_nodes: int, activation_func):
                self.weights = np.random.standard_normal((prev_nodes,␣
 ↪num_nodes))
                self.weights = np.r_[np.zeros((1, num_nodes)), self.weights]
                self.num_nodes = num_nodes
                self.activation_func = activation_func
```

1.6 Graph Class

[7]:
```
class Graph:
        def __init__(self, input_count, n_layers, n_neurons, activation_funcs):
                self.layers = []
                self.n_layer = n_layers
                for i in range(n_layers):
                        if (i==0):
                                new_layer = Layer(input_count, n_neurons[i],␣
 ↪activation_funcs[i])
                        else:
                                new_layer = Layer(self.layers[i-1].num_nodes,␣
 ↪n_neurons[i], activation_funcs[i])
                        self.add_layer(new_layer)

                self.output_activation = self.layers[len(self.layers)-1].
 ↪activation_func

        def add_layer(self, layer: Layer):
                self.layers.append(layer)

        def get_layer_output(self, layer: Layer, inputs: np.array):
                layer.inputs = np.c_[np.ones((inputs.shape[0], 1)), inputs]
                layer.net_value = np.dot(layer.inputs, layer.weights)

                if layer.activation_func == ActivationFunction.SIGMOID:
                         layer.output = Activations.sigmoid(layer.net_value)
                elif layer.activation_func == ActivationFunction.RELU:
                        layer.output = Activations.relu(layer.net_value)
                elif layer.activation_func == ActivationFunction.LINEAR:
                        layer.output = Activations.linear(layer.net_value)
                elif layer.activation_func == ActivationFunction.SOFTMAX:
                        layer.output = Activations.softmax(layer.net_value)

        def predict(self, x: np.ndarray):
                for i in range(len(self.layers)):
                        if i==0:
```

```
                                self.get_layer_output(self.layers[i], x)
                        else:
                                self.get_layer_output(self.layers[i], self.
↪layers[i-1].output)
                return self.layers[len(self.layers)-1].output

        def get_layer_deriv(self, delta: np.ndarray, lr: float, layer: Layer, y:
↪ np.ndarray = None):
                if y is not None:
                        if layer.activation_func == ActivationFunction.SIGMOID:
                                de_dnet = Activations.d_sigmoid(layer.
↪net_value) * (y-layer.output)
                        elif layer.activation_func == ActivationFunction.RELU:
                                de_dnet = Activations.d_relu(layer.net_value) *␣
↪(y-layer.output)
                        elif layer.activation_func == ActivationFunction.LINEAR:
                                de_dnet = Activations.d_linear(layer.net_value)␣
↪* (y-layer.output)
                        elif layer.activation_func == ActivationFunction.
↪SOFTMAX:
                                de_dnet = Activations.d_softmax(layer.output, y)
                else:
                        if layer.activation_func == ActivationFunction.SIGMOID:
                                de_dnet = delta * Activations.d_sigmoid(layer.
↪net_value)
                        elif layer.activation_func == ActivationFunction.RELU:
                                de_dnet = delta * Activations.d_relu(layer.
↪net_value)
                        elif layer.activation_func == ActivationFunction.LINEAR:
                                de_dnet = delta * Activations.d_linear(layer.
↪net_value)

                deriv = np.dot(np.transpose(de_dnet), layer.inputs)
                grad = np.dot(de_dnet, np.transpose(layer.weights))[:,1:]
                layer.weights += lr*np.transpose(deriv)
                return grad

        def backpropagate(self, learning_rate, y):
                delta = self.get_layer_deriv(None, learning_rate, self.
↪layers[len(self.layers)-1], y)
                for i in range(len(self.layers)-2,-1,-1):
                        delta = self.get_layer_deriv(delta, learning_rate, self.
↪layers[i])

        def error(self, yhat: np.ndarray, y: np.ndarray, activation_func:␣
↪ActivationFunction):
```

```
            if activation_func == ActivationFunction.SOFTMAX:
                return -np.log(yhat[np.arange(yhat.shape[0]), y.
↪flatten()]).sum()/yhat.shape[0]
            else:
                return np.sum(np.square(y-yhat))/2

    def train(self, x_train: np.ndarray, y_train: np.ndarray, lr,␣
↪err_thresh, batch_size, max_iter=10000, print_per_iter=1000):
            count_iter = 0
            while True:
                err = 0
                all_batch_x = []
                all_batch_y = []
                n_batches = x_train.shape[0]//batch_size

                for i in range(n_batches):
                    mini_batch_x=x_train[i*batch_size:
↪(i+1)*batch_size,:]
                    all_batch_x.append(mini_batch_x)

                    mini_batch_y=y_train[i*batch_size:
↪(i+1)*batch_size,:]
                    all_batch_y.append(mini_batch_y)

                if x_train.shape[0]%batch_size!=0:
                    mini_batch_x=x_train[(i+1)*batch_size:,:]
                    all_batch_x.append(mini_batch_x)

                    mini_batch_y=y_train[(i+1)*batch_size:,:]
                    all_batch_y.append(mini_batch_y)

                    n_batches+=1

                for i in range(n_batches):
                    yhat = self.predict(all_batch_x[i])
                    err += self.error(yhat, all_batch_y[i], self.
↪output_activation)
                    self.backpropagate(lr, all_batch_y[i])

                count_iter+= 1
                if count_iter % print_per_iter == 0:
                    print(f'Iterasi {count_iter}: error {err:.7f}')
                if count_iter >= max_iter:
                    print("Iterasi maksimum")
                    break
                if err <= err_thresh:
                    print("Mencapai nilai error di bawah batas")
```

```
                    break
            print(f'Berakhir pada iterasi: {count_iter}')
            print(f'Nilai error final: {err}')

    def display_table(self):
            for i,layer in enumerate(self.layers):
                    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
                    print(f'Layer {i}:')
                    print()
                    print("Weights:")
                    print(layer.weights)
                    print()
                    print(f"Activation Function: {layer.activation_func}")
                    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
                    print()
            print(f"Jumlah hidden layer: {self.n_layer - 1}")
```

1.7 Main Function

Fungsi aktivasi Softmax hanya dipakai untuk layer output. Berikut adalah hasil program untuk fungsi aktivasi sigmoid, linear, dan ReLU beserta perbandingannya dengan hasil sklearn MLPClassifier dengan fungsi aktivasi dan learning rate yang sama.

1.7.1 Load Dataset

[8]:
```
from sklearn import datasets
from sklearn.neural_network import MLPClassifier
iris = datasets.load_iris()
x, y = iris.data, iris.target
```

1.7.2 Sigmoid Activation

Create Neural Network and Train

[25]:
```
grafSigmoid = Graph(len(iris.feature_names), 2, [3, len(iris.target_names)],␣
 ↪['sigmoid', 'softmax'])
#x, y, learning rate, error threshold, batch size, max iter?, print per iter?
grafSigmoid.train(x, y.reshape(-1,1), 1e-2, 2e-2,50)
```

```
Iterasi 1000: error 1.6630800
Iterasi 2000: error 0.2583772
Iterasi 3000: error 0.2131004
Iterasi 4000: error 0.1852525
Iterasi 5000: error 0.1621115
Iterasi 6000: error 0.1559171
Iterasi 7000: error 0.1446728
Iterasi 8000: error 0.1473589
Iterasi 9000: error 0.1370553
```

```
Iterasi 10000: error 0.1375363
Iterasi maksimum
Berakhir pada iterasi: 10000
Nilai error final: 0.1375362946368754
```

Show Model

[26]:
```
grafSigmoid.display_table()
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 0:

Weights:
[[  2.74918621   0.15710989 -10.02987639]
 [ -3.59834072   0.15456866 -11.45842846]
 [  7.70916153  -2.51921782  -1.54086399]
 [ -0.27510784   1.31759582  14.03993498]
 [ -3.47608161  -1.66360663  11.48571861]]

Activation Function: sigmoid
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 1:

Weights:
[[ -6.90245537   7.67772312  -0.77526775]
 [ 18.65579164  -1.63012924 -20.47740615]
 [ -1.73111449   1.77378346   0.9413586 ]
 [ -4.98029069  -5.00669068   7.53054733]]

Activation Function: softmax
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Jumlah hidden layer: 1
```

Hasil Prediksi

[27]:
```
yhat_1 = grafSigmoid.predict(x)
yhat = np.argmax(yhat_1, axis=1)
print(iris.target_names[yhat])
```

```
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
```

```
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'virginica' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'virginica'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
'virginica' 'virginica' 'virginica']
```

MLPClassifier Result

[28]:
```
sklearn_sigmoid = MLPClassifier(hidden_layer_sizes=(3,), activation='logistic',␣
↪max_iter=10000, batch_size=50, learning_rate='constant',␣
↪learning_rate_init=1e-2)
sklearn_sigmoid.fit(x,y)
iris.target_names[sklearn_sigmoid.predict(x)]
```

[28]:
```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica', 'virginica',
```

```
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'versicolor', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
'virginica', 'virginica', 'virginica'], dtype='<U10')
```

1.7.3 ReLU Activation

Create Neural Network and Train

[29]:
```
grafRelu = Graph(len(iris.feature_names), 2, [3, len(iris.target_names)],␣
 ↪['relu', 'softmax'])
#x, y, learning rate, error threshold, batch size, max iter?, print per iter?
grafRelu.train(x, y.reshape(-1,1), 1e-4, 2e-2,50)
```

```
Iterasi 1000: error 1.2127796
Iterasi 2000: error 0.7749713
Iterasi 3000: error 0.5844361
Iterasi 4000: error 0.4820500
Iterasi 5000: error 0.4171609
Iterasi 6000: error 0.3726297
Iterasi 7000: error 0.3409879
Iterasi 8000: error 0.3170229
Iterasi 9000: error 0.2982381
Iterasi 10000: error 0.2830913
Iterasi maksimum
Berakhir pada iterasi: 10000
Nilai error final: 0.28309126071707835
```

Show Model

[30]:
```
grafRelu.display_table()
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 0:

Weights:
[[ 1.38452176  0.          0.        ]
 [ 1.86039067 -0.30034126 -0.92095352]
 [ 1.8655005   0.74222744  0.32894113]
 [-2.84158011 -1.66083625 -0.61676459]
 [-2.32330972 -0.28552651  0.2470814 ]]

Activation Function: relu
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 1:

Weights:
[[-4.54403955  0.91263322  3.63140633]
 [ 2.10624908  1.27707952 -3.03125637]
 [-0.35383836 -0.2110127  -1.05054517]
 [ 1.88911902 -1.11631424 -0.01339836]]

Activation Function: softmax
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Jumlah hidden layer: 1
```

Hasil Prediksi

[31]:
```
yhat_1 = grafRelu.predict(x)
yhat = np.argmax(yhat_1, axis=1)
print(iris.target_names[yhat])
```

```
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'virginica'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica']
```

MLPClassifier Result

[32]:
```python
sklearn_relu = MLPClassifier(hidden_layer_sizes=(3,), activation='relu', max_iter=10000, batch_size=50, learning_rate='constant', learning_rate_init=1e-4)
sklearn_relu.fit(x,y)
iris.target_names[sklearn_relu.predict(x)]
```

[32]:
```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'virginica', 'versicolor', 'virginica', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica'], dtype='<U10')
```

1.7.4 Linear Activation

Create Neural Network and Train

[33]:
```python
grafLinear = Graph(len(iris.feature_names), 3, [3, 3, len(iris.target_names)], ['linear', 'sigmoid', 'softmax'])
#x, y, learning rate, error threshold, batch size, max iter?, print per iter?
grafLinear.train(x, y.reshape(-1,1), 1e-4, 2e-2,50)
```

```
Iterasi 1000: error 1.9378099
Iterasi 2000: error 0.8478677
Iterasi 3000: error 0.5984444
Iterasi 4000: error 0.4753763
Iterasi 5000: error 0.4034580
Iterasi 6000: error 0.3568364
Iterasi 7000: error 0.3244343
Iterasi 8000: error 0.3006924
Iterasi 9000: error 0.2825906
Iterasi 10000: error 0.2683712
Iterasi maksimum
Berakhir pada iterasi: 10000
Nilai error final: 0.2683711525638142
```

Show Model

[34]:
```
grafLinear.display_table()
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 0:

Weights:
[[ 1.37230779  0.1401563   0.29778531]
 [ 1.33262776 -0.00731426 -0.33150972]
 [ 1.26766773  1.1198438  -0.37250186]
 [-1.95858695  0.63673126 -0.68699408]
 [-1.84924861 -0.55419071 -1.04302786]]

Activation Function: linear
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 1:

Weights:
[[-3.64659024e-05 -4.93899927e-01  3.18439884e-02]
 [ 3.18984793e-01 -3.50145774e+00  1.55826242e+00]
 [ 1.46588805e-02 -3.99120544e-01 -6.33778232e-01]
 [-1.91885725e+00 -6.62989715e-01  1.07772499e+00]]

Activation Function: sigmoid
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Layer 2:

Weights:
[[-0.613368    1.2106436  -0.5972756 ]
```

```
 [-0.51619637  1.27655383 -0.80533024]
 [-2.89714963 -2.89298769  4.29552799]
 [ 4.90591372 -2.59394055 -0.67624181]]

Activation Function: softmax
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Jumlah hidden layer: 2
```

Hasil Prediksi

[35]:
```
yhat_1 = grafLinear.predict(x)
yhat = np.argmax(yhat_1, axis=1)
print(iris.target_names[yhat])
```

```
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'virginica'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica']
```

MLPClassifier Result

[36]:
```
from sklearn.neural_network import MLPClassifier
iris_classifier_sklearn = MLPClassifier(hidden_layer_sizes=(3,),␣
 ↪activation='identity', max_iter=10000, batch_size=50,␣
 ↪learning_rate='constant', learning_rate_init=1e-4)
iris_classifier_sklearn.fit(x,y)
iris.target_names[iris_classifier_sklearn.predict(x)]
```

[36]:
```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica'], dtype='<U10')
```