Tugas Besar C - IF3270 Pembelajaran Mesin

Authors:

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

Install Libraries

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

Load libraries

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

Enums

In [ ]:
```
class ActivationTypes:
        SIGMOID = "sigmoid"
        RELU = "relu"
        SOFTMAX = "softmax"
        LINEAR = "linear"
```

Utility Functions

In [ ]:
```
class Utils:
        @staticmethod
        def get_activation_func(activation_type: str):
                if activation_type == 'sigmoid':
                  return ActivationTypes.SIGMOID
                elif activation_type == 'linear':
                  return ActivationTypes.LINEAR
                elif activation_type == 'relu':
                  return ActivationTypes.RELU
                elif activation_type == 'softmax':
                  return ActivationTypes.SOFTMAX

        @staticmethod
        def install(package):
                subprocess.check_call([sys.executable, "-m", "pip", "install", ''.join(package)])

        @staticmethod
        def listdivision(arr1, arr2):
                '''
                Fungsi untuk pembagian list dengan panjang sama
                Asumsi untuk zero division nilai yang diberikan adalah 0
                '''
                ret= []
                for i in range(len(arr1)):
                        if (arr2[i] !=0):
                                ret.append(arr1[i]/arr2[i])
                        else:
                                ret.append(0)
                return ret

        @staticmethod
        def confusionMatrix(true, pred):
                '''
                true: array of true values
                pred: array of predicted values
                '''
                num_class = len(np.unique(true))
                result = np.zeros((num_class, num_class))

                for i in range(len(true)):
                        result[true[i]][pred[i]] += 1

                return result

        @staticmethod
        def getTP(cm):
                #total diagonal cm
                return np.diag(cm)

        @staticmethod
        def getTN(cm):
                #Semua kecuali cell kelas dan row col dari kelas
                TN = []
                for i in range(len(cm)):
                                temp = np.delete(cm, i, 0)
                                temp = np.delete(temp, i, 1)
                                TN.append(sum(sum(temp)))
                return TN

        @staticmethod
        def getFP(cm):
                #total di kolom kecuali cell class
```

```
                return np.sum(cm, axis=0) - Utils.getTP(cm)

        @staticmethod
        def getFN(cm):
                #total di row kecuali cell kelas
                return np.sum(cm, axis=1) - Utils.getTP(cm)

        @staticmethod
        def accuracy(true, pred) -> float:
                #true/total
                benar = 0
                for i in range(len(true)):
                        if (true[i] == pred[i]):
                                benar+=1

                return benar/len(true)

        @staticmethod
        def precision(true, pred) -> float:
                #true positive/(truepositive + false positive)
                #true positive/(total pred positive)
                num_class = len(np.unique(true))
                total = 0
                cm = Utils.confusionMatrix(true, pred)
                TP = Utils.getTP(cm)
                FP = Utils.getFP(cm)
                prec= Utils.listdivision(TP,(TP+FP))
                total = np.sum(prec)/num_class
                return total

        @staticmethod
        def recall(true, pred) -> float:
                #true positive/(true positive + false negative)
                #true positive/(total actual positive)
                num_class = len(np.unique(true))
                total = 0
                cm = Utils.confusionMatrix(true, pred)
                TP = Utils.getTP(cm)
                FN = Utils.getFN(cm)
                rec = Utils.listdivision(TP,(TP+FN))

                total = np.sum(rec)/num_class
                return total

        @staticmethod
        def f1(true, pred) -> float:
                #2*(precision*recall)/(precision+recall)
                #TP/(TP+ 0.5(FP+FN))

                cm = Utils.confusionMatrix(true, pred)
                TP = Utils.getTP(cm)
                FN = Utils.getFN(cm)
                FP = Utils.getFP(cm)
                f = TP/(TP+ 0.5*(FP+FN))
                return np.sum(f)/len(np.unique(true))
```

In [ ]:
```
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
```

Layer Class

In [ ]:
```
class Layer:
        def __init__(self, num_nodes: int, activation_func, weights=None, prev_nodes=None):
                if weights!=None:
                        self.weights = weights
                else:
                        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

        def layer_dict(self):
                return {
                        "weights" : self.weights.tolist(),
                        "num_nodes":self.num_nodes,
                        "activation_function":self.activation_func
                }
```

Graph Class

In [ ]:
```
class Graph:
        def __init__(self, input_count=None, n_layers=None, n_neurons=None, activation_funcs=None, file_name=None):
                '''
                input_count: int
```

```
                Jumlah elemen input
        n_layers: int
                Jumlah layer
        n_neurons: list[int]
                Jumlah neuron untuk tiap layer
        activation_func: list[ActivationTypes]
                Fungsi aktivasi untuk tiap layer
        '''
        self.layers = []
        self.loss = []
        if file_name!=None:
                self.load_graph(file_name)
        else:
                if n_layers is not None:
                        self.n_layer = n_layers
                        for i in range(self.n_layer):
                                if (i==0):
                                        new_layer = Layer(n_neurons[i], activation_funcs[i], None, input_count)
                                else:
                                        new_layer = Layer(n_neurons[i], activation_funcs[i], None, self.layers[i-1].num_nodes)
                                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 == ActivationTypes.SIGMOID:
                layer.output = Activations.sigmoid(layer.net_value)
        elif layer.activation_func == ActivationTypes.RELU:
                layer.output = Activations.relu(layer.net_value)
        elif layer.activation_func == ActivationTypes.LINEAR:
                layer.output = Activations.linear(layer.net_value)
        elif layer.activation_func == ActivationTypes.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 == ActivationTypes.SIGMOID:
                        de_dnet = Activations.d_sigmoid(layer.net_value) * (y-layer.output)
                elif layer.activation_func == ActivationTypes.RELU:
                        de_dnet = Activations.d_relu(layer.net_value) * (y-layer.output)
                elif layer.activation_func == ActivationTypes.LINEAR:
                        de_dnet = Activations.d_linear(layer.net_value) * (y-layer.output)
                elif layer.activation_func == ActivationTypes.SOFTMAX:
                        de_dnet = Activations.d_softmax(layer.output, y)
        else:
                if layer.activation_func == ActivationTypes.SIGMOID:
                        de_dnet = delta * Activations.d_sigmoid(layer.net_value)
                elif layer.activation_func == ActivationTypes.RELU:
                        de_dnet = delta * Activations.d_relu(layer.net_value)
                elif layer.activation_func == ActivationTypes.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: ActivationTypes):
        if activation_func == ActivationTypes.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=-1):
        '''
        x_train: np.ndarray
                Input training
        y_train: np.ndarray
                Output dari x_train
        lr: float
                Learning rate
        err_thresh: float
                Error threshold
        max_iter: int
                Jumlah maksimum iterasi
        print_per_iter: int
                Print nilai error setiap berapa iterasi
        '''
        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])


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

        def export_graph(self):
                all_dict = []
                for layer in self.layers:
                        all_dict.append(layer.layer_dict())
                json_obj = json.dumps(all_dict, indent=4)
                with open("model.json", "w") as outfile:
                        outfile.write(json_obj)

        def load_graph(self, filename: str):
                with open(filename, 'r') as openfile:
                        json_object = json.load(openfile)
                for obj in json_object:
                        new_layer = Layer(obj["num_nodes"], obj["activation_function"], obj["weights"], None)
                        self.layers.append(new_layer)

        def loss_graph(self, label="Loss"):
                plt.plot(range(0, len(self.loss)), self.loss , label=label)
                plt.title(label)
                plt.xlabel('Epoch')
                plt.ylabel('Loss')
                plt.show()
```

Main Function

Load Dataset

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

Prediksi Batch

Create Neural Network and Train

Neural Network Hasil Implementasi

In [ ]:
```
#input_count, n_layers, n_neurons, activation_funcs
graf = 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?
graf.train(x, y.reshape(-1,1), 1e-2, 2e-2,50, print_per_iter=1000)
```

```
Iterasi 1000: error 0.3458198
Iterasi 2000: error 0.1991806
Iterasi 3000: error 0.1324378
Iterasi 4000: error 0.1180149
Iterasi 5000: error 0.1132253
Iterasi 6000: error 0.1107779
Iterasi 7000: error 0.1092365
Iterasi 8000: error 0.1081487
Iterasi 9000: error 0.1073258
Iterasi 10000: error 0.1066737
Iterasi maksimum
Berakhir pada iterasi: 10000
Nilai error final: 0.10667366707501996
```

In [ ]:
```
graf.export_graph()
```

Neural Network MLPClassifier Sklearn

In [ ]:
```
sklearn= MLPClassifier(hidden_layer_sizes=(3,), activation='logistic', max_iter=10000, batch_size=50, learning_rate='constant', learning_rate_init=1e-2)
sklearn.fit(x, y)
```

Out[ ]:
```
MLPClassifier(activation='logistic', batch_size=50, hidden_layer_sizes=(3,),
              learning_rate_init=0.01, max_iter=10000)
```

Prediksi

In [ ]:
```
#hasil sendiri
yhat_1 = graf.predict(x)
yhat = np.argmax(yhat_1, axis=1)

#MLPClassifier
yhat_sklearn = sklearn.predict(x)
```

Perbandingan confusion matrix dan kinerja dengan sklearn

Pengukuran Kinerja FFNN Hasil Implementasi

In [ ]:
```
print("Fungsi Kinerja Hasil implementasi sendiri")
print("Confusion Matrix")
cm =Utils.confusionMatrix(y, yhat)
print(cm)
print(f"Accuracy Score: {Utils.accuracy(y, yhat)}")
print(f"Precision Score: {Utils.precision(y, yhat)}")
print(f"Recall Score: {Utils.recall(y,yhat)}")
print(f"F1 Score: {Utils.f1(y, yhat)}")

print()
print("Fungsi Kinerja Sklearn")
print("Confusion Matrix")
print(metrics.confusion_matrix(y,yhat))
print(f"Accuracy Score: {metrics.accuracy_score(y, yhat)}")
print(f"Precision Score: {metrics.precision_score(y, yhat, average='macro', zero_division=0)}")
print(f"Recall Score: {metrics.recall_score(y, yhat, average='macro', zero_division=0)}")
print(f"F1 Score: {metrics.f1_score(y, yhat, average='macro', zero_division=0)}")
```

```
Fungsi Kinerja Hasil implementasi sendiri
Confusion Matrix
[[50.  0.  0.]
 [ 0. 49.  1.]
 [ 0.  0. 50.]]
Accuracy Score: 0.9933333333333333
Precision Score: 0.9934640522875817
Recall Score: 0.9933333333333333
F1 Score: 0.9933326665999934

Fungsi Kinerja Sklearn
Confusion Matrix
[[50  0  0]
 [ 0 49  1]
 [ 0  0 50]]
Accuracy Score: 0.9933333333333333
Precision Score: 0.9934640522875817
Recall Score: 0.9933333333333333
F1 Score: 0.9933326665999934
```

Pengukuran Kinerja MLPCLassifier Sklearn

In [ ]:
```
print("Fungsi Kinerja Hasil implementasi sendiri")
print("Confusion Matrix")
cm =Utils.confusionMatrix(y, yhat_sklearn)
print(cm)
print(f"Accuracy Score: {Utils.accuracy(y, yhat_sklearn)}")
print(f"Precision Score: {Utils.precision(y, yhat_sklearn)}")
print(f"Recall Score: {Utils.recall(y,yhat_sklearn)}")
print(f"F1 Score: {Utils.f1(y, yhat_sklearn)}")

print()
print("Fungsi Kinerja Sklearn")
print("Confusion Matrix")
print(metrics.confusion_matrix(y,yhat_sklearn))
print(f"Accuracy Score: {metrics.accuracy_score(y, yhat_sklearn)}")
print(f"Precision Score: {metrics.precision_score(y, yhat_sklearn, average='macro', zero_division=0)}")
print(f"Recall Score: {metrics.recall_score(y, yhat_sklearn, average='macro', zero_division=0)}")
print(f"F1 Score: {metrics.f1_score(y, yhat_sklearn, average='macro', zero_division=0)}")
```

```
Fungsi Kinerja Hasil implementasi sendiri
Confusion Matrix
[[50.  0.  0.]
 [ 0. 48.  2.]
 [ 0.  1. 49.]]
Accuracy Score: 0.98
Precision Score: 0.980125383486728
Recall Score: 0.98
F1 Score: 0.97999799979998

Fungsi Kinerja Sklearn
Confusion Matrix
[[50  0  0]
 [ 0 48  2]
 [ 0  1 49]]
Accuracy Score: 0.98
Precision Score: 0.980125383486728
Recall Score: 0.98
F1 Score: 0.97999799979998
```

Pembelajaran FFNN

Skema Split Train 90%, Test 10%

In [ ]:
```
# init & config
from sklearn.model_selection import train_test_split
from sklearn import metrics

TRAIN_SIZE = 0.90
sklearn_sigmoid = MLPClassifier(hidden_layer_sizes=(3,), activation='logistic', max_iter=10000, batch_size=50, learning_rate='constant', learning_rate_init=1e-2)
graf_iris = Graph(len(iris.feature_names), 2, [3, len(iris.target_names)], ['sigmoid', 'softmax'])

# split data
x_iris_train, x_iris_test, y_iris_train, y_iris_test = train_test_split(x, y, train_size=TRAIN_SIZE)

# fit & predict
sklearn_fitted = sklearn_sigmoid.fit(x_iris_train, y_iris_train)
sklearn_prediction = sklearn_fitted.predict(x_iris_test)

graf_iris.train(x_iris_train, y_iris_train.reshape(-1,1), 1e-2, 2e-2,50)
graf_prediction_1 = graf.predict(x_iris_test)
```

```
graf_prediction = np.argmax(graf_prediction_1, axis=1)

# kinerja & confusion  matrix
yhat = np.argmax(sklearn_prediction)
```

Pengukuran Kinerja FFNN Hasil Implementasi Sendiri

In [ ]:
```
print("Fungsi Kinerja Implementasi Sendiri")
print("Confusion Matrix")
print(Utils.confusionMatrix(y_iris_test, graf_prediction))
print(f"Accuracy Score: {Utils.accuracy(y_iris_test, graf_prediction)}")
print(f"Precision Score: {Utils.precision(y_iris_test, graf_prediction)}")
print(f"Recall Score: {Utils.recall(y_iris_test, graf_prediction)}")
print(f"F1 Score: {Utils.f1(y_iris_test, graf_prediction)}")
print()
print("Fungsi Kinerja Sklearn")
print("Confusion Matrix")
print(metrics.confusion_matrix(y_iris_test, graf_prediction))
print(f"Accuracy Score: {metrics.accuracy_score(y_iris_test, graf_prediction)}")
print(f"Precision Score: {metrics.precision_score(y_iris_test, graf_prediction, average='macro', zero_division=0)}")
print(f"Recall Score: {metrics.recall_score(y_iris_test, graf_prediction, average='macro', zero_division=0)}")
print(f"F1 Score: {metrics.f1_score(y_iris_test, graf_prediction, average='macro', zero_division=0)}")
```

```
Fungsi Kinerja Implementasi Sendiri
Confusion Matrix
[[5. 0. 0.]
 [0. 6. 0.]
 [0. 0. 4.]]
Accuracy Score: 1.0
Precision Score: 1.0
Recall Score: 1.0
F1 Score: 1.0

Fungsi Kinerja Sklearn
Confusion Matrix
[[5 0 0]
 [0 6 0]
 [0 0 4]]
Accuracy Score: 1.0
Precision Score: 1.0
Recall Score: 1.0
F1 Score: 1.0
```

Pengukuran Kinerja FFNN MLPClassifier Sklearn

In [ ]:
```
print("Fungsi Kinerja Implementasi Sendiri")
print("Confusion Matrix")
print(Utils.confusionMatrix(y_iris_test, sklearn_prediction))
print(f"Accuracy Score: {Utils.accuracy(y_iris_test, sklearn_prediction)}")
print(f"Precision Score: {Utils.precision(y_iris_test, sklearn_prediction)}")
print(f"Recall Score: {Utils.recall(y_iris_test, sklearn_prediction)}")
print(f"F1 Score: {Utils.f1(y_iris_test, sklearn_prediction)}")

print()
print("Fungsi Kinerja Sklearn")
print("Confusion Matrix")
print(metrics.confusion_matrix(y_iris_test, sklearn_prediction))
print(f"Accuracy Score: {metrics.accuracy_score(y_iris_test, sklearn_prediction)}")
print(f"Precision Score: {metrics.precision_score(y_iris_test, sklearn_prediction, average='macro', zero_division=0)}")
print(f"Recall Score: {metrics.recall_score(y_iris_test, sklearn_prediction, average='macro', zero_division=0)}")
print(f"F1 Score: {metrics.f1_score(y_iris_test, sklearn_prediction, average='macro', zero_division=0)}")
```

```
Fungsi Kinerja Implementasi Sendiri
Confusion Matrix
[[5. 0. 0.]
 [0. 5. 1.]
 [0. 0. 4.]]
Accuracy Score: 0.9333333333333333
Precision Score: 0.9333333333333332
Recall Score: 0.9444444444444445
F1 Score: 0.9326599326599326

Fungsi Kinerja Sklearn
Confusion Matrix
[[5 0 0]
 [0 5 1]
 [0 0 4]]
Accuracy Score: 0.9333333333333333
Precision Score: 0.9333333333333332
Recall Score: 0.9444444444444445
F1 Score: 0.9326599326599326
```

Skema 10-fold Cross Validation

In [ ]:
```
# init & config
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score, cross_validate, KFold, StratifiedKFold

K_FOLDS = 10
kf = StratifiedKFold(n_splits=K_FOLDS)

sklearn_sigmoid = MLPClassifier(hidden_layer_sizes=(3,), activation='logistic', max_iter=10000, batch_size=50, learning_rate='constant', learning_rate_init=1e-2)
graf_iris = Graph(len(iris.feature_names), 2, [3, len(iris.target_names)], ['sigmoid', 'softmax'])

# NOTE: Yang ini versi lebih compact, tapi hasilnya ga bisa dipipeline ke fungsi score yang kita buat
# Jadi mungkin yang bakal dipake yang versi manual
# scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
# scores = cross_validate(sklearn_sigmoid, x, y, scoring=scoring, cv=kf)
# print(f'Accuracy: %.3f (%.3f)' % (mean(scores["test_accuracy"]), std(scores["test_accuracy"])))
# print(f'Precision: %.3f (%.3f)' % (mean(scores["test_precision_macro"]), std(scores["test_precision_macro"])))
# print(f'Recall: %.3f (%.3f)' % (mean(scores["test_recall_macro"]), std(scores["test_recall_macro"])))
# print(f'F1: %.3f (%.3f)' % (mean(scores["test_f1_macro"]), std(scores["test_f1_macro"])))

# Yang ini versi manual
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

utils_accuracy_scores = []
utils_precision_scores = []
```

```
utils_recall_scores = []
utils_f1_scores = []

for train_idx, test_idx in kf.split(x, y):
  # split data
  x_iris_train, x_iris_test, y_iris_train, y_iris_test = x[train_idx], x[test_idx], y[train_idx], y[test_idx]

  # fit & predict
  sklearn_sigmoid.fit(x_iris_train, y_iris_train)
  prediction_values = sklearn_sigmoid.predict(x_iris_test)

  graf_iris.train(x_iris_train, y_iris_train.reshape(-1,1), 1e-2, 2e-2,50)
  graf_prediction_1 = graf.predict(x_iris_test)
  graf_prediction = np.argmax(graf_prediction_1, axis=1)

  # Sklearn
  accuracy = metrics.accuracy_score(y_iris_test, prediction_values)
  precision = metrics.precision_score(y_iris_test, prediction_values, average='macro', zero_division=1)
  recall = metrics.recall_score(y_iris_test, prediction_values, average='macro', zero_division=1)
  f1 = metrics.f1_score(y_iris_test, prediction_values, average='macro', zero_division=1)

  accuracy_scores.append(accuracy)
  precision_scores.append(precision)
  recall_scores.append(recall)
  f1_scores.append(f1)

  # Utils
  utils_accuracy = Utils.accuracy(y_iris_test, graf_prediction)
  utils_precision = Utils.precision(y_iris_test, graf_prediction)
  utils_recall = Utils.recall(y_iris_test, graf_prediction)
  utils_f1 = Utils.f1(y_iris_test, graf_prediction)

  utils_accuracy_scores.append(utils_accuracy)
  utils_precision_scores.append(utils_precision)
  utils_recall_scores.append(utils_recall)
  utils_f1_scores.append(utils_f1)
```

Pengukuran Kinerja FFNN Hasil Implementasi Sendiri

In [ ]:
```
print("Fungsi Kinerja Implementasi Sendiri")
print(f"(Average) Accuracy Score: {mean(utils_accuracy_scores)}")
print(f"(Average) Precision Score: {mean(utils_precision_scores)}")
print(f"(Average) Recall Score: {mean(utils_recall_scores)}")
print(f"(Average) F1 Score: {mean(utils_f1_scores)}")

print()
print("Fungsi Kinerja Sklearn")
print(f"(Average) Accuracy Score: {mean(utils_accuracy_scores)}")
print(f"(Average) Precision Score: {mean(utils_precision_scores)}")
print(f"(Average) Recall Score: {mean(utils_recall_scores)}")
print(f"(Average) F1 Score: {mean(utils_f1_scores)}")
```

```
Fungsi Kinerja Implementasi Sendiri
(Average) Accuracy Score: 0.9933333333333334
(Average) Precision Score: 0.9944444444444445
(Average) Recall Score: 0.9933333333333334
(Average) F1 Score: 0.9932659932659933

Fungsi Kinerja Sklearn
(Average) Accuracy Score: 0.9933333333333334
(Average) Precision Score: 0.9944444444444445
(Average) Recall Score: 0.9933333333333334
(Average) F1 Score: 0.9932659932659933
```

Pengukuran Kinerja FFNN MLPClassifier Sklearn

In [ ]:
```
print("Fungsi Kinerja Implementasi Sendiri")
print(f"(Average) Accuracy Score: {mean(accuracy_scores)}")
print(f"(Average) Precision Score: {mean(precision_scores)}")
print(f"(Average) Recall Score: {mean(recall_scores)}")
print(f"(Average) F1 Score: {mean(f1_scores)}")

print()
print("Fungsi Kinerja Sklearn")
print(f"(Average) Accuracy Score: {mean(accuracy_scores)}")
print(f"(Average) Precision Score: {mean(precision_scores)}")
print(f"(Average) Recall Score: {mean(recall_scores)}")
print(f"(Average) F1 Score: {mean(f1_scores)}")
```

```
Fungsi Kinerja Implementasi Sendiri
(Average) Accuracy Score: 0.9800000000000001
(Average) Precision Score: 0.9811111111111112
(Average) Recall Score: 0.9800000000000001
(Average) F1 Score: 0.97993265993266

Fungsi Kinerja Sklearn
(Average) Accuracy Score: 0.9800000000000001
(Average) Precision Score: 0.9811111111111112
(Average) Recall Score: 0.9800000000000001
(Average) F1 Score: 0.97993265993266
```

Instance Baru

In [ ]:
```
new_graf = Graph(file_name="model.json")
```

In [ ]:
```
pred_values = new_graf.predict(np.array([[6.6, 3.1, 5.7, 2.3],[4.4, 2.1, 1.5, 0.3],[5.6, 2.9, 4.3, 1.5],[5.3, 3.9, 1.3, 0.5]]))
yhat_2 = np.argmax(pred_values, axis=1)

correct_answers = [2,0,1,0]
print("Confusion Matrix")
print(Utils.confusionMatrix(correct_answers, yhat_2))
print(f"Accuracy Score: {Utils.accuracy(correct_answers, yhat_2)}")
print(f"Precision Score: {Utils.precision(correct_answers, yhat_2)}")
print(f"Recall Score: {Utils.recall(correct_answers, yhat_2)}")
print(f"F1 Score: {Utils.f1(correct_answers, yhat_2)}")
```

```
Confusion Matrix
[[2. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Accuracy Score: 1.0
Precision Score: 1.0
Recall Score: 1.0
F1 Score: 1.0
```

Analisis Perbandingan *Confusion Matrix* dan Perhitungan Kinerja

Skema *Batch*

-	Confusion Matrix dan Kinerja Hasil Implementasi		Confusion Matrix dan Kinerja Sklearn	
FFNN Hasil Implementasi	*Confusion Matrix*	$\begin{bmatrix} 50. & 0. & 0. \\ 0. & 49. & 1. \\ 0. & 0. & 50. \end{bmatrix}$	*Confusion Matrix*	$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 49 & 1 \\ 0 & 0 & 50 \end{bmatrix}$
	Accuracy Score	0.9933333333333333	*Accuracy Score*	0.9933333333333333
	Precision Score	0.9934640522875817	*Precision Score*	0.9934640522875817
	Recall Score	0.9933333333333333	*Recall Score*	0.9933333333333333
	F1 Score	0.9933326665999934	*F1 Score*	0.9933326665999934
MLPClassifier Sklearn	*Confusion Matrix*	$\begin{bmatrix} 50. & 0. & 0. \\ 0. & 48. & 2. \\ 0. & 1. & 49. \end{bmatrix}$	*Confusion Matrix*	$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 48 & 2 \\ 0 & 1 & 49 \end{bmatrix}$
	Accuracy Score	0.98	*Accuracy Score*	0.98
	Precision Score	0.980125383486728	*Precision Score*	0.980125383486728
	Recall Score	0.98	*Recall Score*	0.98
	F1 Score	0.97999799979998	*F1 Score*	0.97999799979998

Skema *Split*

-	Confusion Matrix dan Kinerja Hasil Implementasi		Confusion Matrix dan Kinerja Sklearn	
FFNN Hasil Implementasi	*Confusion Matrix*	$\begin{bmatrix} 5. & 0. & 0. \\ 0. & 6. & 0. \\ 0. & 0. & 4. \end{bmatrix}$	*Confusion Matrix*	$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 4 \end{bmatrix}$
	Accuracy Score	1.0	*Accuracy Score*	1.0
	Precision Score	1.0	*Precision Score*	1.0
	Recall Score	1.0	*Recall Score*	1.0
	F1 Score	1.0	*F1 Score*	1.0
MLPClassifier Sklearn	*Confusion Matrix*	$\begin{bmatrix} 5. & 0. & 0. \\ 0. & 5. & 1. \\ 0. & 0. & 4. \end{bmatrix}$	*Confusion Matrix*	$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 1 \\ 0 & 0 & 4 \end{bmatrix}$
	Accuracy Score	0.9333333333333333	*Accuracy Score*	0.9333333333333333
	Precision Score	0.9333333333333332	*Precision Score*	0.9333333333333332
	Recall Score	0.9444444444444445	*Recall Score*	0.9444444444444445
	F1 Score	0.9326599326599326	*F1 Score*	0.9326599326599326

Berdasarkan hasil *confusion matrix* dan kinerja FFNN dan MLPClassifier Sklearn pada skema batch dan skema split, FFNN hasil implementasi kami menghasilkan prediksi yang lebih baik karena memiliki *confusion matrix* yang lebih bagus dan kinerja yang lebih tinggi. Dapat dilihat juga bahwa hasil implementasi fungsi kinerja kami menghasilkan nilai yang sesuai dengan fungsi kinerja Sklearn.