TUGAS BESAR A MACHINE LEARNING

- Arjuna Marcelino - 13519021
- Sharon Bernadetha Marbun - 13519092
- Epata Tuah - 13519120
- Giant Andreas Tambunan - 13519127

Membaca Data dan Model

In [ ]:

```
# Membaca data
import csv

def readData():
    data = []
    target_class = []
    dataFile = open('data.csv', "r")
    reader = csv.DictReader(dataFile)
    for row in reader:
        data.append([int(row["x1"]), int(row["x2"])])
        target_class.append(int(row["f"]))

    return data, target_class


# Membaca Model
# filename ex: modelsigmoid.txt
def readModel(filename):
    file = open(filename, "r")
    lines = file.readlines()

    # init array
    activation = []
    weigth = []
    bias = []

    num_atr = int(lines[0])
    i = 1
    while i < len(lines):
        splitline = lines[i].strip("\n").split(" ")
        # baris yang mengandung fungsi aktivasi
        if(("sigmoid" in splitline) or ("relu" in splitline) or ("linear" in splitline) or "softmax" in splitline):
            activation.append(splitline[1])

            i +=1
```

```
            splitline = lines[i].strip("\n").split(" ")
            bias.append(list(map(int, splitline)))

            i+=1
        else:
            w = []
            for num in range(num_atr):
                splitline = lines[i].strip("\n").split(" ")
                w.append(list(map(int, splitline)))
                i +=1

            weigth.append(w)

    return bias, weigth, activation
```

In [ ]:

```
# Fungsi Aktivasi
import math
import numpy as np

def linear(x, kwargs=None):
    return x

def sigmoid(x):
    value = float(1 / (1 + math.exp(x * -1)))
    threshold = 0.1
    if value < threshold:
        return 0
    else:
        return 1

def relu(x):
    alpha = 0.0
    max_value = 1.0
    threshold = 0.0
    if x < threshold:
        if x > x*alpha:
            return x
        else:
            return x*alpha
    else:
        return min(x, max_value)

def softmax(arr):
    arr_exp = np.exp(arr)
    return arr_exp / arr_exp.sum()
```

Neural Network Class and Layer Class

Layer

In [ ]:

```
class Layer:
    def __init__(self,bias:list[int], weight:list[list[int]], activation:str):
        self.weight = weight
        self.bias = bias
        self.activation = activation


    def get_sigma(self, input:list[int]) -> list[int]:
        if(len(self.weight) == len(bias)):
            result = []
            for i in range(len(self.bias)):
                res = 1*self.bias[i]
                for j in range(len(input)):
                    res = res + input[j]*self.weight[j][i]

                result.append(res)

            return result

    def get_result(self, input:list[int]):
        sigma = self.get_sigma(input)
        result = []
        for i in range(len(sigma)):
            result.append(self.activate_function(sigma[i]))
        return result

    def activate_function(self, x):
        if(self.activation == "linear"):
            return linear(x)
        elif(self.activation == "sigmoid"):
            return sigmoid(x)
        elif(self.activation == "relu"):
            return relu(x)
        elif(self.activation == "softmax"):
            return softmax(x)

    def getDiGraph(self, index, max):
        # untuk visualisasi Digraph
        arr = []

        # bias
        i = 1
        for bs in self.bias:
            a = f'b{index}'
            b = f'h{index+1}_{i}'
            if(index+1 == max):
```

```
                b = f'y'
            arr.append([a, b, bs])
            i += 1

        # weight
        i = 1
        for x in self.weight:
            j = 1
            for w in x:
                a = f'x{index}_{i}'
                if(index>0):
                    a = f'h{index}_{i}'

                b = f'h{index+1}_{j}'
                if(index+1 == max):
                    b = f'y'
                arr.append([a, b, w])

                j += 1
            i += 1

        return arr


    def solve(self, input:list[int]):
        print("= = = = = = = = =")
        print(f"Input \t\t: {input}")
        print(f"weight \t\t: {self.weight}")
        print(f"Activation\t: {self.activation}")
        print(f"Sigma \t\t: {self.get_sigma(input)}")
        print()
        print(f"Result \t\t: {self.get_result(input=input)}")
        print("= = = = = = = = =")
```

Neural Network Class

In [ ]:

```
class FNNN:
    def __init__(self, bias:list[list[int]], weight:list[list[int]], activation:list[str], input:list[list[int]]):
        self.layers = []
        self.input = input
        # layer paling awal index 0
        i = 0
        j = 0
        for i in  range(len(activation)):
            layer = Layer(bias=bias[i], weight=weight[i], activation=activation[i])
            self.add_layer(layer)

    def add_layer(self, layer:Layer):
```

```
        self.layers.append(layer)

    def resolve(self):
        result = []
        i = 1
        for x in self.input:
            print(f"\n#### INPUT {i} ####")
            input = x
            for layer in self.layers:
                idx = self.layers.index(layer)
                print(f"\n====LAYER {idx}====")
                layer.solve(input)
                input = layer.get_result(input)

            result.append(input[0])
            i += 1

        print()
        print(f"RESULT ==> {result}")

    def get_results(self):
        result = []
        for x in self.input:
            input = x
            for layer in self.layers:
                input = layer.get_result(input)

            result.append(input[0])

        return result
```

Eksekusi

In [ ]:

```
print("# # # # # # # # # # # # # # # # # # #")
print("# Feed Forward Neural Network : XOR #")
print("# # # # # # # # # # # # # # # # # # #\n")

bias, weigth , activation = readModel("model relu-linier.txt")
data, target = readData()
fnnn = FNNN(bias, weigth, activation, data)
result = fnnn.get_results()

# print result etc
fnnn.resolve()
print()
print(f"TARGET CLASS\t\t: {target}")
print(f"RESULT CLASS\t\t: {result}")
print("==========================")
```

```python
if(result == target):
    print("Result: GOOD PREDICT")
    print("GOOD")
else:
    print("Result: BAD PREDICT")
    print("NO GOOD")


### VISUALIZATION
from graphviz import Digraph

dGraph = Digraph("FNNN: XOR", filename="model relu-linier.gv")

max = len(fnnn.layers)
for layer in fnnn.layers:
    idxL = fnnn.layers.index(layer)
    edges = layer.getDiGraph(idxL, max)
    for ed in edges:
        dGraph.edge(ed[0], ed[1], str(ed[2]))

dGraph.view()
```

```
# # # # # # # # # # # # # # # # # # #
# Feed Forward Neural Network : XOR #
# # # # # # # # # # # # # # # # # # #


#### INPUT 1 ####

====LAYER 0====
= = = = = = = = =
Input           : [0, 0]
weight          : [[1, 1], [1, 1]]
Activation      : relu
Sigma           : [0, -1]

Result          : [0, -0.0]
= = = = = = = = =

====LAYER 1====
= = = = = = = = =
Input           : [0, -0.0]
weight          : [[1], [-2]]
Activation      : linear
Sigma           : [0.0]

Result          : [0.0]
= = = = = = = = =

#### INPUT 2 ####
```

```
====LAYER 0====
= = = = = = = = =
Input           : [0, 1]
weight          : [[1, 1], [1, 1]]
Activation      : relu
Sigma           : [1, 0]

Result          : [1, 0]
= = = = = = = = =

====LAYER 1====
= = = = = = = = =
Input           : [1, 0]
weight          : [[1], [-2]]
Activation      : linear
Sigma           : [1]

Result          : [1]
= = = = = = = = =

#### INPUT 3 ####

====LAYER 0====
= = = = = = = = =
Input           : [1, 0]
weight          : [[1, 1], [1, 1]]
Activation      : relu
Sigma           : [1, 0]

Result          : [1, 0]
= = = = = = = = =

====LAYER 1====
= = = = = = = = =
Input           : [1, 0]
weight          : [[1], [-2]]
Activation      : linear
Sigma           : [1]

Result          : [1]
= = = = = = = = =

#### INPUT 4 ####

====LAYER 0====
= = = = = = = = =
Input           : [1, 1]
weight          : [[1, 1], [1, 1]]
Activation      : relu
Sigma           : [2, 1]

Result          : [1.0, 1]
```

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

====LAYER 1====
= = = = = = = = =
Input            : [1.0, 1]
weight           : [[1], [-2]]
Activation       : linear
Sigma            : [-1.0]

Result           : [-1.0]
= = = = = = = = =

RESULT ==> [0.0, 1, 1, -1.0]

TARGET CLASS             : [0, 1, 1, 0]
RESULT CLASS             : [0.0, 1, 1, -1.0]
==========================
Result: BAD PREDICT
NO GOOD
```

Out[ ]: 'model relu-linier.gv.pdf'