Feed Forward Neural Network

Anggota Kelompok:

- Daffa Pratama Putra / 13518033
- Muchammad Ibnu Sidqi / 13518072
- Muhammad Firas / 13518117
- Muhammad Daffa Dinaya / 13518141

Data Reader

In [40]:
```
import csv

def readData(filename="xor-data.csv"):
    data = []
    target_class = []
    with open(filename, newline="") as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            data.append([int(row["x1"]), int(row["x2"])])
            target_class.append(int(row["f"]))
    return data, target_class


def readWeight(filename):
    f = open(filename, "r")
    fl = f.readlines()
    activation = []
    bias = []
    weight = []
    allowed = ['relu', 'sigmoid', 'linear', 'softmax']

    n_attr = int(fl[0])
    line = 1
    while (line < len(fl)):
        splitted = fl[line].strip('\n').split(' ')
        if (('relu' in splitted) or ('linear' in splitted) or ('sigmoid' in splitted) or ('s
oftmax' in splitted)):
            activation.append(splitted[1])

            line += 1
            splitted = fl[line].strip('\n').split(' ')

            bias.append(list(map(int, splitted)))
        else:
            temp = []
            count = 0
            while (count < n_attr):
                splitted = fl[line].strip('\n').split(' ')
                temp.append(list(map(int, splitted)))
                count += 1
                line += 1
            if (count == n_attr):
                weight.append(temp)
                line -= 1
        line += 1
    return activation, bias, weight
```

Activation Function

In [41]:
```
import math
import numpy as np

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

def sigmoid(x, kwargs=None):
    value = float(1 / (1 + math.exp(x * -1)))
    threshold = kwargs.get("threshold", None)
    if threshold == None:
        return value
    else:
        if value < threshold:
            return 0
        else:
            return 1

def relu(x, kwargs):
    alpha = kwargs.get("alpha", 0.0)
    max_value = kwargs.get("max_value", 1.0)
    threshold = kwargs.get("threshold", 0.0)
    if x < threshold:
        return max(x, x * alpha)
    else:
        if max_value == None:
            return x
        else:
            return min(x, max_value)

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

Neural Network

In [42]:
```
from activation.activationFunction import linear, sigmoid, relu, softmax
from dataReader import *

# For iterating per item in array. Except for linear function because single parameter is au
tomate to iterate per item
linear = np.vectorize(linear)
sigmoid = np.vectorize(sigmoid)
relu = np.vectorize(relu)


class NeuralNetwork:
    def __init__(self):
        self.base_layer = []
        self.current_layer = []

    def get_total_layer(self):
        return len(self.layer)

    def enqueue_layer(self, layer):
        self.base_layer.append(layer)

    def deque_layer(self):
        self.base_layer.pop(0)

    def solve(self):
        self.current_layer = self.base_layer.copy()
        for idx in range(len(self.current_layer)):
            print("")
            print("LAYER === " + str(idx))
            if idx != 0:
                self.current_layer[idx].input_value = self.current_layer[idx-1].result
            else:
                print("Input layer \t:", self.current_layer[idx].input_value)
            self.current_layer[idx].compute()


class InputLayer:
    def __init__(self, arr=[]):
        self.input_value = np.array(arr)
        self.result = self.input_value

    def compute(self):
        pass


class Layer(InputLayer):
    def __init__(self, arr_weight, arr_bias, activation_function, **kwargs):
        super().__init__([])
        self.weight = np.array(arr_weight)
        self.bias = np.array(arr_bias)
        self.result = np.array([])
        self.activation_function = activation_function
        self.kwargs = kwargs

    def activate(self):
        self.result = self.activation_function(self.result, self.kwargs)

    def sigma(self):
        # case 1 Dimension
        if(len(self.weight[0]) == 1):
            self.result = np.matmul(
                self.input_value, self.weight.flatten()) + self.bias

        else:
            self.result = np.matmul(self.input_value, self.weight) + self.bias
        print("Sigma \t: ", self.result)

    def compute(self):
        print("Input \t: ", self.input_value)
        self.sigma()
        self.activate()

        print("Weight \t: ", self.weight)
        print("Result \t: ", self.result)
```

Main Program

In [43]:
```
def main():
    print('Feed Forward Neural Network : XOR')
    print("================================")

    data_training, target = readData()
    activation, bias, weight = readWeight('model 1.txt')
    neural_network = NeuralNetwork()
    result = []
    layer = []

    print("Activation \t: ", end="")
    for i in range(len(activation)):
        act = None
        if (activation[i] == 'sigmoid'):
            act = sigmoid
        elif (activation[i] == 'linear'):
            act = linear
        elif (activation[i] == 'relu'):
            act = relu
        elif (activation[i] == 'softmax'):
            act = softmax
        print(activation[i], end=" ")
        layer.append(Layer(weight[i], bias[i], act, threshold=0.1))
    print("")
    for data in data_training:
        layer.insert(0, InputLayer(data))
        neural_network.base_layer = layer
        neural_network.solve()
        result.append(neural_network.current_layer[-1].result)
        neural_network.deque_layer()
    print("")
    print("Target Class \t: ", target)
    print("Predict Class \t: ", result)
    print("================================")
    if (result == target):
        print("Result : Good Predict")
    else:
        print("Result : Wrong Predict")


    # Model Visualization
    from graphviz import Digraph
    f = Digraph('Feed Forward Neural Network', filename='model 1.gv')
    f.attr('node', shape='circle', fixedsize='true', width='0.9')

    for i in range(len(neural_network.current_layer)):
        if i != 0:
            if i == 1:
                for j in range(len(neural_network.current_layer[i].weight)):
                    # f.edge("x{j}",'hidden{j}")
                    for k in range(len(neural_network.current_layer[i].weight[j])):
                        f.edge(f'x{j}', f'h{i},{k}', str(
                            neural_network.current_layer[i].weight[j][k]))
                for j in range(len(neural_network.current_layer[i].bias)):
                    f.edge(f'bx', f'h{i},{j}', str(
                        neural_network.current_layer[i].bias[j]))

            else:
                for j in range(len(neural_network.current_layer[i].weight)):
                    for k in range(len(neural_network.current_layer[i].weight[j])):
                        f.edge(f'h{i-1},{j}', f'h{i},{k}',
                               str(neural_network.current_layer[i].weight[j][k]))
                for j in range(len(neural_network.current_layer[i].bias)):
                    f.edge(f'bh{i-1}', f'h{i},{j}',
                           str(neural_network.current_layer[i].bias[j]))


    f.view()

if __name__ == "__main__":
    main()
```

```
Feed Forward Neural Network : XOR
================================
Activation      : sigmoid sigmoid

LAYER === 0
Input layer     : [0 0]

LAYER === 1
Input   :  [0 0]
Sigma   :  [-10  30]
Weight  :  [[ 20 -20]
 [ 20 -20]]
Result  :  [0 1]

LAYER === 2
Input   :  [0 1]
Sigma   :  [-10]
Weight  :  [[20]
 [20]]
Result  :  [0]

LAYER === 0
Input layer     : [0 1]

LAYER === 1
Input   :  [0 1]
Sigma   :  [10 10]
Weight  :  [[ 20 -20]
 [ 20 -20]]
Result  :  [1 1]

LAYER === 2
Input   :  [1 1]
Sigma   :  [10]
Weight  :  [[20]
 [20]]
Result  :  [1]

LAYER === 0
Input layer     : [1 0]

LAYER === 1
Input   :  [1 0]
Sigma   :  [10 10]
Weight  :  [[ 20 -20]
 [ 20 -20]]
Result  :  [1 1]

LAYER === 2
Input   :  [1 1]
Sigma   :  [10]
Weight  :  [[20]
 [20]]
Result  :  [1]

LAYER === 0
Input layer     : [1 1]

LAYER === 1
Input   :  [1 1]
Sigma   :  [ 30 -10]
Weight  :  [[ 20 -20]
 [ 20 -20]]
Result  :  [1 0]

LAYER === 2
Input   :  [1 0]
Sigma   :  [-10]
Weight  :  [[20]
 [20]]
Result  :  [0]

Target Class    :  [0, 1, 1, 0]
Predict Class   :  [array([0]), array([1]), array([1]), array([0])]
================================
Result : Good Predict
```

In [ ]:
```

```