In [17]:

```
from math import e

def linear(sigma):
    return sigma

def relu(sigma):
    return max(0, sigma)

def sigmoid(sigma):
    return 1/(1+e**(-sigma))

def softmax(sigma):
    return sigma

class Model:
    def __init__(self,size_of_input_layer):
        self.layers = []
        self.size_of_input_layer = size_of_input_layer

    def add(self, array_of_weights, activation_function):
        self.layers.append(Layer(array_of_weights, activation_function))

    def predict(self, input_to_process):
        results = []
        for batch in range(len(input_to_process)):

            input = input_to_process[batch]
            input.insert(0,1)

            for layer in range(len(self.layers)):
                self.layers[layer].layer_output = []

            self.layers[0].calculate(input)
            input_layer_output = self.layers[0].layer_output
            for i in range(1, len(self.layers)):
                self.layers[i].calculate(input_layer_output)
                input_layer_output = self.layers[i].layer_output

            if self.layers[-1].activation_function != "softmax":
                results.append(input_layer_output[1]) #y_hat
            else:
                results.append(input_layer_output.index(max(input_layer_output[1:])))
        return results

class Layer:
    def __init__(self, array_of_weights, activation_function):
        self.neurons = []
        self.layer_output = []
        self.total_softmax_exponents = 0
        self.activation_function = activation_function
        for weights in array_of_weights:
            self.neurons.append(Neuron(activation_function, weights))

    def calculate(self, input_to_process):
        if self.activation_function != "softmax":
            for neuron in self.neurons:
                neuron.calculate(input_to_process)
                self.layer_output.append(neuron.h)
            self.layer_output.insert(0,1)
        else:
            for neuron in self.neurons:
                self.total_softmax_exponents = e**(neuron.sigma(input_to_process))
            for neuron in self.neurons:
                neuron.h = e**(neuron.sigma(input_to_process)) / self.total_softmax_expo
nents
                self.layer_output.append(neuron.h)
            self.layer_output.insert(0,1)
```

```
class Neuron:
    def __init__(self, activation_function, weights):
        self.activation_function = activation_function
        self.weights = weights

    def sigma(self, x):
        # First element of x must be bias
        result = 0
        for idx, weight in enumerate(self.weights):
            result += weight * x[idx]
        return result

    def calculate(self ,input_to_process):
        sigma_result = self.sigma(input_to_process)
        if self.activation_function == 'linear':
            self.h = linear(sigma_result)
        if self.activation_function == 'relu':
            self.h = relu(sigma_result)
        if self.activation_function == 'sigmoid':
            self.h = sigmoid(sigma_result)

def summary(model):
    paramList = []
    param = model.size_of_input_layer
    for layer in range(len(model.layers)):
        param = (param + 1) * len(model.layers[layer].neurons)
        print("Coefficient:")
        for neuron in model.layers[layer].neurons:
            print("w{}{} : {}".format(layer,model.layers[layer].neurons.index(neuron),ne
uron.weights))
        print("")
        print("Output Shape:")
        print(len(model.layers[layer].neurons))
        print("")
        print("Activation Function:")
        print(neuron.activation_function)
        print("")
        print("Param")
        print(param)
        paramList.append(param)
        param = len(model.layers[layer].neurons)
        print("==================================================")
    print("Total params: {}".format(sum(paramList)))
    print("Trainable params: {}".format(sum(paramList)))

# Testing
# num_of_layer = 3
# model = Model(2)
# model.add([[-10,20,20,30],[30,-20,-20,30]], 'softmax')
# model.add([[-30,20,20,20],[-30,20,20,20]], 'softmax')
# y_hat = model.predict([[1,1,1],[0,0,2],[1,0,3]])
# print(y_hat)
# summary(model)

def parse_model_from_file(file_name):
    model = []
    with open(file_name, encoding = 'utf-8') as f:
        hidden_layer_num = int(f.readline().rstrip("\n"))
        input_layer_neuron_num = int(f.readline().rstrip("\n"))

        model = Model(input_layer_neuron_num)
        for i in range(hidden_layer_num):
            hidden_layer_attributes = f.readline().rstrip("\n").split(",")
            hidden_layer_neuron_count = int(hidden_layer_attributes[0])
            hidden_layer_act_func = hidden_layer_attributes[1]

            array_of_weights = []
            for neuron in range(2, 2 + hidden_layer_neuron_count):
                weights = (list(map(int, hidden_layer_attributes[neuron].split(";"))))
                array_of_weights.append(weights)
```

```
        model.add(array_of_weights, hidden_layer_act_func)
    return model
```

Sigmoid

In [20]:

```
model = parse_model_from_file('tc1.txt')
summary(model)
```

```
Coefficient:
w00 : [-10, 20, 20]
w01 : [30, -20, -20]

Output Shape:
2

Activation Function:
sigmoid

Param
6
==================================================
Coefficient:
w10 : [-30, 20, 20]

Output Shape:
1

Activation Function:
sigmoid

Param
3
==================================================
Total params: 9
Trainable params: 9
```

Sigmoid input (0,0)

In [21]:

```
print(model.predict([[0,0]]))
```

```
[4.543910487654594e-05]
```

Sigmoid input (0,1)

In [22]:

```
print(model.predict([[0,1]]))
```

```
[0.999954519621495]
```

Sigmoid input (1,0)

In [23]:

```
print(model.predict([[1,0]]))
```

```
[0.999954519621495]
```

Sigmoid input (1,1)

In [24]:

```
print(model.predict([[1,1]]))
```

```
[4.543910487654586e-05]
```

```
[4.543910487654586e-05]
```

Sigmoid input Batch atau ((0,0),(0,1)(1,0),(1,1))

In [25]:

```python
print(model.predict([[0,0],[0,1],[1,0],[1,1]]))
```

```
[4.543910487654594e-05, 0.999954519621495, 0.999954519621495, 4.543910487654586e-05]
```

ReLU

In [26]:

```python
model = parse_model_from_file('tc2.txt')
summary(model)
```

```
Coefficient:
w00 : [0, 1, 1]
w01 : [-1, 1, 1]

Output Shape:
2

Activation Function:
relu

Param
6
==================================================
Coefficient:
w10 : [0, 1, -2]

Output Shape:
1

Activation Function:
linear

Param
3
==================================================
Total params: 9
Trainable params: 9
```

ReLU input (0,0)

In [27]:

```python
print(model.predict([[0,0]]))
```

```
[0]
```

ReLU input (0,1)

In [28]:

```python
print(model.predict([[0,1]]))
```

```
[1]
```

ReLU input (1,0)

In [29]:

```python
print(model.predict([[1,0]]))
```

```
[1]
```

ReLU input (1,1)

In [30]:

```
print(model.predict([[1,1]]))
```

```
[0]
```

ReLU input Batch atau ((0,0),(0,1)(1,0),(1,1))

In [31]:

```
print(model.predict([[0,0],[0,1],[1,0],[1,1]]))
```

```
[0, 1, 1, 0]
```