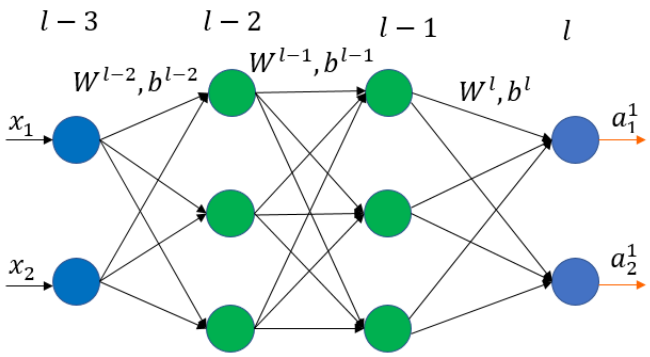
IMPLEMENT NEURAL NETWORK



1. CLASS LAYER

```
from abc import abstractmethod

class Layer:
    def __init__(self):
        self.input = None
        self.output = None
        self.input_shape = None
        self.output_shape = None
        raise NotImplementedError

    @abstractmethod
    def input(self):
        return self.input

    @abstractmethod
    def output(self):
        return self.output

    @abstractmethod
    def input_shape(self):
        return self.input_shape

    @abstractmethod
    def output_shape(self):
        return self.output_shape

    @abstractmethod
    def forward_propagation(self, input):
        raise NotImplementedError

    @abstractmethod
    def backward_propagation(self, output_error, learning_rate):
        raise NotImplementedError
```

2. FULL CONNECTED LAYER

```
# from .layer import Layer
import numpy as np


class FCLayer(Layer):
    def __init__(self, input_shape, output_shape):
        """

        :param input_shape:  (1, 3)
        :param output_shape: (1, 4)
        (1X3) (3x4) => (1, 4)
        (3, 1) (1, 4) => (3x4)
        """
        self.input_shape = input_shape
        self.output_shape = output_shape
        self.weights = np.random.rand(input_shape[1], output_shape[1]) - 0.5
        self.bias = np.random.rand(1, output_shape[1]) - 0.5

    def forward_propagation(self, input):
        self.input = input
        self.output = np.dot(self.input, self.weights) + self.bias
        return  self.output

    def backward_propagation(self, output_error, learning_rate):
        curent_layer_err =  np.dot(output_error, self.weights.T)
        dweight = np.dot(self.input.T, output_error)

        self.weights -= dweight*learning_rate
        self.bias -= learning_rate*output_error

        return curent_layer_err
```

3. CLASS ACTIVATION LAYER

```
# from .layer import Layer


class ActivationLayer(Layer):
    def __init__(self, input_shape, output_shape, activation,
activation_prime):
        """
        :param input_shape:  đầu vào input mảng (1, 4)
        :param output_shape:  mảng
        :param activation:  hàm
        :param activation_prime:  hàm
        """
```

```
        self.input_shape = input_shape
        self.output_shape = output_shape
        self.activation = activation
        self.activation_prime = activation_prime

    def forward_propagation(self, input):
        self.input = input
        self.output = self.activation(input)
        return self.output

    def backward_propagation(self, output_error, learning_rate):
        return self.activation_prime(self.input)*output_error
```

4. CLASS NEURAL NETWORK LAYER

```
class Network:
    def __init__(self):
        self.layers = []
        self.loss = None
        self.loss_prime = None

    def add(self, layer):
        self.layers.append(layer)

    def setup_loss(self, loss, loss_prime):
        self.loss = loss
        self.loss_prime = loss_prime

    def predict(self, input):
        """

        :param input: [[1, 3]] = > 1 , [[1, 3], [3, 5], [3, 4]]
        :return: kết quả dự đoán
        """
        result = []
        n = len(input)
        for i in range(n):
            output = input[i]

            for layer in self.layers:
                output = layer.forward_propagation(output)
            result.append(output)

        return result

    def fit(self, x_train, y_train, learning_rate, epochs):

        n = len(x_train)
        for i in range(epochs):
            err = 0
```

```
            for j in range(n):
                #lan truyen tiến
                output = x_train[j]
                for layer in self.layers:
                    output = layer.forward_propagation(output)

                #tính lỗi của từng
                err += self.loss(y_train[j], output)

                #lan truyền ngược
                error = self.loss_prime(y_train[j], output)
                for layer in reversed(self.layers):
                    error = layer.backward_propagation(error, learning_rate)

            err = err / n

            print('epoch : %d/%d  err = %f'%(i, epochs, err))
```

5. CẤU HÌNH DỰ ĐOÁN DỰA TRÊN NEURAL NETWORK

```
#from network.network import Network
#from layers.FCLayer import FCLayer
#from layers.activation_layer import ActivationLayer
import numpy as np


def relu(z):
    """

    :param z:  numpy array
    :return: 0 nếu z <= 0
         z nếu z > 0
         [1, -3, 9, -7] => [1, 0, 9, 0]
    """
    return np.maximum(0, z)


def relu_prime(z):
    """

    :param z: numpy array
    :return: array 1, 0 z> 0 => 1, z < 0 => 0
    """
    z[z<0]=0
    z[z>0]=1
    return z


def loss(y_true, y_pred):
```

```
    return 0.5*(y_pred-y_true)**2


def loss_prime(y_true, y_pred):
    return y_pred-y_true

x_train = np.array([[[0, 0]], [[0, 1]], [[1, 0]], [[1, 1]]])
y_train = np.array([[[0]], [[1]], [[1]], [[0]]])

net = Network()
net.add(FCLayer((1, 2), (1, 3)))
net.add(ActivationLayer((1, 3), (1, 3), relu, relu_prime))
net.add(FCLayer((1, 3), (1, 1)))
net.add(ActivationLayer((1, 1), (1, 1), relu, relu_prime))

net.setup_loss(loss, loss_prime)

net.fit(x_train, y_train, epochs=1000, learning_rate=0.01)

out = net.predict([[0, 1]])

print(out)
```