Date :
Page :

Neural Network From Scratch

Here, how neural network is implemented from scratch is discussed. Here all mathmatical expressions needed to train a simple neural network is explained. This consists of mainly six major parts. They are :-

1) Implementation design
2) Create the base layer
3) Create Dense layer
4) Create Activation layer
5) Implement activation functions and loss functions
6) Solve XOR problem.

Before heading into implementation design, first revising machine learning steps :

Step 1 :- Feed input : Data flows from layer to layer : Gives output (prediction)

Let $y = network(x, w)$

where y is prediction
x is input / features
w is parameter to train (weights & bias)
and network() is a neural network arch.

Step 2: Calculate error.

eg. $E = \frac{1}{2}(y - \hat{y})^2$

where y is prediction
$\hat{y}$ is true.

*Note: These notation might be different on different sources.

Step 3: Adjust the parameter using gradient descent (Back propagation).

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w}$$

where α is learning rate

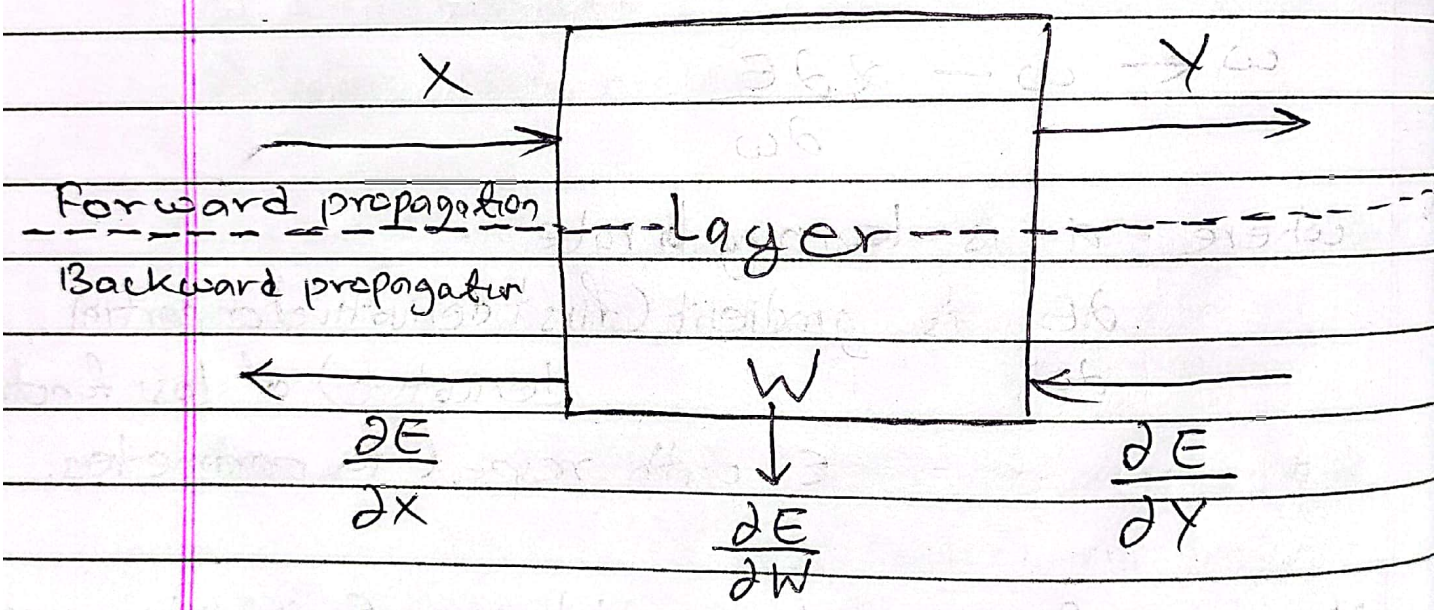
$\frac{\partial E}{\partial w}$ is gradient (first derivative or partial derivative) of loss function E with respect to parameters.

Step 4: Start again, until error E is minimum.

Now getting into our main purpose.

1) Implementation design:

First of all, we need a ~~layer~~ model which can fit every types of layers whether it's fully connected, convolution, drop out or activation. Since a layer is a big function, hence, it is represented as a box. Below



For every layer, when an input 'X' is given it will produce output 'Y'. This is called forward propagation.

Similarly the other step when training is called backward propagation. when the layer update its parameters. During back propagation.

Date:
Page:

If a layer is given the derivative of the error w.r.t its output, it should give back the derivative of the error w.r.t its input.

Ultimately, for every trainable layer there is set of parameters W that need to be updated according to gradient descent or similar other. In order to update parameters, the layer needs to compute the derivative of error w.r.t. the parameter.

Then,

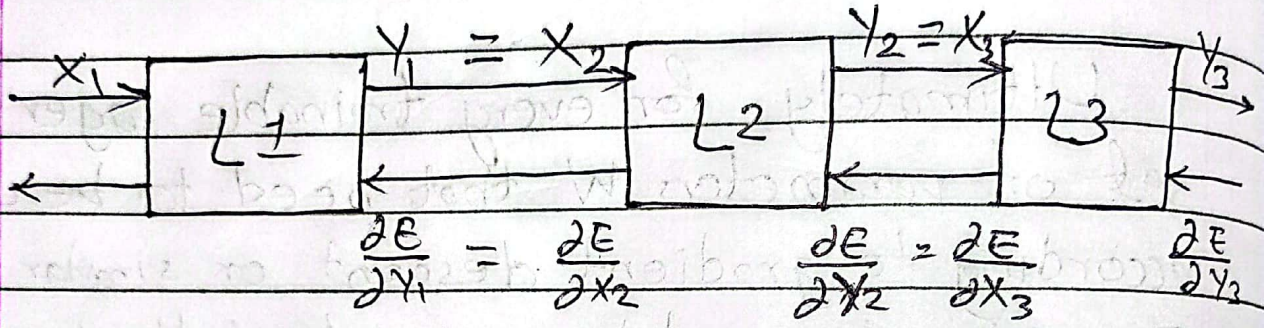
$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y} \frac{\partial Y}{\partial W}$$

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} \frac{\partial Y}{\partial X}$$

Now suppose

Date: .....
Page: .....

Now suppose there are multiple layers then,



In these types of sequential model, the output of first layer is input for second layer and goes on.

Hence $Y_1 = X_2$, $Y_2 = X_3$

Similarly, for layer 1, derivative of error with respect to output of L1 is equal to derivative of error w.r.t. input of L2 and same for other layer.

i.e. $\frac{\partial E}{\partial Y_1} = \frac{\partial E}{\partial X_2}$ and $\frac{\partial E}{\partial Y_2} = \frac{\partial E}{\partial X_3}$

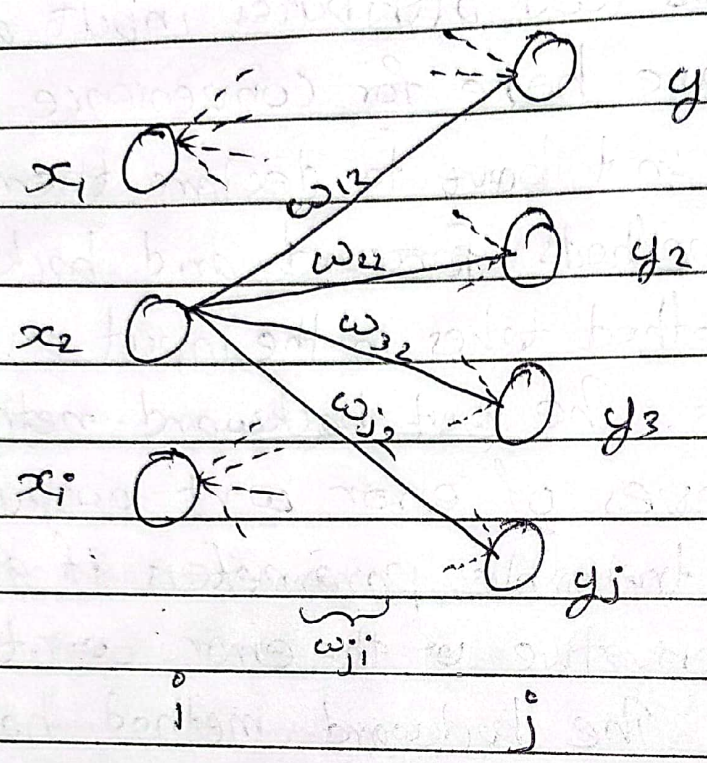
2) Base Layer

This layer has two attributes input and output. these are here for convenience so that other layers doesn't have to declare them. It also has two methods forward and backward. The forward method takes in the input and gives you the output. The backward methods takes in the derivatives of error wrt output and should update trainable parameters if any and return the derivative of the error w.r.t. input to the layer. The backward method has an additional parameter, the learning-rate.

*Note: Code for these classes are written at very last.

Page: .....

3) Dense Layer. (Fully-Connected Layer)



Dense layer, connect a set of 'i' input neurons to set of 'j' O/p neurons. I/P is denoted as x and o/p is denoted as y. In dense layer each input neuron is connected to every O/p neuron. Each connection represents weights. Represented by w_{ji}, which means, weight which connect output neuron j with input neuron i.

Every o/p value is computed as sum of all inputs multiplied by the weights and an additional term called bias denoted as b. Biases are also trainable parameters like weight

Date:
Page:

Here,

$$y_1 = x_1 w_{11} + x_2 w_{12} + \cdots + x_i w_{1i} + b_1$$
$$y_2 = x_1 w_{21} + x_2 w_{22} + \cdots + x_i w_{2i} + b_2$$
$$y_3 = x_1 w_{31} + x_2 w_{32} + \cdots + x_i w_{3i} + b_3$$
$$\vdots$$
$$y_j = x_1 w_{j1} + x_2 w_{j2} + \cdots + x_i w_{ji} + b_j$$

Rewriting in matrix form

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_j \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1i} \\ w_{21} & w_{22} & \cdots & w_{2i} \\ \vdots & & & \\ w_{j1} & w_{j2} & \cdots & w_{ji} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \end{bmatrix}$$

$j \times 1 \qquad j \times i \qquad i \times 1 \qquad j \times 1$

Weight matrix has a shape of $j \times i$ and bias is a column vector of shape $j \times 1$.

Finally, we can represent it as

$$Y = W \cdot X + B$$

This layer inherits from the base layer class. The constructor of layer takes two parameters: input size and output size, which are number

of neurons in input and output respectively. Inside the constructor weights and bias are initialize randomly.

The forward method simply computes y equals $wx + b$.

For backward propagation

Let as assume we have derivative of error w.r.t. output of dense layer ie $\frac{dE}{dY}$ is given then we have to calculate.

~~the~~ two things.

$\frac{dE}{dY}$ → $\frac{dE}{dW}, \frac{dE}{dB} = ?$

$\frac{dE}{dY}$ → $\frac{\partial E}{\partial X} = ?$

Using $\frac{dE}{dY}$ we need to calculate derivative of error w.r.t. weights and bias, as they are trainable parameter, and ther value have to be adjusted according to gradient descent.
On the other, we need derivative of eror w.r.t input. this is because, it will be passed to

Date: _____
Page: _____

the layer before the one computing this.

(i) For $\frac{\partial E}{\partial W}$

Starting with derivative of error with respect to W.

We have

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \frac{\partial E}{\partial y_2} \\ \vdots \\ \frac{\partial E}{\partial y_j} \end{bmatrix}$$

$j \times 1$

Now

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{11}} & \frac{\partial E}{\partial w_{12}} & \cdots & \frac{\partial E}{\partial w_{1i}} \\ \frac{\partial E}{\partial w_{21}} & \frac{\partial E}{\partial w_{22}} & \cdots & \frac{\partial E}{\partial w_{2i}} \\ \vdots & & & \\ \frac{\partial E}{\partial w_{j1}} & \frac{\partial E}{\partial w_{j2}} & \cdots & \frac{\partial E}{\partial w_{ji}} \end{bmatrix}$$

$j \times i$

Taking first term only

$$\frac{\partial E}{\partial \omega_{12}} = \frac{\partial E}{\partial y_1}\underbrace{\frac{\partial y_1}{\partial \omega_{12}}}_{x_2} + \frac{\partial E}{\partial y_2}\underbrace{\frac{\partial y_2}{\partial \omega_{12}}}_{0} + \cdots + \frac{\partial E}{\partial y_j}\underbrace{\frac{\partial y_j}{\partial \omega_{12}}}_{0}$$

$$\therefore \frac{\partial E}{\partial \omega_{12}} = \frac{\partial E}{\partial y_1} x_2$$

In general

$$\frac{\partial E}{\partial \omega_{ji}} = \frac{\partial E}{\partial y_j} x_i$$

Now using this equation

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial y_1} x_1 & \frac{\partial E}{\partial y_1} x_2 & \cdots & \frac{\partial E}{\partial y_1} x_i \\ \frac{\partial E}{\partial y_2} x_1 & \frac{\partial E}{\partial y_2} x_2 & \cdots & \frac{\partial E}{\partial y_2} x_i \\ \vdots & & & \\ \frac{\partial E}{\partial y_j} x_1 & \frac{\partial E}{\partial y_j} x_2 & \cdots & \frac{\partial E}{\partial y_j} x_i \end{bmatrix}_{j \times i} \checkmark$$

This can be further more simplified by converting in matrix multiplication form

Date: _____

Page: _____

$$\therefore \frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \frac{\partial E}{\partial y_2} \\ \vdots \\ \frac{\partial E}{\partial y_j} \end{bmatrix} \times \begin{bmatrix} x_1 & x_2 & \cdots & x_i \end{bmatrix}$$

$j \times 1$ $\qquad$ $1 \times i$

$j \times i$

$$\therefore \frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y} \cdot X^T$$

[X^T is transpose of X]

(ii) For $\frac{\partial E}{\partial B}$

$$\frac{\partial E}{\partial B} = \begin{bmatrix} \frac{\partial E}{\partial b_1} \\ \frac{\partial E}{\partial b_2} \\ \vdots \\ \frac{\partial E}{\partial b_3} \end{bmatrix}$$

For

$$\frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial b_1} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial b_1} + \cdots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial b_1}$$

$\frac{\partial y_1}{\partial b_1} \rightsquigarrow 1$, $\frac{\partial y_2}{\partial b_1} \rightsquigarrow 0$, $\frac{\partial y_j}{\partial b_1} \rightsquigarrow 0$

$$\therefore \frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial y_1}$$

In general,

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y_j}$$

$$\therefore \boxed{\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}}$$

(iii) For $\partial E/\partial X$.

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \\ \vdots \\ \frac{\partial E}{\partial x_i} \end{bmatrix}$$

Date:
Page:

$$\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \cdots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_1}$$

$\frac{\partial y_1}{\partial x_1} = w_{11}$, $\frac{\partial y_2}{\partial x_1} = w_{21}$, $\frac{\partial y_j}{\partial x_1} = w_{j1}$

In general,

$$\therefore \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_1} w_{1i} + \frac{\partial E}{\partial y_2} w_{2i} + \cdots + \frac{\partial E}{\partial y_j} w_{ji}$$

$$\therefore \frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial y_1} w_{11} + \frac{\partial E}{\partial y_2} w_{21} + \cdots + \frac{\partial E}{\partial y_j} w_{j1} \\ \frac{\partial E}{\partial y_1} w_{12} + \frac{\partial E}{\partial y_2} w_{22} + \cdots + \frac{\partial E}{\partial y_j} w_{j2} \\ \vdots \\ \frac{\partial E}{\partial y_1} w_{1i} + \frac{\partial E}{\partial y_2} w_{2i} + \cdots + \frac{\partial E}{\partial y_j} w_{ji} \end{bmatrix}_{i \times 1}$$

Expressing it into matrix multiplication

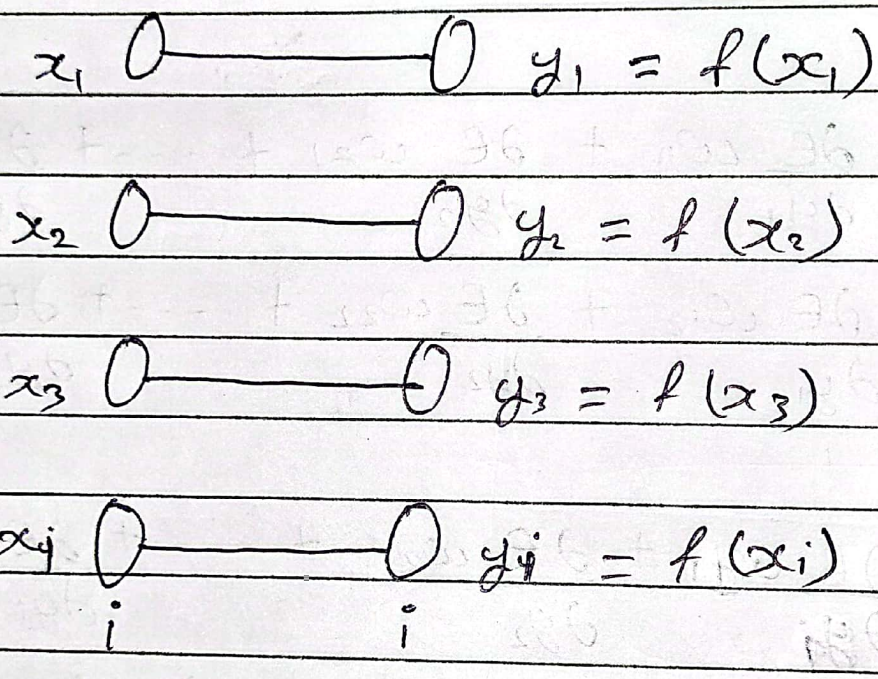
$$\frac{\partial E}{\partial X} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{j1} \\ w_{12} & w_{22} & \cdots & w_{j2} \\ \vdots & & & \\ w_{1i} & w_{2i} & \cdots & w_{ji} \end{bmatrix}_{i \times j} \begin{bmatrix} \partial E/\partial y_1 \\ \frac{\partial E}{\partial y_2} \\ \vdots \\ \partial E/\partial y_j \end{bmatrix}_{j \times 1}$$

$$\therefore \frac{\partial E}{\partial X} = W^T \cdot \frac{\partial E}{\partial Y}$$

Page:

Now we can implement these three equations for backward propagation

4) Activation Layer



The activation layer takes some input neurons and passes them through activation function. Thus, for that layer the output has the same shape as the input.

The forward propagation is expressed as

- $\boxed{Y = f(X)}$

where f is activation function

Date:
Page:

The activation layer inherites from the base layer class. The constructor takes two parameters the activation and its derivatives.

The forward method simply applies the activation to the input.

The backward method should return the derivative of the error w.r.t the input

As always let we have derivative of E w.r.t Y then we have to calculate derivative of E w.r.t X

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \\ \vdots \\ \frac{\partial E}{\partial x_i} \end{bmatrix}$$

$$y_1 = f(x_1)$$
$$y_2 = f(x_2)$$
$$\vdots$$
$$y_i = f(x_i)$$

$i \times 1$.

$$\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial y_1}\underbrace{\frac{\partial y_1}{\partial x_1}}_{f'(x_1)} + \frac{\partial E}{\partial y_2}\underbrace{\frac{\partial y_2}{\partial x_1}}_{0} + \cdots + \frac{\partial E}{\partial y_i}\underbrace{\frac{\partial y_i}{\partial x_1}}_{0}$$

$$\therefore \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} f'(x_i)$$

Page:

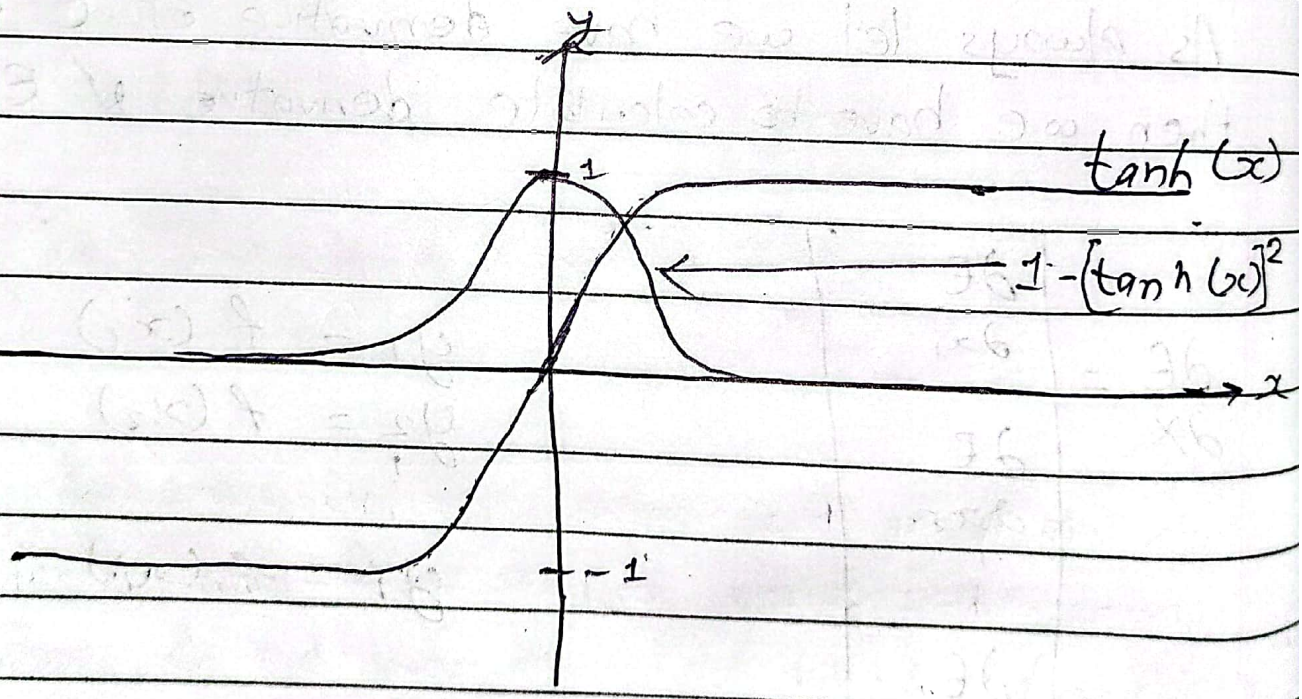
Hence,

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} \odot f'(X)$$

⊙ element wise multiplication

5) Implementation of activation function and loss function

Hyperbolic Tangent as activation function



$$f(x) = \tanh(x)$$

$$f'(x) = 1 - [\tanh(x)]^2$$

Date:
Page:

For implementation of tanh activation, we inherit from activation class and pass hyperbolic tangent function and its derivatives to super constructor.

Loss function

Here mean squared error is implemented. Given a vector y that represents the desired output denoted as y and another vector that represent the actual output represented by $\hat{y}$.

Error is defined as

$$E = \frac{1}{n} \sum_{i} (\hat{y}_i - \hat{y}_i)^2$$

Now,

From the begining we have supposed derivative of E w.r.t. Y is given. during back propagation. For L1 layer, it is given by L2 layer and for L2 it is given by L3 and so on. At very last layer the output of last layer is the output of whole neuron network. Hence it is the output used to calculate error. Hence derivative of E is calculated using above function.

Date:
Page:

$$\therefore \frac{\partial E}{\partial y_1} = \frac{\partial}{\partial y_1} \frac{1}{n} \left[(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_n - y_n)^2 \right]$$

$$= \frac{\partial}{\partial y_1} \frac{1}{n} (\hat{y}_1 - y_1)^2$$

$$= \frac{2}{n} (y_1 - \hat{y}_1)$$

In general

$$\frac{\partial E}{\partial y_i} = \frac{2}{n} (y_i - \hat{y}_i)$$

In matrix form

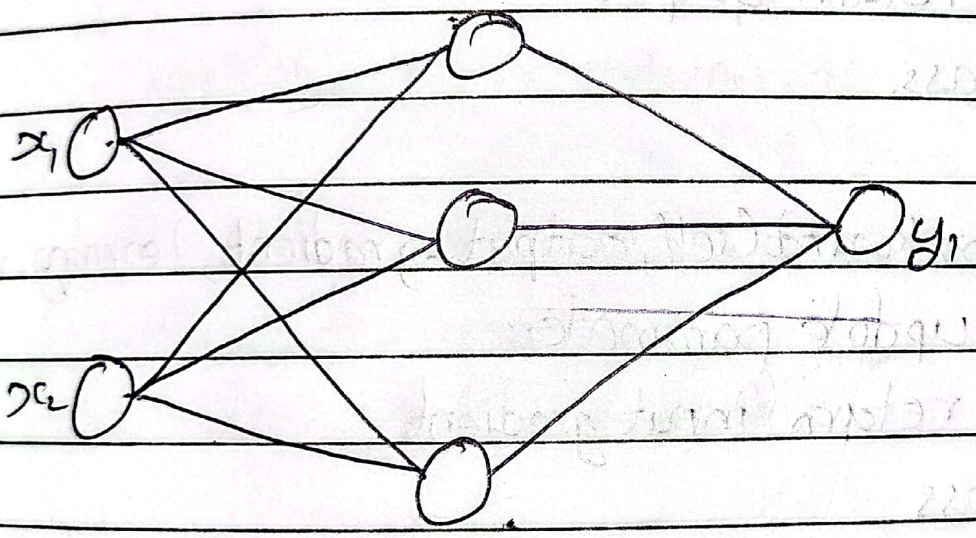
$$\boxed{\frac{\partial E}{\partial Y} = \frac{2}{n} (Y - \hat{Y})}$$

Now two different function will be made for these two task i.e. to calculate error and to calculate derivative of E w.r.t. Y.

Date:
Page:

6) Soluing for X-OR Problem.

x_1	x_2	$\hat{y}$
0	0	0
0	1	1
1	0	1
1	1	0.



Here two input neuron is kept as there are two inputs for each row and one output neuron for one output value. Also three neurons are used for one hidden layer.

CODE

Page:

layer.py

```
class Layer:
    def __init__(self):
        self.input = None
        self.output = None

    def forward(self, input):
        # return output
        pass.

    def backward(self, output_gradient, learning_rate):
        # update parameters
        # return input gradient
        pass
```

$output\_gradient = \frac{\partial E}{\partial Y}$ input = X

output = Y

$input\_gradient = \frac{\partial E}{\partial X}$ input.T = X^T

$weight\_gradient = \frac{\partial E}{\partial W}$ weight.T = W^T

$bias\_gradient = \frac{\partial E}{\partial B}$ np.dot() → matrix multiplication

Date:
Page:

dense.py

```
import numpy as np
from layer import Layer

class Dense(Layer):
    def __init__(self, input_size, output_size):
        self.weights = np.random.randn(output_size,
                                       input_size)
        self.bias = np.random.randn(output_size, 1)

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

    def backward(self, output_gradient, learning_rate):
        weights_gradient = np.dot(output_gradient, self.input.T)
        input_gradient = np.dot(self.weights.T, output_gradient)
        self.bias =
        self.weight -= learning_rate * weights_gradient
        self.bias -= learning_rate * output_gradient
        return input_gradient
```

activation.py

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

class Activation (Layer):
    def __init__ (self, activation, activation-prime):
        self.activation = activation
        self.activation-prime = activation-prime

    def forword(self, input):
        self.input = input
        return self.activation (self.input)

    def backword (self, output-gradient, learning-rate):
        return np.multiply (output-gradient, self.activation-
                                        prime(self.input))
```

activation = f(x)
activation_prime = f'(x)

np.multiply () → element wise multiplication.

```
class Tanh(Activation):
    def __init__(self):
        def tanh(x)
            return np.tanh(x)

        def tanh_prime(x):
            return 1 - np.tanh(x) ** 2

        super().__init__(tanh, tanh_prime)
```

losses.py

```
import numpy as np.

def mse(y_true, y_pred):
    return np.mean(np.power(y_true-y_pred, 2))

def mse_prime(y_true, y_pred)
    return 2*(y_pred - y_true)/np.size(y_true)
```

$y\_true = \hat{y}$

$y\_pred = y$

$np.size() = n$.

train.py

```
def predict(network, input):
    output = input
    for layer in ~~layers~~ network:
        output = layer.forward(output)
    return output


def fit(network, loss, loss_prime, x_train, y_train,
        epochs=1000, learning_rate=0.01,):
    for e in range(epochs):
        error = 0
        for x, y in zip(x_train, y_train):
            # forward propagation
            output = predict(network, x)

            # error calculation
            error += loss(y, output)

            # backward propagation
            grad = loss_prime(y, output)
            for layer in reverse(network):
                grad = layer.backward(grad, learning_rate)
        error /= len(x_train)
        print(f"{e+1}/{epochs}, error={error}")
```

Date:
Page:

xor.py

```
import numpy as np
from dense import Dense
from activation import Tanh
from losses import mse, mse_prime
from network import predict, fit

X = np.reshape([[0,0], [0,1], [1,0], [1,1]], (4,2,1))
Y = np.reshape([[0], [1], [1], [0]], (4,1,1))

network = [ Dense(2, 3),
            Tanh(),
            Dense(3, 1)
            Tanh() ]

fit(network, mse, mse_prime, X, Y, epochs=1000,
    learning_rate=0.1)
```