CNN From Scratch

Here five major topic of CNN is discussed

1. Convolution operation
2. Convolution layer.
3. Reshape Layer
4. Binary Cross Entropy Loss
5. Sigmoid Activation.

1. Convolution / Correlation Operation

Cross-Correlation

Suppose we have given input matrix, arbitrarily big and the smaller one - kernel or filter. The cross-correlation between input and kernel will create another matrix. The values contain by output matrix are obtained by sliding the kernel on the top of input and by performing a multiplication of adjacent values then summing them up.

The size of output will be size of input minus size of kernel plus one.

$\therefore Y = I - K + 1$

Date:
Page

for example.

1	6	2
5	3	1
7	0	9

input 3X3

$\star$

1	2
-1	0

kernel 2X2

$=$

a	b
c	d

output 2X2.

where

$a = 1 \times 1 + 6 \times 2 + 5 \times (-1) + 3 \times 0 = 8$

$b = 6 \times 1 + 2 \times 2 + 3 \times (-1) + 1 \times 0 = 7$

$c = 5 \times 1 + 3 \times 2 + 7 \times (-1) + 0 \times 0 = 4$

$d = 3 \times 1 + 1 \times 2 + 0 \times (-1) + 4 \times 0 = 5$

$\therefore$ Output matrix $= \begin{bmatrix} 8 & 7 \\ 4 & 5 \end{bmatrix}_{2 \times 2}$

Output size $= 3 - 2 + 1 = 2.$

$\star$ is cross correlation operator

$*$ is convolution operator.

rot 180° ≡ horizontal flip followed by vertical flip.

Date:
Page:

Convolution Operation

It is same as cross-correlation operation. Only the difference is kernel is rotated by 180°.

i.e.

$$\begin{bmatrix} 1 & 6 & 2 \\ 5 & 3 & 1 \\ 7 & 0 & 4 \end{bmatrix} * \begin{bmatrix} 0 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 5 \\ 11 & 3 \end{bmatrix}$$

input kernel output

rot180(kernel)

In other word convolution between I and k is cross correlation between I and rotated version of k.

$$conv(I, k) = I \star rot\,180^\circ(k)$$

$$\therefore I * k = I \star rot180(k)$$

Date:
Page:

Valid Cross-correlation

$$\begin{bmatrix} 1 & 6 & 2 \\ 5 & 3 & 1 \\ 7 & 0 & 4 \end{bmatrix} \star \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 7 \\ 4 & 5 \end{bmatrix}$$

Start computing the product by placing the kernel entirely on the top of input and stop sliding the kernel when it hits the boarder of input.

Full Cross-Correlation

$$\begin{bmatrix} 1 & 6 & 2 \\ 5 & 3 & 1 \\ 7 & 0 & 4 \end{bmatrix} \overset{\text{full}}{\star} \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

In this mode we start computing the product as soon as there is intersection between kernel and the input. Hence,

$a = 0 \times 1 = 0$

$b = -1 \times 1 + 0 \times 6 = -1$

$c = -1 \times 6 + 0 \times 2 = -6$

$d = -1 \times 2 = -2$

$e = 2 \times 1 + 0 \times 5 = 2$

$f = 1 \times 1 + 2 \times 6 + -1 \times 5 + 0 \times 3 = 8$

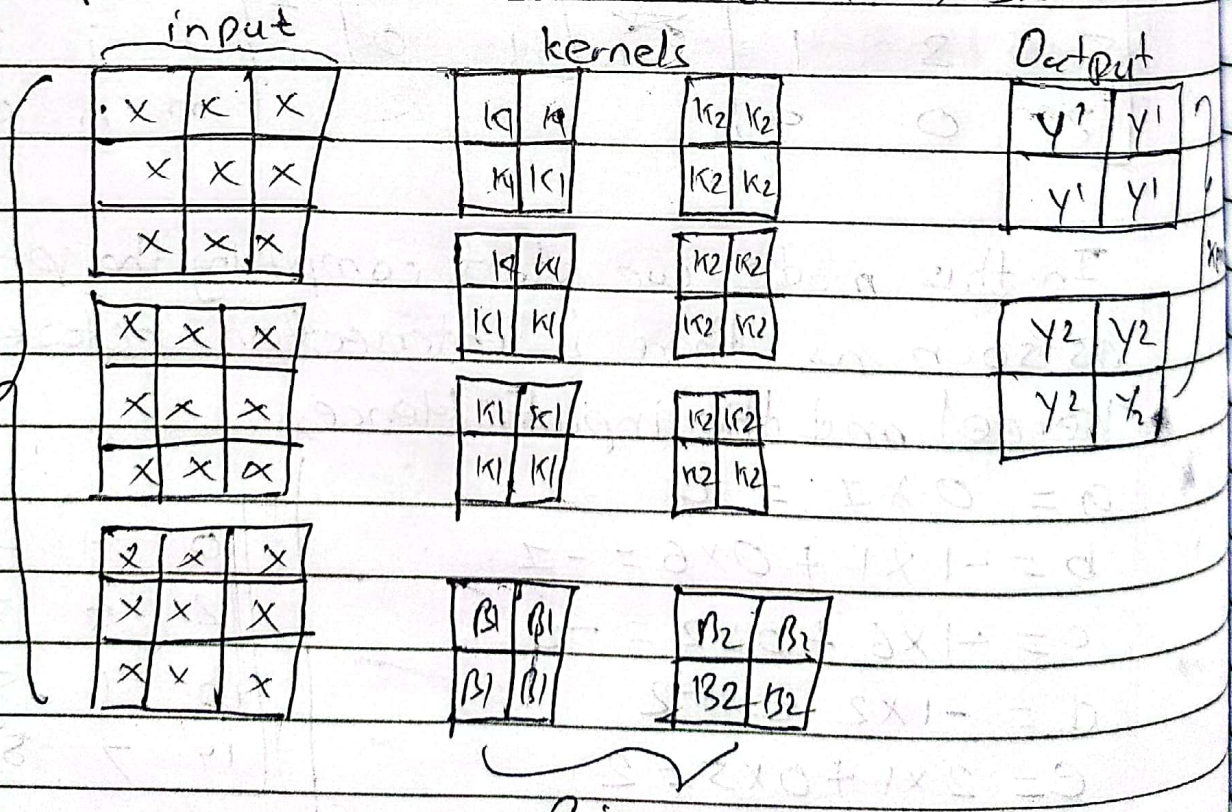
.....

$$\begin{bmatrix} 0 & -1 & -6 & -2 \\ 2 & 8 & 7 & 1 \\ 10 & 4 & 5 & -3 \\ 14 & 7 & 8 & 4 \end{bmatrix}$$

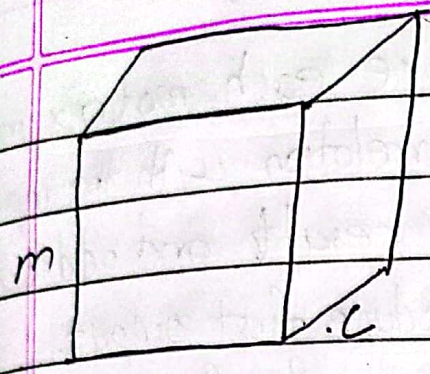
Date:
Page:

2) Convolution Layer

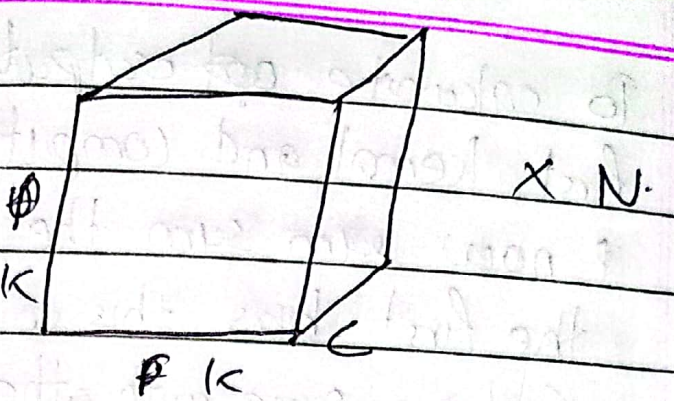
The convolution layer takes a 3-dimensional block of data as input, in that case the depth of i/p is 3. The layer has trainable parameters among them the kernels. Each kernel is also a three dimensional block that extend the full depth of input. The layer may have multiple kernels all of which must extend the full depth of i/p. To each kernel we associate a bias matrix that will have same shape as the outputs. Finally the layer will produce the three dimensional data as output. The depth of output is the same as no. of kernels.



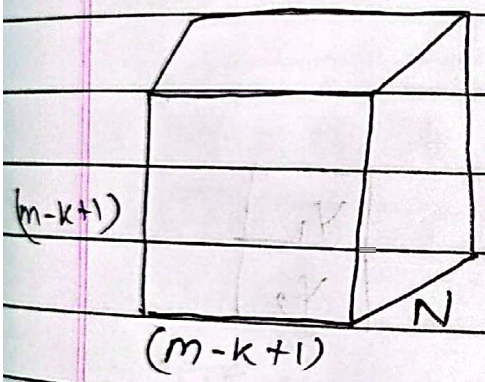
Date:
Page:



Input of (m, m) height & height and 'c' depth



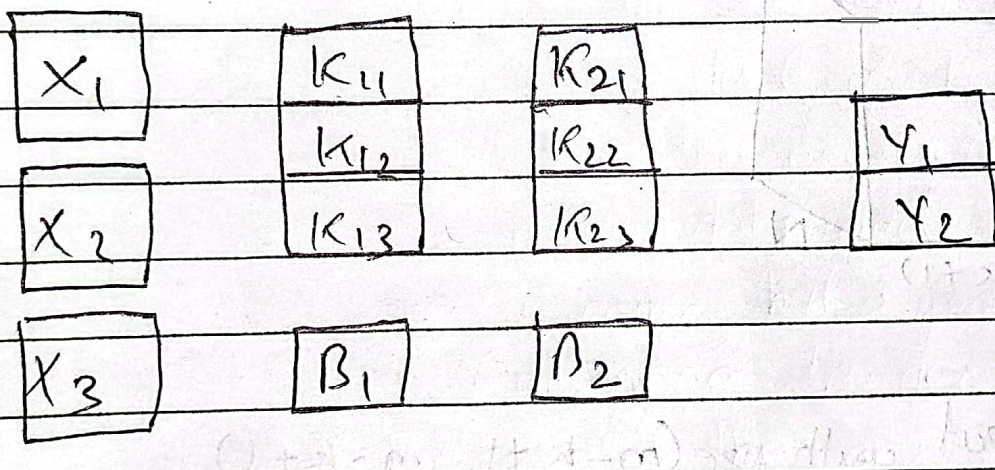
kernel of size (k, k) with depth 'c'
N is number of such kernl



Output with size (m-k+1, m-k+1) and N as depth.

Date:
Page:

To calculate ~~opt~~ output, take each matrix in first kernal and compute correlation with the input ~~f~~ now sum sum the three result and add up the first bias, this will produce first output.
Continue same with other values ~~for~~ first output.
Like wise second output is computed by taking the sum of correlation between ~~ip~~ input and second kernal plus second bias



$$Y_1 = B_1 + X_1 \star K_{11} + X_2 \star K_{12} + X_3 \star K_{13}$$
$$Y_2 = B_2 + X_1 \star K_{21} + X_2 \star K_{22} + X_3 \star K_{23}$$

For d number of kernals.

$$Y_d = B_d + X_1 \star K_{d1} + X_2 \star K_{d2} + X_3 \star K_{d3}$$

~~For input of depth D.~~

Page

For input of depth n.

$$Y_1 = B_1 + X_1 \star K_{11} + \cdots + X_n \star K_{1n}$$
$$Y_2 = B_2 + X_1 \star K_{21} + \cdots + X_n \star K_{2n}$$
$$\vdots$$
$$Y_d = B_d + X_1 \star K_{d1} + \cdots + X_n \star K_{dn}.$$

This is forward propagation of convolution layer.

$$\therefore \boxed{Y_i = B_i + \sum_{j=1}^{n} X_j \star K_{ij}}, \quad i = 1 \cdots d.$$

Here we cannot directly perform matrix operation, as variables as not scalor, they are matrix and $\star$ operator is not just multiplication operator.

let $\cdot 1\star$ is an operator which can perform same task then we can write.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_d \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_d \end{bmatrix} + \begin{bmatrix} K_{11} & K_{12} \cdots K_{1n} \\ K_{21} & K_{22} \cdots K_{2n} \\ \vdots & \vdots \\ K_{d1} & K_{d2} \cdots K_{dn} \end{bmatrix} \cdot 1\star \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

$$\therefore Y = B + K \cdot 1\star X$$

Date :
Page :

It is the same equation for dense layer, except that instead of having matrices of scalar, we have matrices of matrices and instead of regular dot product we have cross-correlated dot product.

If each of these matrices were one dimensional (i.e. each matrix contains a single element) then the cross correlation between two matrices containing one element is equal to one-dimensional matrix containing the product of both of these elements and therefore this new operator ends of being a regular dot product. Hence convolution layer is nothing but a generalization of dense layer. (or rather the dense layer is a particular case of the convolution layer).

Backward propagation

As earlier, assume we have $\frac{\partial \epsilon}{\partial y_i}$ then we have to compute derivative of ϵ w.r.t to trainable parameter of the layer ie kernels and biases and also need to compute the derivative of ϵ w.r.t to input of the layer

$$\frac{\partial \epsilon}{\partial y_i} \rightarrow \frac{\partial \epsilon}{\partial k_{ij}}, \frac{\partial \epsilon}{\partial B_i}$$

$$\frac{\partial \epsilon}{\partial y_i} \rightarrow \frac{\partial \epsilon}{\partial x_j}$$

(1) for $\frac{\partial \epsilon}{\partial k_{ij}}$

From eqn of forward propagation

$$Y_i = B_i + \sum_{j=1}^{n} X_j * K_{ij}$$

Expanding sum

$$Y_i = B_i + X_1 * K_{i1} + \cdots + X_n * K_{in}$$

Simplifying this eqn

$$Y_i = B_i + X_1 * K_{i1}$$

Date :
Page :

$$\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} + \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$$

$$y_{11} = b_{11} + k_{11}x_{11} + k_{12}x_{12} + k_{21}x_{21} + k_{22}x_{22}$$
$$y_{12} = b_{12} + k_{11}x_{12} + k_{12}x_{13} + k_{21}x_{22} + k_{22}x_{23}$$
$$y_{21} = b_{21} + k_{11}x_{21} + k_{12}x_{22} + k_{21}x_{31} + k_{22}x_{32}$$
$$y_{22} = b_{22} + k_{11}x_{22} + k_{12}x_{23} + k_{21}x_{32} + k_{22}x_{33}$$

Now,

$$\frac{\partial E}{\partial k_{11}} = \frac{\partial E}{\partial y_{11}}\frac{\partial y_{11}}{\partial k_{11}} + \frac{\partial E}{\partial y_{12}}\frac{\partial y_{12}}{\partial k_{11}} + \frac{\partial E}{\partial y_{21}}\frac{\partial y_{21}}{\partial k_{11}} + \frac{\partial E}{\partial y_{22}}\frac{\partial y_{22}}{\partial k_{11}}$$

or, $$\frac{\partial E}{\partial k_{11}} = \frac{\partial E}{\partial y_{11}}x_{11} + \frac{\partial E}{\partial y_{12}}x_{12} + \frac{\partial E}{\partial y_{21}}x_{21} + \frac{\partial E}{\partial y_{22}}x_{22}$$

Similarly

$$\frac{\partial E}{\partial k_{12}} = \frac{\partial E}{\partial y_{11}}x_{12} + \frac{\partial E}{\partial y_{12}}x_{13} + \frac{\partial E}{\partial y_{21}}x_{22} + \frac{\partial E}{\partial y_{22}}x_{23}$$

$$\frac{\partial E}{\partial k_{21}} = \frac{\partial E}{\partial y_{11}}x_{21} + \frac{\partial E}{\partial y_{12}}x_{22} + \frac{\partial E}{\partial y_{21}}x_{31} + \frac{\partial E}{\partial y_{22}}x_{32}$$

$$\frac{\partial E}{\partial k_{22}} = \frac{\partial E}{\partial y_{11}}x_{22} + \frac{\partial E}{\partial y_{12}}x_{23} + \frac{\partial E}{\partial y_{21}}x_{32} + \frac{\partial E}{\partial y_{22}}x_{33}$$

This is cross-correlation between input matrix and the output gradient.

$$\begin{bmatrix} \frac{\partial E}{\partial k_{11}} & \frac{\partial E}{\partial k_{12}} \\ \frac{\partial E}{\partial k_{21}} & \frac{\partial E}{\partial k_{22}} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} \frac{\partial E}{\partial y_{11}} & \frac{\partial E}{\partial y_{12}} \\ \frac{\partial E}{\partial y_{21}} & \frac{\partial E}{\partial y_{22}} \end{bmatrix}$$

$$\therefore \frac{\partial E}{\partial K} = X * \frac{\partial E}{\partial Y}$$

This is for only if $Y = B + X * K$.

Now

For $y_i = B_1 + \sum_{j=1}^{n} X_j * K_{ij} \quad i = 1 \dots d$

$$y_1 = B_1 + X_1 * K_{11} + \dots + X_n * K_{1n}$$
$$y_2 = B_2 + X_1 * K_{21} + \dots + X_n * K_{2n}$$
$$\vdots$$
$$y_d = B_d + X_1 * K_{d1} + \dots + X_n * K_{dn}$$

$$\times \frac{\partial E}{\partial K_{21}} = \frac{\partial E}{\partial y_1} \cdot \frac{\partial y_1}{\partial K_{21}} + \frac{\partial E}{\partial y_2} \cdot \frac{\partial y_2}{\partial K_{21}} + \dots + \frac{\partial E}{\partial y_d} \cdot \frac{\partial y_d}{\partial K_{21}}$$

It is not valid, because Y & K are not scalar, there are matrices, so derivative of matrix is not valid.

$$\frac{d}{dx}(y_1 + y_2 + y_3) = \frac{\partial y_1}{\partial x} + \frac{\partial y_2}{\partial x} + \frac{\partial y_3}{\partial x}$$

Derivative of sum = Sum of derivative

Date:
Page:

$$\frac{\partial E}{\partial K_{21}} = ?$$

Looking for K_{21} in forward propagation eqⁿ K_{21} appears only in second eqⁿ and since the derivative of sum is the sum of the derivative. As far as K_{21} is concerned y_2 only boils down to

$$Y_2 = B_2 + X_1 * K_{21}.$$

So we can write

$$\frac{\partial E}{\partial K_{21}} = X_1 * \frac{\partial E}{\partial Y_2}$$

In general,

$$\boxed{\frac{\partial E}{\partial K_{ij}} = X_j * \frac{\partial E}{\partial Y_i}}$$

(iii) For $\frac{\partial E}{\partial B_i}$ (Bias gradient)

As earlier simplifying forward equation to

$$y_i = B_i + x_1 * K_{i1}$$

$$\frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial y_{11}} \frac{\partial y_{11}}{\partial b_{11}} + \frac{\partial E}{\partial y_{12}} \frac{\partial y_{12}}{\partial b_{11}} + \frac{\partial E}{\partial y_{21}} \frac{\partial y_{21}}{\partial b_{11}} + \frac{\partial E}{\partial y_{22}} \frac{\partial y_{22}}{\partial b_{11}}$$

$$\therefore \frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial y_{11}}$$

Similarly,

$$\frac{\partial E}{\partial b_{12}} = \frac{\partial E}{\partial y_{12}}$$

$$\frac{\partial E}{\partial b_{21}} = \frac{\partial E}{\partial y_{21}}$$

$$\frac{\partial E}{\partial b_{22}} = \frac{\partial E}{\partial y_{22}}$$

$$\therefore \frac{\partial E}{\partial B} = \frac{\partial B}{\partial y}.$$

This is simplified version of problem. So,

$$\frac{\partial E}{\partial B_1} = \frac{\partial E}{\partial y_1}$$

In general

$$\boxed{\frac{\partial E}{\partial B_i} = \frac{\partial E}{\partial y_i}}$$

Date :
Page :

(iii) for $\frac{\partial E}{\partial x_j}$ (input gradient)

Working on simplified problem

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial y_{11}}\frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial E}{\partial y_{12}}\frac{\partial y_{12}}{\partial x_{11}} + \frac{\partial E}{\partial y_{21}}\frac{\partial y_{21}}{\partial x_{11}} + \frac{\partial E}{\partial y_{22}}\frac{\partial y_{22}}{\partial x_{11}}$$

or, $\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial y_{11}} k_{11}$

$\frac{\partial E}{\partial x_{12}} = \frac{\partial E}{\partial y_{11}} k_{12} + \frac{\partial E}{\partial y_{12}} k_{11}$

$\frac{\partial E}{\partial x_{13}} = \frac{\partial E}{\partial y_{12}} k_{12}$

$\frac{\partial E}{\partial x_{21}} = \frac{\partial E}{\partial y_{11}} k_{21} + \frac{\partial E}{\partial y_{21}} k_{11}$

$\frac{\partial E}{\partial x_{22}} = \frac{\partial E}{\partial y_{11}} k_{22} + \frac{\partial E}{\partial y_{12}} k_{21} + \frac{\partial E}{\partial y_{21}} k_{12} + \frac{\partial E}{\partial y_{22}} k_{11}$

$\frac{\partial E}{\partial x_{23}} = \frac{\partial E}{\partial y_{12}} k_{22} + \frac{\partial E}{\partial y_{22}} k_{12}$

$\frac{\partial E}{\partial x_{31}} = \frac{\partial E}{\partial y_{21}} k_{21}$

$\frac{\partial E}{\partial x_{32}} = \frac{\partial E}{\partial y_{21}} k_{22} + \frac{\partial E}{\partial y_{22}} k_{21}$

$\frac{\partial E}{\partial x_{33}} = \frac{\partial E}{\partial y_{22}} k_{22}$

It is the full cross-correlation between output gradient and rotated kernel

$$\begin{bmatrix} \frac{\partial E}{\partial x_{11}} & \frac{\partial E}{\partial x_{12}} & \frac{\partial E}{\partial x_{13}} \\ \frac{\partial E}{\partial x_{21}} & \frac{\partial E}{\partial x_{22}} & \frac{\partial E}{\partial x_{23}} \\ \frac{\partial E}{\partial x_{31}} & \frac{\partial E}{\partial x_{32}} & \frac{\partial E}{\partial x_{33}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_{11}} & \frac{\partial E}{\partial y_{12}} \\ \frac{\partial E}{\partial y_{21}} & \frac{\partial E}{\partial y_{22}} \end{bmatrix} \overset{\text{full}}{\star} \begin{bmatrix} K_{22} & K_{21} \\ K_{12} & K_{11} \end{bmatrix}$$

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial Y} \overset{\text{full}}{\star} \text{rot}180(K)$$

Also we can say

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial Y} \overset{\text{full}}{*} K$$

This is simplied version.

For actual forward equation.

$$\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial y_1} \overset{\text{full}}{*} K_{11} + \cdots + \frac{\partial E}{\partial Y_d} \overset{\text{full}}{*} K_{d1}$$

$$\therefore \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial Y_1} \overset{\text{full}}{*} K_{1j} + \cdots + \frac{\partial E}{\partial Y_d} \overset{\text{full}}{*} K_{dj}$$

$$\therefore \frac{\partial E}{\partial x_j} = \sum_{i=1}^{d} \frac{\partial E}{\partial y_i} \overset{\text{full}}{*} K_{ij} \qquad j = 1 \cdots n$$

Date:

Page:

from scipy import signal.

```
class Convolution(Layer):
    def __init__(self, input_shape, kernel_size, kernel_number):
        input_depth, input_height, input_width = input_shape
        self.kernel_number = kernel_number
        self.input_shape = input_shape
        self.output_shape = (kernel_number, input_height - kernel_size + 1,
                             input_width - kernel_size + 1)
        self.kernels_shape = (kernel_number, input_depth,
                              kernel_size, kernel_size)
        self.kernels = np.random.randn(*self.kernels_shape)
        self.biases = np.random.randn(*self.output_shape)

    def forward(self, input):
        self.input = input
        self.output = np.copy(self.biases)
        for i in range(self.kernel_number):
            for j in range(self.input_depth):
                self.output[i] += signal.correlate2d(self.input[j],
                                  self.kernels[i,j], "valid")
        return self.output
```

```
def backward (self, output_gradient, learning_rate):
    kernels_gradient = np.zeros (self.kernels_shape)
    input_gradient = np.zeros (self.input_shape)

    ~~for i in range (self.depth):~~
        ~~for j in range (self~~
    for i in range (self.kernel_number):
        for j in range (self.input_depth):
            kernels_gradient [i, j] = signal.correlate2d (
                    self.input [j], output_gradient [i], "valid")
            input_gradient [j] += signal.convolve2d (
                    output_gradient [i], self.kernels [i, j], "full")

    # bias gradient should not be calculated as it is equal to
    # output gradient given.
    self.kernels -= learning_rate * kernels_gradient
    self.biases -= learning_rate * output_gradient
    return input_gradient.
```

3) Reshape Layer

It is needed because the output of the convolution layer is 3d block and at the end of network we use dense layers which takes in a column vector as input. Therefore we need to reshape data. hence the reshape layer

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

class Reshape(Layer):
    def __init__(self, input_shape, output_shape):
        self.input_shape = input_shape
        self.output_shape = output_shape

    def forward(self, input):
        return np.reshape(input, self.output_shape)

    def backward(self, output_gradient, learning_rate):
        return np.reshape(output_gradient, self.input_shape)
```

4) Binary Cross Entropy

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \end{bmatrix} \rightarrow \text{Predicted}$$

$y_i \in \{0, 1\}$

$$\hat{Y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_i} \end{bmatrix} \rightarrow \text{Actual}$$

$$E = -\frac{1}{n} \sum_{i=1}^{n} \hat{y_i} \log(y_i) + (1-\hat{y_i}) \log(1-y_i)$$

$$\frac{\partial E}{\partial y_i} = \frac{\partial}{\partial y_1} \left(-\frac{1}{n} \sum_{i=1}^{n} \hat{y_i} \log(y_i) + (1-\hat{y_i}) \log(1-y_i) \right)$$

$$= \frac{\partial}{\partial y_1} \left(-\frac{1}{n} \left(\hat{y_1} \log(y_1) + (1-\hat{y_1}) \log(1-y_1) \right) - \cdots \right.$$

$$\left. \cdots - \frac{1}{n} \left(\hat{y_n} \log(y_n) + (1-\hat{y_n}) \log(1-y_n) \right) \right]$$

$$= \frac{\partial}{\partial y_1} \left(-\frac{1}{n} \left(\hat{y_1} \log(y_1) + (1-\hat{y_1}) \log(1-y_1) \right) \right)$$

$$= -\frac{1}{n} \left(\frac{\hat{y_1}}{y_1} - \frac{1-\hat{y_1}}{1-y_1} \right)$$

$$= \frac{1}{n} \left(\frac{1-\hat{y_1}}{1-y_1} - \frac{\hat{y_1}}{y_1} \right)$$

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

$$\frac{\partial E}{\partial y_i} = \frac{1}{n}\left(\frac{1-y_i}{1-\hat{y}_i} - \frac{y_i}{\hat{y}_i}\right)$$

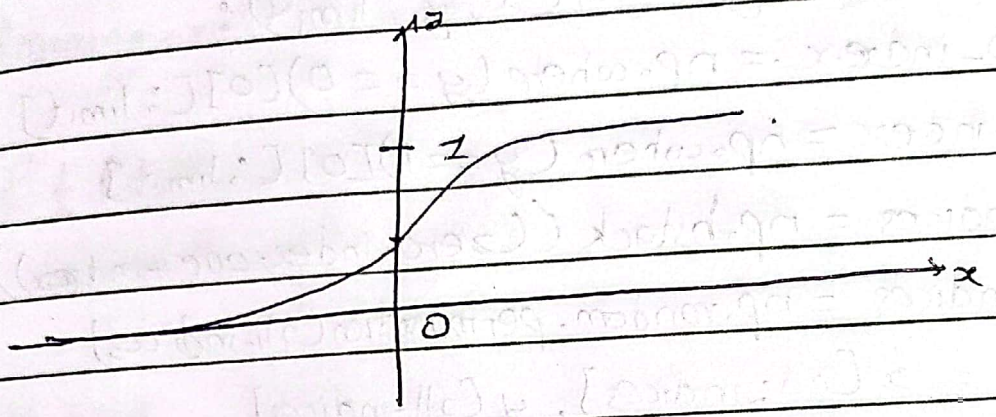
```
import numpy as np.

def binary_cross_entropy(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) +
                    (1-y_true) * np.log(1-y_pred))

def binary_cross_entropy_prime(y_true, y_pred):
    return ((1-y_true)/(1-y_pred) - y_true/y_pred)/np.size(y_true)
```

(5) Sigmod Activation

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

```
import numpy as np
from activation import Activation
class Sigmoid (Activation):
    def __init__(self):
        def sigmoid(x):
            return 1 / (1 + np.exp(-x))

        def sigmoid_prime(x):
            s = sigmoid(x)
            return s * (1 - s)
        super().__init__(sigmoid, sigmoid_prime)
```

Page:

Applying CNN in mnist. ~~for~~ for only two classes

```
#import * all necessary ~~libraries~~ modules made earlier,
from keras.datasets import mnist
from keras.utils import np_utils
def preprocessing_data(x, y, limit):
    zero_index = np.where(y==0)[0][:limit]
    one_index = np.where(y==1)[0][:limit]
    all_indices = np.hstack((zero_index, one_index))
    all_indices = np.random.permutation(all_indices)
    x, y = x[all_indices], y[all_indices]
    x = x.reshape(len(x), 1, 28, 28)
    x = x.astype("float32")/255
    y = np_utils.to_categorical(y)
    y = y.reshape(len(y), 2, 1)
    return x, y

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, y_train = preprocessing_data(x_train, y_train, 100)
x_test, y_test = preprocessing_data(x_test, y_test, 100)

network = [
        Convolutional((1, 28, 28), 3, 5),
        Sigmoid(),
        Reshape((5, 26, 26), (5*26*26, 1)),
        Dense(5*26*26, 100),
        Sigmoid(),
```

```
    Dense(100, 2),
    Sigmoid() ]

epochs = 20
learning_rate = 0.1

fit(network, binary_cross_entropy, binary_cross_entropy_pr
    x_train, y_train, epochs, learning_rate)

# test
for x, y in zip(x_test, y_test):
    output = x
    for layer in network:
        output = layer.forward(output)
    print

for x, y in zip(x_test, y_test):
    output = predict(network, x)
    print(f"pred: {np.argmax(output)},
          true: {np.argmax(y)}")
```