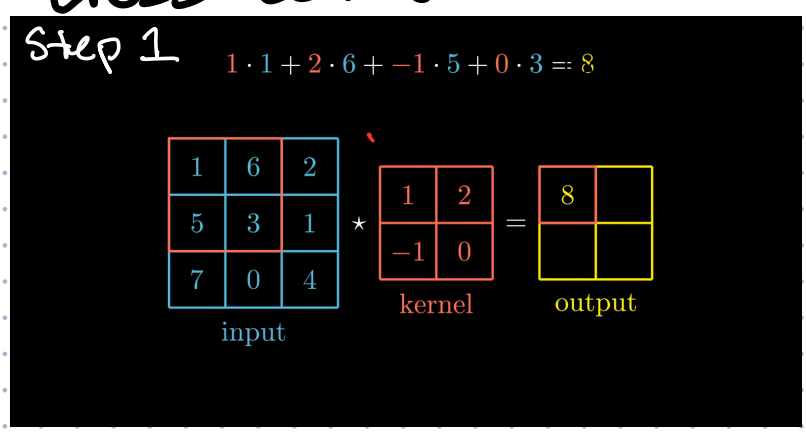
These notes were mainly derived from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

Overview:

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

Convolution walk through

Cross-correlation



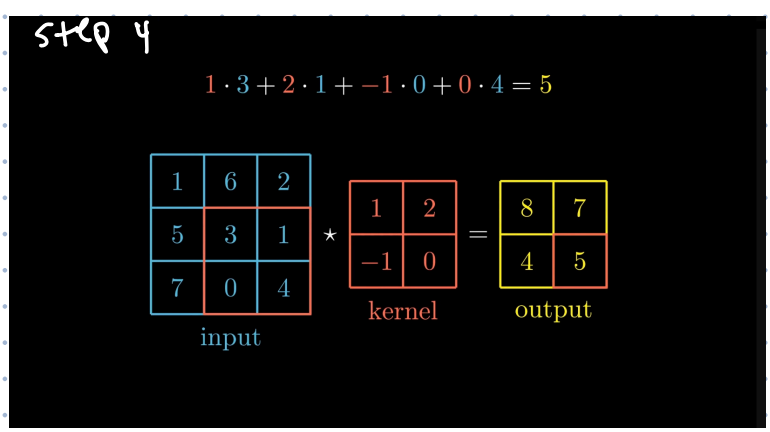
$$\begin{bmatrix} 1 & 6 \\ 5 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}$$

$$= 1 \cdot 1 + 6 \cdot 2 + 5 \cdot -1 + 3 \cdot 0$$

$$1 + 12 + -5 + 0$$

$$8$$

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code



$$\begin{bmatrix} 3 & 1 \\ 0 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}$$

$$= 3 \cdot 1 + 1 \cdot 2 + 0 \cdot -1 + 4 \cdot 0$$

$$3 + 2 + 0 + 0$$

$$5$$

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

Size of Output

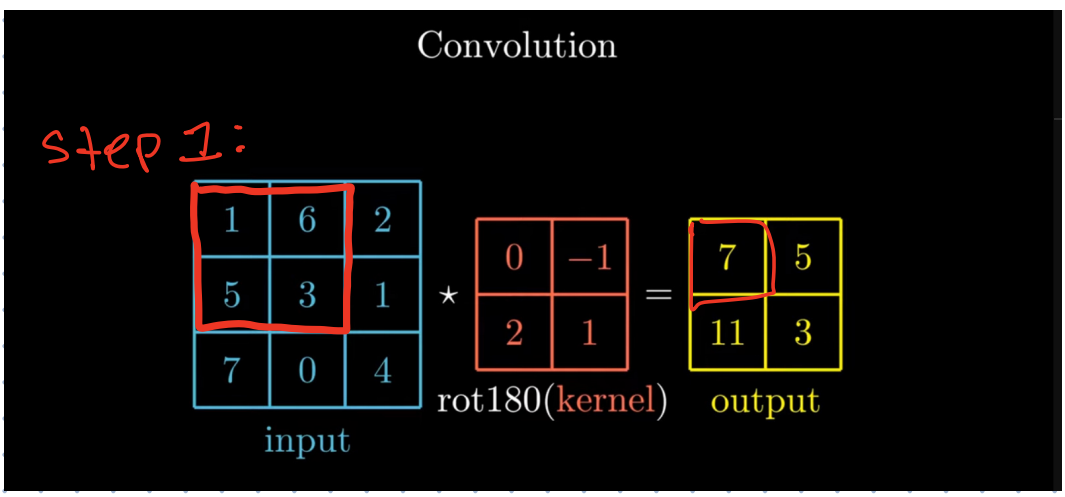
$Y = I - K + 1$

size of output is the size of input subtracted by size of the output plus 1

Example:

$2 = 3 - 2 + 1$

Convolution



*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

step 1: $\begin{bmatrix} 1 & 6 \\ 5 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & -1 \\ 2 & 1 \end{bmatrix} = 1 \cdot 0 + 6 \cdot -1 + 5 \cdot 2 + 1 \cdot 3$

$0 + -6 + 10 + 3$

$[7]$

Cross-correlation ≠ convolution

Math Notation

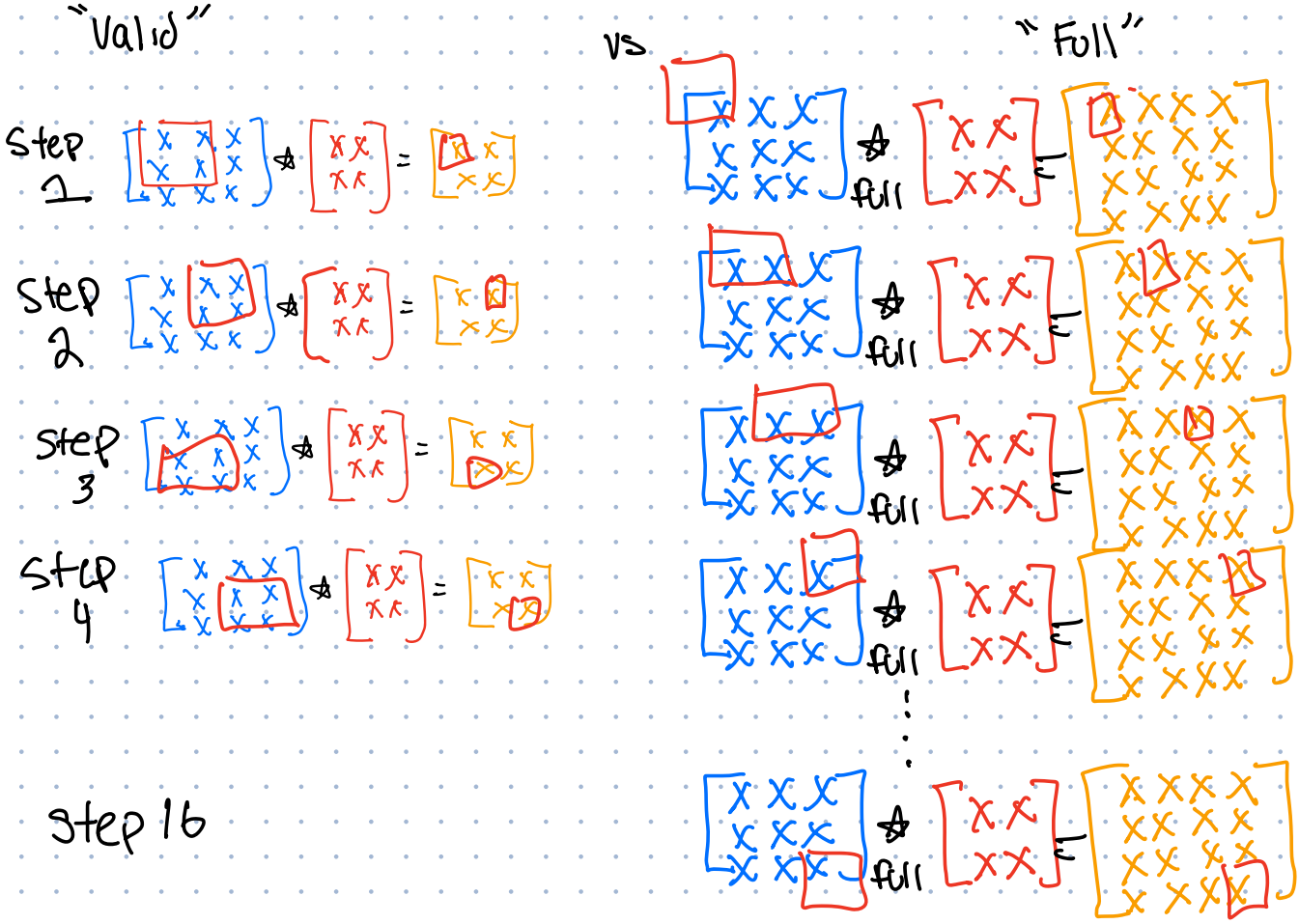
I = input matrix
K = Kernal matrix
O = output matrix

$$Conv(I, K) = I * rot180(K)$$

$$I * K = I \star rot180(K)$$

$*$ → Convolution

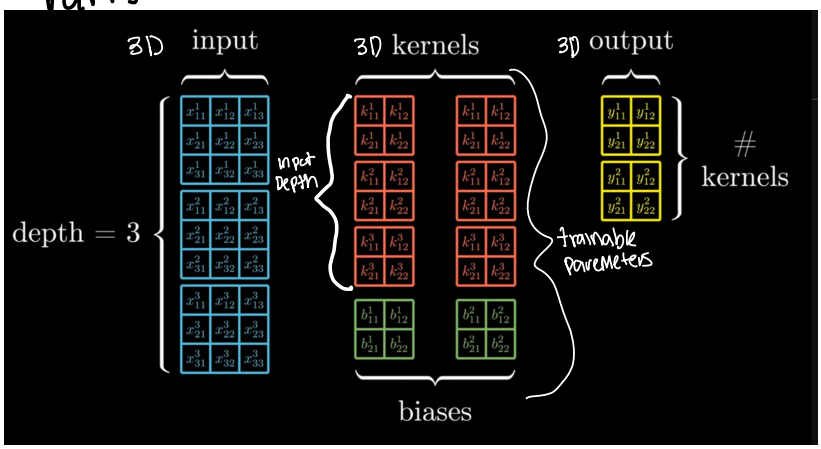
$\star$ → Cross-Correlation



*Screenshot from Neural Network from Scratch | Math

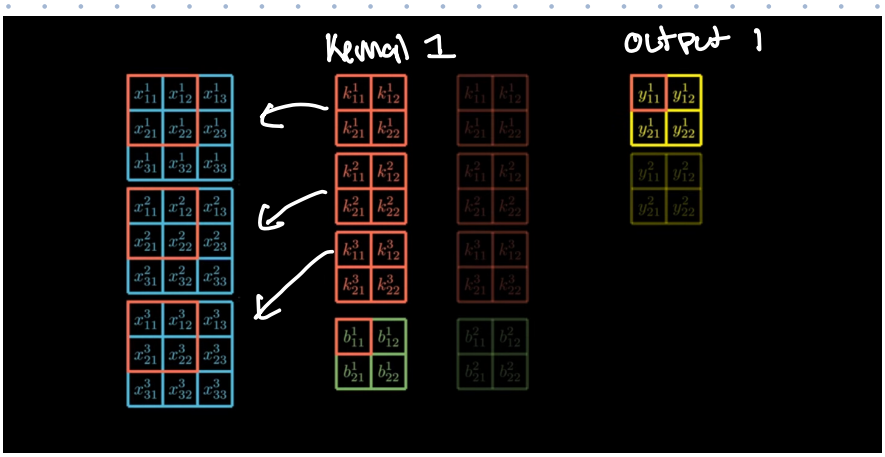
Convolutional Layer

Parts:



*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

output calculation



*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

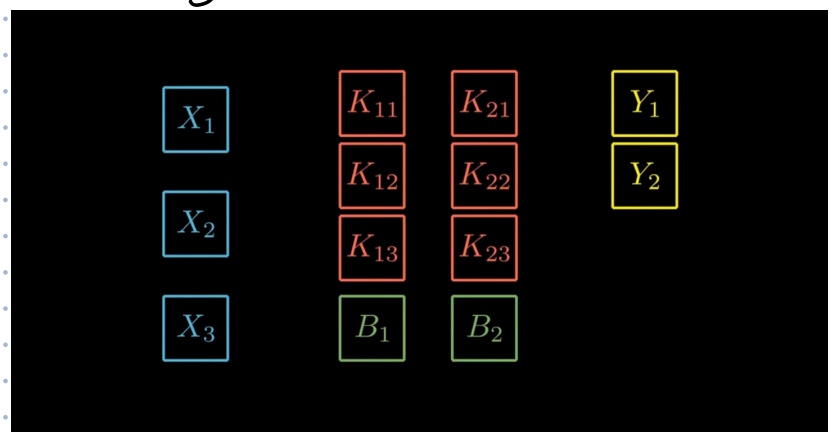
$$\begin{bmatrix} x^1_{11} & x^1_{12} \\ x^1_{21} & x^1_{22} \end{bmatrix} * \begin{bmatrix} k^1_{11} & k^1_{12} \\ k^1_{21} & k^1_{22} \end{bmatrix} = O_1$$

$$\begin{bmatrix} x^2_{11} & x^2_{12} \\ x^2_{21} & x^2_{22} \end{bmatrix} * \begin{bmatrix} k^2_{11} & k^2_{12} \\ k^2_{21} & k^2_{22} \end{bmatrix} = O_2$$

$$\begin{bmatrix} x^3_{11} & x^3_{12} \\ x^3_{21} & x^3_{22} \end{bmatrix} * \begin{bmatrix} k^3_{11} & k^3_{12} \\ k^3_{21} & k^3_{22} \end{bmatrix} = O_3$$

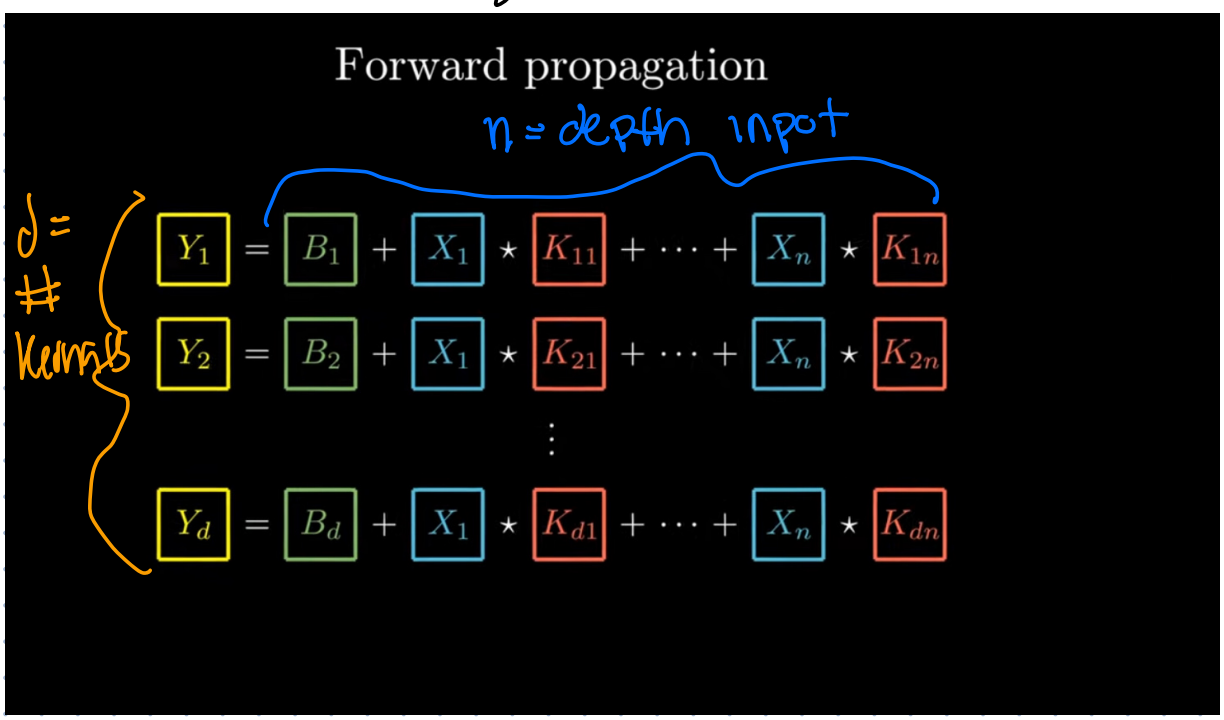
$$O_1 + O_2 + O_3 + B = y^1_{11}$$

Abstracting the matrixs :



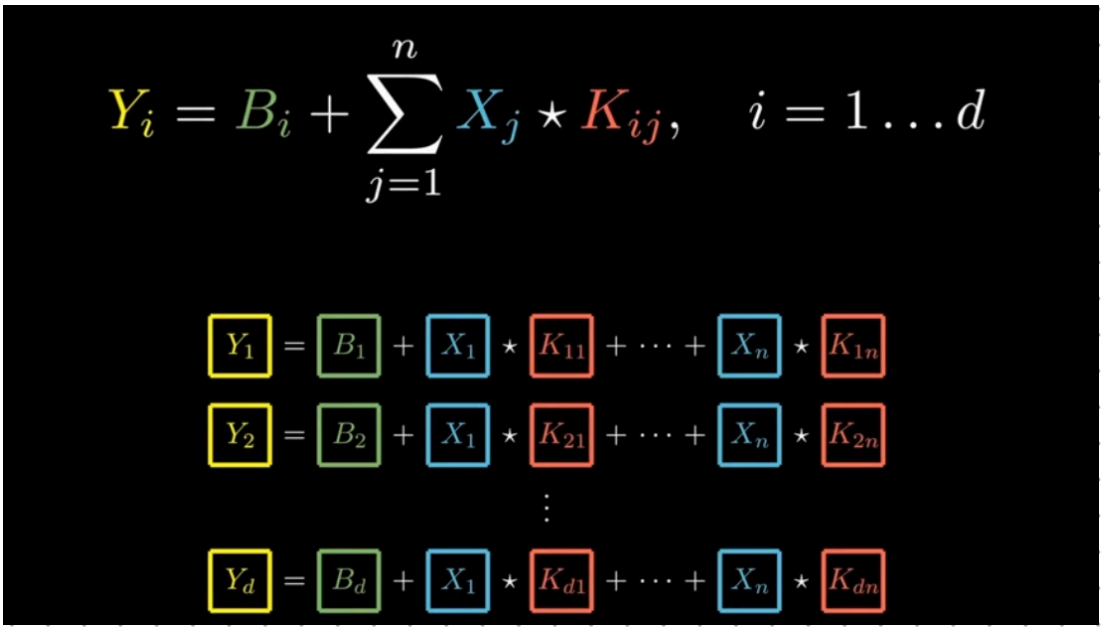
*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

Forward Propgration mode



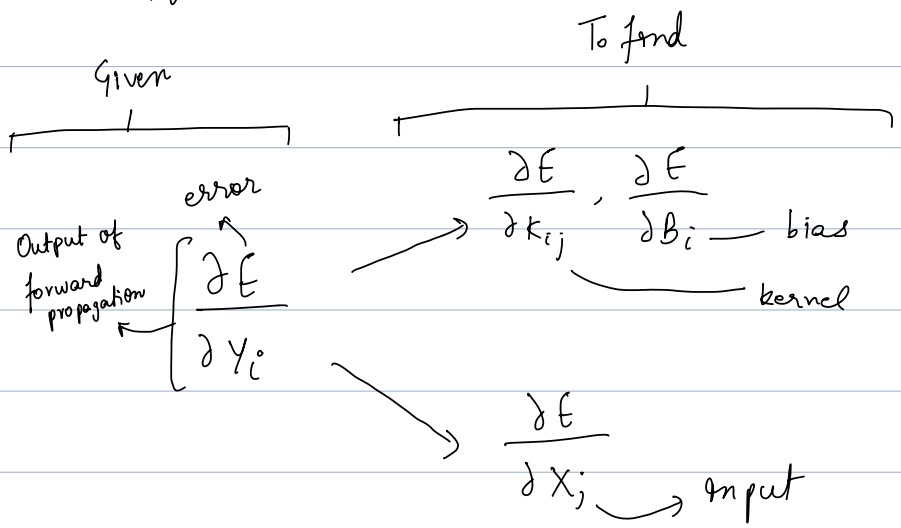
*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

Written in Summation form



*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

✡ Backward Propagation Overview



① $\frac{\partial E}{\partial k_{ij}}$: Looking at single term of foward propagation(Y_i) to generalize:

✡

$\frac{\partial E}{\partial Y} =$  →

Expanded

$$Y = B + X * K$$

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

To find: expanded using chain rule

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

$$\therefore \frac{\partial}{\partial k_{11}}(y_{11}) = \frac{\partial}{\partial k_{11}}(k_{11}x_{11}) = x_{11}$$

$$\Rightarrow x_{11} = \frac{\partial y_{11}}{\partial k_{11}}$$

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

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

$\therefore$ we have $\frac{\partial E}{\partial K_{ij}} = X_j * \frac{\partial E}{\partial Y_i}$

(2) $\frac{\partial E}{\partial B_i}$: Looking at single term of foward propagation(Y_i) to generalize:

$Y = B + X * K$

$$\frac{\partial E}{\partial Y} = \begin{bmatrix} \frac{\partial E}{\partial y_{11}} & \frac{\partial E}{\partial y_{12}} \\ \frac{\partial E}{\partial y_{21}} & \frac{\partial E}{\partial y_{22}} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\partial E}{\partial b_{11}} & \frac{\partial E}{\partial b_{12}} \\ \frac{\partial E}{\partial b_{21}} & \frac{\partial E}{\partial b_{22}} \end{bmatrix}$$

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

$\Downarrow$

$$\frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial y_{11}}\underbrace{\frac{\partial y_{11}}{\partial b_{11}}}_{1} + \frac{\partial E}{\partial y_{12}}\underbrace{\frac{\partial y_{12}}{\partial b_{11}}}_{0} + \frac{\partial E}{\partial y_{21}}\underbrace{\frac{\partial y_{21}}{\partial b_{11}}}_{0} + \frac{\partial E}{\partial y_{22}}\underbrace{\frac{\partial y_{22}}{\partial b_{11}}}_{0}$$

$$\frac{\partial y_{11}}{\partial b_{11}} = \frac{\partial (b_{11} + k \cdot x_{11} + \dots)}{\partial b_{11}} = 1$$

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

$\therefore$ we have $\frac{\partial E}{\partial B_i} = \frac{\partial E}{\partial Y_i}$

(3) $\frac{\partial E}{\partial X_i}$

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial y_{11}} k_{11} + \frac{\partial E}{\partial y_{12}} \cdot \overbrace{\frac{\partial y_{12}}{\partial x_{11}}}^{0} + \frac{\partial E}{\partial y_{21}} \cdot \overbrace{\frac{\partial y_{21}}{\partial x_{11}}}^{0} + \frac{\partial E}{\partial y_{22}} \cdot \overbrace{\frac{\partial y_{22}}{\partial x_{11}}}^{0}$$

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

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

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

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

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

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

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

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

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

$\Rightarrow$

$$\begin{bmatrix} \frac{\partial E}{\partial y_{11}} & \frac{\partial E}{\partial y_{12}} \\ \frac{\partial E}{\partial y_{21}} & \frac{\partial E}{\partial y_{22}} \end{bmatrix} \overset{*}{full} \begin{bmatrix} k_{22} & k_{21} \\ k_{12} & k_{11} \end{bmatrix}$$

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

$$\Rightarrow \frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} \underset{full}{*} rot180(K) \quad [\text{full cross-correlation}]$$

$$= \frac{\partial E}{\partial Y} \underset{full}{*} K \quad [\text{full-convolution}]$$

High Level Explanations of the rest of the Implementation

Note: these parts are not CNN specific

3) Reshape Layer

- ☆ has the functionality to reshape the data
  - ↳ necessary b/c the output of the conv. is 3D
  - ↳ we use dense layers typically @ end of network
    - ☆ specifically, column vectors

→ uses numpy to reshape input → output shape ; output gradient → input shape

4) Binary Cross Entropy Loss

↳ used for classification of only 2 digits of MNIST (we will be attempting a diff version since we want to classify more than 2 digits if we can)

☆ uses log function

5) Sigmoid Activation

→ inputs bounded between 0-1, since Bin. CEL. can't take negative inputs as it uses a log function

Cross Entropy Loss

$Y^* \rightarrow \begin{bmatrix} y_1^* \\ \downarrow \\ y_i^* \end{bmatrix}$ ✡ is the "right"/desired answer outputs

→ for binary, all y_i^* values can either be 0 or 1 since we're only classifying 2

$Y \rightarrow \begin{bmatrix} y_1 \\ \downarrow \\ y_i \end{bmatrix}$ ✡ is the actual outputs

✡ this is where we compute $\frac{dE}{dY}$ for the backward prop. of the last layer of neural network

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

$$\frac{\partial E}{\partial Y} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \frac{\partial E}{\partial y_2} \\ \vdots \\ \frac{\partial E}{\partial y_i} \end{bmatrix}$$

$$\begin{aligned}
\frac{\partial E}{\partial y_1} &= \frac{\partial}{\partial y_1}\left(-\frac{1}{n}\sum_{i=1}^{n} y_i^* log(y_i) + (1 - y_i^*) log(1 - y_i) \right) \\
&= \frac{\partial}{\partial y_1}\left(-\frac{1}{n}(y_1^* log(y_1) + (1 - y_1^*) log(1 - y_1)) - \cdots - \frac{1}{n}(y_n^* log(y_n) + (1 - y_n^*) log(1 - y_n)) \right) \\
&= \frac{\partial}{\partial y_1}\left(-\frac{1}{n}(y_1^* log(y_1) + (1 - y_1^*) log(1 - y_1)) \right) \\
&= -\frac{1}{n}\left(\frac{y_1^*}{y_1} - \frac{1 - y_1^*}{1 - y_1} \right) \\
&= \frac{1}{n}\left(\frac{1 - y_1^*}{1 - y_1} - \frac{y_1^*}{y_1} \right)
\end{aligned}$$

all these terms disappear since they do not have y_1 terms, ∴ we will be performing a derivative

here we derive by applying chain rule

*Screenshot from Neural Network from Scratch | Mathematics & Python Code by The Independent Code

Sigmoid Activation

* gets applied multiple times throughout network

Sigmoid Function

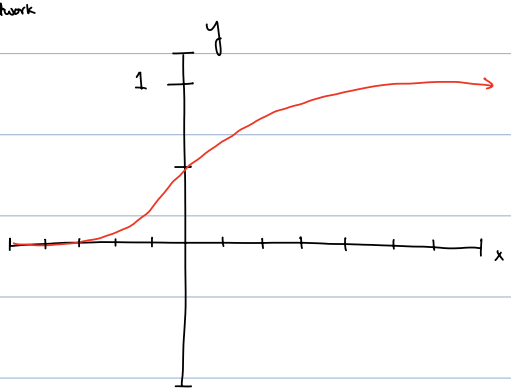
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Sigmoid Derivative

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

or

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



$\sigma(x)$

* as you can see, it is bounded between 0 & 1