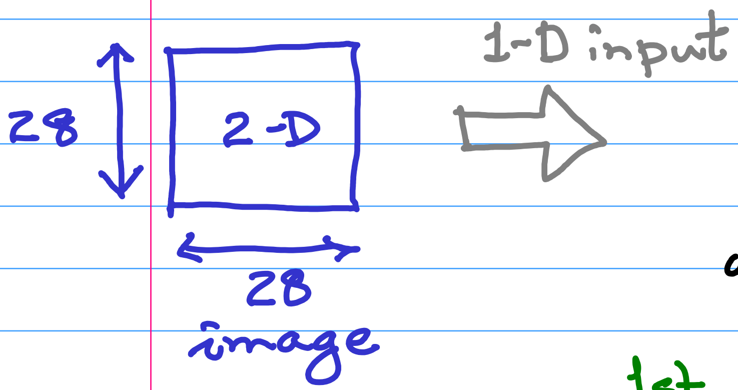
Building Block (Input: 2-D: Image)

MNIST Numeral database: 28×28 images
(greyscale: not binary (not 0 and 255, or normalised 0 and 1))
~smooth
→ shades of grey as well, though most of the image is black (0) or white (255, or 1, normalised)

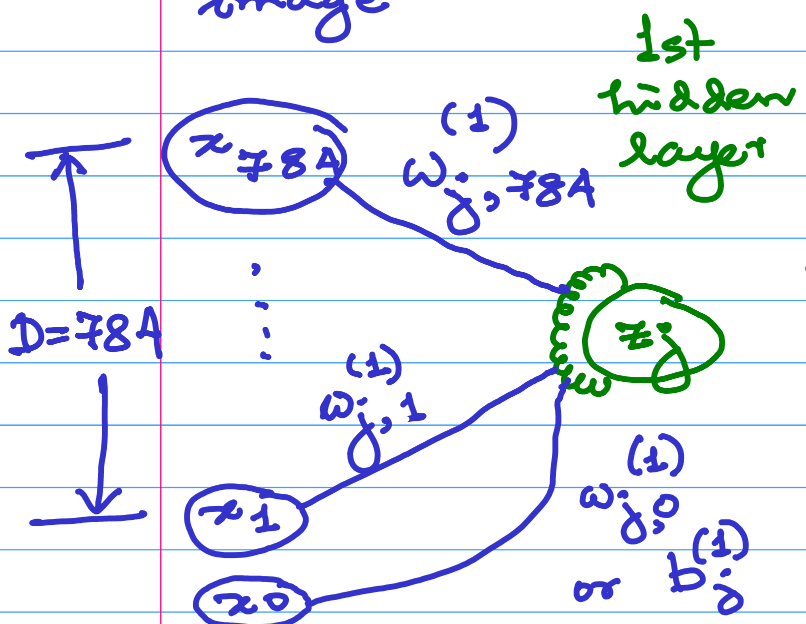
Images of the 10 numerals: 0 to 9

Basic structure: an MLP with 2 hidden layers
→ of specific interest is the first layer of connections



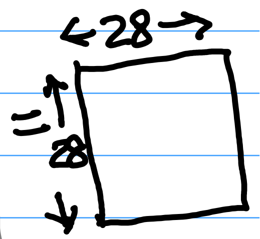
784 × 1 input vector

$w_{ji}^{(1)} \in \mathbb{R}$, a real number



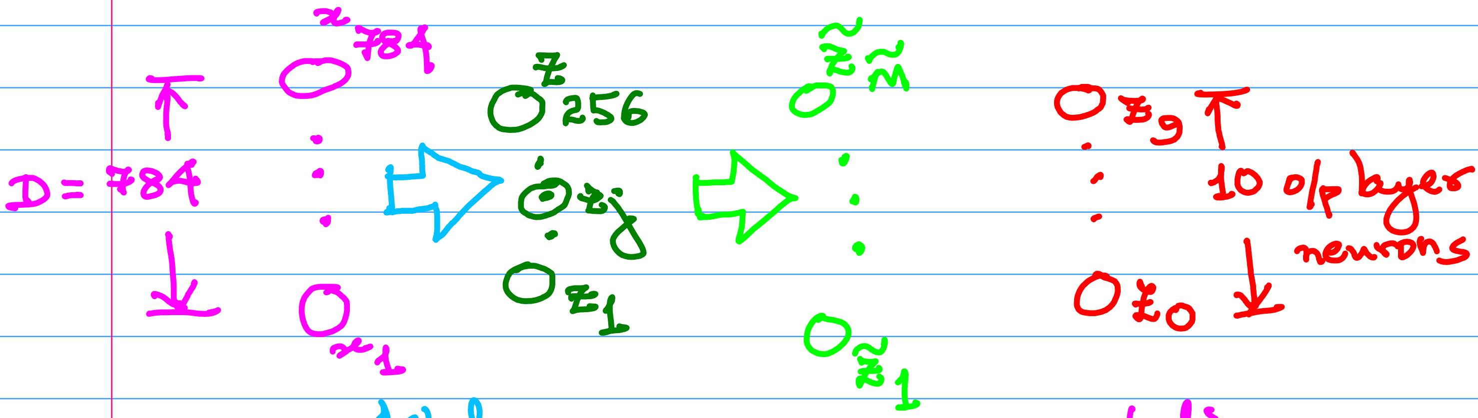
$$\underset{784\times1}{\underline{w}^{(1)}} = \underset{784\times1}{\begin{bmatrix} w_{j,784}^{(1)} \\ \vdots \\ w_{j,1}^{(1)} \end{bmatrix}}$$

$\equiv$



"image" of the weights

Take-home point: The input is not an ordinary 1-D vector, but a 2-D image
$\Rightarrow$ we can visualise the weights not as an ordinary 1-D vector, but a 2-D image with a similar spatial configuration / arrangement as the input itself.



1st layer of connections (fully connected)

(to visualise)

notation

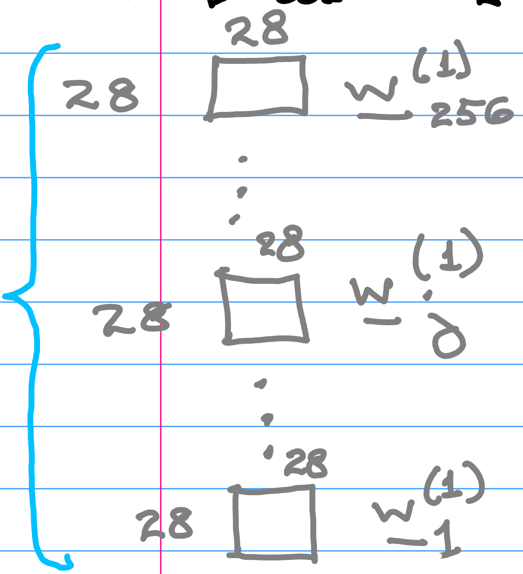
$z_{10} \cdots z_1$

z_9 ("9") $\quad$ z_0 ("0")

classification example

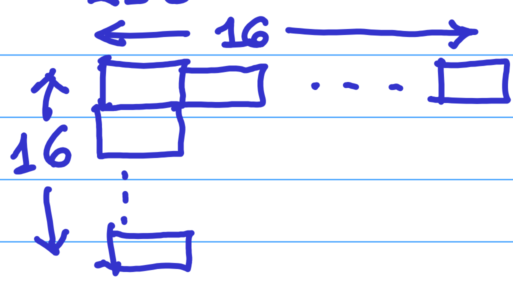
In this example, we are given 256 neurons in the first hidden layer. We will visualise the 784×1 weight vector associated with each of the first hidden layer neurons e.g., $\#j$, $j \in \{1, 256\}$

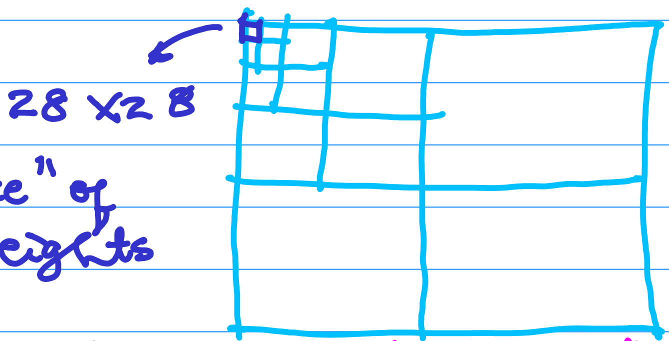
⟶ as a 28×28 "image"



Stack these together

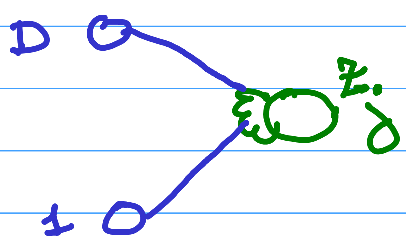
$256 = 2^8 = 2^4 \times 2^4$





Empirical observation: these "images" correspond to directional edge detectors, or directional derivatives, for a network which is subjected to weight learning.

1-D example



$$a_j^{(1)} = \underline{w}_j^{(1)} \underline{x} + b_j^{(1)}$$

or $w_{0,j}^{(1)}$

$$= \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$$

Empirical observation $\to$ most of the learnt weights are zero, except for a select few "close to" the neuron in question.

e.g., $a_j^{(1)} = w_{j,l}^{(1)} x_l + w_{j,l+1}^{(1)} x_{l+1}$

and the other $w_{ji}^{(1)}$'s are $\simeq$ zero

and $w_{j,l}^{(1)} = 1$, $w_{j,l+1}^{(1)} = -1$

$$\Rightarrow a_j^{(1)} = \underbrace{x_l - x_{l+1}}_{\text{difference}}$$

$y = f(x)$

$$\frac{\partial f}{\partial x} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{(x+\delta x) - (x)} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x}$$

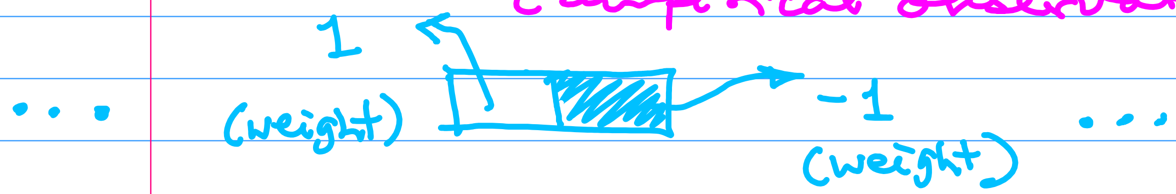
discrete case δx : 1 sampling point

$\delta x = 1$

derivate $\to$ $f[x+1] - f[x]$

discrete approximation of the derivative : in 1-D.

The first layer : computes a "local" derivative.
(Empirical observation)



CONVOLUTION

Neural Networks:

Activation $a = \underbrace{\underline{w}^T \underline{x}}_{\text{sum of pairwise products}} + b$

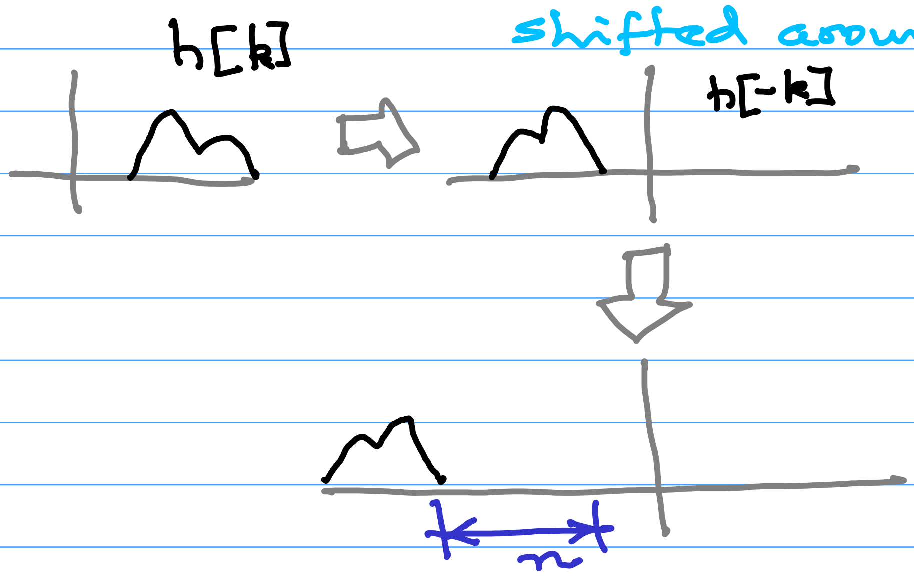
or w_0

[Discrete domain : easier to understand]

Formula for Convolution:

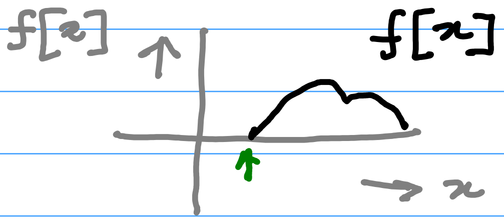
$$x[n] * h[n] \triangleq \sum_k \underbrace{x[k]}_{\text{input}} \; \underbrace{h[n-k]}_{\text{"mask" or system response}}$$

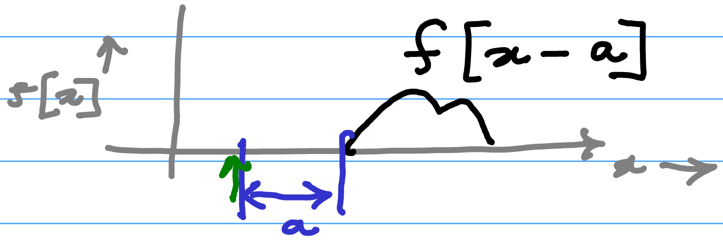
'flipped' and then shifted around



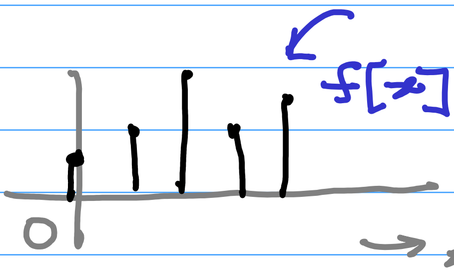
Notes on Convolution:

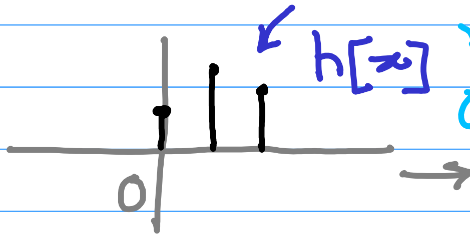
1-D discrete functions





"delayed function"





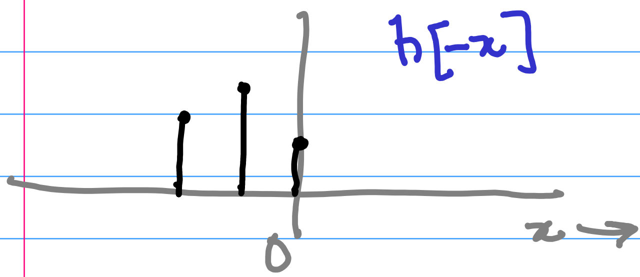
"mask" (the smaller one)

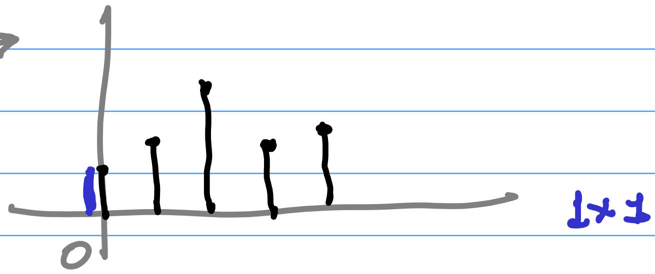
$$f[x] * h[x] = \sum_k f[k]\, h\underbrace{[x-k]}$$

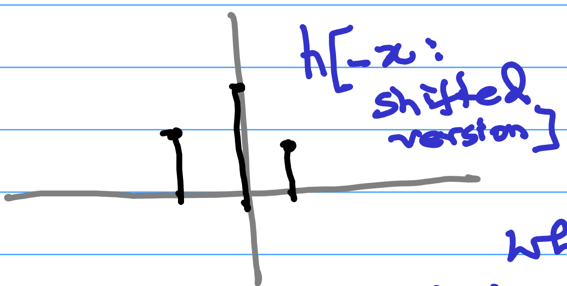
h[−k]: flipped mask

$h[-k+x]$

shift it all around

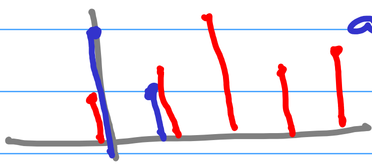






→ shifting all around & looking for overlaps.

wherever there is an overlap, we perform pairwise mult & sum all of them up

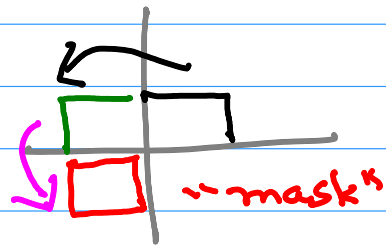


How many overlaps

A + B − 1 (Brigham's Result)

Efficient manner: using arrays

2-D: Conv.



Correlation:
no flipping!