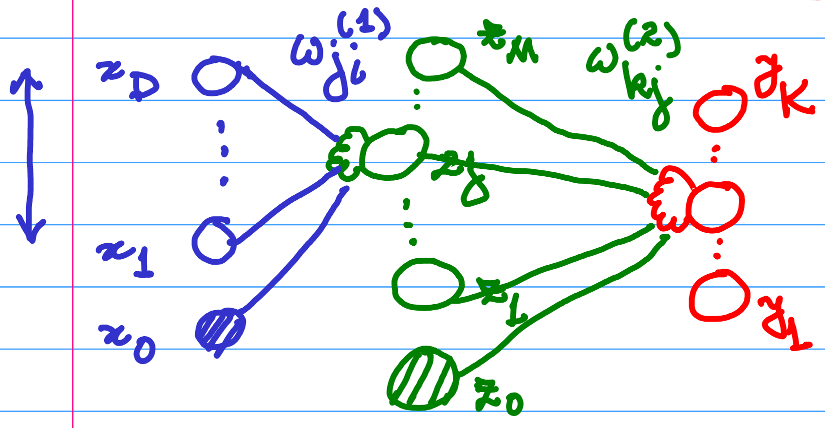
A representative example

BACKPROPAGATION (contd.) [An example]



Assume

→ $h(\cdot)$: activation function is tanh( )

→ $\sigma(\cdot)$: activation function is a unit function

[2 layers of connections]

Why is this called 'Backpropagation'?

→ $\underline{w}$ → Training phase : error is at the output : this is fed back to update the weights

$n \in \{1, N\}$

FOR each pattern in the training set in turn, we follow the following operations.

① A 'forward propagation'

(j: hidden layer)

$$a_j^{(1)} = \underline{w}_j^{(1)T} \underline{x} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = \tanh(a_j)$$

$$y_k = \underline{w}_k^{(2)T} \underline{z} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

② Evaluate δ_k's for each output unit

$$\delta_k = y_k - t_k$$ What is this, and how?

$$E_n \triangleq \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

$$\delta_k \triangleq \frac{\partial E_n}{\partial a_k} = \frac{1}{2}\cdot 2\,(y_k - t_k)\,\frac{\partial y_k}{\partial a_k}$$

output layer activation

$= 1$ as $y_k = a_k$ [$\sigma(\cdot) =$ unit fn.]

$\delta_k = y_k - t_k$ (Else, according to the specific activation function $\sigma(\cdot)$ at the output layer)

(3) <u>Backpropagate these to obtain δ_j's for the hidden layer units</u>

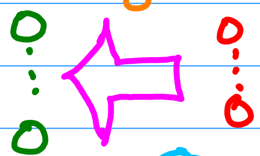
$$\delta_j = (1 - z_j^2)\sum_{k=1}^{K} w_{kj}\,\delta_k$$

What is this, and how?

[hidden layer]

previous step (step #2)

[output layer]



$$\delta_j \triangleq \frac{\partial E_n}{\partial a_j} = \sum_{\forall k} \frac{\partial E_n}{\partial a_k}\,\frac{\partial a_k}{\partial a_j}$$

δ_k [step #2]

$$a_k = \underline{w}^{(2)T}\underline{z} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \sum_{j=0}^{M} w_{kj}^{(2)} h(a_j)$$

$$\Rightarrow \frac{\partial a_k}{\partial a_j} = w_{kj}^{(2)}\frac{\partial h(a_j)}{\partial a_j} = w_{kj}^{(2)} h'(a_j) = w_{kj}^{(2)}(1 - z_j^2)$$

$$\delta_j = \sum_{\forall k} \delta_k\, w_{kj}^{(2)}(1 - z_j^2) = (1 - z_j^2)\sum_{\forall k} w_{kj}^{(2)}\,\delta_k$$

$$E_n \triangleq \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

$$\delta_k \triangleq \frac{\partial E_n}{\partial a_k} = \frac{1}{2}\cdot 2\,(y_k - t_k)\frac{\partial y_k}{\partial a_k}$$

output layer activation

$\frac{\partial y_k}{\partial a_k} = 1$ as $y_k = a_k$ [$\sigma(\cdot)$ = unit fn.]

$\delta_k = y_k - t_k$ (Else, according to the specific activation function $\sigma(\cdot)$ at the output layer)

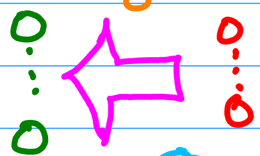
(3) **Backpropagate these to obtain δ_j's for the hidden layer units**

$$\delta_j = (1 - z_j^2)\sum_{k=1}^{K} w_{kj}\,\delta_k$$

What is this, and how?

δ_j → [hidden layer]

δ_k → previous step (step #2) [output layer]



$$\delta_j \triangleq \frac{\partial E_n}{\partial a_j} = \sum_{\forall k}\frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

$\frac{\partial E_n}{\partial a_k}$ → δ_k [step #2]

$$a_k = \underline{w}^{(2)T}\underline{z} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \sum_{j=0}^{M} w_{kj}^{(2)} h(a_j)$$

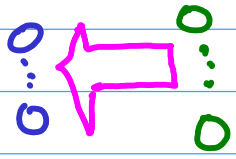
$$\Rightarrow \frac{\partial a_k}{\partial a_j} = w_{kj}^{(2)}\frac{\partial h(a_j)}{\partial a_j} = w_{kj}^{(2)} h'(a_j)$$
$$= w_{kj}^{(2)}(1 - z_j^2)$$

$$\delta_j = \sum_{\forall k}\delta_k\, w_{kj}^{(2)}(1 - z_j^2) = (1 - z_j^2)\sum_{\forall k} w_{kj}^{(2)}\delta_k$$

④ Use the chain rule to evaluate the gradient

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \, , \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

What is this, and how?



$$\vec{\nabla} E = \begin{bmatrix} \vdots \\ \partial E_n / \partial w \\ \vdots \end{bmatrix}$$

all weights (1st & 2nd layers of connections)

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}} \; ; \quad a_j = \underline{w}_j^{(1)T} \underline{x} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

δ_j (previous step: step # 3)

$$\Rightarrow \frac{\partial a_j}{\partial w_{ji}^{(1)}} = x_i$$

$$\Rightarrow \frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} \; ; \quad a_k = \underline{w}_k^{(2)T} \underline{z} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

δ_k (step # 2)

$$\Rightarrow \frac{\partial a_k}{\partial w_{kj}^{(2)}} = z_j$$

$$\Rightarrow \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Side topic: Numerical Evaluation of the gradient

Empirically, all of these alternative methods are numerically not as good as Backpropagation

(*) $$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji})}{\underbrace{\epsilon}_{(w_{ji}+\epsilon)-(w_{ji})}} + o(\epsilon)$$

OR: symmetrical central differences

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon)}{2\epsilon} + o(\epsilon)$$

Why? The first direct formula?

$$E_n(w_{ji}+\epsilon) = E_n(w_{ji}) + \epsilon \frac{\partial E_n}{\partial w_{ji}} + \frac{1}{2}\epsilon H \epsilon + \text{higher order terms}$$

$$\Rightarrow \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji})}{\epsilon} = \frac{\partial E_n}{\partial w_{ji}} + \frac{1}{2}\epsilon H$$

Why? The symmetrical central differences formula:

$$E_n(w_{ji}+\epsilon) = E_n(w_{ji}) + \epsilon \frac{\partial E_n}{\partial w_{ji}} + \frac{1}{2}\epsilon H \epsilon + o(\epsilon^3)$$

$$E_n(w_{ji}-\epsilon) = E_n(w_{ji}) - \epsilon \frac{\partial E_n}{\partial w_{ji}} + \frac{1}{2}\epsilon H \epsilon - o(\epsilon^3)$$

$-$ $\qquad$ $-$ $\qquad$ $+$ $\qquad$ $-$ $\qquad$ $+$

$$E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon) = 2\epsilon \frac{\partial E_n}{\partial w_{ji}} + 2\, o(\epsilon^3)$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon)}{2\epsilon} + O(\epsilon^2)$$

Physical Significance: NN-based solution vis-a-vis the linear (or restricted linear) method done before.

(*) NN: the weight parameters in the first layer are shared between the outputs
linear: each classification is performed independently.

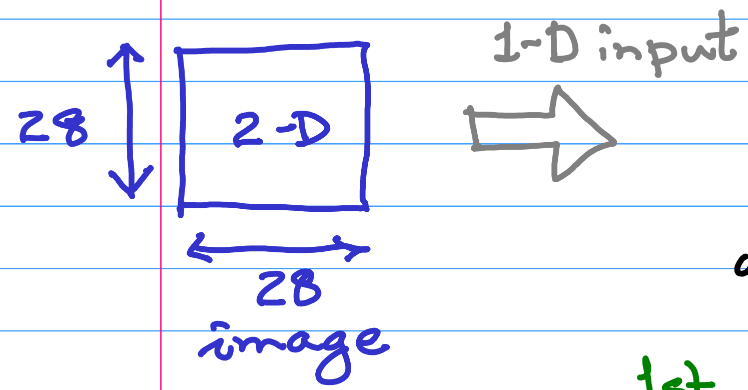
(*) The first layer of the network can be viewed as performing a non-linear feature extraction, and sharing the features between different outputs can lead to savings on computation and also lead to improved generalisation.

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)

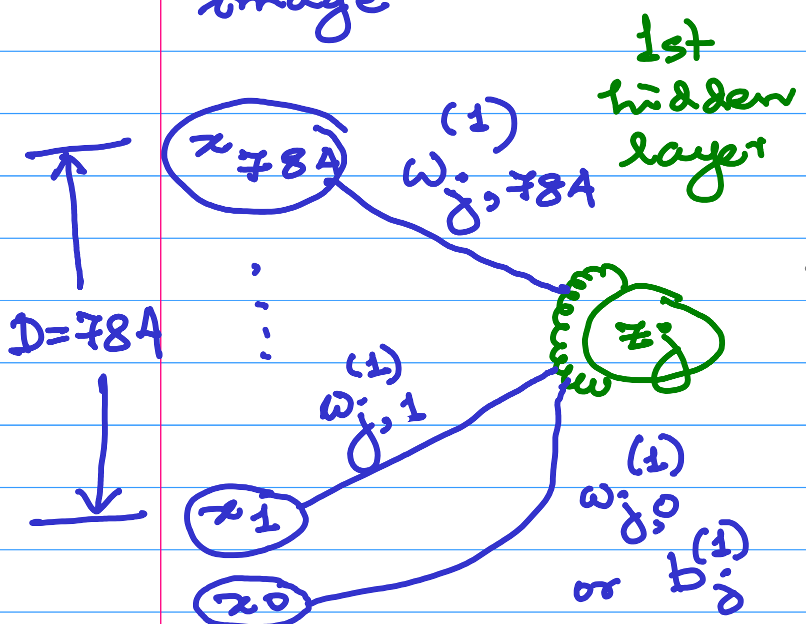
Images of the 10 numerals: 0 to 9

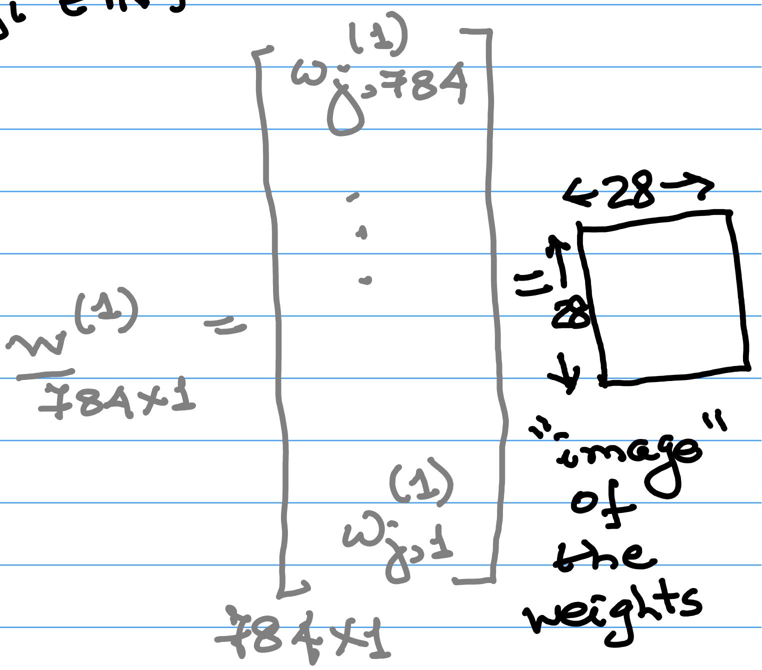
Basic structure: an MLP with 2 hidden layers
→ of specific interest is the first layer of connections



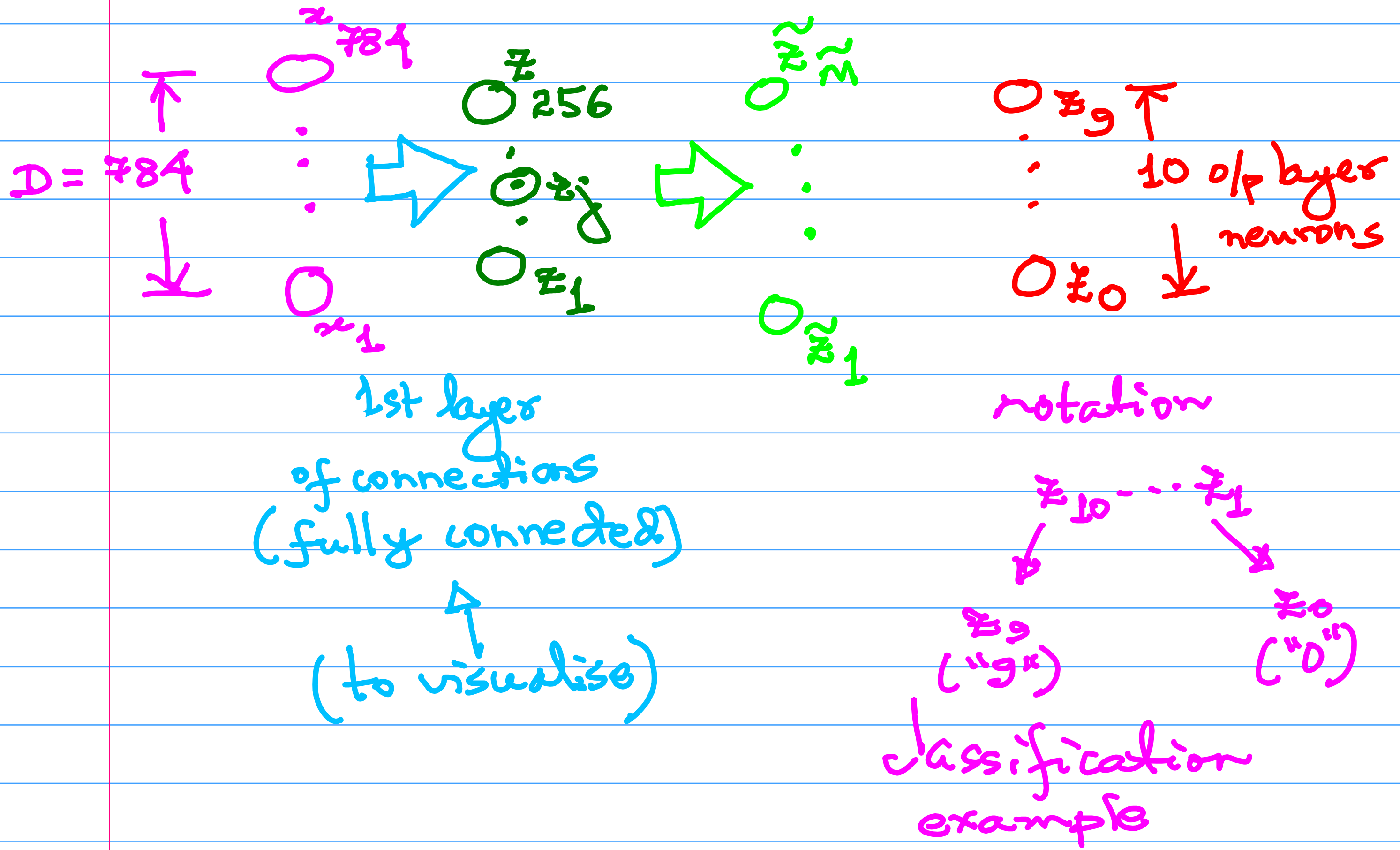
784 × 1 input vector

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



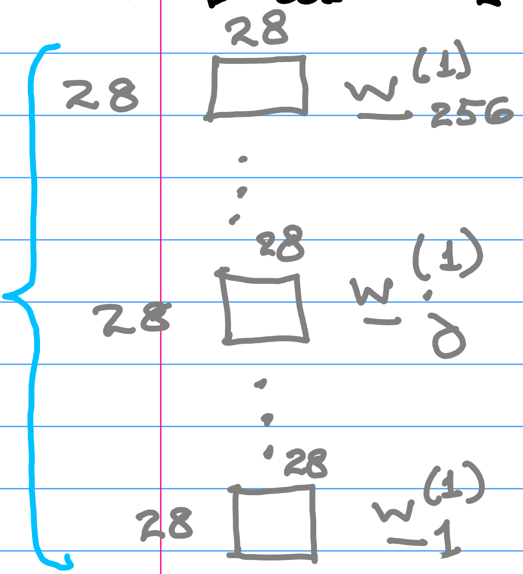


Take-home point: The input is not an ordinary 1-D vector, but a 2-D image
⇒ we can visualise the weights not as an ordinary 1-D vector, but a 2-D image
with a similar spatial configuration/arrangement as the input itself.



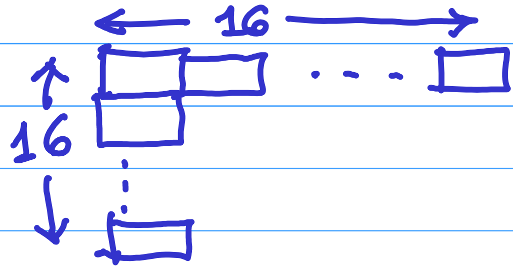
In this example, we are given 256 neurons in the first hidden layer. We will visualise the 784×1 weight vector associated with each of the first hidden layer neurons e.g., #j, $j \in \{1, 256\}$

$\longrightarrow$ as a 28×28 "image"



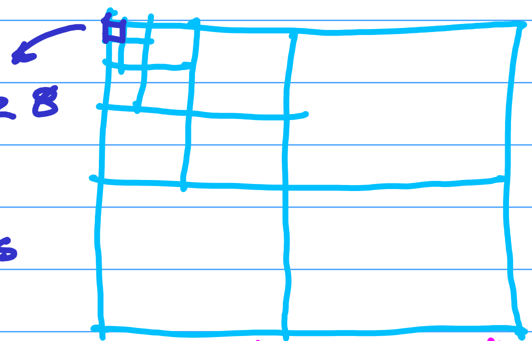
Stack these together

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



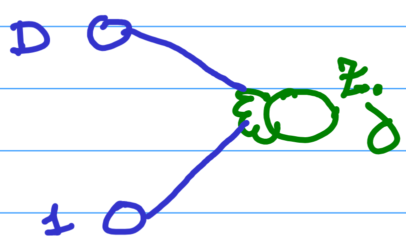
28×28

"image" of the weights



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

1-D example



$$a^{(1)}_j = \underline{w}^{(1)}_j \underline{x} + b^{(1)}_j$$

or $w^{(1)}_{0,j}$

$$= \sum_{i=1}^{2} w^{(1)}_{ji} x_i + b^{(1)}_j$$

Empirical observation → 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 ≃ zero

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

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

$y = f(x)$

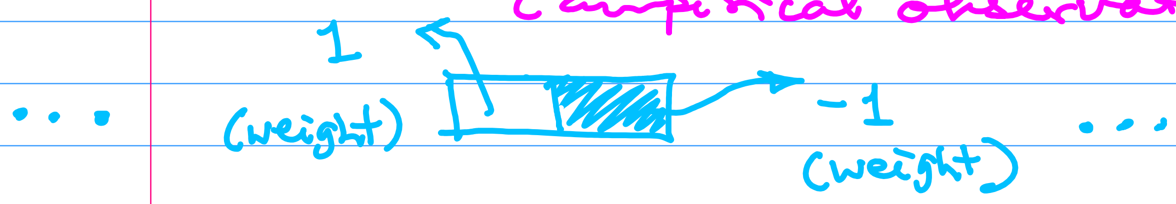
$$\frac{\partial f}{\partial x} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{(x+\delta x) - (x)} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x}$$

discrete case δx : 1 sampling point

$\delta x = 1$

derivate → $f[x+1] - f[x]$
discrete approximation of the derivative : in 1-D.

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



$\boxed{\text{CONVOLUTION}}$

Neural Networks :

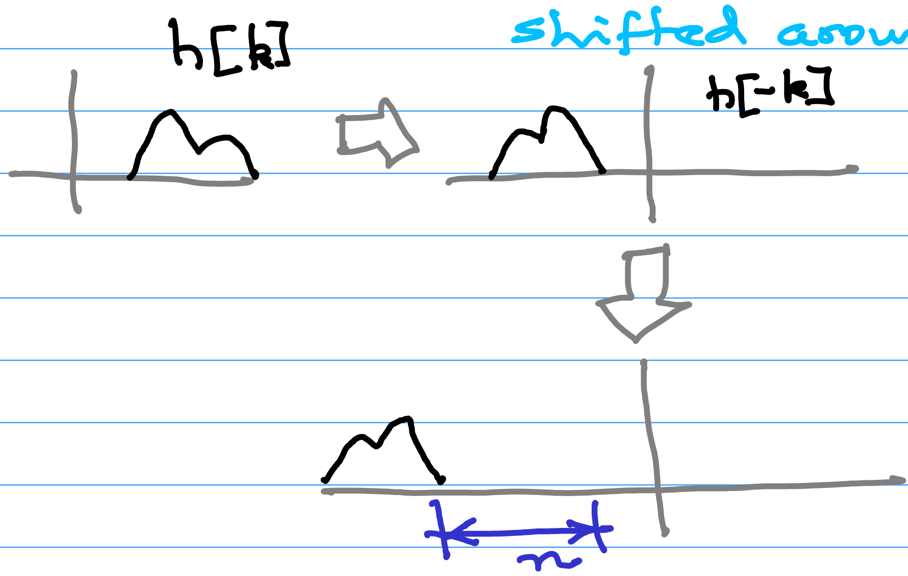
Activation $a = \underbrace{\underline{w}^T \underline{x}}_{\text{sum of pairwise products}} + b$ or w_0

[Discrete domain : easier to understand]

Formula for Convolution:

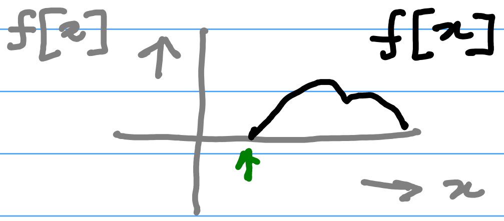
$$x[n] * h[n] \triangleq \sum_k \underbrace{x[k]}_{\text{input}} \; \underbrace{h[n-k]}_{\text{"mask" or system response}}$$

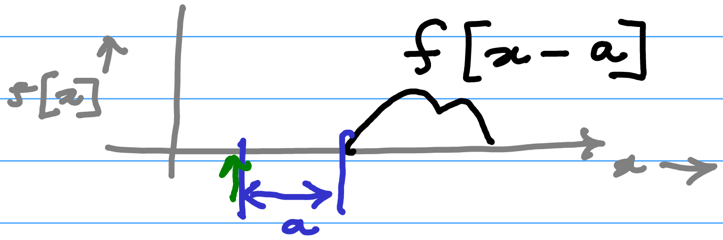
'flipped' and then shifted around



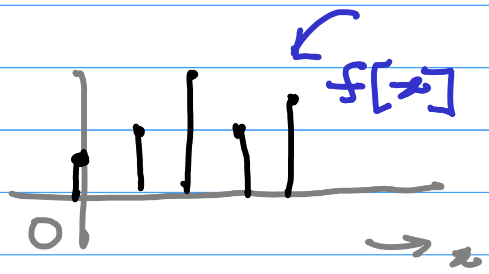
Notes on Convolution:

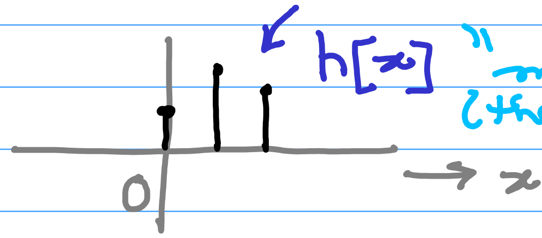
1-D discrete functions





"delayed function"



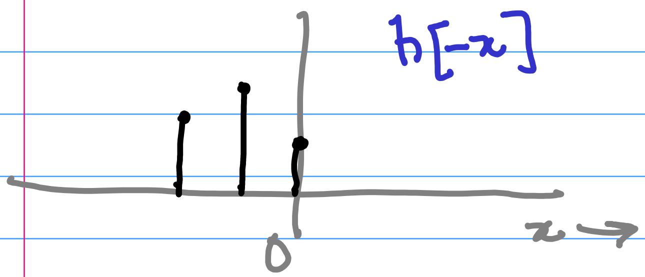


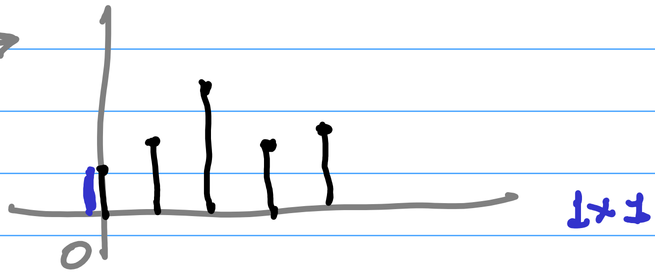
"mask" (the smaller one)

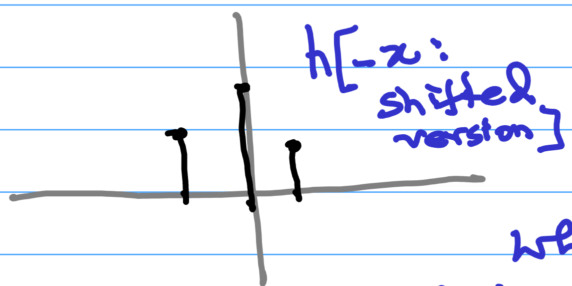
$$f[x] * h[x] = \sum_k f[k]\, \underbrace{h[x-k]}$$

$h[-k]$: flipped "mask"

$h[-k+x]$ → shift it all around

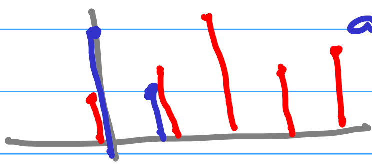






→ shifting all around & looking for overlaps.

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



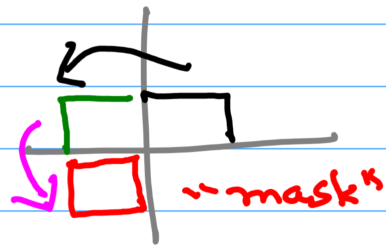
How many overlaps

$A + B - 1$ (Brigham's Result)

Efficient manner: using arrays

2-D:

Conv.

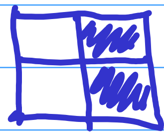


Correlation:

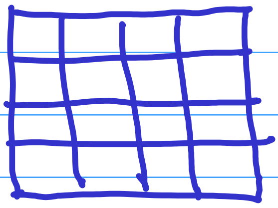
no flipping!

$$\begin{matrix} \vdots \\ -1 \\ 1 \\ \vdots \end{matrix}$$ → other weights ≈ 0 (negligible)

2-D:

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$ captures a "vertical" edge 

$$\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$ captures a "horizontal" edge

larger neighbourhood  capture edges in different orientations.

Interpretation: (deep network)

o ⋮ o [oriented local edge detection] o ⋮ o ... [mid-level features] (semantic) ... o ⋮ o o/p → layer

input layer

↓ first layer of connections

[high level features]

e.g., image of a cat / dog

↓ also corresponds to the action in V1/V2 of the visual cortex ⇒ [1) simple derivative features 2) only 'local', not global

$x[n]$ ⟶ system ⟶ output
input

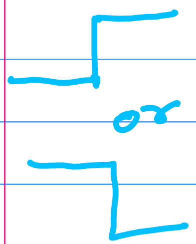
with a system response $h[n]$

[convolution]
(We will justify this soon!)

input waveform/ electrical signal ⟶ Electrical circuit ⟶ output

road profile ⟶ vehicle suspension ⟶ output

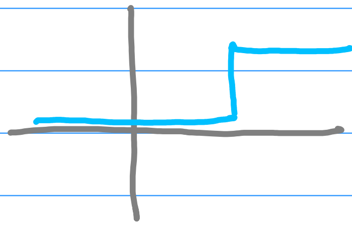
(pothole)

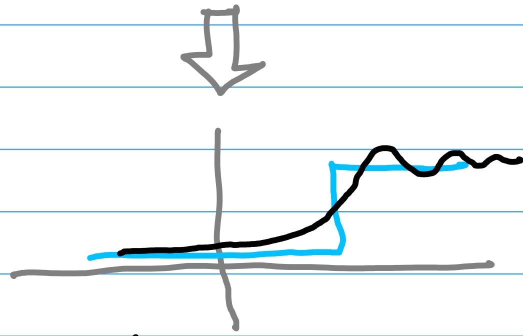


or

(unit step function/ extreme case of a sigmoid)

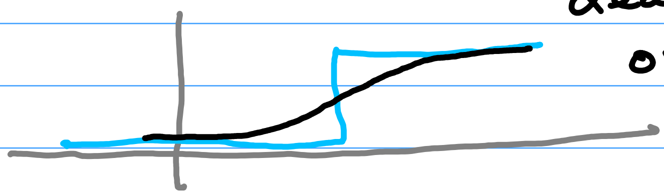
smoothing agent/ smoothing filter/ Low-pass filter





'damping'

good suspension



dead-beat/ overdamped system

(On the other hand, for a chronometer: a time-keeping device, we want an underdamped system)

LSI SYSTEMS (LINEAR SHIFT INVARIANT) SYSTEMS

Approximation

↳ the nature of the output is the same, just shifted in time/space

Electrical: diode/BJT linear region

Mechanical: Mass-spring system

$F = -kx$

e.g., weighing scales with a tray to keep objects

Why are people obsessed with LSI systems?

* Many practical systems can be approximated by LSI systems

(EE specific:
- LSI permit an equivalent freq domain analysis
- complex exponentials / sinusoids are eigenfunctions of LSI systems)

What is linearity?

[— Additivity
  — Homogeneity

$x_1[n] \longrightarrow y_1[n]$

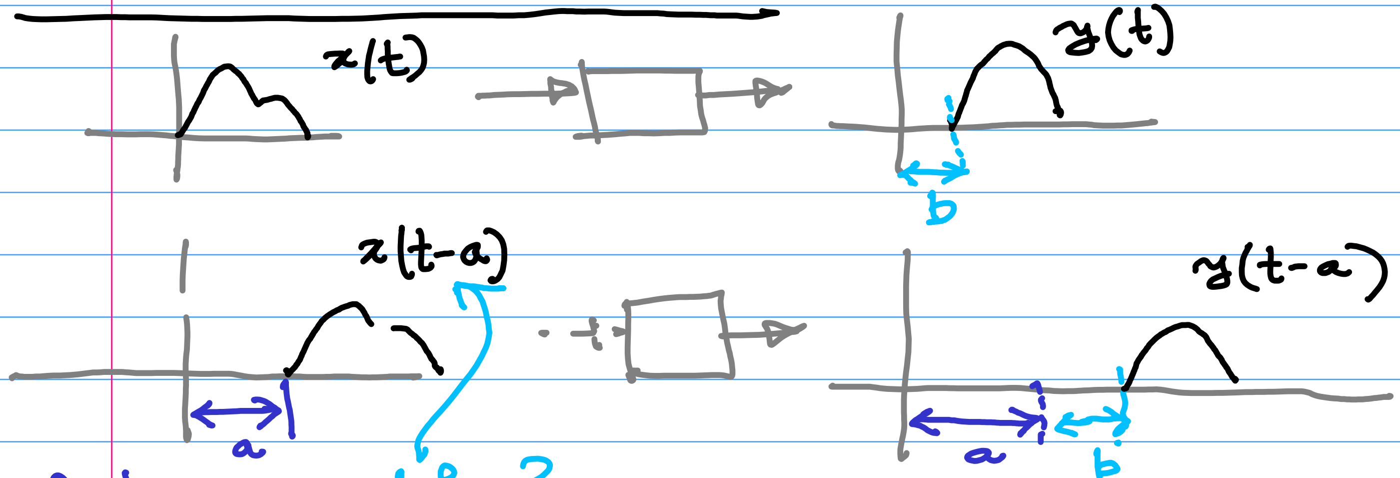
$x_2[n] \longrightarrow y_2[n]$

Additivity $\Rightarrow$ $x_1[n] + x_2[n] \longrightarrow y_1[n] + y_2[n]$

Homogeneity $\Rightarrow$ $\alpha x_1[n] \longrightarrow \alpha y_1[n]$

"Superposition Principle"

What is Shift Invariance?



delay = a

why?

it starts at $t-a=0$ i.e., $t=a$ instead of starting at $t=0$

How do we 'characterise' a system?

can be constructed out of different classes of

e.g., electronic components
- vacuum tubes
- transistors

e.g., mechanical components (time measurement)
- pendulum
- flywheel

we need a "standardisation of the equivalence"

"standard" input → $\boxed{\text{system}}$ → output

e.g., monitor (CRT/LCD/LED/Plasma)
input (switching on)
characterisation: how quickly does it respond

This output is a waveform which "characterises" the system irrespective of the exact internal hardware construction (as a black box)

→ This allows us to treat a system (electrical/mechanical) as a <u>waveform</u> itself

→ The name of the EE course "Signals and Systems"

$$\delta[n] \longrightarrow \boxed{LSI} \longrightarrow h[n]$$

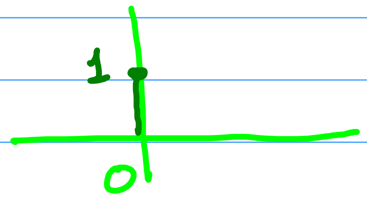
Impulse (Kronecker Delta / 'bar') This is a standard input

$\underbrace{\text{'Impulse Response'}}$ is the characterisation of the system

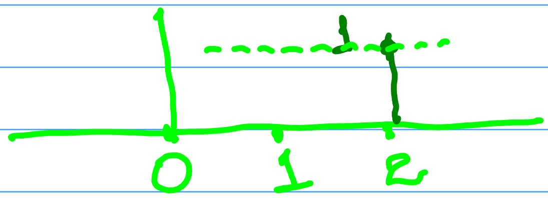
Civil Engg example: Bridge: (Suspension bridge / ... /)

input: perturbation

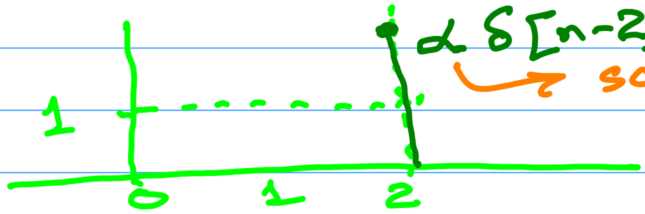
output: vibration response

① 

$\delta[n] \triangleq 1$, at $n=0$ (origin)
0, otherwise

② 

$\delta[n-2] = 1$, at $(n-2)=0$
$\Rightarrow n=2$
$= 0$, otherwise

③ 

$\alpha\,\delta[n-2]$
α → scaling constant, we can call it $x[2]$

$\boxed{\text{Step I}}$ $\quad \delta[n] \longrightarrow \boxed{LSI} \longrightarrow h[n]$

"Impulse Response"

$\boxed{\text{Step II}}$ $\quad \delta[n-k] \longrightarrow \boxed{LSI} \longrightarrow h[n-k]$

delayed the impulse by 'k' units

(Shift Invariance)

i.e., impulse is not at n=0, but at k

$\boxed{\text{Step III}}$ $\quad x[k]\,\delta[n-k] \longrightarrow \boxed{LSI} \longrightarrow x[k]\,h[n-k]$

The impulse at k is no longer a unit impulse, but has a magnitude x[k]

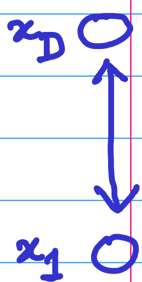
(Homogeneity)

$\boxed{\text{Step IV}}$ $\quad \underbrace{\sum_k x[k]\,\delta[n-k]}$ $\longrightarrow \boxed{LSI} \longrightarrow \sum_k x[k]\,h[n-k]$

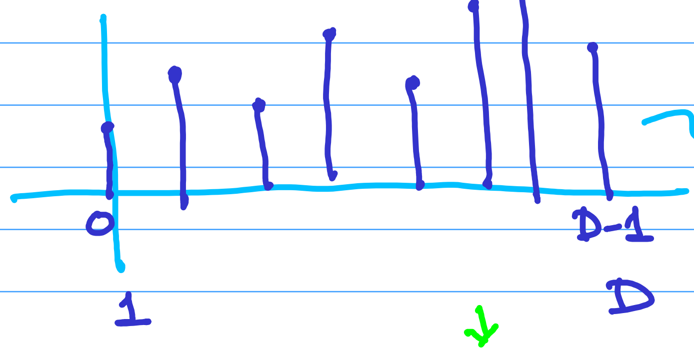
(Additivity)

This is the input waveform itself! A collection of scaled unit impulses which are appropriately delayed.

this is the definition of Convolution itself!



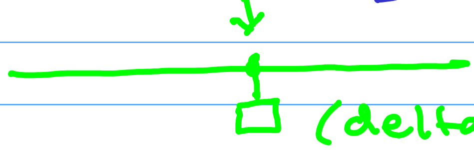
$$\begin{bmatrix} x_D \\ \vdots \\ x_1 \end{bmatrix}$$



scaled & delayed Kronecker Delta.

Kronecker Delta:

Dirac/Delta



beam

(delta function)