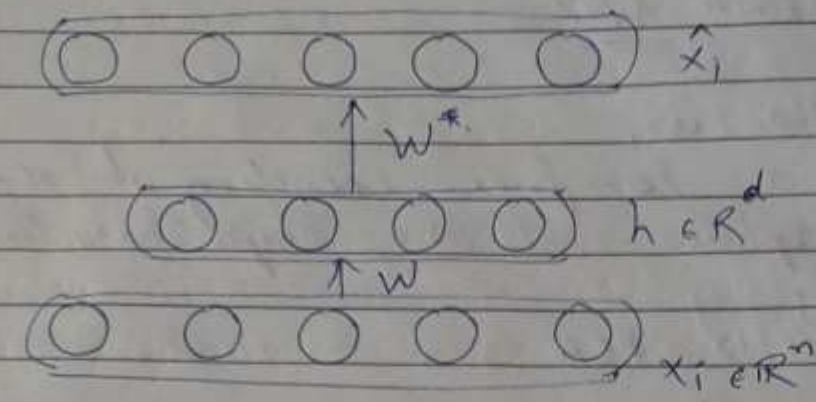
Auto Encoder



→ It's like a feed-forward NN

① → It Encodes its input x_i into a hidden representation h

$$h = g(\underbrace{Wx_i}_{\text{Linear transformation}} + b)$$

$h \in \mathbb{R}^{d \times n}$, x_i: $n \times 1$, b: $d \times 1$

can be any kind of sigmoid fn (Linear or non-linear transformation ie)

② Decodes the i/p again from this hidden representation & reconstructs the i/p

$$\hat{x}_i = f(w^* h + c)$$

⇒ x_i should be similar to $\hat{x}_i$ → very similar to PCA

Case I

⇒ Let us consider the case when $\dim(h) < \dim(x_i)$

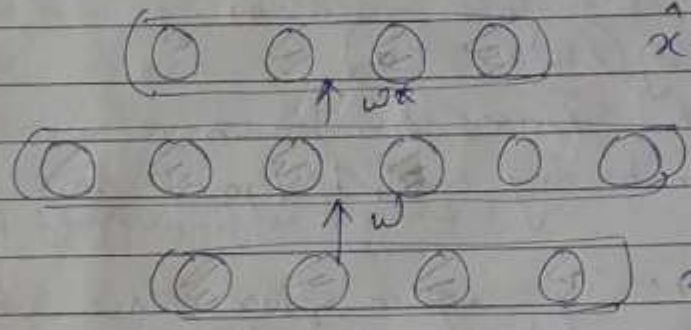
⇒ If we are still able to reconstruct

$\hat{x}_i$ perfectly from h. Then what does it say about h?

→ It tells us,
"h is a loss-free encoding of x_i.
It captures all the imp. characteristics of x_i.
- Similar to PCA.

→ An encoder where, [dim(h) < dim(x_i)] is called an [undercomplete autoencoder]

II Case :- Consider, dim(h) ≥ dim(x_i)



- In such a case, the autoencoder could learn a trivial encoding by simply copying x_i into h & then copying h into $\hat{x}_i$.

Eg
Compression → we have 10 bits → 4 bits → (encoding info. process)

But 10 bits → 16 bits → 10 bits (often 6 bits blank)

PCA linearly transforms data into new coordinate sys where most of the variation in the data can be described with fewer dimensions than the original data

PAGE NO:
DATE: / /

⇒ Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

⇒ Such as autoencoder is where, $dim(h) \geq dim(x_i)$ is called an over-complete autoencoder (hidden layer more neurons than ip)

⇒ Eg: whether a person is likely to get a disease or not
- we want parameters like, height, wt & BMI
⇒ we only computed BMI.
⇒ We conclude at the end, that the person has high likelihood of getting diabetes depending on BMI
⇒ We want to know, whether it was the height or whether it was the wt which was responsible for this.
⇒ So original ip, features are entangled & we want to disentangle it.
(⇒ i.e. we want to go from smaller feature space to larger feature space)

⇒ so here, there is a need to prevent the identity fn (representation)

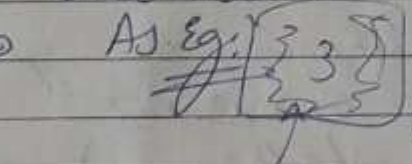
Regularization in Autoencoders

→ Why we need Regularization?
To avoid overfitting or
to Enable Generalization

→ In case of [over Complete] autoencoder Overfitting is likely.
→ Not generalized to unseen data

→ Overfitting - happens for more no of parameters
∴ overcomplete autoencoders have more no of parameters
So we need Regularization.

→ It can happen in undercomplete autoencoder also.

→ As Eg: 
100x100 = 10K

Lot of data is not imp, ~~Hence~~
After shrinking we are still having large no of parameters as, even after shrinking also there is so much redundancy in the i/p that shrinking still leads to a large no of parameters ∴ we could still overfit. So we can need regularizer

→ In Case of Over-complete Autoencoder

Solution:

The simplest solution is to add a "L2-Regularization" term to the objective fn.

$$\min_{\theta, W, W^*, b, c} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

$L(\theta)$ $\Omega(\theta)$

$\theta \rightarrow$ all the parameters i.e. weights & put it in large vector.

$\theta = (w_1 w_2 w_3 \ldots)$

$\frac{\partial L(\theta)}{\partial w_1}$

- This is easy to implement & just add a term λW to the gradient $\frac{\partial L(\theta)}{\partial W}$

(& similarly for other parameters)

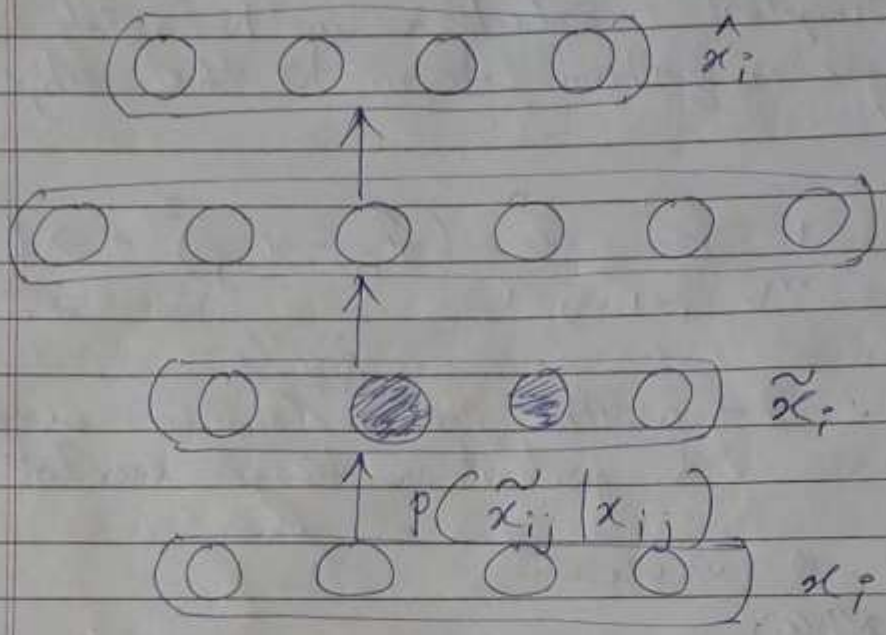
(2) Another trick is to tie the weights of the encoder & decoder ie. $W^* = W^T$

⇒ Tying the wts means it reduces the parameter space of auto encoder. So it reduces the complexity of the model which reduces overfitting

→ parameter sharing: we force certain weights to share the same value

- we divide the parameters or weights of a model into groups by leveraging prior knowledge, & all parameters in each group are constrained to take the same value

Denoising Autoencoders (DAE)



→ A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij} | x_{ij})$) before feeding it to the network.

→ A simple $P(\tilde{x}_{ij} | x_{ij})$ used in practice is the following:-

$$P(\tilde{x}_{ij} = 0 | x_{ij}) = q$$

ie.

with probability q, it will set (corrupting) $x_{ij} = 0$ & with probability $(1-q)$ it is kept as it is (retaining data)

(Assume i/ps are binary 0 & 1)

⇒ Here we have corrupted data as i/p & getting o/p as $\hat{x}_i$ similar to x_i.

[→ ie the objective fn. is still to reconstruct the original (uncorrupted) x_i

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{d} (\hat{x}_{ij} - x_{ij})^2$$

Eg:

→ we corrupt, still it should predict it as '3'.

So it should be more robust

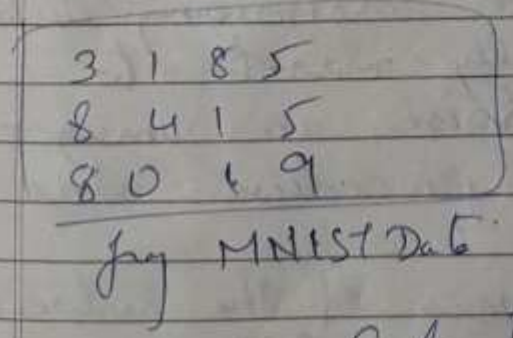
So it should not just memorize the data so to learn better we are feeding corrupted data.

for eg:

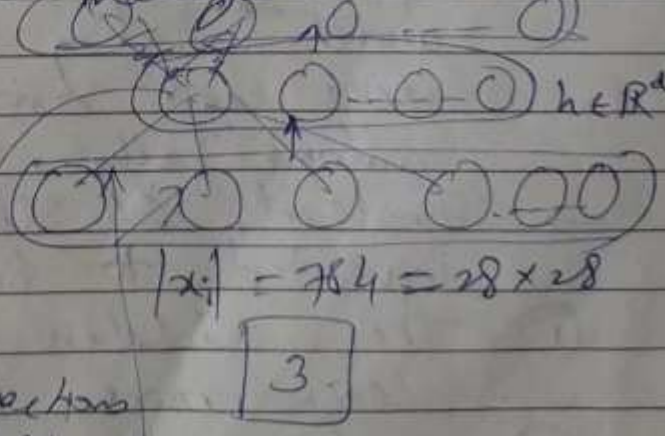
it will have to learn to reconstruct a corrupted x_{ij} correctly by relying on its interactions with other elements of x_i.

Eg

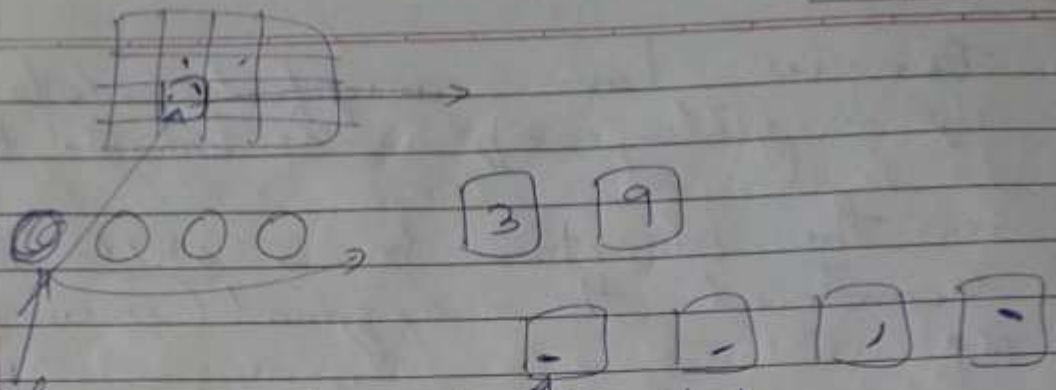
Hand-written digit Recognition



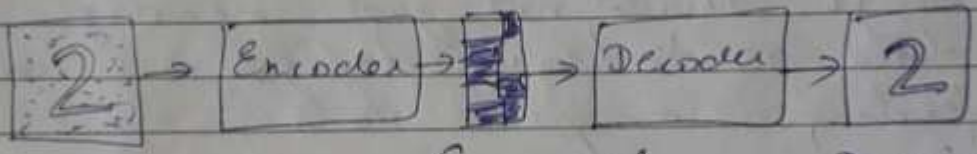
for MNIST Data



Captures interactions betn diff neurons



This neuron fires for this stroke, other neurons for other strokes so together all neurons can decide that the combination of these strokes is 3 or 9.



Noisy i/p

Compressed representation

Denoised image

- Denoising autoencoders are an extension of simple autoencoders, however it's worth noting that denoising autoencoders were not originally meant to automatically denoise an image.
- Instead, this procedure was invented to help :-
  1. the hidden layers of the autoencoder learn more robust filters
  2. Reduce the risk of overfitting in the autoencoder.

PAGE NO.
DATE / /

(3) Prevent the autoencoder from learning a simple identity function

→ Noise was stochastically (ie. randomly) added to the input data & then the autoencoder was trained to recover the original, nonperturbed signal

Eg: preprocessing an image to improve the accuracy of an optical character recognition (OCR) algorithm.

(2) Sparse Auto Encoders: Ensures - that it is actually learning latent representations instead of redundant info. in our i/p data

- A hidden neuron with sigmoid activation will have values between 0 & 1.
- We say that neuron is activated when its o/p is close to 1 & not activated when o/p is 0.

* A sparse Encoder tries to ensure the neuron is inactive most of the times (ie. it is close to 0 for most of the i/ps)
ie the Average activation of a neuron is close to 0.

→ If the neuron l is sparse (ie mostly inactive) then $\hat{\rho}_l \to 0$.

- The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

2 diff ways to constrain sparsity penalty:
↳ ① L1 regularization ② KL divergence

PAGE NO.
DATE / /

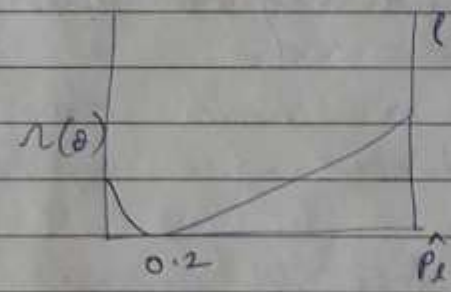
R ⇒ A sparse autoencoder (absolute value) uses a 'sparsity parameter' ρ (rho) (typically very close to 0, say 0.005) & tries to enforce the constraint $\hat{\rho}_i = \rho$.

⇒ To ensure $\hat{\rho}_i = \rho$, we add the term

$$\Omega(\theta) = \sum_{i=1}^{K} \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_i}$$

$$\left(L'(\theta) = L^s(\theta) + \Omega(\theta) \right)$$

* this fn takes its minimum value when $\hat{\rho}_i = \rho$



- for a single k, we take $\rho = 0.2$, we reach to zero only when $\hat{\rho}_i = 0$.

- $L(\theta)$ is the squared error loss or cross entropy loss & $\Omega(\theta)$ is the sparsity constraint

Finally, $\frac{\partial L'(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial w} + \frac{\partial \Omega(\theta)}{\partial w}$

PAGE NO.
DATE / /

(3) Contractive Autoencoders :-

- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function
- It does so, by adding the following regularization term to the loss fn;

$$\Omega(\theta) = \| J_x(h) \|_F^2$$

- where, $J_x(h)$ is the Jacobian of the encoder

⇒ If the i/p has n-dimensions & the hidden layer has K dimensions then, the Jacobian Matrix is,

$$J_x(h) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial h_K}{\partial x_1} & & \frac{\partial h_K}{\partial x_n} \end{bmatrix}$$

* { its a partial derivative of every neuron in the 1st hidden layer w.r.t the 1st i/p. ie. Jacobian is derivative of h w.r.t x

⇒ Each J_i^{th} entry shows :- how much does h_i change with a small change in x_k

Similar to Euclidean norm

PAGE NO.
DATE / /

- Frobinious Norm =

$$\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j}\right)^2$$ Should be 0

Contradictory statements

$\rightarrow n(\theta)$

it is sum of squares of all elements of a matrix

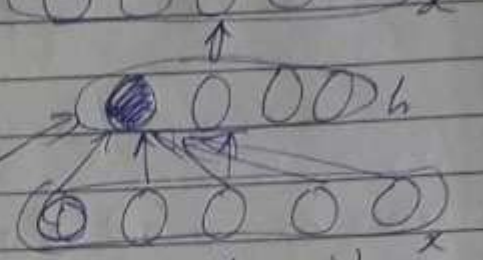
we want to Min^n $L(\theta) + r(\theta)$ (Loss fn)

Frobinious norm = 0

if $r(\theta) = 0$, then $L(\theta)$ will be very high

$\Rightarrow$

Consider $\frac{\partial h_l}{\partial x_j} = 0$



$\rightarrow$ It means that this neuron is not very sensitive to variations in the i/p x_j

$\rightarrow$ But, doesn't this contradict our other goal of minimizing $L(\theta)$ which requires h to capture variations in the i/p

$\rightarrow$ $L(\theta)$ - wants to capture the variations

$\Rightarrow$

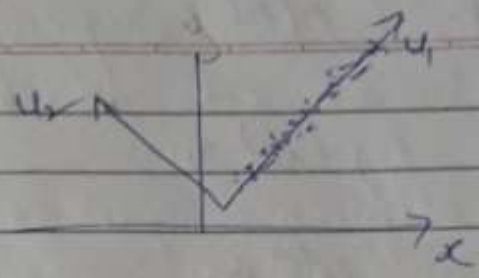
By putting these 2 contradicting objectives against each other we ensure that 'h' is sensitive to only very important variations as observed in the training data

$\rightarrow$ $L(\theta)$ - Capture important variations in data

$\rightarrow$ $r(\theta)$ - do not capture variations in data

$\Rightarrow$ Trade off - Capture only very important variations in the data

Eg:



- u_1 & u_2 are the dimensions.
- u_1 is important - as it has lots of variance
- u_2 → variance is out of line which doesn't seem important

→ Its similar to PCA -> try to capture the variations across the important dimensions but not across non-important axes.