



## \* Principle Component Analysis

lets say,

n attributess inputs			0
x	y	z	
1	1	1	$\vec{z}_1$
0.5	0	0	$\vec{x}_1$
0.25	1	1	$\vec{y}_1$
0.35	2.5	1.5	$\vec{z}_1$
:	:	:	
:	:	:	
:	:	:	
:	:	:	
1	1	1	$\vec{z}_m$

Here x & y are highly correlated thus if any one is ignored it wouldn't make much difference in result.

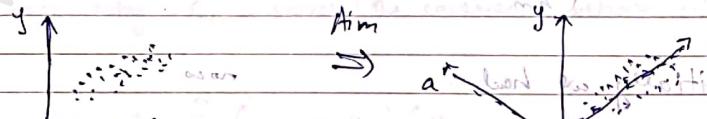
↳ 3D components are:

Here if in following operation where  $y_i = \bar{y} + \hat{y}$  &  $\bar{y} = \bar{y}_i$  then after updating all  $y_i$  if we calculate  $\bar{y}_i$  then if it is  $\bar{y}_i = 0$  then it's called 0 mean.

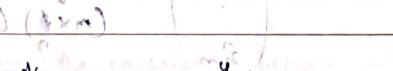
$$P_{yz} = \sum_{i=1}^n (y_i - \bar{y})(z_i - \bar{z})$$

$$(x,y,z) \text{ covariance} = \sum_{i=1}^n (y_i - \bar{y})^2 \times \sum_{i=1}^n (z_i - \bar{z})^2$$

Notes



- In this first figure the data points have notable variance across both axis



- Aim is to find basis in the dimension such that dimension can be reduced

→ Here in 2nd figure a & b are independent vectors that can form entire plane. But unlike x, y variance of points is significantly affected by b and it can be safely ignored (approx. 90% of variance). Thus we reduce dimensions from newly formed vectors.

→ Initially make data 0-mean & unit variance by Z-Score normalization

Now, representing  $\pi$ , using basis  $P$

all statements should be made clearly

$$f_{\text{total}}(x) = \alpha_1 p_1 + \alpha_2 p_2 + \dots + \alpha_n p_n$$

Unter der geschafften Lärmen stand

for orthonormal basis

$$\text{radioactive fraction of } \text{Li-7} = n_{\text{Li}}^{\text{radioactive}}$$

~~18. De grotten zijn niet~~

~~west~~ ~~in~~ ~~gladly~~ ~~as~~ ~~if~~

$\alpha \cdot \mathbf{r}_i^T \cdot \mathbf{P}$  } Dimension =  $(1 \times n)$

## new B leading

$$[1, \frac{n}{n}] [1, n]$$

Note for m vectors

$$\left( \begin{array}{c} \vec{n}_1 \\ \vec{n}_2 \\ \vdots \\ \vec{n}_m \end{array} \right) \quad \left( \begin{array}{c} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vec{s}_n \end{array} \right)$$

initially, we had

no co

Previous

$$\Rightarrow \begin{cases} \leftarrow x_{11}, x_{12}, \dots, x_{1n} \rightarrow \\ \leftarrow x_{21}, x_{22}, \dots, x_{2n} \rightarrow \\ \vdots \\ \leftarrow x_{m1}, x_{m2}, \dots, x_{mn} \rightarrow \end{cases}$$

test whether heterogeneity is due to sampling bias or not can

\* Now here we've got new dimension. Here we want to prove that these new dimensions have low covariance i.e. they are not interrelated.

Theorem

(1)

If  $X$  is a matrix such that its columns have zero mean and if  $\mathbf{q}^T \mathbf{q} = \mathbf{q}^T \mathbf{X} \mathbf{P}$  then the columns of  $\hat{\mathbf{X}}$  will also have zero mean.

proof →

$$\mathbf{q}^T (\mathbf{X}^T \hat{\mathbf{X}}) \perp = \mathbf{q}^T (\mathbf{q} \mathbf{X}) \perp = \mathbf{q}^T \mathbf{q} \perp = 0$$

$$\text{now } \hat{\mathbf{X}} = \mathbf{X} \mathbf{P} \quad \mathbf{q}^T \mathbf{X} \mathbf{X}^T \hat{\mathbf{X}} + \mathbf{I}^T \hat{\mathbf{X}} \perp = \mathbf{I}^T \mathbf{X} \mathbf{P}$$

$$= \mathbf{0} \mathbf{P}$$

(2)

Theorem

$\mathbf{q}^T \mathbf{X} \mathbf{X}^T \mathbf{q}$  is symmetric in  $\mathbf{q}$  if  $\mathbf{X}^T \mathbf{X}$  is symmetric.

proof →  $(\mathbf{q}^T \mathbf{X} \mathbf{X}^T \mathbf{q})^T = (\mathbf{q}^T (\mathbf{X} \mathbf{X}^T))^T = \mathbf{q}^T \mathbf{X}^T \mathbf{X} \mathbf{q}$  since  $\mathbf{X}^T \mathbf{X}$  is symmetric.

→ If  $X$  is a matrix whose columns are zero mean then  $\frac{1}{m} \mathbf{X}^T \mathbf{X}$  is the covariance matrix. This means that each entry  $\Sigma_{ij}$  stores the covariance between column  $i$  &  $j$  of  $X$ .

Explanation

$$C_{ij} = \frac{1}{m} \sum_{k=1}^m (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j) \quad \begin{array}{l} \text{covariance} \\ \text{ML Book - (1)} \end{array}$$

$$= \frac{1}{m} \sum_{k=1}^m x_{ki} x_{kj} \quad (\because \bar{x}_i = \bar{x}_j = 0)$$

$$= \frac{1}{m} \mathbf{x}_i^T \mathbf{x}_j \quad \text{resp. entry of } \mathbf{X}^T \mathbf{X}$$

Similarly matrix is formed

$$\frac{1}{m} (\mathbf{X}^T \mathbf{X})_{ij}$$

$\frac{1}{m} \hat{X}^T \hat{X}$  → Covariance matrix of transformed data

$\Sigma = \frac{1}{m} X^T X$  → Covariance matrix of original data

DELUXE  
PAGE NO.:  
DATE:

CloudTech

Now,  $\hat{X}' = X P$

and as seen earlier  $\frac{1}{m} \hat{X}'^T \hat{X}'$  is Covariance Matrix has

$$\therefore \frac{1}{m} \hat{X}'^T \hat{X}' = \frac{1}{m} (X P)^T X P = \frac{1}{m} (P^T)^T X P$$

mean bias and variance be merged  $= P^T \Sigma P$

$$Q_X = P^T \left( \frac{1}{m} X^T X \right) P \quad Q_X = \hat{\Sigma}$$

$$\frac{1}{m} \hat{X}'^T \hat{X}' = P^T \Sigma P$$

Here, in  $\frac{1}{m} \hat{X}'^T \hat{X}'$  is covariance matrix where we

want all  $i, j$  where  $i \neq j$  "0" since if  $i=j$ , it is covariance of  $i$  &  $j$  column if  $i=j$  it is variance

Left over  $\begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$  Diagonal matrix

$$\begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$$

Now, as seen earlier

$\frac{1}{m} \hat{X}'^T \hat{X}'$  is symmetric  $\rightarrow$  Theorem (2)

$$\text{Thus } P^T = P^{-1}$$

( $P$  is collection of eigen vectors of  $\frac{1}{m} \hat{X}'^T \hat{X}'$ )

and  $P^T \Sigma P = \text{Diagonalization of gives Diagonal matrix}$



The method of transforming data to new basis with low covariance

(1) low covariance - proved

(2) high variance - not proved

is called PCA

→

$$n_i = \sum_{j=1}^n \alpha_{ij} p_j$$

Now, if we approximate by taking only  $k$  dimensions out of  $n$  then we can reduce dimensions

$$\hat{n}_i = \sum_{j=1}^k \alpha_{ij} p_j$$

So we select this  $k$  such that

$$\text{Error} = \sum_{i=1}^m (n_i - \hat{n}_i)^T (n_i - \hat{n}_i)$$

$$= \sum_{i=1}^m \left( \sum_{j=1}^n \alpha_{ij} p_j - \sum_{j=1}^k \alpha_{ij} p_j \right)^2$$

all dimensions       $k$  dimensions

$$= \sum_{i=1}^m \left( \sum_{j=k+1}^n \alpha_{ij} p_j \right)^2 = \sum_{i=1}^m \left( \sum_{j=k+1}^n \alpha_{ij} p_j \right)^T \left( \sum_{j=k+1}^n \alpha_{ij} p_j \right)$$

$$= \sum_{i=1}^m \sum_{j=k+1}^n \alpha_{ij} p_j^T p_j \alpha_{ij}$$

vectors  
entries

DELUXE

Dimension of probability distribution is  $m \times n$

$$F = \sum_{i=1}^m \sum_{j=k+1}^n \alpha_{ij}^2 \quad (\because p_j^T p_j = 1, p_i^T p_j = 0 \forall i \neq j)$$

Leverage row  $i$  contribute less  
less value  $\alpha_{ij}$

$$= \sum_{i=1}^m \sum_{j=k+1}^n (x_i^T p_j)^2 \quad \text{A.S. better if } \alpha_{ij}$$

$\left( p_j^T x_i \right) \left( x_i^T p_j \right)$

Now we want  $\sum_{j=k+1}^n w_{jk} p_j^T \left( \sum_{i=1}^m \alpha_{ij} x_i \right) p_j$  which is equal to  $w_{jk} \alpha_{jk}$

Now

lets say

$$\sum_{i=1}^m \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix} \begin{bmatrix} p_{j1} & p_{j2} & p_{j3} \end{bmatrix} = \sum_{i=1}^m \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \alpha_{i3} \end{bmatrix} \begin{bmatrix} x_{i1} & x_{i2} & x_{i3} \end{bmatrix}$$

$$= \begin{bmatrix} (x_{11}^2 + x_{12}^2 + x_{13}^2) & (x_{11} x_{12} + x_{12} x_{13}) & (x_{11} x_{13} + x_{12} x_{13}) \\ (x_{12} x_{11} + x_{13} x_{11}) & (x_{12}^2 + x_{13}^2) & (x_{12} x_{13} + x_{13} x_{12}) \\ (x_{13} x_{11} + x_{13} x_{12}) & (x_{13} x_{12} + x_{12} x_{13}) & (x_{13}^2 + x_{12}^2) \end{bmatrix}$$

$$(0.1^2 3) (0.1^2 3) = (0.1^2 3) 3$$

$$(0.1^2 3) \cdot \sum_{i=1}^m \left[ x_{i1}^2 + x_{i2}^2 + \dots + x_{im}^2 \right] = 0.1^2 \cdot 3 \cdot (x_{11}^2 + x_{12}^2 + \dots + x_{1m}^2 + x_{21}^2 + x_{22}^2 + \dots + x_{2m}^2 + \dots + x_{m1}^2 + x_{m2}^2 + x_{m3}^2)$$

The Matrix  $\sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T$  is the covariance matrix

$$\begin{array}{c} \text{Let } \mathbf{x} = \begin{pmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} \\ \mathbf{x}_{21} & \mathbf{x}_{22} \\ \mathbf{x}_{31} & \mathbf{x}_{32} \end{pmatrix} \\ \text{column } i \quad \text{column } j \\ + \end{array}$$
$$\begin{array}{c} \mathbf{x}_{11}^2 = \mathbf{x}_{11} \mathbf{x}_{11}^T \text{ sp. } \mathbf{P}^2 \\ + \mathbf{x}_{21} \mathbf{x}_{21}^T \\ + \mathbf{x}_{31} \mathbf{x}_{31}^T \\ \vdots \\ \text{column } i \quad \text{column } j \\ \text{center map. if } i = j \\ \text{center map. if } i \neq j \end{array}$$

Now

the  $\sum_{i=1}^n p_i^T m c p_i$  is Norm.  $\left[ \text{Let } \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T \in \frac{\mathbf{x}^T \mathbf{x}}{m} = C \right]$   
 $p_i^T m c p_i$  is called eigen value of  $m c$ .  
so  $\sum_{i=1}^n p_i^T m c p_i$  is sum of eigen values.

$$C = \sum_{j=1}^k p_j^T m c p_j$$

(prob. of coffee |  $j=1, 2, \dots, k$ ) signified  $A^{-1} \mathbf{y}$

$$\min_{j=1, 2, \dots, k} (\mathbf{y}^T \mathbf{v}) \text{ s.t. } p_j^T m c p_j \text{ such that } p_j^T p_j = 1$$

P1, P2, ..., Pn

statement says ref along steers direction  $\rightarrow$  center map.  $\rightarrow$  theorem  
Deep learning book (7)

$p_j$  are smallest eigen vectors

BV2 has few result polynomials and integers part 68

→ How to prove high variance - ② condition as mentioned earlier

mean of eigenvalues is mean of column  $\rightarrow$  center map.  $\rightarrow$  variance

$$\mathbf{x}_i = \mathbf{x} p_i$$

variance

$$\frac{(\hat{\mathbf{x}}_i - \mu_i)^T (\hat{\mathbf{x}}_i - \mu_i)}{m} = \frac{\hat{\mathbf{x}}_i^T (\hat{\mathbf{x}}_i - \mu_i)}{m} \quad (\because \mu_i = 0) \quad \text{zero mean}$$

$$\begin{aligned} \text{from } \frac{1}{m} \sum_i p_i^T (X^T X) p_i &\rightarrow A_n = \lambda n \\ &= \frac{1}{m} \sum_i p_i^T (\lambda_i p_i) \\ &= \frac{\lambda_i}{m} \underbrace{p_i^T p_i}_{\text{normed vector}} \\ &= \frac{\lambda_i}{m} \quad \} \text{ eigen values} \end{aligned}$$

If we remove the small eigen values then we are left with large eigen values & large eigen value means high variance.

→ PCA Example (refer notes/week 6/ pca example.png)

→ Singular Value Decomposition (SVD)

Eigen values & vectors work only for square symmetric matrix

But when matrix are rectangular then we use SVD

Let's say  $\underline{A}$  is a  $n \times m$  matrix

matrix  $A$  transforms a vector from  $n$  dimension to  $m$  dimension

$$\text{i.e. } A v' = v''$$

$$\begin{array}{ccc} \downarrow & & \downarrow \\ (n \times n) & \text{mxn} & n \times 1 \\ \text{mxn} & \text{nx1} & \text{mx1} \end{array}$$

$$(n \times n)(n \times 1) = n \times 1$$

Now (let us assume) that  $\{v_i\}_{i=1}^k$  are such that

( $n \times 1$ ) vectors whose sum is zero, then we find

$\{v_i\}$  is basis of vector space  $\mathbb{R}^n$  and also

( $m \times 1$ ) basis of (vector) space  $\mathbb{R}^m$  and now

then,

$$Av = \sum_{i=1}^k \alpha_i v_i = \text{non-zero}$$

$$Av = \sigma_i u_i \quad \text{can be possible}$$

$$[v = v] \quad \Rightarrow \quad v = v^T v$$

$$\text{Now } v \text{ is a vector such that } v = \sum_{i=1}^k \alpha_i v_i$$

$$[v = v] \quad v^T v = A^T A$$

$\therefore$

$$Av = \sum_{i=1}^k \alpha_i Av_i \quad v \in \text{range}(A) \Rightarrow (\text{columns})$$

$$= \sum_{i=1}^k \alpha_i \sigma_i u_i \quad \Rightarrow \quad u \in (\text{columns})$$

$$(v^T v)^{-1} (v^T v) = A^T A \quad \therefore$$

$$v^T v = A^T A$$

Here  $k$  is used because there are  $k$  linearly independent vectors in rowspace & columnspace of  $A$  for these  $k$  vectors as input  $A$  is non-zero

$$A^T A = \text{columns}$$

Since  $v_i \perp u_i$  common term  $k$  is used in summation

$n - \text{dim}$        $m - \text{dim}$

$$\rightarrow \text{Thus we can write} \quad A_{m \times n} \times V_{n \times k} = \begin{bmatrix} U_{m \times k} & X \end{bmatrix} \sum_{i=1}^k \begin{bmatrix} I_{m \times m} \\ 0_{(m-k) \times m} \end{bmatrix} = \begin{bmatrix} I_m \\ 0_{(m-k) \times m} \end{bmatrix} A$$

$\downarrow \quad \downarrow \quad \downarrow$

$n \times k$  vectors       $m \times k$  vectors      diagonal matrix

Now there are  $(n \times k)$  &  $(m \times k)$  vectors to solve  
but we need  $n \times n$  &  $m \times m$  vectors  
so by (Gram-Schmidt orthogonalization) we  
can find  $(n-k)$  &  $(m-k)$  orthogonal vectors respectively.

$$\therefore A_{m \times n} V_{n \times n} = U_{m \times m} \Sigma_{m \times n}$$

Thus  $U^T A V = \Sigma$  [  $U^{-1} = U^T$  ]

$$A = U \Sigma V^T$$
 [  $V^{-1} = V^T$  ]

→ Now we find  $U \leq V$  &  $A = U \Sigma V^T$

let say  $U, V \in \mathbb{R}^{n \times n}$  exist

$$U = (U_i) \in \mathbb{R}^{n \times n}$$

$$\therefore A^T A = (U^T \Sigma V)^T (U^T \Sigma V)$$

$$= V \Sigma^T U^T U \Sigma V$$

Similarly  $A^T A^T = U^T \Sigma^2 U$  {  $U, V$  are eigen vectors of  $A^T A, A^T A^T$  respectively }

$$\begin{aligned} A &= \begin{bmatrix} 1 & 1 & 1 \\ u_1 & u_2 & \dots & u_k \\ \downarrow & \downarrow & \ddots & \downarrow \end{bmatrix}_{m \times n} = \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_k & \\ & & & 0 \end{bmatrix}_{k \times k} \times \begin{bmatrix} v_1 & v_2 & \dots & v_n \\ \downarrow & \downarrow & \ddots & \downarrow \\ u_1 & u_2 & \dots & u_k \end{bmatrix}_{k \times n} \\ &= \sum_{i=1}^k \sigma_i u_i v_i^T \end{aligned}$$

complexity in storage for  $A = mxn$

improved

$$= (m+n+1) k$$

$\downarrow \quad \downarrow \quad \downarrow$

(8x5) stored stuff

## SVD

$\min \|A - WA\|^2$  is given by

$$A = U_{:,k} \Sigma_{k,k} V^T$$

largest /  
first k dominant  
eigen vectors

(first k) columns of  
U & first k  
rows of V

then have at most  $k$  non-zero entries of each matrix

losses in  $\|A - WA\|^2$

columns of  $U$  are  $\{u_1, u_2, \dots, u_m\}$  if  $\left\|u_i\right\|_2 = 1$

columns of  $V$  are  $\{v_1, v_2, \dots, v_n\}$  if  $\left\|v_i\right\|_2 = 1$

good grade of  $\sigma_k$

most significant loss due to  $\sigma_k$

$$\sigma_k + \Delta \sigma_k = \sigma_k$$

and  $\sigma_k$  end same position and value of  $\sigma_k$

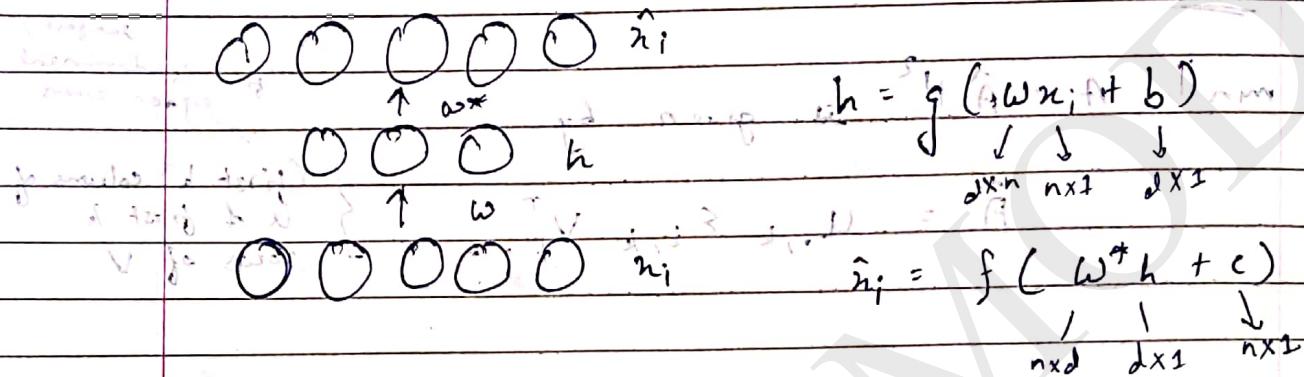
$$(\sigma_k + \Delta \sigma_k)^2 = \sigma_k^2 + \Delta \sigma_k^2$$

values change for  $\sigma_k$  in contrast to  $\sigma_k$  without  $\Delta \sigma_k$

WEEK 7

~~nxm  $\rightarrow$  n ref operations, m isolations~~~~if (m < n) =~~~~Isolations~~

## \* Auto Encoders



Aim here is to reconstruct  $\hat{x}_i$  from  $h$  such that error

$\|x_i - \hat{x}_i\|^2$  is minimized.

→ if  $\dim(h) \geq \dim(x_i) \Rightarrow$  Over complete autoencoders  
 if  $\dim(h) < \dim(x_i) \Rightarrow$  Under complete autoencoders  
 ↓ as in above case

→ If the  $x_i$  nodes have real numbers then

$$\hat{x}_i = w^T h + c$$

If  $x_i$  nodes have binary numbers 0, 1 then

$$\hat{x}_i = \text{sigmoid}(w^T h + c)$$

f in different cases

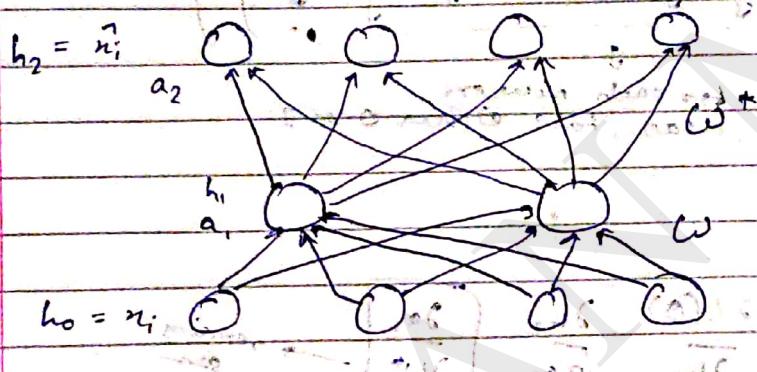
→ lets say loss function is sum of squared error

$$\text{ie } \min_{w, w^*, c, b} \sum_{m=1}^M \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

$$\Rightarrow \min_{w, w^*, c, b} \sum_{m=1}^M \sum_{i=1}^n (\hat{x}_i - x_i)^2$$

→ Backprop (for Real  $x_i$ )

$$L(\theta) = (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)$$



Now earlier we did backprop with cross entropy loss

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial L(\theta)}{\partial h_2} \left[ \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial w^*} \right] \xrightarrow{\text{remains same}} \downarrow \text{changes}$$

$$\frac{\partial L(\theta)}{\partial w} = \frac{\partial L(\theta)}{\partial h_2} \left[ \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w} \right] \xrightarrow{\text{remains same}}$$

Now

$$\frac{\partial L(\theta)}{\partial h_2} = \frac{\partial L(\theta)}{\partial \hat{x}_i} = \sum_j (\hat{x}_{ij} - x_{ij})^2 \xrightarrow{\text{scaled}}$$

$$= 2(\hat{x}_i - x_i) \xrightarrow{\text{refer machine learning book ①}}$$

→ Back Propagation (for binary  $n_i$ )

$$\begin{matrix} 0 & 0 & 0 & 0 \\ \text{Classified} & 0 & (0 - n_i) & 1 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{matrix}$$

} we will use binary cross entropy loss

no. of entries : no. of neurons/inputs

$$\min \left( \sum_{i=1}^m \sum_{j=1}^4 \left[ n_{ij} \log \hat{n}_{ij} + (1-n_{ij}) \log (1-\hat{n}_{ij}) \right] \right)$$

for each neuron  
can take either 0 or 1

Now

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \left[ \frac{\partial a_2}{\partial w^*} \right]$$

and you can see it's like operation like we did in back propagation

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \left[ \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w^*} \right]$$

Now, loss for 1 data entry

$$\frac{\partial L(\theta)}{\partial w^*} = \sum_{j=1}^6 \left[ n_{ij} \log \hat{n}_{ij} + (1-n_{ij}) \log (1-\hat{n}_{ij}) \right]$$

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial L(\theta)}{\partial h_2} = -n_{ij} + 1 - n_{ij}$$

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial h_2}{\partial a_2} = \sigma(a_2)(1-\sigma(a_2))$$

① good

## \* Link between PCA and Autoencoders with

→ Conditions of noise in  $X$  for autoencoder to be

- (1) Linear decoder  $\rightarrow f$  is a linear function (given)
  - (2) Linear encoder  $\rightarrow g$  is a linear function (prove) in order to minimize loss
  - (3) Loss function  $\rightarrow$  sum of squared errors (given) linear encoder is required
  - (4) Generalization  $\rightarrow$   $x_{ij} = \frac{1}{\sqrt{m}} (x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj})$
- Normalization zero mean

$$\rightarrow x_{ij} = \frac{1}{\sqrt{m}} (x_{ij} - \text{mean}) \text{ zero mean}$$

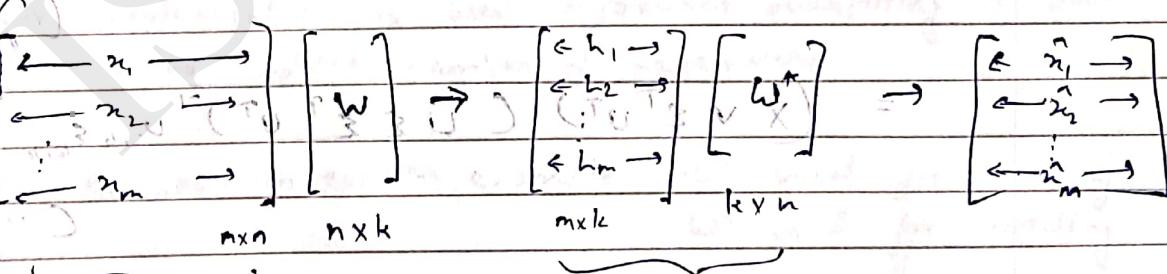
$$\therefore X = \frac{1}{\sqrt{m}} X'$$

$$X^T X = \frac{1}{m} (X')^T (X')$$

now it just needs to prove that  $X$  is precessed result of  $X'$

→ Now loss function is

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



$$\min_{w^*} \|X - H W^*\|_F^2 = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - \hat{x}_{ij})^2$$

Now from SVD theorem it is suggested that

Best approximation of  $X$  is given by  $HW^* \approx \hat{X}$

$$\hat{X} = U \Sigma V^T$$

$$H = U_{:,k} \Sigma_{k,k} V_{:,k}^T$$

Now we can say,

$$H = U_{:,k} \Sigma_{k,k}$$

$$W^* = V_{:,k}$$

Now given all statement we try to prove  
 $H$  is linear encoding of  $X$ .

$$H = U_{:,k} \Sigma_{k,k} = (x x^T) (x x^T)^{-1} U_{:,k} \Sigma_{k,k}$$

$$= (x v \Sigma^T u^T) (u \Sigma^T v^T u^T)^{-1} U_{:,k} \Sigma_{k,k}$$

$$( \because x = u \Sigma v^T )$$

$$= (x v \Sigma^T u^T) (v \Sigma^T v^T u^T)^{-1} U_{:,k} \Sigma_{k,k}$$

$$( \because v^T v = I )$$

$$= x v \Sigma^T u^T (v \Sigma^T)^{-1} u^T U_{:,k} \Sigma_{k,k} \quad [ \because (ABC)^{-1} = C^{-1} B^{-1} A^{-1} ]$$

$$= x v \Sigma^T (v \Sigma^T)^{-1} u^T U_{:,k} \Sigma_{k,k}$$

$$= x v \Sigma^T \Sigma^{-1} u^T U_{:,k} \Sigma_{k,k} \quad [ \because (AB)^{-1} = B^{-1} A^{-1} ]$$

(considering  $\mathbf{X} \mathbf{U} \mathbf{V}^{-1} \mathbf{T}_{\mathbf{I}, \mathbf{k}} \mathbf{T}_{\mathbf{I}, \mathbf{k}}^T \mathbf{U}^T \mathbf{V}$ , each column of  $\mathbf{U} \mathbf{V}^T \mathbf{U}$ ,  $\mathbf{U}_{\cdot, \mathbf{k}}$  =  $\mathbf{I}_{\mathbf{k}, \mathbf{k}}$ )

$$= \mathbf{X} \mathbf{V} \mathbf{I}_{\cdot, \mathbf{k}} \quad (\mathbf{V}^T \mathbf{I}_{\cdot, \mathbf{k}} = \mathbf{I}_{\mathbf{k}, \mathbf{k}})$$

$$= \mathbf{X} \mathbf{U}_{\cdot, \mathbf{k}}$$

linear combination of  $\mathbf{X}$  &  $\mathbf{U}_{\cdot, \mathbf{k}}$

↓  
is weight matrix  $\mathbf{w}$

## \* Regularization in Autoencoders

Coverfitting can happen in both undercomplete &

overcomplete Autoencoders

Two types are i) L1 regularization ii) L2 regularization

$$(1) \text{ Loss Function} + \frac{\lambda \|\mathbf{w}\|^2}{2}$$

} updating the weights using L2 norm  
flatten all  $\mathbf{w}$  matrix & put them in a vector  $\mathbf{w}$

$$\text{Ans: } \frac{\partial L}{\partial \mathbf{w}} = \text{nderivative of loss function} + \lambda \mathbf{w}$$

## (2) Weights tying

since regularization is used to prevent overfitting of data  
we need to reduce number of parameters

∴ As seen in earlier example,  $\mathbf{w}$  is used for encoding  
and  $\mathbf{w}^*$  is used for decoding

in this method  $\mathbf{w}^* = \mathbf{w}^T$ . Learning is as follows

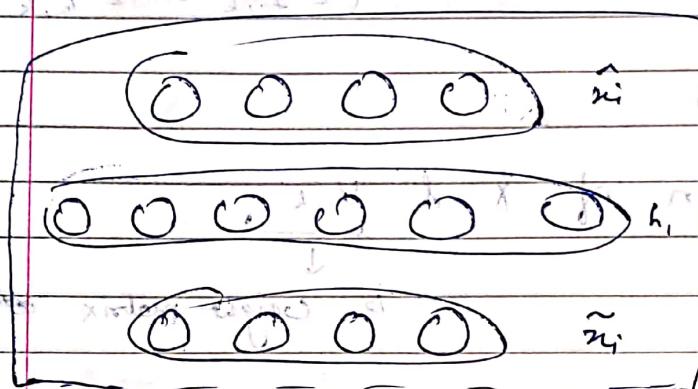
At each instance only  $\mathbf{w}$  is updated.

who fix the subsequent new autoencoder of

The loss is for least squares error or cross entropy

the gradient is taken w.r.t.  $\mathbf{w}$  for fitting at each

## \* Denoising Autoencoders (Type of Regularization)



Autoencoder

denoising autoencoder

Denoising autoencoder (n\_i - n\_i-hat)

(original input - denoised input)

Here,  $n_i$  is original input which is corrupted using a function & all ones are converted to zeros & vice versa in  $\tilde{n}_i$  is formed from  $n_i$  + (random noise)

Now,  $\hat{n}_i$  is obtained w/ loss function. Here will be

$$\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{n}_{ij} - n_{ij})^2$$

loss function (8)

model for  $\hat{n}_i$  is not  $\left( \frac{1}{m} \sum_{i=1}^m (\hat{n}_{ij} - n_{ij})^2 \right)$  it makes no sense

thus, since it is overcomplete autoencoder  
the  $\hat{n}_i$  won't go in  $\tilde{n}_i$  straight away because

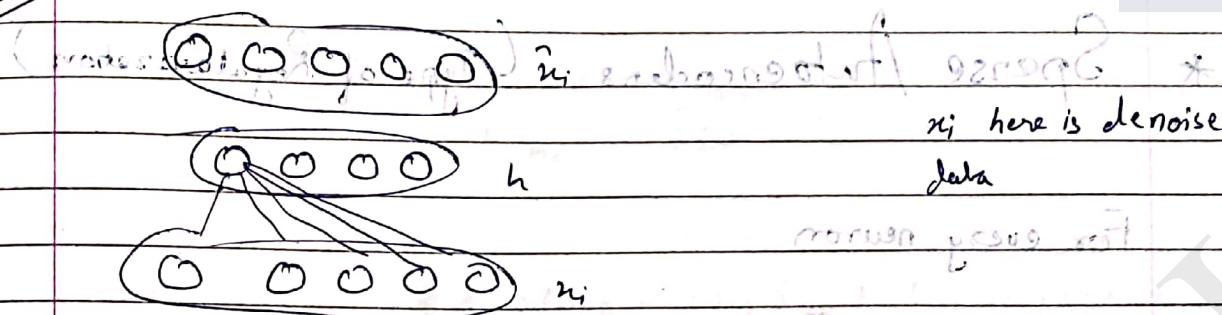
$\tilde{n}_i$  is compared with  $n_i$  w/ loss function

loss function is as given below

By reconstructing over corrupted  $\tilde{n}_i$ , it also learns some important features of  $n_i$  and will be able to predict if some data is missing.

Experiment

→ shows that denoising the inputs helps learn better feature representations from input.



$$\text{Each neuron in } h_j = \sigma(w_j^T n_i)$$

Now each neuron fires when  $(w_j^T n_i)$  is maximum.

maximal because it is at point 1 because at this point sigmoid reaches max value 1.

Message passed back pass at each of them is zero.

$$\max \{ w_j^T n_i \} \text{ such that } \|x_i\|^2 = n_i^T n_i = 1$$

at this point  $n_i$  stands on unit circumference where  $n_i$  is normalized.

$w_j^T n_i$  if dot product is maximum when  $\theta = 90^\circ$  in same direction.

$$n_i = \frac{w_j}{\|w_j\|}$$

$$(a-i) \text{ and } (a-i) + \sqrt{1 - a^2} \rightarrow (a) \cdot \omega_j$$

Each input here helps respective neuron to fire

$$\frac{w_1}{\sqrt{w_1^T w_1}}, \frac{\sqrt{w_2^T w_2}}{\sqrt{w_2^T w_2}}, \dots, \frac{w_n}{\sqrt{w_n^T w_n}}$$

$n_1, n_2, \dots, n_n$  are orthogonal to  $n_i$ .

variation of mean over first 5 epochs - (B) JG

This experiment is done after training set of all inputs

## \* Sparse Autoencoders (Type of Regularization)

↳ for every neuron in

For every neuron

$$\hat{p}_e = \frac{1}{m} \sum_{i=1}^m g(\omega_i^\top, u_i) \quad \text{is calculated}$$

↓

This gives us (at) an average whether the neuron will fire or not. If  $\hat{p}_e$  is less than it is a sparse neuron & will be zero most of times. Since we have taken average it will be close to zero and hence sparse.

$I = \omega^\top u = 11.811$  but since

$p$  is sparsity parameter which we choose close to zero lets say 0.005

Now we want  $\hat{p}_e = p$  (at an average every neuron should be sparse)

∴ we use

$$\Omega(\theta) = \sum_{i=1}^k p \log \frac{p}{\hat{p}_e} + (1-p) \log \frac{1-p}{1-\hat{p}_e}$$

$$L(\theta) = L(\theta) + \Omega(\theta)$$

→ will be minimum if  $p = \hat{p}_e$

→ Back Propagation

$\frac{\partial L(\theta)}{\partial \omega}$  \* charges thus we need to calculate

$\frac{\partial L(\theta)}{\partial \omega}$  (gradient w.r.t. weight)

$$\frac{\partial \ell(\theta)}{\partial \omega} = + \frac{\partial \ell(\theta)}{\partial p_i} \times \frac{\partial p_i}{\partial \omega}$$

Now,

$$\frac{\partial \ell(\theta)}{\partial p_i} = \lambda \log p_i - \log \hat{p}_i + (-\lambda) \log (1-p_i) -$$

$$= \lambda \left( (\lambda + \log(\lambda)) p_i \right) - \lambda \left( (1-\lambda) \log(1-\hat{p}_i) \right)$$

$$\frac{\partial \hat{p}_i}{\partial \omega}$$

$$= \lambda \left( \begin{array}{c} -\lambda \\ \frac{-\lambda}{\hat{p}_i(1-\hat{p}_i)} \end{array} \right) \quad \begin{array}{l} \text{vector with each} \\ \text{element representing} \\ \text{a neuron} \end{array}$$

(softmax is used to convert probabilities into raw values)

Now

$$\frac{\partial \hat{p}_i}{\partial w_{ij}} = \lambda \left( \frac{1}{m} \sum_{i=1}^m g(w_{ij} x_{ij} + b_i) \right)$$

scalar

$\frac{\partial w_{ij}}{\partial w_{ij}}$

$$= \frac{1}{m} \sum_{i=1}^m \left[ \underbrace{g'(w_{ij} x_{ij} + b_i)}_{\text{scalar}} \times \underbrace{x_{ij}}_{\text{scalar}} \right]$$

$$\frac{\partial \hat{p}_i}{\partial w_{ij}} = \lambda \frac{1}{m} \sum_{i=1}^m \left[ \underbrace{g(w_{ij} x_{ij} + b_i)}_{\text{scalar}} \times \underbrace{x_{ij}}_{\text{vector}} \right]$$

vector

$$= \frac{1}{m} \sum_{i=1}^m \left[ \underbrace{g'(w_{ij} x_{ij} + b_i)}_{\text{scalar}} \times \underbrace{x_i}_{\text{vector}} \right]$$

$$\frac{\partial \hat{p}_i}{\partial \omega} = \lambda \frac{1}{m} \sum_{i=1}^m \left( g(w x_i + b) \right)$$

matrix

$\frac{\partial \hat{p}_i}{\partial \omega}$

$$= \frac{1}{m} \sum_{i=1}^m \left[ \underbrace{g'(w_n; + b)}_{\text{vector}} \times \underbrace{n_i^T}_{\text{vector}} \right]$$

→ Final Derivative

$$\frac{\partial Q(\theta)}{\partial w} = \frac{1}{m} \sum_{i=1}^m \left( \underbrace{\begin{pmatrix} -p & (1-p) \\ \hat{p}_x & (1-\hat{p}_x) \end{pmatrix}}_{\text{vector from } \textcircled{1}} \odot \underbrace{g'(w_n; + b)}_{\text{vector}} \times \underbrace{n_i^T}_{\text{vector}} \right)$$

## \* Contractive Autoencoder

we use Jacobian of  $h$  (vector) over  $n$  (vector)  
 refer the notes of machine learning book 1.

$$\therefore J = \begin{bmatrix} \frac{\partial h_1}{\partial n_1} & \frac{\partial h_1}{\partial n_2} & \dots & \frac{\partial h_1}{\partial n_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_k}{\partial n_1} & \frac{\partial h_k}{\partial n_2} & \dots & \frac{\partial h_k}{\partial n_m} \end{bmatrix}$$

$n$  has dimension  $m$  (input layer vector) we call  
 $h$  has dimension  $k$  (hidden layer)

$$Q(\theta) = \sum_{j=1}^k \sum_{l=1}^L \left( \frac{\partial h_k}{\partial z_j} \right)^2$$

[using  $\frac{\partial h_k}{\partial z_j} = J_{kj}$ ]

$$\min (L(\theta) + \alpha(\theta)) \rightarrow \text{aim}$$

$$\therefore \sigma(\theta) = 0$$

(residual sum of squares means don't capture variations in data)

$L(\theta) = \text{capture important variations in data}$

$\{\epsilon_i\} - \{\epsilon_i\}' = \{\epsilon_i\} (\text{changes with respect to } \theta)$

longer  $\sigma^2(\theta) = \text{don't capture variations in data}$

(residual sum of squares doesn't change with respect to  $\theta$ )

Trade off - capture only very important variations in data

$\{\epsilon_i\}' - \{\epsilon_i\} = \{\epsilon_i\} \text{ training}$

(Residuals =  $\{\epsilon_i\}' - \{\epsilon_i\}$ )

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

lengths of becomes too large (and address with

lengths sum of becomes too long due

WEEK 8

\* Bias: (Simple models have high bias)

→ Formula

$$\text{Bias}(\hat{f}(x)) = E[\hat{f}(x)] - f(x)$$

\* Variance: (Complex Models) have high variance

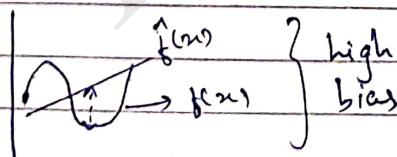
→ Formula

$$\text{Variance}(\hat{f}(x)) = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

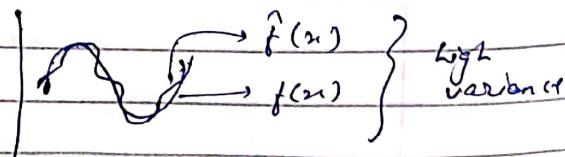
Variance (high)  $\rightarrow$   $E[\hat{f}(x) \neq f(x)]$

Here unlike bias predicted are not compared to original but they are compared to one another

\* Bias - Variance Trade off



Simple model



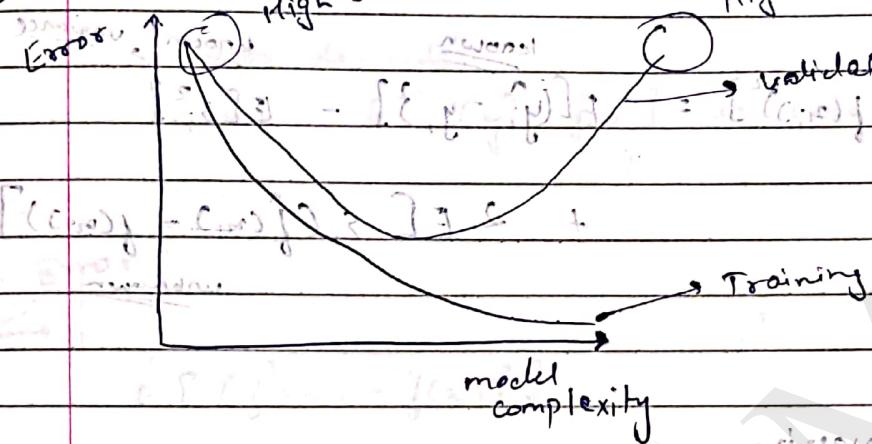
Complex model

It can be proved

$$E[(y - f(x))^2] = \text{Bias}^2 + \text{variance} + \sigma^2 \text{ (irreducible error)}$$

mean squared error

$\rightarrow$  High Bias      High Variance



If there are points in training & in testing point?

$$\text{data to train} \rightarrow \text{train err} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

$$\text{add point to test} \rightarrow \text{test err} = \frac{1}{m} \sum_{i=n+1}^{n+m} (y_i - \hat{f}(x_i))^2$$

Mathematically

we know training data  $(x_i, y_i)$  corresponding to  $f(x_i) + \epsilon_i$

$\therefore$  we can calculate  $\hat{y}_i$

$$[(\hat{y}_i - y_i)^2] = (\hat{y}_i - y_i)^2$$

$E[(\hat{y}_i - y_i)^2]$  can be empirically calculated

$$(\hat{y}_i - y_i)^2 = (\hat{y}_i - \hat{f}(x_i))^2$$

$$\therefore E[(\hat{y}_i - y_i)^2] = E[(\hat{y}_i - \hat{f}(x_i))^2]$$

$$(f(x_i) + \epsilon_i) - (\hat{y}_i - \epsilon_i) = (f(x_i) - \hat{y}_i) + (\epsilon_i - \epsilon_i)$$

Here we assume that we know  $f(x_i)$  but not  $f(x_i)$

$$\begin{aligned}
 & \text{Test Error} = \sum_{i=n+1}^m [f(\bar{x}_i) - y_i]^2 = E[(\hat{f}(\bar{x}_i) - f(\bar{x}_i))^2] \\
 & = E[(\hat{f}(\bar{x}_i) - f(\bar{x}_i))^2] - 2E[\varepsilon_i(\hat{f}(\bar{x}_i) - f(\bar{x}_i))] + E[\varepsilon_i^2] \\
 & = E[(\hat{f}(\bar{x}_i) - f(\bar{x}_i))^2] - 2E[\varepsilon_i(\hat{f}(\bar{x}_i) - f(\bar{x}_i))] + E[\varepsilon_i^2] \\
 & \therefore E[(\hat{f}(\bar{x}_i) - f(\bar{x}_i))^2] = \underbrace{E[(y_i - \hat{y}_i)^2]}_{\text{known}} - \underbrace{E[\varepsilon_i^2]}_{\text{variance}} \\
 & \quad + 2E[\varepsilon_i(\hat{f}(\bar{x}_i) - f(\bar{x}_i))]
 \end{aligned}$$

Test Error

Now, above expression

- $E[(y_i - \hat{y}_i)^2]$  can be empirically written as

$$\frac{1}{m-n} \sum_{i=n+1}^{n+m} (y_i - \hat{y}_i)^2 \quad \left. \begin{array}{l} \text{for test data} \\ \text{this is not exact (E)} \\ \text{but it is value} \\ \text{very close to expectation} \end{array} \right\}$$

Similarly,

$$E[\varepsilon_i^2] \approx \frac{1}{m-n} \sum_{i=n+1}^{n+m} \varepsilon_i^2$$

- $\& E[\varepsilon_i(\hat{f}(\bar{x}_i) - f(\bar{x}_i))] \approx 0$ , minimize covariance  $C[\varepsilon_i, (\hat{f}(\bar{x}_i) - f(\bar{x}_i))]$

$y_i = f(\bar{x}_i) + \varepsilon_i$  for test data  
 $f(\bar{x}_i)$  for test data is calculated  
 using parameters from training  
 data which depend on  
 $\varepsilon_i$  of training data and  
 not  $\varepsilon_i$  of test data

$$\because \text{cov}(x, y) = E[(x - \bar{x})(y - \bar{y})]$$

$$\begin{aligned}
 & \text{cov}(x, y) = E[(x - \bar{x})(y - \bar{y})] \\
 & = E[xy]
 \end{aligned}$$

Now, since  $(x - \bar{x})$  &  $(\hat{f}(\bar{x}_i) - f(\bar{x}_i))$  are independent

$$(x - \bar{x}) \cdot (\hat{f}(\bar{x}_i) - f(\bar{x}_i)) = 0 \quad \text{if } x \perp (\hat{f}(\bar{x}_i) - f(\bar{x}_i))$$

∴ total cov = 0 if  $x$  &  $\hat{f}(\bar{x}_i) - f(\bar{x}_i)$  are independent

$$\therefore \text{we } E[(\hat{f}(x_i) - f(x_i))^2] = \text{Error}$$

$$\text{measured as } \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2 + 3 \sum_{i=1}^m \epsilon_i^2$$

Since function is not linear then  $\epsilon_i^2$  is small constant

$\therefore$  Error depends on training error.

on  $E[(\hat{y}_i - y_i)^2]$  in case of test error

$$E[(\hat{y}_i - y_i)^2] = (m)(f - f(x_i))^2 + \frac{1}{n} \sum_{i=1}^n \epsilon_i^2$$

Train Error

$\rightarrow$  Now we have to find test error i.e.

$$E[(\hat{f}(x_i) - f(x_i))^2]$$

is  $\hat{f}(x_i)$  is fit based upon  $x_i$

$$= \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + - \frac{1}{n} \sum_{i=1}^n \epsilon_i^2 + 2E[\epsilon_i(\hat{f}(x_i) - f(x_i))]$$

here  $\epsilon_i$  is independent of training items with covariance  $(\epsilon_i, (\hat{f}(x_i) - f(x_i)))$

Here  $\epsilon_i$  is independent on training dataset wrt  $x_i$

$\therefore$  covariance  $\neq 0$

Thus  $E[(\hat{y}_i - y_i)^2]$  can't give correct estimate of error

Conclusion

Validation set gives better estimation of error than training set

Test set  $(B)$ ,  $(B) + (C)$  and  $(A)$  will give best result

and assessment set  $(D)$  will give worst result

$\therefore$  Validation set gives better result than training set

Training set for testing

→ Earlier

$$\mathbb{E} [\varepsilon_i (\hat{f}(x_i) - f(x_i))] \text{ was unknown}$$

$\mathbb{E} [\varepsilon_i (\hat{f}(x_i) - f(x_i))] + \text{was unknown}$

Now,

using Stein's Lemma

$$\frac{1}{n} \sum_{i=1}^n \varepsilon_i (\hat{f}(x_i) - f(x_i)) = \frac{\sigma^2 \sum_{i=1}^n \frac{\partial f(x_i)}{\partial y_i}}{n}$$

this value is high when  $\frac{\partial f(x_i)}{\partial y_i}$

is high and it is high

$\frac{\partial f(x_i)}{\partial y_i}$  gives small change in observation causes large change in estimation

so this value is high in complex models

∴ True error = empirical train error

+ small constant term

+

$\alpha$  (model complexity)

→ This we minimize has nothing to do with model

$$\min_{\theta} L_{\text{train}}(\theta) + \alpha R(\theta)$$

for a given  
concept  
of  
regularization

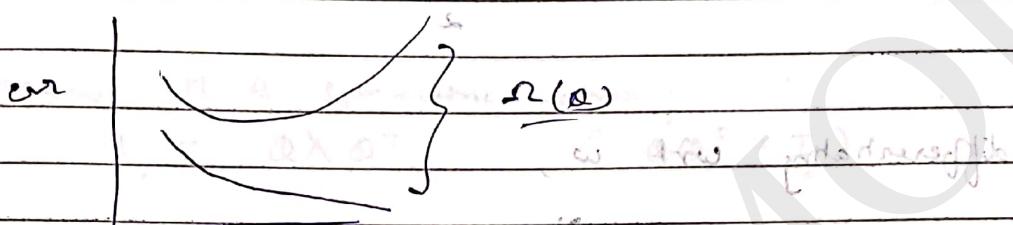
instead of train errors

Now we still don't know this quantity  $\omega(\omega)$   
thus we use

$$(i) H^T(\omega) \leq (d + \omega) J \nabla^T(\omega) + (\omega) J \omega^2 = (d + \omega^2) J$$

any function as  $\omega(\omega)$  such that it is high for complex models & low for simple models.

$$+ \omega - \omega) H^T(d - \omega) \leq +$$



## \* L2 Regularization

$$H^T(d - \omega) = (d - \omega) J \nabla$$

$$\tilde{L}(\omega) = L(\omega) + \frac{1}{2} \alpha \|\omega\|^2$$

$$\nabla \tilde{L}(\omega) = \{\nabla L(\omega) + \alpha \omega\} J \nabla \in (\omega) J \nabla$$

update  $H^T(d - \omega) = (\omega) J \nabla$

$$\omega_{t+1} = \omega_t - \eta (\nabla L(\omega_t) + \alpha \omega_t)$$

Biggest minus sign

Now  $\omega = (\omega) J \nabla$  (definition of  $\tilde{L}$ )

let  $\omega^*$  be such that  $\nabla \tilde{L}(\omega^*) = 0$

Loss without regularization

&  $h = \omega - \omega^*$  where  $(\omega, \omega^*)$  is a point in neighbourhood of  $\omega^*$

Now quest is to find  $\nabla L(\omega)$

(i) from Taylor's series in upto 2<sup>nd</sup> terms note we will 2<sup>nd</sup> derivative  
 $\Rightarrow \text{sum out last}$  9

$$L(\omega^* + h) = L(\omega^*) + (h)^T \nabla L(\omega^*) + \frac{1}{2} (h)^T H(h)$$

and put in last term (ii) in question p no

$$L(\omega) = \underbrace{L(\omega^*)}_{0} + \underbrace{(h - \omega^*)^T \nabla L(\omega^*)}_{0} - (\because \nabla L(\omega^*) = 0 \text{ from } \text{Q})$$

$$+ \frac{1}{2} (\omega - \omega^*)^T H (\omega - \omega^*)$$

(ii) 9

differentiating wrt  $\omega$ ,

$$\nabla L(\omega) = \underbrace{\nabla L(\omega^*)}_{0} + \underbrace{H(\omega^*)}_{0} - (\because \nabla L(\omega^*) = 0 \text{ from Q})$$

$$+ (\omega - \omega^*) H$$

$$\nabla L(\omega) = (\omega - \omega^*) H$$

Now,

$$\tilde{L}(\tilde{\omega}) = \nabla L(\omega) + \alpha \omega + (\omega)^T V = \text{regularized loss function } \tilde{L}(\omega)$$

$$\nabla \tilde{L}(\tilde{\omega}) = (\omega - \omega^*) H + \alpha \omega$$

$$(\omega - \omega^* + \alpha \omega) H = 0$$

Let  $\tilde{\omega}$  be such that  $\tilde{L}(\tilde{\omega}) = 0$  with  $\leftarrow$

①  $\tilde{\omega} = \omega^* - (\alpha \omega) \leftarrow$  best direction to go  
 $\rightarrow$  regularized loss

$$\therefore \text{from } H(\tilde{\omega}, \omega^*) + \alpha \tilde{\omega} = 0 \Rightarrow \tilde{\omega} = -\frac{1}{\alpha} H(\tilde{\omega}, \omega^*)$$

$\leftarrow$  to understand it

$$\therefore (H + \alpha I) \tilde{\omega} = H \omega^*$$

$$\tilde{\omega} = (H + \alpha I)^{-1} H \omega^*$$

Note:  $\alpha \rightarrow 0$  then  $\tilde{\omega} = \omega^*$  (no regularization)

→ Assume  $H$  is symmetric matrix  $(n \times n)$

$$H = Q \Lambda Q^T \quad (Q Q^T = I)$$

$$\tilde{\omega} = (H + \alpha I)^{-1} H \omega^*$$

$$= (Q \Lambda Q^T + \alpha I)^{-1} Q \Lambda Q^T \omega^*$$

$$= (Q \Lambda Q^T + \alpha Q I Q^T)^{-1} Q \Lambda Q^T \omega^*$$

$$= (Q \Lambda (Q^T + \alpha I) Q^T)^{-1} Q \Lambda Q^T \omega^*$$

$$= Q^T (Q^T + \alpha I)^{-1} Q \Lambda Q^T \omega^*$$

$$= Q \boxed{(Q^T + \alpha I)^{-1} Q \Lambda} Q^T \omega^*$$

Diagonal matrix  $D$

$$\tilde{\omega} = Q D Q^T \omega^*$$

rotates  $\omega^*$  scales  $\omega^*$  rotates  $\omega^*$  rotates  $\tilde{\omega}$   
↓ ↓ ↓ ↓  
 rotates  $\tilde{\omega}$  scales  $\tilde{\omega}$  rotates  $\tilde{\omega}$  rotates  $\tilde{\omega}$

Here  $\tilde{\omega}$  &  $\omega^*$  are weight vectors in the solution space.

$$D = \begin{bmatrix} \frac{\lambda_1}{\lambda_1 + \alpha} \cos \theta & \frac{\lambda_1}{\lambda_1 + \alpha} \sin \theta \\ \frac{\lambda_2}{\lambda_2 + \alpha} \cos \theta & \frac{\lambda_2}{\lambda_2 + \alpha} \sin \theta \\ \vdots & \vdots \\ \frac{\lambda_n}{\lambda_n + \alpha} \cos \theta & \frac{\lambda_n}{\lambda_n + \alpha} \sin \theta \end{bmatrix}$$

Condition number ( $\kappa$ )  $\kappa_D = \frac{\lambda_1}{\lambda_n + \alpha}$

$$= (\lambda + \alpha I)^{-1} \text{ where } \lambda \in D \text{ eigenvalues}$$

$(I = \text{Identity}) \quad \text{rank } D = n \rightarrow$

$$\begin{bmatrix} \frac{1}{\lambda_1 + \alpha} & & & & \lambda_1 \\ & \frac{1}{\lambda_2 + \alpha} \cos \theta & & & \lambda_2 \\ & & \ddots & & \vdots \\ & & & \frac{1}{\lambda_n + \alpha} \cos \theta & \lambda_n \end{bmatrix}$$

$$+ \alpha^{-1} \text{rank } D^{-1} (\text{rank } D = n) =$$

Note

$$\text{Now, } \text{rank } D^{-1} (\text{rank } (\lambda + \alpha I) = n)$$

$$\text{if } \lambda_i \gg \alpha, \quad \frac{\lambda_i}{\lambda_i + \alpha} = 1$$

$$\text{if } \lambda_i \ll \alpha, \quad \frac{\lambda_i}{\lambda_i + \alpha} = 0$$

A similar analogy

## \* Dataset Augmentation

Increasing the amount of data

- Rotating images
- Shifting horizontally / vertically
- Blurring, adding noise etc

\* Parameter Sharing & Tying: used in CNN and in Encoder/Decoder

$$\theta = \text{parameters} \quad \text{and} \quad \theta = \text{parameters}$$

\* Adding Noise to Data (Inputs) / (Encoding)

$$[\varepsilon \sim N(0, \sigma^2)] \quad \text{Gaussian distribution}$$

$\tilde{x}_i = x_i + \varepsilon_i \quad (\text{zero mean, unit variance})$

$\tilde{x}_i = x_i + \varepsilon_i \quad (\text{zero mean, multi-variate})$

$$\hat{y} = \sum_{i=1}^n w_i x_i$$

$$\begin{aligned} \tilde{y} &= \sum_{i=1}^n w_i \tilde{x}_i \\ &= \sum_{i=1}^n w_i (x_i + \varepsilon_i) \\ &= \hat{y} + \sum_{i=1}^n w_i \varepsilon_i \end{aligned}$$

Now, we are interested in  $E[(\hat{y} - y)^2]$

$$\therefore E[(\hat{y} - y)^2] = E[(\hat{y} + \sum_{i=1}^n w_i \varepsilon_i - y)^2]$$

$$= E[((\hat{y} - y) + (\sum_{i=1}^n w_i \varepsilon_i))^2]$$

$$= E[(\hat{y} - y)^2] + E[2(\hat{y} - y) \sum_{i=1}^n w_i \varepsilon_i]$$

of expectation of products of independent random variables

$$= 0 + E[(\sum_{i=1}^n w_i \varepsilon_i)^2]$$

by prop.

Here in second term ~~gradient~~ partial derivative

$(\hat{y} - y) \& \epsilon_i$  are independent

~~original values when noise was not added~~

thus covariance  $(\hat{y} - y) \epsilon w_i \epsilon_i = 0$

Covariance (diagonal) total of each gradient

A in third term

gradient of loss function

$$\text{Covariance } (\sum w_i \epsilon_i)^2 \Rightarrow (a + b + c)^2$$

(shown above in previous slide)

$$(\epsilon_i \epsilon_j)^2 \approx 0 \text{ if independent}$$

$$(\epsilon_i^2) \neq 0$$

$$\therefore (\sum w_i \epsilon_i)^2 = \sum w_i^2 \epsilon_i^2$$

$$= E[(\hat{y} - y)^2] + 0 + E[\sum_{i=1}^n w_i^2 \epsilon_i^2]$$

$$\text{Here } E[\sum_{i=1}^n w_i^2 \epsilon_i^2] = (\sum_{i=1}^n w_i^2) E[\epsilon_i^2]$$

since  $w_i$  is not random variable

$$= E[(\hat{y} - y)^2] + \sigma^2 \sum_{i=1}^n w_i^2$$

$$= E[(\hat{y} - y)^2] + \sigma^2 \sum_{i=1}^n w_i^2$$

↓ same as L<sub>2</sub>

regularization

$$= E[(\hat{y} - y)^2] + \sigma^2 \sum_{i=1}^n w_i^2$$

This is used in overcomplete autoencoders for encoding, in order to reduce the complexity by number of dimensions increase.

① Input

## \* Adding Noise to Data (Output) (softmax + noise)

At the output we have

$$\sum_{i=1}^n p_i \log q_i \quad (\omega - \omega_{\text{true}})^H H = (\omega) \Delta \theta$$

$\downarrow \quad \downarrow$   
original      predicted

we add noise to original so ~~data~~ training error doesn't drive to 0

$$p_i = \{0, 0, 1, 0, \dots\} \quad \text{Assuming there are } q \text{ elements}$$
$$p_i = \left[ \frac{1}{q}, \frac{\epsilon}{q}, \frac{1}{q}, \frac{1}{q}, \frac{\epsilon}{q}, \dots \right] \quad \begin{matrix} \text{noise} \\ \text{the correction} \\ \text{removed} \end{matrix}$$

$$q_i = \text{remains as it is } (p_i + \Delta \theta)$$

## \* Early Stopping

Patience parameter  $p$

→ At every epochs check train error & validation error and if for previous  $p$  epoch if validation error has increased or remained same stop

→ In gradient Descent  $\eta$  is pre-given &  $\lambda$

$$\omega_{t+1} = \omega_t + \eta \nabla \omega_t (t+1) - \lambda I = \tilde{\omega}$$

$$= \omega_0 + \eta \sum_{i=1}^t \nabla \omega_i - \lambda I \quad \tilde{\omega} = \omega_0$$

maximum out of them

$$\omega_{t+1} \geq \omega_0 + \eta t I \quad (N \in \mathbb{N})$$

Thus  $t$  controls how far  $\omega$  is from  $\omega^*$ .

→ As seen earlier

$$\nabla L(\omega) = H(\omega - \omega^*)$$

$$\omega_t = \omega_{t-1} + \eta \nabla L(\omega_{t-1})$$

$$\begin{aligned} \text{Epochs move forward along the contours of } & \text{minimizing } L(\omega) \text{ (blue line)} \\ & \text{and } L(\omega) \text{ (red line)} \\ & \text{at each step } \omega_t = \omega_{t-1} + \eta H(\omega_{t-1} - \omega^*) \end{aligned}$$

$$\hat{\omega}_t = (I + \eta H)\omega_{t-1} - \eta H\omega^*$$

$$\therefore \omega_t = (I + \eta H)^{-1}\omega_{t-1} - \eta H^{-1}\omega^*$$

- Using EVD,  $H = Q\Lambda Q^T$  (principal plot)

$$\omega_t = (I + \eta Q\Lambda Q^T)\omega_{t-1} - \eta Q\Lambda Q^T\omega^*$$

If  $\omega_0$  is random and standard variance matrix  $I$

$$\omega_t = Q [Q^T(I + \eta \Lambda)^{-1}Q] Q^T \omega^* \quad \text{Eq. ①}$$

- This  $\Rightarrow$  early stopping is similar to  $L_2$  regularization

$$\tilde{\omega} = Q [I - (\lambda + \alpha I)^{-1}\alpha] Q^T \omega^* \quad \text{Eq. ②}$$

- $\omega_t = \tilde{\omega}$  for following condition

$$(I - \epsilon \Lambda)^{-1} = (\lambda + \alpha I)^{-1}\alpha$$

## \* Ensemble Methods

→ Methods like Bagging (ML book 2) dataset is broken down into parts and models are prepared for each training data and error is evaluated for each part.

### Evaluate

Avg of all errors generated (Here the number of models is  $k$ )

$$\frac{1}{k} \sum_i \varepsilon_i$$

Expectation over the inputs

Expected mean squared error  $\text{E}[ \left( \frac{1}{k} \sum_{i=1}^k \varepsilon_i \right)^2 ]$

equal in accuracy & stability of many bags

$$\text{where } = \text{over } \frac{1}{k^2} \text{ E} \left[ \sum_{i=1}^k \sum_{j=1}^k \varepsilon_i \varepsilon_j \right] + \text{other terms}$$

$$\text{L terms with } \frac{1}{k^2} \text{ E} \left[ \sum_{i=1}^k \sum_{j=1}^k (\varepsilon_i \varepsilon_j) + \text{other terms} \right]$$

$$= \frac{1}{k^2} \left[ \sum_i \text{E}(\varepsilon_i^2) + \sum_i \sum_j \text{E}(\varepsilon_i \varepsilon_j) \right]$$

$$= \frac{1}{k^2} \left[ kV + (k-1)C \right]$$

$$= \frac{V}{k} + \left( \frac{k-1}{k} \right) C$$

experience

average covariance

NoteIf errors are independent  $C = \sigma^2 I$ method is robust (s. good fit)  $\rightarrow$  good fit.   
Thus averaging  $\bar{V}$  when  $V$  has large error matrix  $C$ , does not introduce bias term when  $C = V$ If errors are correlated  $C = V$ 

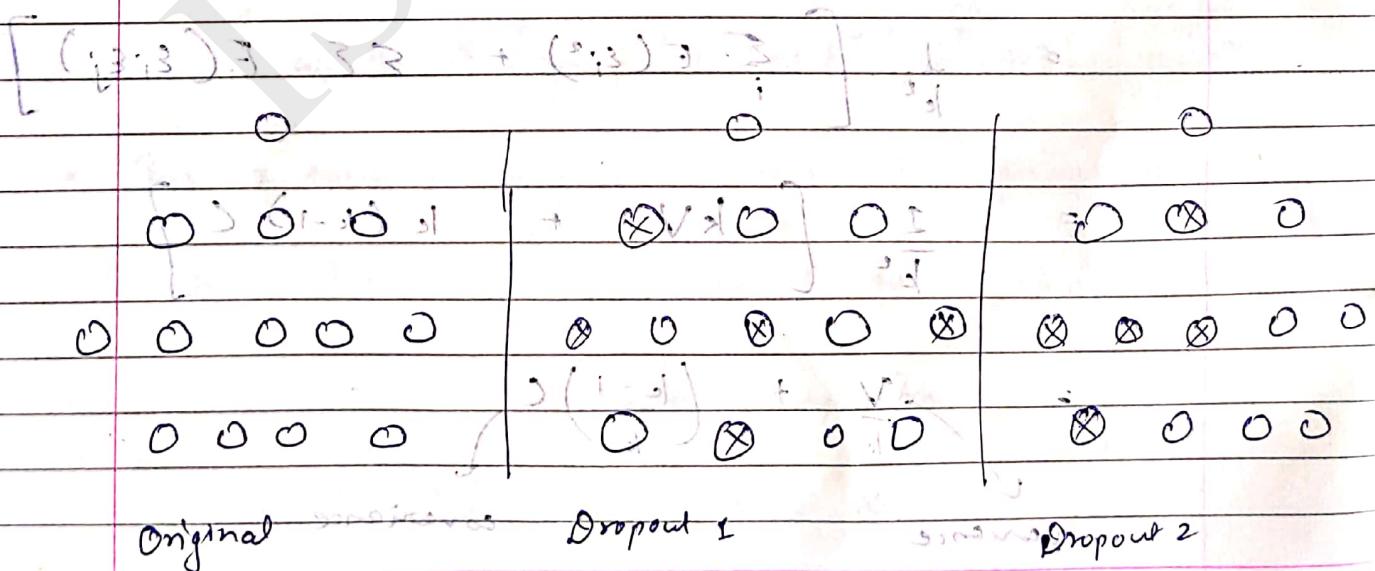
$$mse = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Thus using averaging helps reduce errors in fit.

## \* Dropout

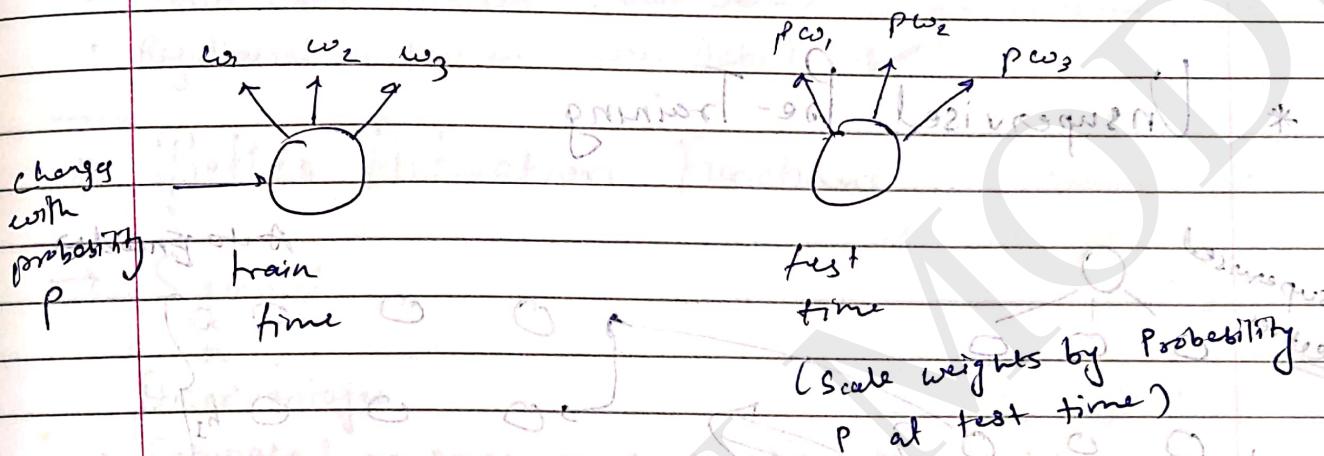
Let  $W$  be the weight matrix after backpropagation

- Dropout means to eliminate neurons in layers with probability  $p$  to form completely new model train for one epoch on that model. update  $W$ .
- Now eliminate from original to get new model & use updated  $W$  and train for 1 epoch



This will make all neurons independent from one another because they can disappear any time.

These neurons will be robust (due to feedback).



Now we can see that the neurons are independent. That is, the initial learning and the subsequent changes in the weights are addressed as if each does not affect, and the new weights are not affected by the previous changes. This is called the separation of weight and bias shift.

Mathematically,  $\frac{\partial \text{loss}}{\partial w_i} = \sum_j \frac{\partial \text{loss}}{\partial w_i}$

The important thing is that neurons are independent. This means that the structure of each neuron is not affected by other neurons.

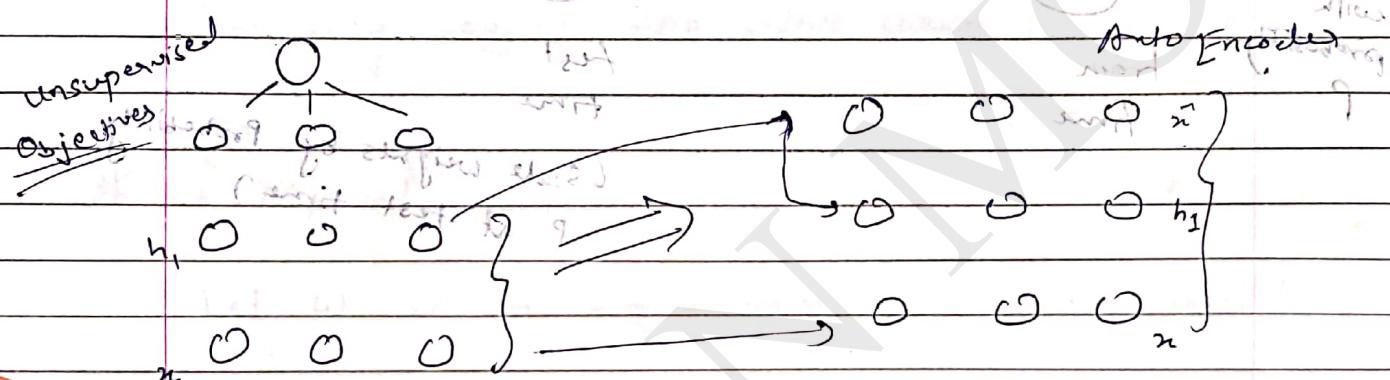
If enough time has passed, these best off as well. This is called learning for optimization.

recently learned neurons enterprises continue to function correctly, providing many consequences.

~~WEEK 9~~ ~~many techniques can increase the learning rates but~~  
~~exit units responsible for soft sigmoid sections~~

→ Gradient at a particular layer depends on input at that layer (refer backprop deep learning (1))

## \* Unsupervised Pre-Training



Thus, between each layer there is an encoder which gives abstraction of previous layer

Thus we are trying to minimize

$$\text{Loss}(Q) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - \hat{x}_{ij})^2$$

After this is done for entire networks the weights will be initialized such that each layer is abstraction of previous.

Instead of random weights we used weight from unsupervised pre-training.

~~supervised  
Objectives~~

$$L(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - f(x_i))^2$$

{ Normal ←  
Backprop ←  
regularization ← }

- Helps converge Faster
- Optimization Problem (Train Data)
- Regularization Problem (Test Data)

## \* Better Activation Functions

→ Sigmoid

Disadvantages

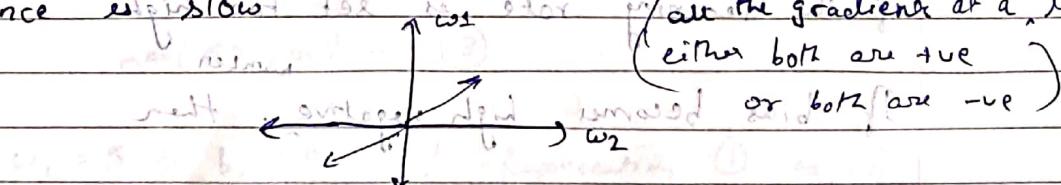
- Saturated neurons cause gradient to vanish

Here, in this case if value of  $n$  is very high or very low then sigmoid( $n$ ) saturates that is, its value becomes 1 and 0 respectively.

thus derivative  $\sigma(n)(1 - \sigma(n))$  which is used in gradient becomes zero. Thus gradient become zero.

- Not zero centered ( $0$  from  $0$  to  $1$ )

∴ the movement of gradient becomes restricted and thus convergence is slow for one or other reason (all the gradients at a layer either both are true)



- Computationally expensive ( $\exp(n)$ )

Time and memory ↓  $O = \text{const}$   $\rightarrow$   $O = \text{const}$

→ Tanh

Disadvantage

- Gradient vanishes at saturation
- Computationally expensive due to addition

Solve

- zero-centered

→ ReLU

$$f(n) = \max(0, n)$$

$$f(n) = \max(0, n+1) - \max(0, n-1) \quad \left\{ \begin{array}{l} \text{equivalent} \\ \text{to sigmoid} \end{array} \right.$$

or first layer is  $\approx$  ReLU-like if  $n$  is ReLU-like in next

∴ both networks  $\approx$  becomes next will pass

Mostly used in NNs bcoz it's fastest and

Only problem ( $\Rightarrow$  (dead neuron problem)) is that

most of neurons will (never) receive input in last

$$\text{Case 1: } \frac{\partial f}{\partial n} = 0, \quad n < 0 \quad \left\{ \begin{array}{l} \text{derivatives} \\ \text{otherwise} \end{array} \right.$$

$$(1, n, n \geq 0) \quad \text{another case later}$$

In practice a large fraction of ReLU units can die

if the learning rate is set too high

(but this had nothing to do with this)

If bias becomes high negative then

$$w_1n_1 + w_2n_2 + b < 0$$

(which is analogous phenomenon)

$$\therefore \frac{\partial f}{\partial n} = 0 \quad \text{& weights are not updated}$$

→ Leaky ReLU ( $\text{ReLU}_{\text{soft}}$ ). non-saturation.  $f(x) = \max(0, \alpha x)$

(solution to dead neuron)

$$f(x) = \max(0, \alpha x) \quad (\text{if } x > 0 \text{ then } f(x) = \alpha x, \text{ if } x \leq 0 \text{ then } f(x) = 0)$$

→ Parametric ReLU

$$f(x) = \max(\alpha x, x) \quad (\text{if } \alpha > 1 \text{ then } f(x) = \alpha x, \text{ if } \alpha < 1 \text{ then } f(x) = x)$$

lower selection parameter not higher performing than  $x$   
higher parameter

→ Exponential Linear Unit

more rd. activation and higher non-linearity which makes it more expressive

$$f(x) = x \quad \text{if } x > 0 \\ = \alpha e^x - 1 \quad \text{if } x \leq 0$$

→ Maxout Network (more power of representation)

$$\max(C_1 w_1^T x + b_1, C_2 w_2^T x + b_2) \quad \text{--- (1)}$$

variant of ReLU  $\max(0, \alpha x)$  at  $\alpha = 1$  (representation)

$$\left. \begin{array}{l} w_1 = 1, b_1 = 0 \\ w_2 = 1, b_2 = 0 \end{array} \right\} \text{Representing } \text{--- (1)} \text{ as } \text{--- (2)}$$

$$\max(\alpha x, x) \quad \text{--- (3)} \quad \text{further higher order}$$

$$\left. \begin{array}{l} w_1 = \alpha, b_1 = 0 \\ w_2 = 1, b_2 = 0 \end{array} \right\} \text{Representing } \text{--- (1)} \text{ as } \text{--- (3)}$$

## \* Better Weight Initialization Strategies

(Methods of initialization)

→ Weights are initialized by zero ( $0.0$ ) or  $\pm 0.01$

$$a_{11} = w_{11}n_1 + w_{12}n_2$$

$$a_{12} = w_{21}n_1 + w_{22}n_2$$

W12 & W21 initialized

$$a_{11} = a_{12} \quad \& \quad h_{11} = h_{12}$$

Thus both weights will get the same update and remain equal

→ Same happens when weights are initialized by same values

(Known as Symmetric Breaking Problem)

→ Initializing weights to very small values

① vanishing gradient problem occurs if more layers are present

→ Initializing weights to large values

variance is low when plotted  
② saturation occurs and thus the vanishing gradient problem persists

Principled Way

$$\rightarrow S_{11} = \sum_{i=1}^n w_{11}n_i$$

$$v(XY) = \underbrace{(\bar{x})^2 v_{\text{ar}}(Y) + (\bar{y})^2 v_{\text{ar}}(X) + v_{\text{ar}}(Y) v_{\text{ar}}(X)}_{\text{Formula}}$$

DELUXE  
PAGE NO.:  
DATE:



$$v(S_{1,1}) = \frac{\text{var}(\sum_{i=1}^n w_i x_i)^2}{n^2} = \frac{\sum_{i=1}^n \text{var}(w_i x_i)}{n}$$

Now  $(\bar{w}_i)$  now rest  $\leftarrow (\bar{w}_i)$  now

$$\text{variance} = \sum_{i=1}^n \left( E[w_i]^2 \text{Var}(x_i) \right)$$

$$\text{rest now } (\bar{w}_i)^2 + E[x_i]^2 \text{Var}(w_i)$$

$$+ \text{Var}(x_i) \text{Var}(w_i)$$

$$\left( \sum w_i \right) \text{Var} X = (\bar{w}) \text{Var} X$$

lets say  $w_i$  &  $w_{i,i}$

rest w.i are 0 mean &

standard deviation

$$\text{var}(x_i) = \text{var}(n), \text{ var}(w_i) = \text{var}(w)$$

$$= \sum_{i=1}^n \text{Var}(x_i) \text{Var}(w_i)$$

$$(S)_{\text{cov}} = (S)_{\text{var}} \quad (S)_{\text{cov}} = (S)_{\text{var}} = w \cdot n$$

$$= n \text{var}(w) \text{Var}(n)$$

$$\therefore \text{Var}(S_{1,1}) = (n \text{var}(w))(\text{Var}(n)) - \text{Eq. 1}$$

so that  $S_{1,1}$  variance  $\downarrow$   $\rightarrow$   $(S)_{\text{cov}}$

Now,

$$\text{var}(S_{2,1}) = \sum_{i=1}^n \text{Var}(S_{1,i}) \text{Var}(w_{2,i})$$

$$= n \text{Var}(S_{1,1}) \text{Var}(w_2)$$

so that variance of  $w_i$  in  $\downarrow$  substituting Eq. 1

$$\text{var}(S_{2,1}) \propto \left[ \frac{n \text{var}(w)]^2 \text{Var}(n)}{n} \right]$$

thus low values & high values

would lead to 0 variance of  $S_{2,1}$   
or very high variance of  $S_{2,1}$



Scanned with OKEN Scanner

$$\text{Var}(s_{ki}) = \frac{1}{n} \text{Var}(\omega) \left[ \frac{\text{Var}(x_i)}{\text{Var}(x_i) + (y)_{\text{cov}}^2 (\bar{x})} \right]$$

if

$n \text{Var}(\omega) = 1$  then  $\text{Var}(s_{ki})$  will remain normal

→ If  $\text{cov}(z) = z \sim N(0, 1)$  and we take

weights such that

$$(x, \omega)_{\text{new}} = (x, \omega)_{\text{old}}$$

$$n \text{Var}(\omega) = n \text{Var}(z)$$

$\sqrt{n}$  is number of neurons in the layer  
norm. var. of each neuron

thus

$$(x, \omega)_{\text{new}} = (x, \omega)_{\text{old}}, (x)_{\text{new}} = z_{\text{old}}$$

since  $(\omega = \frac{z}{\sqrt{n}})$

$$\text{Now } (x, \omega)_{\text{new}} = (x, \omega)_{\text{old}} \quad \left. \begin{array}{l} \text{var}(z) = n \text{Var}(z) \\ (x)_{\text{new}} = \sqrt{n} (x)_{\text{old}} \end{array} \right\} \text{var}(az) = a^2 \text{var}(z)$$

$$= n \times \frac{1}{n} \text{var}(z)$$

$$= (x)_{\text{new}}^2 \text{Var}(\omega) = (x, \omega)_{\text{new}}$$

$$= \text{var}(z) = 1 \text{ since normal distribution}$$

Applicable for Sigmoid & Tanh ( $a, b$ ) new

$$(a, b)_{\text{new}} = (a, b)_{\text{old}}$$

→ For ReLU (since it is 0 for negative values that is approx half of time)

$$\omega = \left[ \frac{(z)_{\text{new}}^2 - (x)_{\text{new}}^2}{\sqrt{\frac{n}{2}}} \right] \times (a, b)_{\text{new}}$$

similarly for sigmoid & tanh with diff.

for sigmoid a & b best choice  
for tanh a & b max

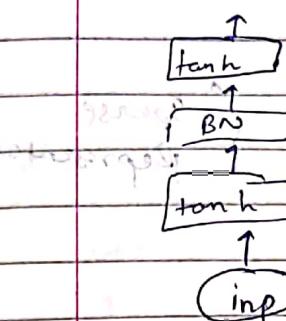
input vector =  $[x_1, x_2, x_3, \dots, x_n]$

here we normalize  
with respect to  
features of  
input

DELUXE  
PAGE NO.:  
DATE:



## \* Batch Normalization



in between layers normalize values so that input to the next layer comes from the same distribution as the entire network

$$\rightarrow \hat{S}_{ik} = \frac{S_{ik} - E[S_{ik}]}{\sqrt{\text{var}(S_{ik})}} \quad 0 \text{ mean, unit variance}$$

for backprop. we need to differentiate the above expression also

If we don't want normalized values we can convert batch Norm to original

$$y^{(k)} = \gamma^{(k)} \hat{S}_{ik} + \beta^{(k)}$$

where, network learns  $\gamma$  &  $\beta$  which should be as follows

$$\gamma^{(k)} = \sqrt{\text{var}(x^{(k)})} \quad \text{and} \quad \beta^{(k)} = E[x^{(k)}]$$

$\rightarrow$  Batch Norms helps network converge faster in some cases

For better understanding refer this file - [deepLearningNotes/week9/batchnorm.tutorial.html](#)

[deeplearning1 notes/ week9 / batchnorm.tutorial.html](#)  
[batchnorm.png](#)



WEEK 10

## \* One-Hot Representation of Words (Sparse Representation)

Corpus → Collection of Sentences

Vocabulary - All unique words in corpus

→ Let's say vocabulary is

cat, dog, truck

One-Hot =  $[1, 0, 0]$ ,  $[0, 1, 0]$ ,  $[0, 0, 1]$

- It Occupies lot of space

- No similarity between words (e.g. Euclidean distance between cat & dog should be less because both are animals)

## \* Distributed Representation of Words

→ Write the individual unique words in corpus as rows and columns

→ Make a co-occurrence matrix with words

words Human, machine, interface, for, computer, applications

Corpus - Cat eats a dog

Dog eats a bone

Cat eats a mouse

vocab - Cat, eats, a, Dog,

bone, mouse

tokens (n-gram)

(word w (i,j) means j-th word)

Table of Co-occurrence matrix  $\Rightarrow k=1$  TMA

context

words	Cat	eats	(w,i)	Dog	Bone	Mouse	TMA
Cat	0	2	0	0	0	0	0
eats	2	6	13	1	0	0	0
	$a(w)$ times a common			b			
Dog				0			
Bone					0	0	0
Mouse						0	

• < (Here)  $k$ , is that if for a current neighbour word we need to check 1 neighbour towards left and right.

→ (Cat) appeared two times around the word eat in a window of 1 word around it.

→ This is still very complex

→ ~~overcomes difficulties of QV2 problem~~

solution

- Remove stopwords from column (context)
- Ignore very frequently occurring words

• Use threshold,  $t = 100$

entry in above matrix

$$x_{ij} = \min (\text{count}(w_i, c_j), t)$$

word context

glossary animal : teacup took = Txy

as Tx for marks less suff

(marks i) associated plurals are

(low if  $\text{count}(w, c)$  is low)

- Use  $\text{PMI } L = \frac{1}{c}$  (high if  $\text{count}(w, c)$  is high)

$$\begin{aligned} - \text{PMI}(w, c) &= \log \frac{p(c|w)}{p(c)} = \log \left( \frac{\text{count}(w, c)}{\text{count}(w)} \right) \\ &= \log \left( \frac{\text{count}(w, c) + N}{\text{count}(c) + \text{count}(w)} \right) \end{aligned}$$

 $N$  is total no. of words

- $\text{count}(w, c) = 0$ ,  $\text{PMI} = -\infty$

(±) now  $\text{PMI}_0(w, c) = \text{PMI}(w, c)$ , if  $\text{count}(w, c) > 0$ 

With word frequency = 0, we set 1, else otherwise

(2)  $\text{PPMI}(w, c)$  based on  $\text{PMI}(w, c)$  and if  $\text{PMI}(w, c) > 0$ 

positive if, known words &amp; 0 if unknown words

extreme available is suff

### \* Using SVD for dimensionality reduction

$$\hat{X}_{m \times n} = \underbrace{U_{m \times k}}_{\text{orthogonal matrix}} \underbrace{\Sigma_{k \times k}}_{\text{diagonal matrix}} \underbrace{V_{k \times n}}_{\text{orthogonal matrix}}$$

$$\hat{X} = U \Sigma V^T$$

rank  $k$  approx  
we construct it using additive  
 $k$  matrix

Now  $(U, \Sigma, V)$  terms

$$\hat{X}\hat{X}^T = \text{dot product} = \text{cosine similarity}$$

if we ignore denominator

Thus each element of  $\hat{X}\hat{X}^T$  is cosine similarity between  $(i^{\text{th}} \text{ row})$  of  $X$  and  $(j^{\text{th}} \text{ column})$  of  $X^T$

Now let initial PPMI table be  $X$  & transformed table be  $\hat{X} \hat{X}^T$

$$\therefore X \leq \hat{X} \hat{X}^T \quad \{ \text{similarity between various}$$

+ terms, no words are similar, rows & cols increases thus it is able to learn relation

(Now, taking) short for row rewriting

$$X = U\Sigma V^T$$

$$XX^T = (U\Sigma V^T)(U\Sigma V^T)^T$$

(d)  $U\Sigma V^T \Sigma V\Sigma^T V^T U^T$

$$= U\Sigma \Sigma^T U^T \quad (\Sigma \Sigma^T) \text{ term is}$$

$$(U\Sigma) = \text{long } U\Sigma \text{ on } (\Sigma U)^T \text{ becomes fast one}$$

$m \times n$   $\Rightarrow m \times k$  &  $n \times k$   $\Rightarrow m \times k$   $\times m \times k$   $\Rightarrow m \times k$   $\times m \times k$   $\Rightarrow m \times k$

in between  $U\Sigma$  with  $W_{word}$   $\left\{ \begin{array}{l} \text{dimension} \\ \text{of } W_{word} = m \times k \\ \text{easy to multiply} \end{array} \right.$

$$XX^T = W_{word} W_{word}^T \quad \text{both have same cosine similarity (dot product)}$$

ignoring denominator

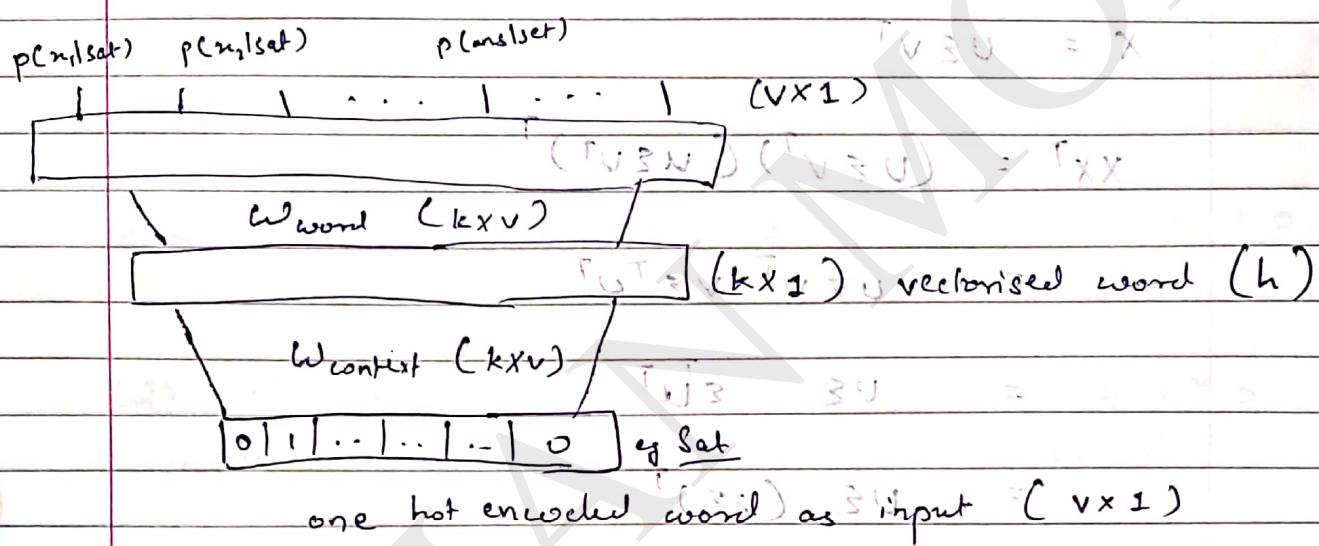
though dimension of  $W_{word}$  is very less than  $X$

old context)  $\rightarrow$   $x$  is old thing having old word  
 $T \times Y$  old

$$W_{\text{context}} = V^T \quad \left( \begin{matrix} n \times k & T \times Y \\ \text{matrix} & \end{matrix} \right) \rightarrow \text{by } \dots$$

part  $\rightarrow$   $W_{\text{context}}$  is the reduction for columns or context words

## \* Continuous Bag of Words (given $n-1$ words predict $n^{\text{th}}$ word)



So, here we have made a Neural Network.

→  $w_{\text{word}}$  &  $w_{\text{context}}$  are weight matrix to be learnt between layers

→ Final activation is Softmax (since we need probability)

(activation function since word and other words)

→ Loss is Cross-Entropy loss.

X Working: this is useful for next words dependent applications

→ When we move from input  $x_i$  to next word

( $\text{W}_{\text{context}}$  is  $X = h$  for the  $i^{\text{th}}$ )

$\rightarrow$   $\text{W}_{\text{context}} \cdot x_i + b = h$

$$\begin{bmatrix} -1 & \{0.5\} & 2 \\ 3 & -1 & -2 \\ -2 & 1.7 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.5 \\ -1 \\ 1.7 \end{bmatrix} = (0)$$

∴ if  $i^{\text{th}}$  index of  $x$  is  $\neq 1$  then

More  $h = i^{\text{th}}$  column of  $\text{W}_{\text{context}}$ .

→ Now,

$$\text{softmax}(\text{Word} \cdot h) = \text{output} (\sim \times 1)$$

Let see any one entry from  $\text{output}$

$$p(\text{con} | \text{sat}) = \frac{e^{(\text{Word} \cdot h)[i]}}{\sum_j e^{(\text{Word} \cdot h)[j]}}$$

$$\text{Word} \cdot h = \text{o/p}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 6 \end{bmatrix} \xrightarrow{j=0} 4$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = 4 \quad | \quad j^{\text{th}} \text{ column of Word} \cdot h$$

Now,  $h = i^{\text{th}} \text{ column of } W_{\text{context}}$  (as marked)

$(j^{\text{th}} \text{ col of } W_{\text{word}}) \times (i^{\text{th}} \text{ col of } W_{\text{context}})$



$$L(\theta) = \left[ e^{-\log \hat{y}_w} \right] - \log (w_i(c))$$

$$h = W_{\text{context}} \cdot n_c = u_c$$

now I is the vector for word  $i$   
 $i^{\text{th}} \text{ col of } W_{\text{context}}$

$$\hat{y}_w = \frac{\exp(u_c \cdot v_w)}{\sum_{w' \in V} \exp(u_c \cdot v_{w'})}$$

$$\exp(u_c \cdot v_w)$$

Now we want to learn  $v_w$

$$\text{Res. } L(\theta) = -\log \hat{y}_w$$

$$= -\log \left( \frac{\exp(u_c \cdot v_w)}{\sum_{w' \in V} \exp(u_c \cdot v_{w'})} \right)$$

$$= -\log(u_c \cdot v_w - \log \sum_{w' \in V} \exp(u_c \cdot v_{w'}))$$

$$\nabla_{v_w} = \frac{\partial}{\partial v_w} (u_c \cdot v_w - \log \sum_{w' \in V} \exp(u_c \cdot v_{w'}))$$

loss funct

wrt  $v_w$

$$L = -u_c(1 - \hat{y}_w)$$

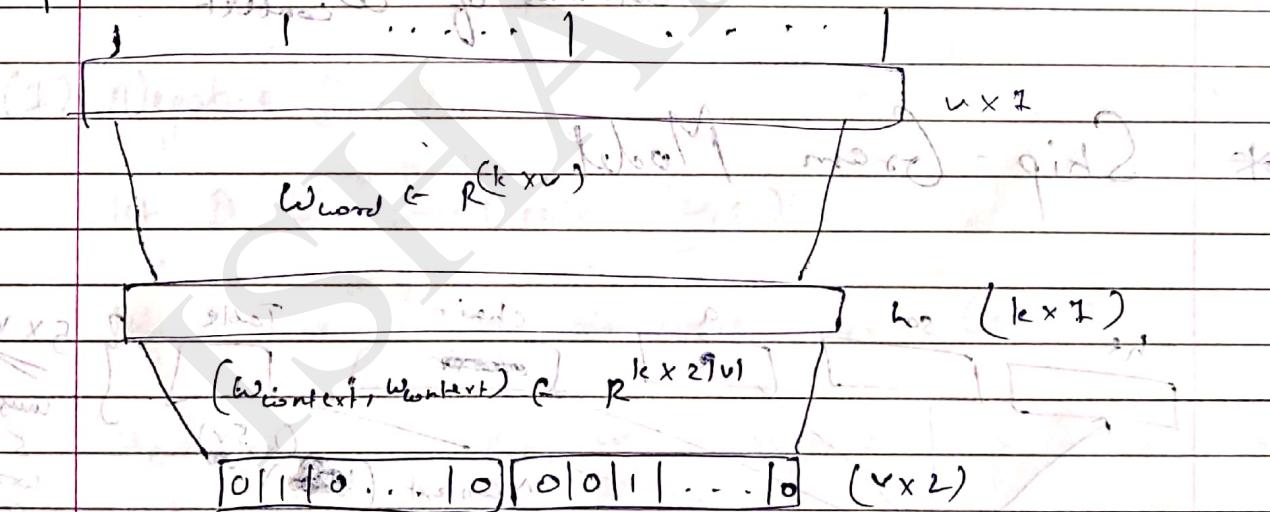
Update rule:

$$\begin{aligned} \Delta V_w &= \eta \nabla V_w - \eta \nabla U_c \\ &= V_w + \eta u_c (1 - y_w) \end{aligned}$$

Note - Current entry of  $V_w$  comes closer to entries in  $U_c$

→ More than one word are given

Let's consider dialogue of going to present  
with a context set up with words



Now,

number of words ( $l \times 2$ )

$$\sum_{i=1}^{l-1} u_{c_i}$$

$h = \sum_{j=1}^l u_{c_j} \rightarrow$  sum of columns in  $W_{context}$  corresponding to one lot of words given

$$\begin{array}{c}
 A+B \\
 \begin{bmatrix} -1 & \{0.5\} & 2 & -1 & 0.5 & \{2\} \\ 3 & \{-1\} & -2 & 3 & -1 & \{-2\} \\ -2 & \{1.7\} & 3 & -2 & 1.7 & \{3\} \end{bmatrix} \xrightarrow{\text{sum along}}
 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \} \text{set} \\
 \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
 W_{\text{context}} \quad W_{\text{word}}
 \end{array}$$

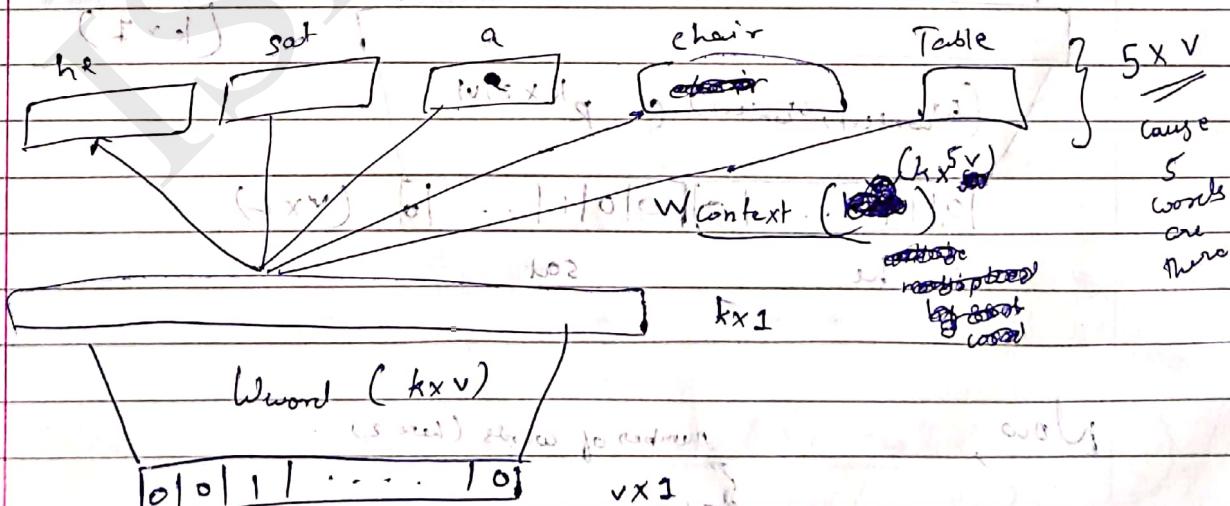
column of words from  $\rightarrow$  for predicting word  $\rightarrow$  Italy

$$= \begin{bmatrix} 2.5 \\ -3 \\ 4.7 \end{bmatrix} = \underline{A+B}$$

Now,

During Backprop, update all values of  $W_{\text{word}}$  & only the participating columns of  $W_{\text{context}}$

## \* Skip-Gram Model.



→ This model allows us to predict context words for a given word, words which can occur on the left or right of the given word.

$$\rightarrow L(\theta) = - \sum_{i=1}^{d+1} \log \hat{y}_{w_i}$$

If there was only 1 context word at output model is same as bag of words = 5  $\times$  5 dimension.

But here 5 words are true random, thus loss function is addition of all loss.

### Problem

- lot of computation at final layer

### Solution

#### (1) Negative Sampling

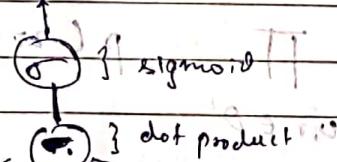
let  $D$  be correct pair  $(w, c)$

let  $D'$  be random incorrect pair  $(w, r)$

$$u_c = c, v_w = w, u_r = r$$

correct context words

$$P(z|w, c)$$



using a red arrow pointing fibering of all words between stuff  $\rightarrow$   
 $p(z=1|w, c) = \sigma(u_c^T v_w)$  the above terms  
 focus on right

$$= \frac{1}{1 + e^{-u_c^T v_w}} \quad (3)$$

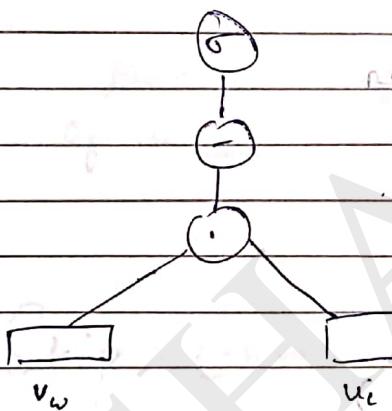
for all  $(w, c) \in D$

the below thoughts is based  $\rightarrow$  and (product of probabilities)  
 maximize  $\prod p(z=1|w, c)$

is maximum over  $\Theta$  over  $(w, c) \in D$  exists and choose  $\theta$  such that  
 good No. of mistakes

~~incorrect context words~~

Now,



now long  $p(z=0|w, c)$  given to let

$$= 1 - \sigma(u_c^T v_w)$$

$$= \frac{1}{1 + e^{u_c^T v_w}}$$

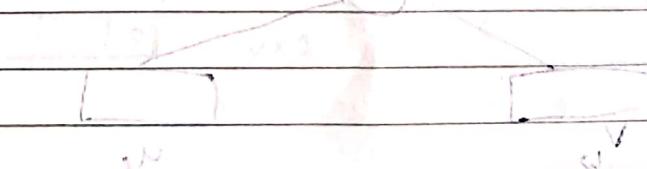
$$= \frac{1}{1 + e^{u_c^T v_w}}$$

$$\frac{1}{1 + e^{u_c^T v_w}}$$

$$\therefore p(z=0|w, c) = \sigma(-u_c^T v_w)$$

$\therefore$  maximize  $\prod p(z=0|w, c)$

$w, c \in D$





Total

$$\underset{(\omega, c) \in D}{\text{maximize}} \prod_{(\omega, c) \in D} p(z=1 | \omega, c) \prod_{(\omega, r) \in D'} p(z=0 | \omega, r)$$

$$\underset{\Theta}{\text{maximize}} \sum_{\omega, c \in D} \frac{1}{1 + e^{-u_c^T v_\omega}} + \sum_{\omega, r \in D'} \frac{1}{1 + e^{u_r^T v_\omega}}$$

$$= \underset{\Theta}{\text{maximized}} \sum_{\omega, c \in D} \log \frac{1}{1 + e^{-u_c^T v_\omega}} + \sum_{\omega, r \in D'} \log \frac{1}{1 + e^{u_r^T v_\omega}}$$

$$\text{Final maximized } \underset{\Theta}{\text{maximized}} \sum_{\omega, c \in D} \log \sigma(u_c^T v_\omega) + \sum_{\omega, r \in D'} \log \sigma(u_r^T v_\omega)$$

bringing  $u_c$  &  $v_\omega$

moving  
 $u_r$  &  $v_\omega$   
away

Number of samples in  $D'$  is  $k$  times more than  $D$

→ Selection of  $(\omega, r)$

The random context word is drawn from a modified unigram distribution

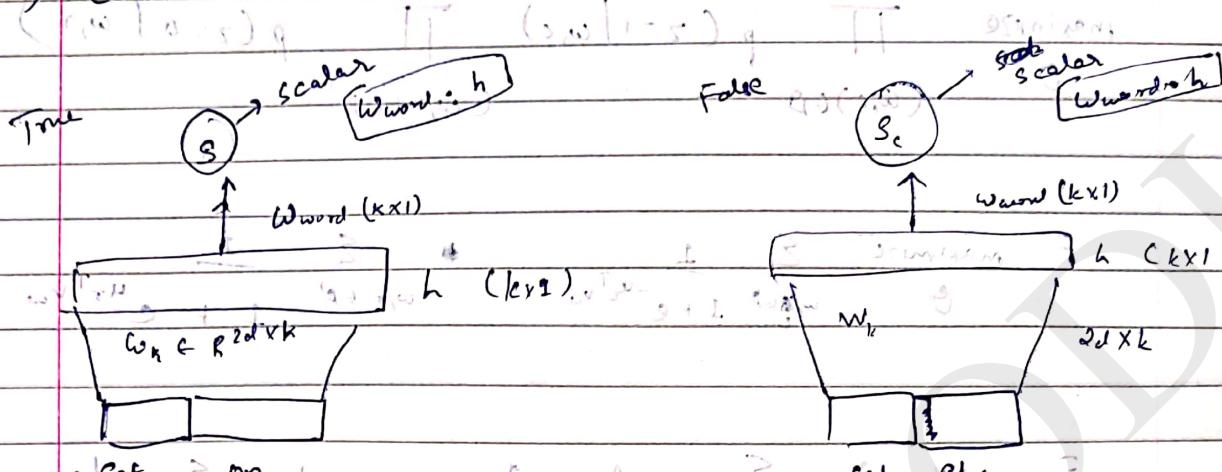
$$\text{(posting, doc, term)} \quad \text{O} = \text{count} \cdot p(r)^{\frac{3}{4}} \quad (\text{count}, r) = \text{each} \quad \text{fi}$$

$$((\text{count}, r) - 1) = \text{each} \quad \text{O} = (\text{count}, r)^{\frac{3}{4}} - 1 \quad \text{fi} \quad \}$$

$$\text{count} \sim \frac{\text{count}(r)}{N}$$

it is better than picking random because frequency of appearance is different.

## (2) Contrastive Estimation



We want to maximize

(large difference)

$s - s_c + \text{margin}$  we want margin between them

Thus we maximize

$$s - (s_c + \text{margin}) \rightarrow \text{optimization}$$

Now, if  $s > s_c + m$  don't do anything

$$\max \min \left[ 0, s - (s_c + m) \right] \quad \left\{ \begin{array}{l} (s, w) \text{ to minimize} \\ \text{loss function} \end{array} \right.$$

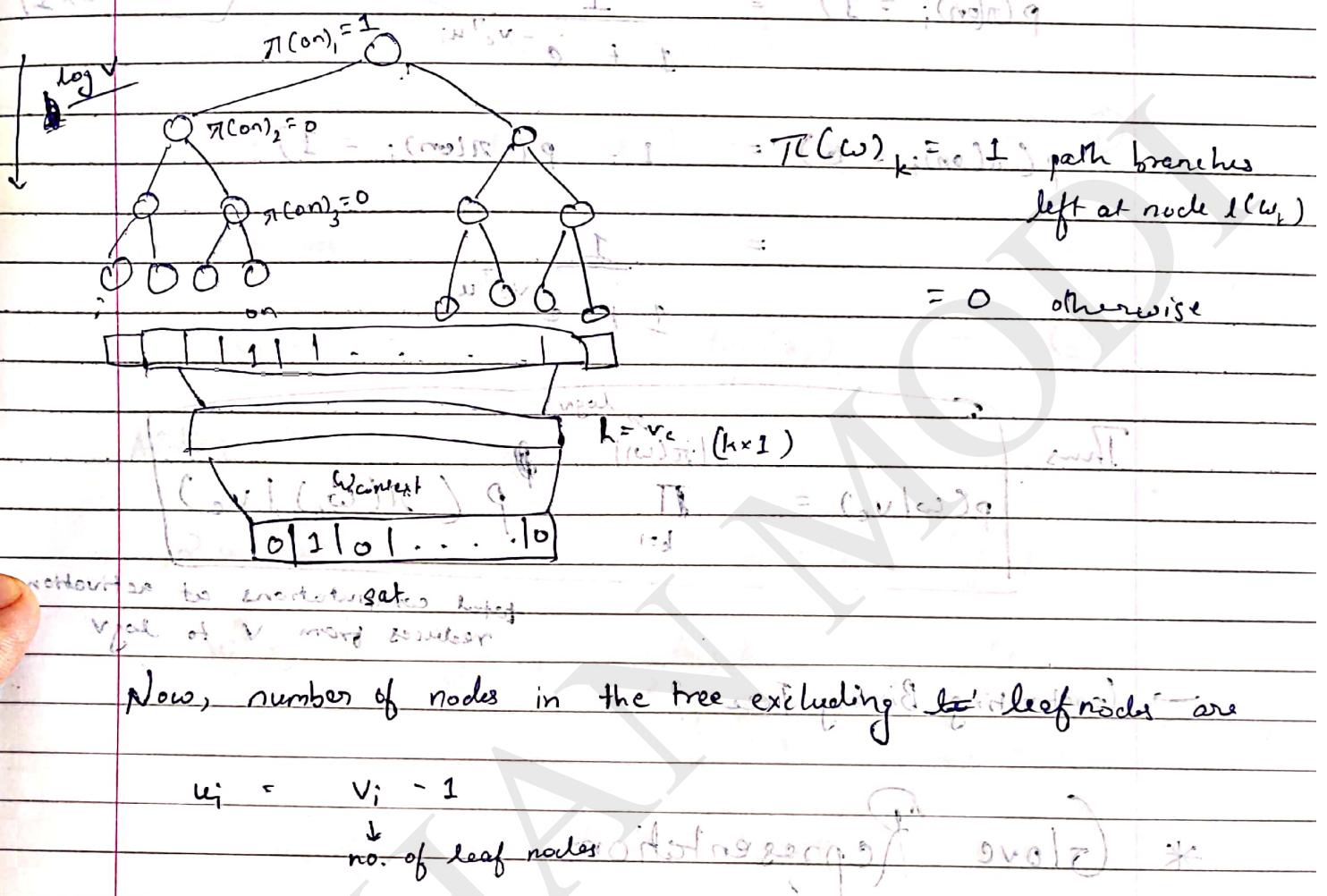
Here  $\min$  network is being fixed, maximize  $s - (s_c + m)$

if  $s - (s_c + m) > 0$ , loss = 0 (don't do anything)

if  $s - (s_c + m) < 0$ , loss =  $s - (s_c + m)$

so passing second max & passing result called as  
loss if it is same message

## (3) Hierarchical Softmax



Now, number of nodes in the tree excluding leaf nodes are

$$u_i = v_i - 1$$

no. of leaf nodes

Thus the parameters for this network will be

$w_{\text{context}} \in u_1, u_2, \dots, u_{v-1}$

vector dimension  $v$   
 all the nodes in tree except leaf nodes  
 every non-leaf node has a vector associated with it

→ Now,

$$p(\text{col } v_c) = \prod_k \pi(w_k | v_c)$$

$$\begin{aligned} p(\text{con} | v_{\text{set}}) &= p(\pi(\text{con})_1 = 1 | v_{\text{set}}) \\ &\quad * p(\pi(\text{con})_2 = 0 | v_{\text{set}}) \\ &\quad * p(\pi(\text{con})_3 = 0 | v_{\text{set}}) \end{aligned}$$

Here

$$p(\pi(\omega_i) = 1) = \frac{1}{1 + e^{-v_c^T u_i}}$$

$v_c^T u_i$  (2)   
 are  $k \times 1$

$$p(\pi(\omega_i) = 0) = 1 - p(\pi(\omega_i) = 1)$$

$$= \frac{1}{1 + e^{v_c^T u_i}}$$

Thus,

$$p(c|v_c) = \prod_{k=1}^{\log N} p(\pi(\omega_k) | v_c)$$

total computations at activation  
reduces from  $V$  to  $\log V$

→ Constructing Binary Tree

## \* Glove Representations

Mixture of SVD based & Predictive based methods

→ Let,

$x_{ij}$  be each entry in the cooccurrence matrix

$$p(j|i) = \underline{x_{ij}} = \underline{x_{ij}}$$

$$\sum_j x_{ij} (\omega_j)_q x_{ij}^T = (\omega_i)_q$$

$$(x_{ij})_q = (x_{ij})_q$$

$$(x_{ij})_q = (x_{ij})_q$$

Now, let's say

$$u_i^T v_j = \log(p(c_{ij})) \quad \text{--- equation 1}$$

$$u_i^T v_j = \log(x_{ij}) - \log x_i \quad \text{--- (1)}$$

Similarly for  $p(c_{ilj})$

$$v_j^T u_i = \log(x_{ij}) - \log(x_{lj}) \quad \text{--- (2)}$$

Adding (1) & (2)

$$2u_i^T v_j = 2\log(x_{ij}) - \log x_i - \log x_j$$

$$u_i^T v_j = \log(x_{ij}) - \frac{1}{2}\log x_i - \frac{1}{2}\log x_j$$

$$\therefore u_i^T v_j + a + b = \log(x_{ij})$$

learnable parameters known

→ loss function

$$\min_{u_i, v_j, a, b} \sum \left( \underbrace{u_i^T v_j + a + b}_{\text{predicted value from model}} - \underbrace{\log x_{ij}}_{\text{value computed from corpus}} \right)$$

## \* Evaluating Word Representations

- Semantic Relatedness

Comparing computer predictions with human perception

- Synonym Detection

Term - derived

Candidates - { imposed, believed, requested, correlated }

$$\text{Synonym} = \arg \max_{v \in C} \cosine(v_{\text{term}}, v_c)$$

- Analogy

- Semantic Analogy

$$V_{\text{apple}} - V_{\text{apple}} + V_{\text{brother}} = V_{\text{grandson}}$$

- Syntactic Analogy

$$V_{\text{works}} = V_{\text{work}} + V_{\text{speaks}} + \dots + V_{\text{speaks}}$$

## \* Relation Between SVD & Word2Vec

- refer / notes / Creek10 / svdrelation.py

$$(W^T \cdot p_k) = d + b + (v^T w)$$