Page No.
Date: / /

n attributes/inputs

x	y	z	
1	1	1	$\vec{x_1}$
0.5	0	0	$\vec{x_2}$
0.25	1	1	$\vec{x_3}$
0.35	1.5	1.5	$\vec{x_4}$
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	$\vec{x_m}$

$X =$

DELUXE
PAGE NO.:
DATE:

* Principle Component Analysis

lets say,

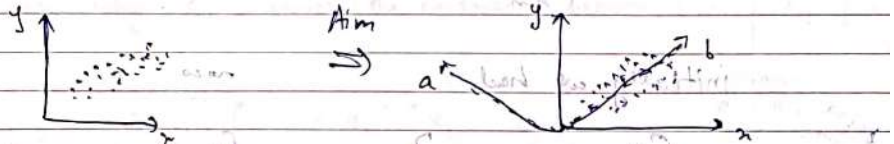
x	y	z
1	1	1
0.5	0	0
0.25	1	1
0.35	1.5	1.5
0.45	1	1
0.57	2	2.1
062	1.1	1
0.735	0.75	0.71
0.72	0.86	0.87

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

Here if in following operation
when $y_i := y_i - \bar{y} \quad \forall i$
then after updating all y_i if we calculate $\bar{y_i}$ then it is $\bar{y_i} = 0$ thus it is called 0 mean.

$$P_{yz} = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(z_i - \bar{z})}{\sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2 \times \sum_{i=1}^{n} (z_i - \bar{z})^2}}$$

Notes



In this figure the data points have notable variance accross 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 not is significantly affected by b and a can be safely ignored
Thus we reduce dimensions from newly formed vectors

Ishan Modi

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

Now, representing x_i using basis P

$$x_i = a_{i,1} p_1 + a_{i,2} p_2 + \dots + a_{i,n} p_n$$

for orthonormal basis

$$a_{ij} = x_i^T \cdot p_j \qquad \left[\leftarrow x_i^T \rightarrow\right] \cdot \begin{bmatrix} \uparrow \\ p_j \\ \downarrow \end{bmatrix}$$

$x_i^T P$ } Dimension = $(1 \times n)$ all x_i

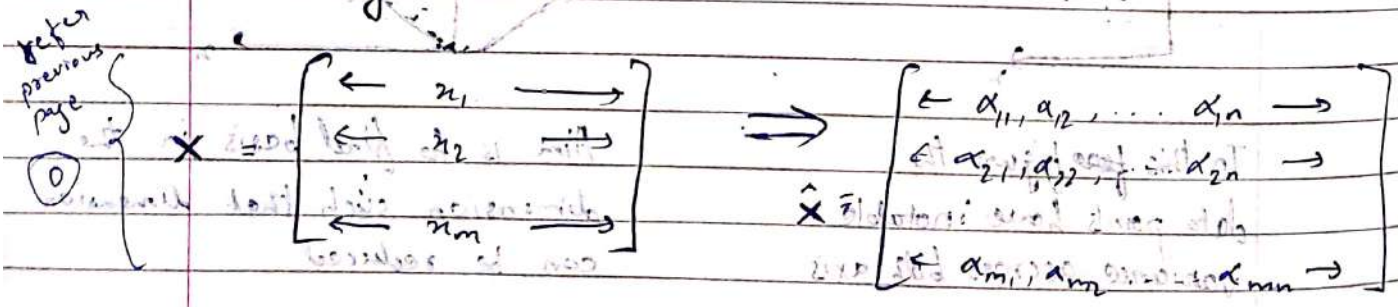
$[1, n] \; [n, n]$

Now for m vectors

$$\begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_m \end{pmatrix} = X \; P \; \} \rightarrow \text{dimension } (m \times n)$$

$(m \times n) \; (n \times n)$

initially we had



* Now here we 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 $\hat{X} = XP$ then the columns of $\hat{X}$ will also have zero mean

proof →

$1^T X = 0$ ... since columns are zero mean

now $\hat{X} = XP \Rightarrow \frac{1}{m} 1^T \hat{X} = \frac{1}{m} 1^T X P$

$= 0P$

$= 0$

Theorem (2)

$X^T X$ is symmetric

proof → $(X^T X)^T = X^T (X^T)^T = X^T X$

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

Explanation

$$C_{ij} = \frac{1}{m} \sum_{k=1}^{m} (X_{ki} - \mu_i)(X_{kj} - \mu_j)$$ → covariance ML Book - (1)

$$= \frac{1}{m} \sum_{k=1}^{m} X_{ki} X_{kj} \quad (\because \mu_i = \mu_j = 0)$$

$= \frac{1}{m} X_i^T X_j$ (one entry)

Similarly matrix is formed

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

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

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

Now, $\hat{X} = XP$

as seen earlier $\frac{1}{m}\hat{X}^T\hat{X}$ is covariance matrix

$$\therefore \frac{1}{m}\hat{X}^T\hat{X} = \frac{1}{m}(XP)^T XP = \frac{1}{m}(P^TX^T)XP$$

(de morgan)

$$= P^T\left(\frac{1}{m}X^TX\right)P$$

$$\frac{1}{m}\hat{X}^T\hat{X} = P^T\Sigma P$$

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 $i \neq j$ it is covariance of i & j column if $i = j$ it is variance

$$\begin{bmatrix} n_1 & 0 & \cdots & 0 \\ 0 & n_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & n_p \end{bmatrix}$$ = Diagonal matrix

Now, as seen earlier

($\frac{1}{m}\hat{X}^T\hat{X}$ is symmetric) — Theorem (2)

Thus $P^T = P^{-1}$ — Deep Learning book (1) ending

(P is collection of eigen vectors of $\frac{1}{m}\hat{X}^T\hat{X}$)

and $P^{-1}\Sigma P$ = Diagonalization ∴ gives diagonal matrix

Ishan Modi

The method of transforming data to new basis with

(1) low covariance – proved

(2) high variance – not proved

is called PCA

→

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

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

$$\hat{x}_i = \sum_{j=1}^{k} \alpha_{ik} p_k$$

So we select this k such that

$$e = \sum_{i=1}^{m} (x_i - \hat{x}_i)^T (x_i - \hat{x}_i)$$

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

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

$$= \sum_{i=1}^{m} (\alpha_{ik+1} p_{k+1} + \ldots + \alpha_{in})^T (\alpha_{ik+1} p_{k+1} + \ldots + \alpha_{in})$$

$$= \sum_{i=1}^{m} \sum_{j=k+1}^{n} \alpha_{ij} \underbrace{p_j^T p_j}_{\text{vectors}} \alpha_{ij} + \sum_{i=1}^{m} \sum_{j=k+1}^{n} \sum_{l=k+1, l \neq k}^{n} \alpha_{ij} p_j^T p_l \alpha_{il}$$

scalars

Ishan Modi

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

$$= \sum_{i=1}^{m} \sum_{j=k+1}^{n} (x_i^T p_j)^2$$

$$= \sum_{i=1}^{m} \sum_{j=k+1}^{n} (p_j^T x_i)(x_i^T p_j)$$

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

Now

let say

$$\sum_{i=1}^{m} \begin{bmatrix} x_{m1} \\ x_{m2} \\ x_{m3} \end{bmatrix} \begin{bmatrix} x_{m1} & x_{m2} & x_{m3} \end{bmatrix}$$

$$= \begin{bmatrix} x_{11}^2 & x_{11}x_{12} & x_{11}x_{13} \\ x_{12}x_{11} & - & - \\ - & - & x_{13}x_{13} \end{bmatrix} + \begin{bmatrix} x_{21}^2 & x_{21}x_{22} & x_{21}x_{23} \\ x_{22}x_{21} & - & - \\ - & - & x_{23}x_{23} \end{bmatrix} + \cdots$$

$$= \begin{bmatrix} x_{11}^2 + x_{21}^2 + \cdots + x_{m1}^2 & x_{11}x_{12} + x_{21}x_{22} + \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & x_{13}^2 + x_{23}^2 + x_{m3}^2 \end{bmatrix}$$

Ishan Modi

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

$$\begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{2,1} & x_{3,1} \\ x_{1,2} & x_{2,2} & x_{3,2} \end{bmatrix} = \begin{bmatrix} x_{1,1}^2 + x_{2,1}^2 + x_{3,1}^2 & \cdots \\ \cdots & \cdots \end{bmatrix}$$

i column, j column — i column, j column

Now

$$\sum_{j=k+1}^{n} p_j^T m C p_j \quad \text{where} \quad \left[\frac{1}{m} \sum_{i=1}^{m} x_i x_i^T = \frac{X^T X}{m} = C \right]$$

$$e = \sum_{j=k+1}^{n} p_j^T m C p_j$$

$$\min \sum_{j=k+1}^{n} p_j^T m C p_j \quad \text{such that} \quad p_j^T p_j = 1$$

$p_{k+1}, p_{k+2}, \ldots, p_n$ → theorem (Deep learning book ①)

p_j are $n-k$ smallest eigen vectors

→ How to prove high variance — (2) condition as mentioned earlier

$$\bar{X}_i = X p_i$$

variance

$$\frac{(\bar{x}_i - \mu_i)^T (\bar{x}_i - \mu_i)}{m} = \frac{\bar{x}_i^T \bar{x}_i}{m} \quad (\because \mu_i = 0)$$

zero mean

Ishan Modi

$$= \frac{1}{m} p_i^T (X^T X p_i) \quad \text{(} Ax = \lambda x \text{)}$$

$$= \frac{1}{m} p_i^T (\lambda_i p_i)$$

$$= \frac{\lambda_i}{m} \underbrace{p_i^T p_i}_{1}$$

$$= \frac{\lambda_i}{m} \} \text{ eigen values}$$

If we remove the $n-k$ small eigen values then we are left we k large eigen values & large eigen value means high variance.

→ 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

matrix A transforms a vector from n dimension to m dimension

ie $A \; v' = v''$

$m \times n \quad n \times 1 \quad m \times 1$

Now let us assume that $v_{(n\times 1)}$ is basis of vector space R^n and $u_{(m\times 1)}$ is basis of (vector space : R^m

then,

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

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

$\therefore$

$$Ax = \sum_{i=1}^{k} \alpha_i A v_i \Rightarrow (n\times 1) \text{ vectors}$$

$$= \sum_{i=1}^{k} \alpha_i \sigma_i u_i \Rightarrow (m\times 1) \text{ vectors}$$

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

Since v_i (n-dim) & u_i (m-dim) } common term k is used in summation

→ Thus we can write

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

($n \times 1$ vectors) ($m \times 1$ vectors) diagonal matrix

Now these are $(n \times k)$ & $(m \times k)$ vectors 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

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

Thus

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

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

→ Now we find U & V

let say V, U & Σ exist

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

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

$$A^T A = V_i^T \Sigma^2 V$$

Similarly $A A^T = U^T \Sigma^L U$

} ∴ U & V are eigen vectors of AA^T & $A^T A$ respectively

$$[A]_{m \times n} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \cdots & u_k \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1 & \rightarrow \\ \leftarrow & v_2 & \rightarrow \\ & \vdots & \\ \leftarrow & v_k & \rightarrow \end{bmatrix}_{k \times n}$$

$$= \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

Ishan Modi

complexity in storage for $A = m \times n$

improved $= (m + n + 1)\, k$

(u_i, v_i, σ_i)

SVD

$\min \| A - \hat{A} \|_F^2$ is given by

largest / first k dominant eigen vectors

$\hat{A} = U_{\cdot,k}\, \Sigma_{k,k}\, V^T_{k,\cdot}$ } (first k) columns of U & first k rows of V

Example of PCA & SVD

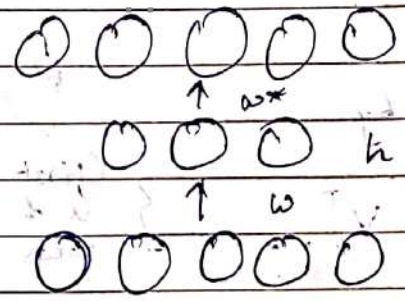
- These images are called eigenfaces and form a basis for representing any face in our database
- In other words, we can now represent a given image (face) as a linear combination of these eigen faces
- In practice, we just need to store $p_1, p_2, \cdots, p_k$ (one time storage)
- Then for each image i we just need to store the scalar values $\alpha_{i1}, \alpha_{i2}, \cdots, \alpha_{ik}$
- This significantly reduces the storage cost without much l quality

$$\sum_{i=1}^{16} \alpha_{1i} p_i$$

Ishan Modi

WEEK 7

* Auto Encoders



$$h = g(Wx_i + b)$$

$d \times n \quad n \times 1 \quad d \times 1$

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

$n \times d \quad d \times 1 \quad n \times 1$

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

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

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

→ If the x_i nodes have real numbers then

$$\hat{x}_i = W^* h + c$$

If x_i nodes have binary numbers 0, 1 then

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

f in different cases

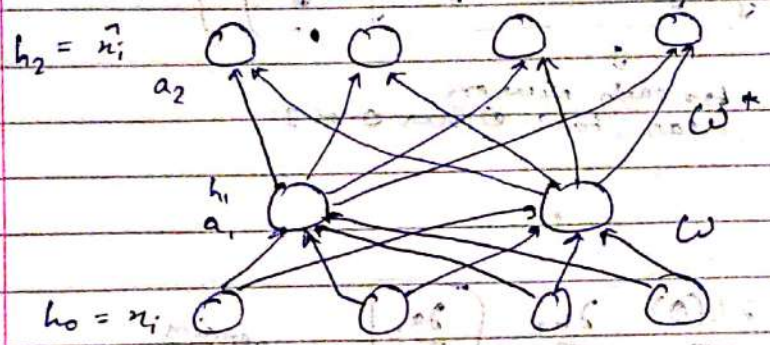
→ Let say loss function is sum of squared error

i.e. $\min_{\omega, \omega^*, c, b} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$

$\Rightarrow \min_{\omega, \omega^*, c, b} \frac{1}{m} \sum_{i=1}^{m} (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)$

→ Backprop (for Real x_i)

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



Now earlier we did backprop with a cross entropy loss

- $\frac{\partial L(\theta)}{\partial \omega^*} = \frac{\partial L(\theta)}{\partial h_2} \left[\frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial \omega^*} \right]$ → remains same
  ↓ changes
- $\frac{\partial L(\theta)}{\partial \omega} = \frac{\partial L(\theta)}{\partial h_2} \left[\frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial \omega} \right]$ → remains same

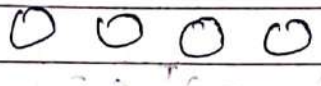
Now

$\frac{\partial L(\theta)}{\partial h_2} = \frac{\partial L(\theta)}{\partial \hat{x}_i} = \frac{\partial \sum (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)}{\partial \hat{x}_i}$ → scalar / vector

$= 2 (\hat{x}_i - x_i)$

refer Machine learning book (1)

→ Back Propagation (for binary x_i)



1 0 1 1 } we will use binary cross entropy loss

$$\min\left(\sum_{i=1}^{m} \left(-\sum_{j=1}^{4} \underbrace{x_{ij} \log \hat{x}_{ij} + (1-x_{ij}) \log (1-\hat{x}_{ij})}_{\text{for each neuron can take either 0 or 1}} \right) \right)$$

m: no. of data entries; 4: no. of neurons/inputs

Now

$$\frac{\partial L(\theta)}{\partial w^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \boxed{\frac{\partial a_2}{\partial w^*}} \rightarrow \text{remains same}$$

$$\frac{\partial L(\theta)}{\partial w'} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \boxed{\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w'}} \rightarrow \text{remain same}$$

Now, loss for 1 data entry

$$-\sum_{j=1}^{4} \left[x_{ij} \log \hat{x}_{ij} + (1-x_{ij}) \log (1-\hat{x}_{ij}) \right]$$

$$\frac{\partial L(\theta)}{\partial h_{ij}} \cdot \frac{\partial L(\theta)}{\partial \hat{x}_{ij}} = -\frac{x_{ij}}{\hat{x}_{ij}} + \frac{1-x_{ij}}{1-\hat{x}_{ij}}$$

$$\frac{\partial L(\theta)}{\partial a_{2j}} \cdot \frac{\partial h_{ij}}{\partial a_{2j}} = \sigma(a_{2j})(1-\sigma(a_{2j}))$$

Ishan Modi

* Link between PCA & Autoencoders

→ Conditions

(1) Linear decoder → f is a linear function (given)

(2) Linear encoder → g is a linear function (prove) (in order to minimise loss linear encoder is required)

(3) Loss function → sum of squared errors (given)

(4) Generalization / Normalization → $\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)$ (zero mean)

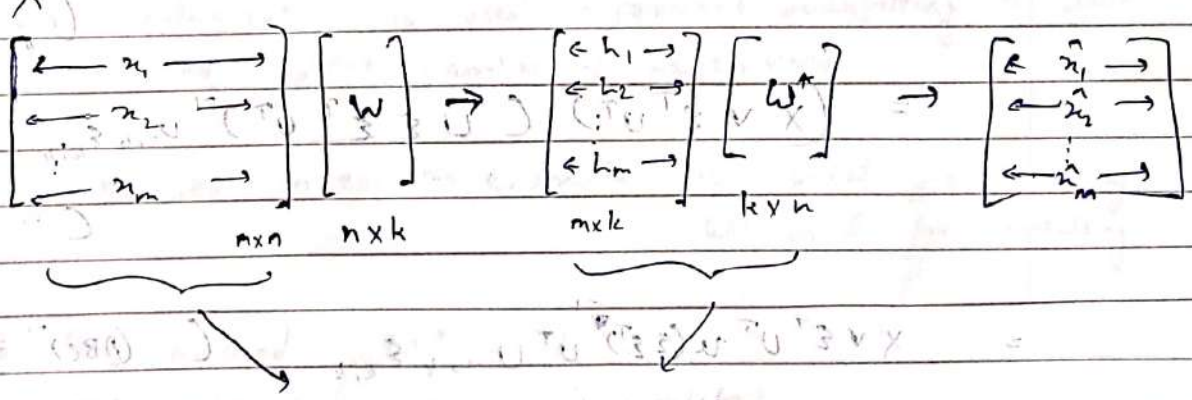
→ $x_{ij} = \frac{1}{\sqrt{m}}$ (zero mean)

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

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

→ Now loss function is

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



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

Ishan Modi

Now from S.V.D. Theorem

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

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

$$\left(\because HW^* = U_{:,\leq k} \; \Sigma_{k,k} \; V^T_{:,\leq k} \right)$$

Now we can say,

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

$$W^* = V^T_{:,k}$$

→ Now given all statements we try to prove H is linear encoding of X

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

$$= (X V \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1} U_{:,k} \; \Sigma_{k,k}$$

$$(\because X = U \Sigma V^T)$$

$$= (X V \Sigma^T U^T)(U \Sigma \Sigma^T U^T)^{-1} U_{:,k} \; \Sigma_{k,k}$$

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

$$= X V \Sigma^T U^T U (\Sigma \Sigma^T)^{-1} U^T U_{:,k} \; \Sigma_{k,k} \qquad (\because (ABC)^{-1} = C^{-1} B^{-1} A^{-1})$$

$$= X V \Sigma^T (\Sigma \Sigma^T)^{-1} U^T U_{:,k} \; \Sigma_{k,k}$$

$$= X V \Sigma^T \Sigma^{T-1} \Sigma^{-1} U^T U_{:,k} \; \Sigma_{k,k} \qquad (\because (AB)^{-1} = B^{-1} A^{-1})$$

Ishan Modi

$= X V \Sigma^{-1} I_{:,k} \Sigma_{k,k}$ $\quad (\because U^T U_{:,k} = I_{:,k})$

$= X V I_{:,k}$ $\quad (\Sigma^{-1} I_{:,k} = \Sigma^{-1}_{k,k})$

$= X V_{:,k}$

linear combination of X & $V_{:,k}$

↓

is weight matrix W

* Regularization in Autoencoders

(Overfitting can happen in both undercomplete & overcomplete Autoencoders)

Two types

(1) $\left(\text{Loss Function} + \frac{\lambda \|W\|^2}{2}\right)$ } updating the weights using L2 norm flatten all W matrix & put them in a vector w

$\frac{\partial L}{\partial w}$ = derivative of loss function + λw

(2) Weights tieing

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

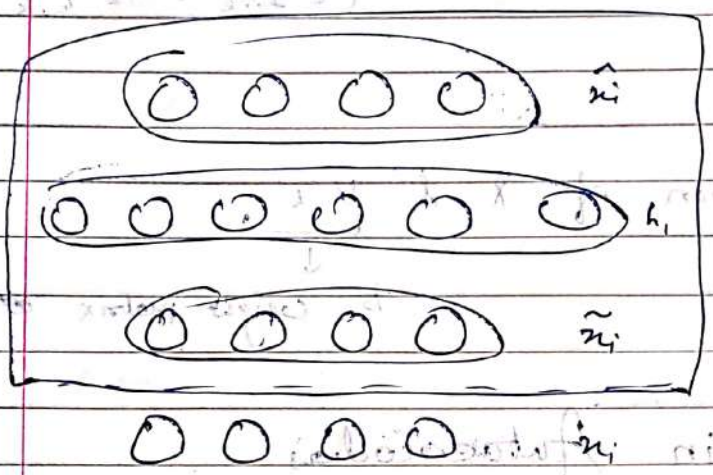
∴ As seen in earlier example W used for encoding

W^* used for decoding

in this method $W^* = W^T$.

At each instance only W is updated.

* Denoising Autoencoders (Type of Regularization)



Here, x_i is original input which is corrupted using a function & all ones are converted to zeros & visa versa in $\tilde{x}_i$ is formed from x_i

Now,

$\hat{x}_i$ is obtained & loss function here will be

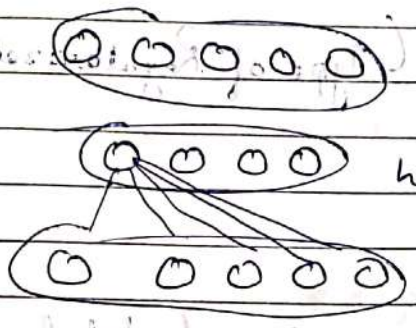
$$\frac{1}{m} \sum_{i=1}^{\tilde{m}} \sum_{j=1}^{\hat{m}} (\hat{x}_i - x_i)^2$$

& not $\left(\frac{1}{m} \Sigma \Sigma (\hat{x}_i - \tilde{x}_i)^2 \right)$

thus since it is over complete auto encoder the $\tilde{x}_i$ won't go in $\hat{x}_i$ straight away because $\hat{x}_i$ is compared with x_i

By reconstructing over corrupted $\tilde{x}_i$ it also learns some important features of x_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.



x_i here is denoised data

Each neuron in $h_j = \sigma(w_j^T x_i)$

Now each neuron fires when $(w_j^T x_i)$ is maximum because at this point sigmoid is 1.

$\therefore \max_{x_i} \{ w_j^T x_i \}$ such that $\underbrace{\|x_i\|^2 = x_i^T x_i = 1}_{x_i \text{ is normalized}}$

$w_j^T . x_i$ } dot product is maximum when $\theta = 0$ same direction

$\therefore$

$$x_i = \frac{w_j}{\sqrt{w_j^T w_j}}$$

Each input here helps respective neuron to fire

$$\underbrace{\frac{w_1}{\sqrt{w_1^T w_1}}}_{x_1}, \underbrace{\frac{w_2}{\sqrt{w_2^T w_2}}}_{x_2}, \ldots, \underbrace{\frac{w_n}{\sqrt{w_n^T w_n}}}_{x_n}$$

$\underbrace{\qquad\qquad}$ set of all inputs

This experiment is done after training

Ishan Modi

* Sparse Autoencoders (Type of Regularization)

For every neuron

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} g(\omega_j^T x_i) \quad \text{is calculated}$$

↓

This gives us at an average when the neuron will fire. If ρ_j 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.

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

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

∴ we use

$$\Omega(\theta) = \sum_{j=1}^{\text{all neurons of a layer}} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_j)}$$

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

↳ will be minimum if $\rho = \hat{\rho}_j$

→ Back Propagation

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

$\frac{\partial \Omega(\theta)}{\partial \omega}$

$$\frac{\partial \Omega(\theta)}{\partial \omega} = \frac{\partial \Omega(\theta)}{\partial \hat{p}_{\cdot i}} \times \frac{\partial \hat{p}_{\cdot i}}{\partial \omega}$$

Now,

$$\frac{\partial \Omega(\theta)}{\partial \hat{p}_i} = \frac{\partial \left(p \log p - p \log \hat{p}_i + (1-p) \log (1-p) - (1-p) \log (1-\hat{p}_i) \right)}{\partial \hat{p}_i}$$

$$= \begin{pmatrix} -\frac{p}{\hat{p}_1} - \frac{(1-p)}{(1-\hat{p}_1)} \\ \vdots \end{pmatrix}$$ vector with each element representing a neuron

(1)

Now

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

scalar

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

$$\frac{\partial \hat{p}_i}{\partial \omega_{i,\cdot}} = \frac{\partial \frac{1}{m} \sum_{i=1}^{m} \left[g(\omega_{i,\cdot} x_i + b_i) \right]}{\partial \omega_{i,j}}$$

vector

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

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

matrix

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

(× : dot product)

→ Final Derivative

$$\frac{\partial \Omega(\theta)}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} \left(\left[\underbrace{\begin{pmatrix} -\rho & -(1-\rho) \\ \hat{\rho}_1 & (1-\hat{\rho}_1) \end{pmatrix}}_{\text{vector from ①}} \odot \underbrace{g'(wx_i + b)}_{\text{vector}} \right] \times \underbrace{x_i^T}_{\text{vector}} \right)$$

(× : dot product)

* Contractive Autoencoder

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

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

$$\therefore \quad J = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & & & \frac{\partial h_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial h_k}{\partial x_1} & & & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

x has dimension n (input layer)
h has dimension k (hidden layer)

$$\Omega(\theta) = \sum_{j=1}^{n} \sum_{i=1}^{k} \left(\frac{\partial h_i}{\partial x_j} \right)^2$$

$\min (L(\theta) + \Omega(\theta))$ → aim

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

$\frac{\partial h_\theta}{\partial x_i} = 0$ } means: don't capture variations in data

$\therefore$ $L(\theta)$ = capture important variations in data
(how h_θ changes with respect to x_i)

$\Omega(\theta)$ = don't capture variations in data
(how h_θ doesn't change with x_i) *

Trade off - capture only very important variations in data

DELUXE
PAGE NO.:
DATE:

WEEK 8

* Bias ... (Simple models have high bias)

→ Formula

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

($\hat{f}(x)$: predicted, $f(x)$: original)

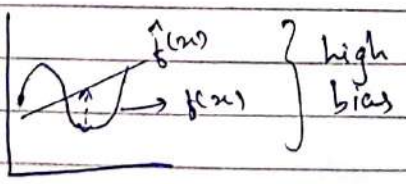
* Variance ... (Complex Models have high variance)

→ Formula

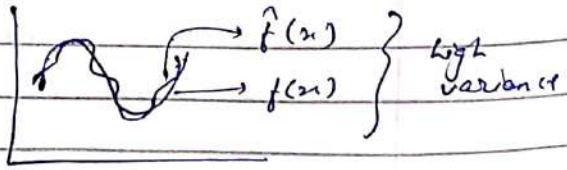
$$\text{Variance}(\hat{f}(x)) = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

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

* Bias - Variance Trade off



Simple model



Complex model

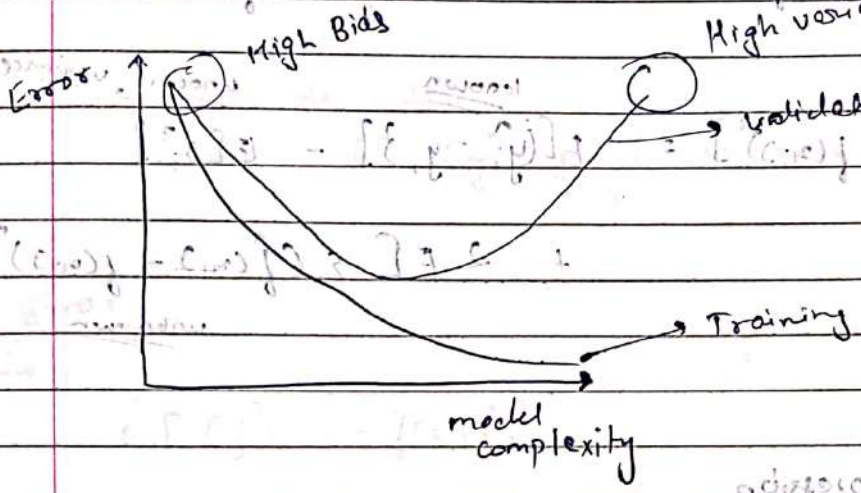
Ishan Modi

It can be proved

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

mean squared error

→



If there are n training & m testing point :

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

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

→ Mathematically

We know training data (ie y_i) corresponding x_i

∴ we can calculate $\hat{y}_i$

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

∴ $E[(y_i - y_i)^2] = E[(\hat{f}(x_i) - f(x_i) - \varepsilon_i)^2]$

$\because y_i = f(x_i) + \varepsilon_i$

Here we assume that we know $\hat{f}(x_i)$ but not $f(x_i)$

Ishan Modi

$$= E\left[(\hat{f}(x_i) - f(x_i))^2 - 2\varepsilon_i(\hat{f}(x_i) - f(x_i)) + \varepsilon_i^2\right]$$

$$= E\left[(\hat{f}(x_i) - f(x_i))^2\right] - 2E\left[\varepsilon_i(\hat{f}(x_i) - f(x_i))\right] + E[\varepsilon_i^2]$$

known — known — variance

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

unknown

Test Error

→ Now, in above expression

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

$$\frac{1}{m}\sum_{i=n}^{n+m}(\hat{y}_i - y_i)^2$$ for test data

this is not exact (E) but it is value very close to Expectation

Similarly,

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

$y_i = f(x_i) + \varepsilon_i$ for test data (independent)

$\hat{f}(x_i)$ for test data is calculated using parameters from training data which depend on ε_i of training data and not ε_i of test data

- & $2E\left[\varepsilon_i(\hat{f}(x_i) - f(x_i))\right]$

Covariance $(\varepsilon_i, (\hat{f}(x_i) - f(x_i)))$

$\because Cov(X, Y) = E[(X - \bar{X})(Y - \bar{Y})]$

zero mean

$= E[XY]$

Now, since ε_i & $(\hat{f}(x_i) - f(x_i))$ are independent

$\therefore (\varepsilon_i) \cdot (\hat{f}(x_i) - f(x_i)) = 0$ & $\varepsilon_i \perp (\hat{f}(x_i) - f(x_i))$

∴ covariance = 0

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

$\frac{1}{m} \sum_{i=n}^{n+m} (\hat{y}_i - y_i)^2 + \frac{1}{m} \sum_{i=n}^{m+n} \varepsilon_i^2$

small constant

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

Train Error

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

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

covariance $(\varepsilon_i, (\hat{f}(x_i) - f(x_i))$

Here ε_i is dependent on training data

$\therefore$ 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

Ishan Modi

→ Earlier

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

Now,

using Stein's lemma

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

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

$\frac{\partial \hat{f}(x_i)}{\partial y_i}$ 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

+

Ω(model complexity)

→ This tries minimize

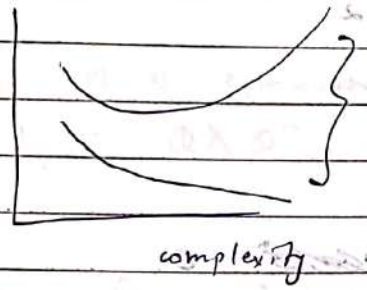
$$\min_{w.r.t\ \theta} L_{train}(\theta) + \Omega(\theta) = L(\theta)$$

concept of regularization

instead of train errors

Ishan Modi

→ Now we still don't know this quantity ie $\Omega(\theta)$ thus we use any function as $\Omega(\theta)$ such that it is high for complex models & low for simple models.



* L2 Regularization

$$\bar{L}(\bar{w}) = L(w) + \frac{1}{2}\alpha \|w\|^2$$

$$\nabla \bar{L}(\bar{w}) = \nabla L(w) + \alpha w$$

↳ update

$$w_{t+1} = w_t - \eta(\nabla L(w_t) + \alpha w_t)$$

→ Now

Let w^* be such that $\underbrace{\nabla L(w^*)}_{\text{Loss without regularization}} = 0$ — (1)

& $h = w - w^*$ } where w is a point in neighbourhood of w^*

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

from Taylor series upto 2nd terms (2nd derivative)

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

$$L(w) = L(w^*) + (w - w^*)^T \nabla L(w^*) - (\because \nabla L(w^*) = 0 \text{ from } ①)$$

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

differentiating wrt w,

$$\nabla L(w) = \nabla L(w^*) + 0 - (\because \nabla L(w^*) = 0 \text{ from } ①)$$

$$+ (w - w^*) H$$

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

Now,

$$\nabla \tilde{L}(\tilde{w}) = \nabla L(w) + \alpha \tilde{w} \quad \} \text{ regularized loss function } \tilde{L}(w)$$

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

→

Let $\tilde{w}$ be such that $\tilde{L}(\tilde{w}) = 0$ ←

regularized loss

$$\therefore H(\tilde{w} - w^*) + \alpha \tilde{w} = 0$$

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

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

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

→ Assume H is symmetric matrix

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

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

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

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

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

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

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

↓

Diagonal matrix D

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

- Q ↓ rotates ω^*
- D ↓ scales ω^*
- Q^T ↓ rotates ω^*

Here, $\tilde{\omega}$ & ω^* are weight vectors

Ishan Modi

$$D = \begin{bmatrix} \frac{\lambda_1}{\lambda_1+\alpha} & & & \\ & \frac{\lambda_2}{\lambda_2+\alpha} & & \\ & & \ddots & \\ & & & \frac{\lambda_n}{\lambda_n+\alpha} \end{bmatrix}$$

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

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

Note

if $\lambda_i >> \alpha$, $\frac{\lambda_i}{\lambda_i+\alpha} = 1$

if $\lambda_i << \alpha$, $\frac{\lambda_i}{\lambda_i+\alpha} \approx 0$

* Dataset Augmentation

Increasing the amount of data

→ Rotating images
→ Shifting horizontaly / vertically
→ Blurring, adding noise etc

* Parameter Sharing & Tying
used in CNN & in Encoder/Decoders

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

$[\varepsilon \sim \mathcal{N}(0, \sigma^2)]$ Gaussian distribution
(zero mean, unit variance)
or (zero mean, multivariate)

$\tilde{x}_i = x_i + \varepsilon_i$

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

$\tilde{y} = \sum_{i=1}^{n} w_i \tilde{x}_i = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i \varepsilon_i$

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

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

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

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

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

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

Ishan Modi

Here in second term

$(\hat{y} - y)$ & ε_i are independent

($\hat{y}$ - original values when noise was not added ; y)

thus covariance $(\hat{y} - y) \sum \omega_i \varepsilon_i = 0$

& in third term

$$\left(\sum \omega_i \varepsilon_i\right)^2 \Rightarrow (a + b + c)^2$$

$$(\varepsilon_i \varepsilon_j)^2 \approx 0 \quad ; \text{ independent}$$

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

$$\therefore \left(\sum \omega_i \varepsilon_i\right)^2 = \sum \omega_i^2 \varepsilon_i^2$$

$$= E\left[(\hat{y} - y)^2\right] + 0 + E\left[\sum_{i=1}^{n} \omega_i^2 \varepsilon_i^2\right]$$

Here $E\left[\sum_{i=1}^{n} \omega_i^2 \varepsilon_i^2\right] = \left(\sum_{i=1}^{n} \omega_i^2\right) * \left(\sum_{i=1}^{n} E[\varepsilon_i^2]\right)$, where $E[\varepsilon_i^2] = \sigma^2$

since ω_i is not random variable

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

↓ same as L_2 regularization

This is used in overcomplete auto encoders for encoding, in order to reduce the complexity as number of dimensions increase.

Input

Ishan Modi

* Adding Noise to Data (Output)

At the output we have

$$\sum_{i=1}^{n} p_i \log q_i$$

p_i → original (correct), q_i → predicted

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

p_i = { 0 , 0 , 1 , 0 , ... } (correct) Assuming there are q elements

p_i = $\left\{ \frac{\varepsilon}{q}, \frac{\varepsilon}{q}, 1-\varepsilon, \frac{\varepsilon}{q}, \ldots \right\}$ if ε = noise

q_i = remains as it is

* Early Stopping

Patience parameter p

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

→ In gradient Descent

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$= w_0 - \eta \sum_{i=1}^{t} \nabla w_i$$

∇w_i → maximum out of them

$$w_{t+1} \geq w_0 - \eta t \Gamma$$

Thus t controls how far w_t is from w_0

→ As seen earlier

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

$$w_t = w_{t-1} + \eta \nabla L(w_{t-1})$$

$$= w_{t-1} + \eta H(w_{t-1} - w^*)$$

$$= (I + \eta H) w_{t-1} - \eta H w^*$$

$$\therefore w_t = (I + \eta H) w_{t-1} - \eta H w^*$$

- Using EVD, $H = Q \Lambda Q^T$

$$w_t = (I + \eta Q \Lambda Q^T) w_{t-1} - \eta Q \Lambda Q^T w^*$$

- If $w_0 = 0$

$$w_t = Q [I - (I - \varepsilon \Lambda)^t] Q^T w^* \quad \text{--- (1)}$$

- Thus ∴ early stopping is similar to L_2 regularization

$$\tilde{w} = Q [I - (\Lambda + \alpha I)^{-1} \alpha] Q^T w^* \quad \text{--- (2)}$$

- $w_t = \tilde{w}$ for following condition

$$(I - \varepsilon \Lambda)^t = (\Lambda + \alpha I)^{-1} \alpha$$

Ishan Modi

DELUXE
PAGE NO.:
DATE:

* 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 (Here the number of models is k)

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

Expected mean squared error (Expectation over the inputs)

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

$$= \frac{1}{k^2} E\left[\sum_{i=i} \sum_{j=j} \varepsilon_i \varepsilon_j + \sum_i \sum_{j \neq i} \varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k^2} E\left[\sum_{i=1}^{k} (\varepsilon_i^2) + \sum_i \sum_{j \neq i} \varepsilon_i \varepsilon_j\right]$$

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

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

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

$\frac{V}{k}$: variance

$\left(\frac{k-1}{k}\right)C$: covariance

Ishan Modi

Note

If errors are independent $C = 0$

$$mse = \frac{V}{k}$$

If errors are correlated $C = V$

$$mse = V$$

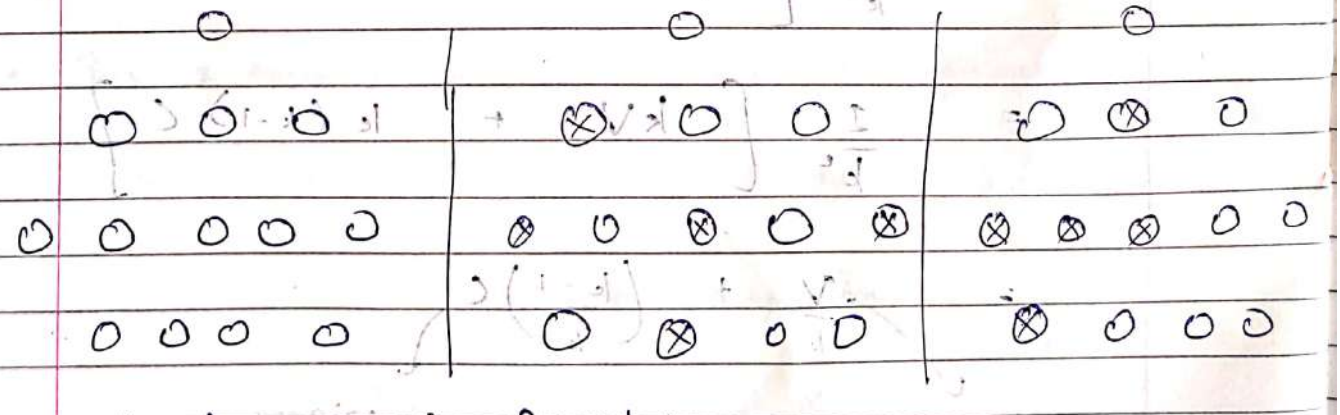
Thus using bagging helps reduce error.

* Dropout

Let W be the weight matrix

→ Dropout means to eliminate neurons in layers with probability $\frac{1}{2}$ form completely new model train for the one epoch on that model. Update W.

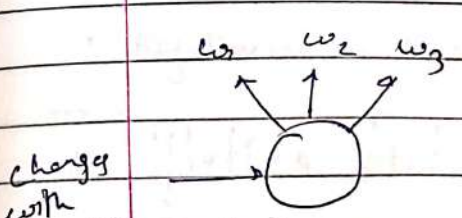
→ Now eliminate from original to get new model 1 use updated W and train for 1 epoch



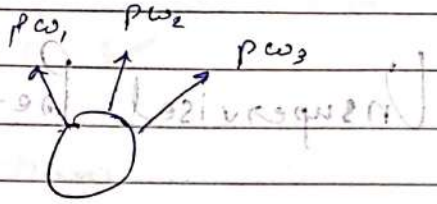
Original | Dropout 1 | Dropout 2

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

Thus neurons will be robust



train time



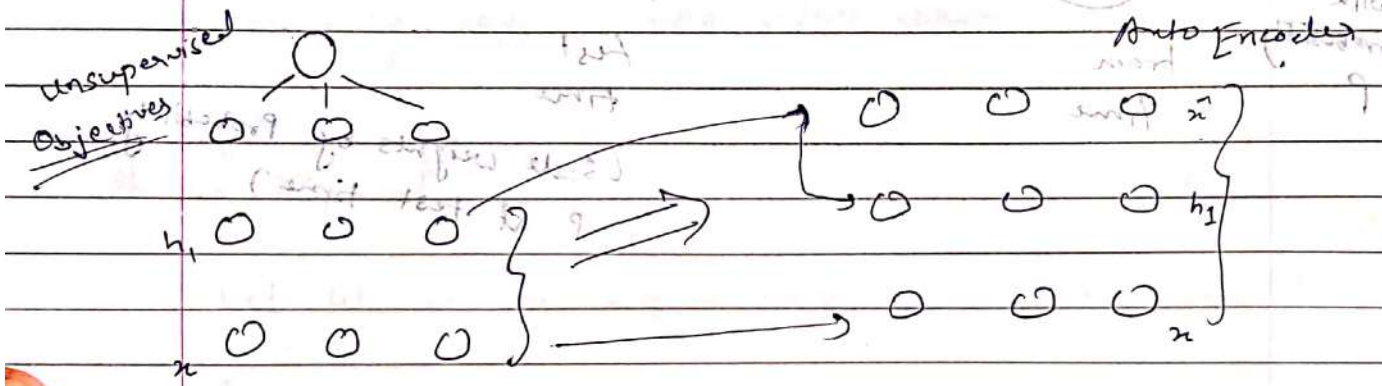
test time

(Scale weights by Probability p at test time)

WEEK 9

→ Gradient at a particular layer depend 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

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

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

- Helps converge Faster
- Optimization Problem (train Data)
- Regularization Problem (test Data)

* Better Activation Functions

→ Sigmoid

Disadvantages

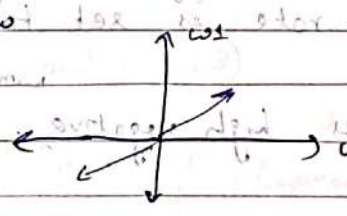
- Saturated neurons cause gradient to vanish

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

Thus derivative $\sigma(x) \times (1 - \sigma(x))$ which is used in gradient becomes (zero). Thus gradient becomes (zero)

- Not zero centered (from 0 to 1)

the movement of gradient becomes restricted and thus convergence is slow.



(all the gradients at a particular layer either both are +ve or both are -ve)

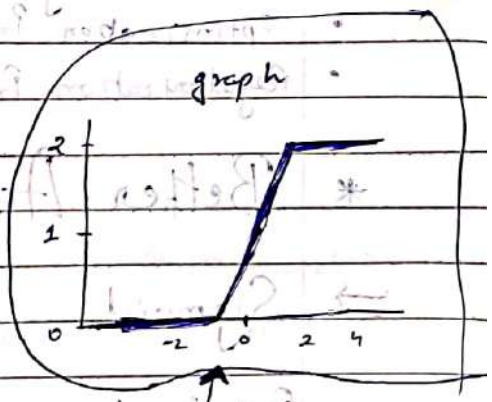
- Computationally expensive ($\exp(x)$)

Ishan Modi

→ Tanh.

Disadvantage

- Gradient vanishes at saturation
- Computationally expensive
- Solves
- zero centered.



→ ReLU

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

$$f(x) = \underbrace{\max(0, x+1)}_{\text{ReLU}} - \underbrace{\max(0, x-1)}_{\text{ReLU}}$$ } equivalent to sigmoid

Mostly used in CNNs

- Only problem (Dead Neuron problem)

$$\frac{\partial \text{ReLU}}{\partial x} = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$ } derivatives

In practice a large fraction of ReLU units can die if the learning rate is set too high

If bias becomes high negative number then

$$w_1 x_1 + w_2 x_2 + b < 0$$

$$\therefore \frac{\partial \text{ReLU}}{\partial x} = 0$$ & weights are not updated

→ Leaky ReLU (solution to dead neuron)

$f(x) = \max(0.01x, x)$

→ Parametric ReLU

$f(x) = \max(\alpha x, x)$

↓

trainable parameter not hyper parameter

→ Exponential Linear Unit

$f(x) = x \quad \text{if } x > 0$

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

→ Maxout Neuron

$\max(w_1^T x + b_1, w_2^T x + b_2)$ — (1)

variant of ReLU $\max(0, x)$ — (2)

$w_1 = 0 \quad b_1 = 0$

$w_2 = 1 \quad b_2 = 0$

} Representing (1) as (2)

$\max(\alpha x, x)$ — (3)

$w_1 = \alpha \quad b_1 = 0$

$w_2 = 1 \quad b_2 = 0$

} Representing (1) as (3)

Ishan Modi

* Better Weight Initialization Strategies

→ Weights are initialized by zero

$$a_{11} = w_{11} x_1 + w_{12} x_2$$
$$a_{12} = w_{21} x_1 + w_{22} x_2$$

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

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.
vanishing gradient problem occurs if more layers are present

→ Initializing weights to large values
Saturation occurs and thus the vanishing gradient problem persists

Variance is low when plotted

Principled Way

→ $s_{11} = \sum_{i=1}^{n} w_{1i} x_i$

$$V(XY) = \underbrace{(\bar{x})^2 \text{var}(Y) + \bar{y}^2 \text{var}(X) + \text{var}(Y)\,\text{var}(X)}_{\text{formula}}$$

$$\text{var}(S_{1,}) = \text{var}\left(\sum_{i=1}^{n} \omega_{1,i} x_i\right) = \sum_{i=1}^{n} \text{var}(\omega_{1,i} x_i)$$

$$= \sum_{i=1}^{n} \Big((E[\omega_{1,i}])^2 \text{Var}(x_i) + (E[x_i])^2 \text{Var}(\omega_{1,i}) + \text{Var}(x_i)\,\text{Var}(\omega_{1,i}) \Big)$$

Let say x_i & $\omega_{1,i}$ are 0 mean &

$$\text{var}(x_i) = \text{var}(x), \quad \text{var}(\omega_{1,i}) = \text{var}(\omega)$$

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

$$= n\,\text{Var}(\omega)\,\text{Var}(x)$$

$$\therefore \text{Var}(S_{1,}) = (n\,\text{Var}(\omega))(\text{Var}(x)) \quad \text{— (1)}$$

Now,

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

$$= n\,\text{Var}(S_{1,})\,\text{Var}(\omega_2) \quad \text{→ substituting (1)}$$

$$\text{var}(S_{2,}) \propto \left[[n\,\text{var}(\omega)]^2 \text{var}(x) \right]$$

↓

thus low values & high values would lead to 0 variance of S_{21} or very high variance of S_{21}

Ishan Modi

$$Var(S_{ki}) = [n Var(w)]^k Var(x_i)$$

if $n Var(w) = 1$ then $var(S_{ki})$ will remain normal

→ If we take $z \sim N(0,1)$ and we take weights such that

$$n var(w) = n var\left(\frac{z}{\sqrt{n}}\right)$$

n → number of neurons in the layer

$\sqrt{n}$ → normalizing variable

thus

$$= n var\left(\frac{z}{\sqrt{n}}\right)$$

since $\left(w = \frac{z}{\sqrt{n}}\right)$

$$= n var\left(\frac{z}{\sqrt{n}}\right) \quad \left\{ var(az) = a^2 var(z) \right.$$

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

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

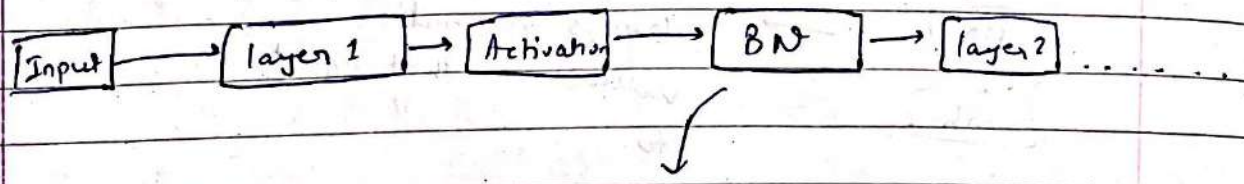
Applicable for Sigmoid & Tanh

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

$$w = \frac{[z]}{\sqrt{\frac{n}{2}}}$$

Ishan Modi

* Batch Normalization



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

Note - Batch Normalization as the name suggests is applied on the ~~entire~~ each batch that passes through the network.

Also, batch norm is applied accross individual features of the input & not the entire input

eg.

	x_1	x_2	x_3	x_4
input 1 →	0.3	0.9	0.7	0.2
input 2 →	0.4	0.7	0.2	0.1
⋮	⋮	⋮	⋮	⋮
input n →	0.9	0.8	0.8	0.4

Let say we have a batch with n inputs

each column in the above case should be normalized separately

→ Formula

As seen earlier, S_{ki} is the result of dot product of ~~out~~ input & weights

S_{ki} → output of i^{th} neuron of k^{th} layer

A_{ki} → output of i^{th} neuron of k^{th} layer after activation

- $\hat{A}_{ki} = \dfrac{A_{ki} - E[A_{ki}]}{\sqrt{var(A_{ki})}}$ 0 mean, unit variance

The above expression is differentiable so we can easily backpropagate the error

- Now, if we don't want normalization because normalizing values is ~~having~~ not allowing us to learn effectively we can convert batch norm to original

$$y^{(k)} = \gamma^k \hat{A}_{ki} + \beta^{(k)}$$

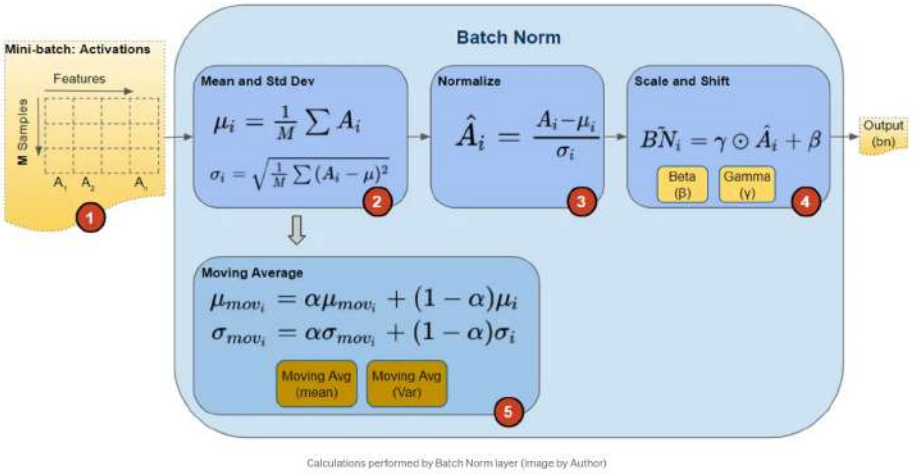
where, network learns γ & β which should be as follows

$$\gamma^k = \sqrt{var(A_{ki})} \quad \& \quad \beta^k = E[A_{ki}]$$

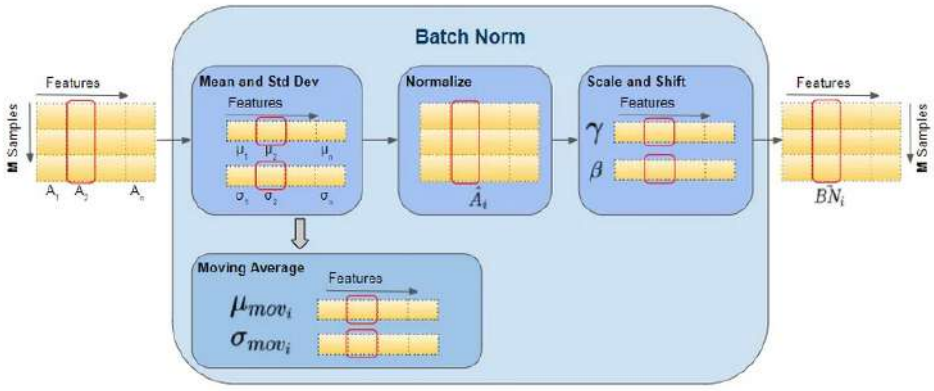
Note - Batch Norms helps network converge faster in most cases

Batch Normalization During Training

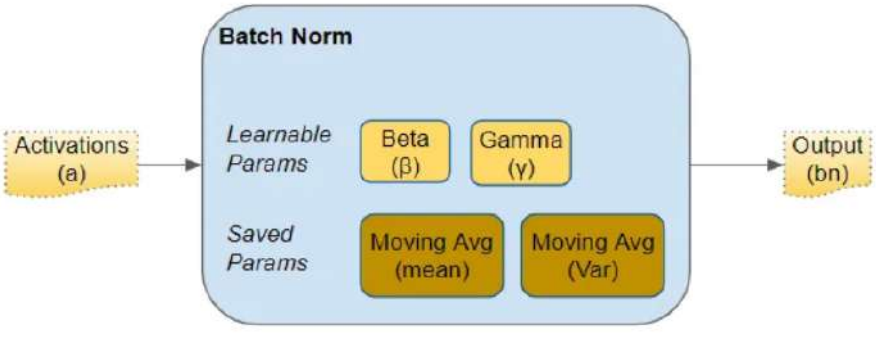
—> Example shows one mini batch at a time



Calculations performed by Batch Norm layer (Image by Author)



Shapes of Batch Norm vectors (Image by Author)

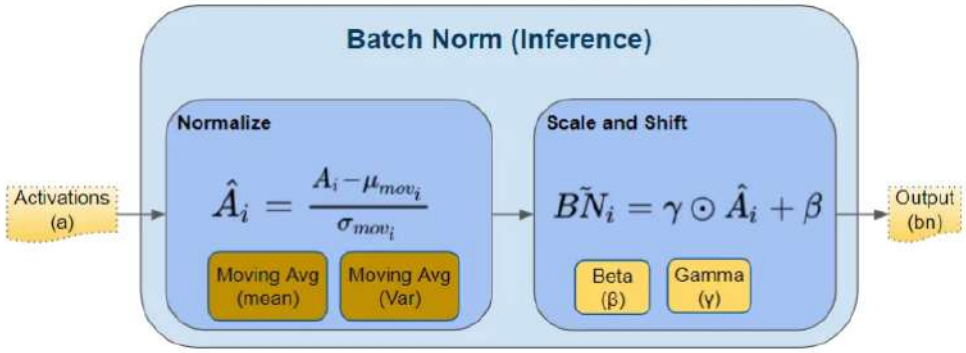


Parameters of a Batch Norm layer (Image by Author)

Ishan Modi

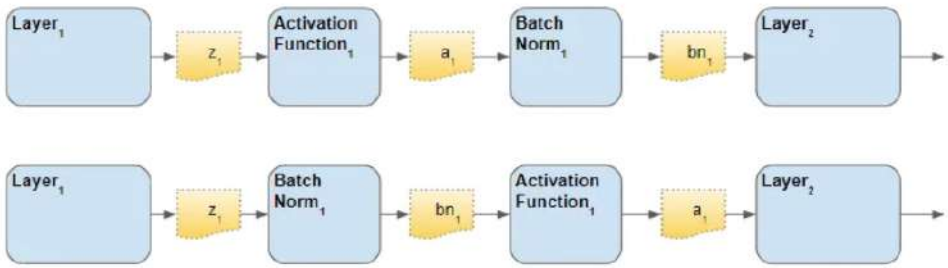
Batch Normalization During Inference

—> Example shows one mini batch at a time



Batch Norm calculation during Inference (Image by Author)

—> Finally, below is the overall functioning of batch normalization in a neural network



Batch Norm can be used before or after activation (Image by Author)

Ishan Modi

WEEK 10

* One-Hot Representation of Words (Sparse Representation)

Corpus - Collection of Sentences

Vocabulary - ~~Tool~~ All unique words in corpus

→ Lets say vocabulary is (words in corpus)

cat, dog, towel

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

- It Occupies lot of space
- No similarity between words (eg- 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

~~vocab - 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

Table - Co-occurence matrix ⇒ k = 1

words \ context	Cat	eats	a	Dog	Bone	Mouse
Cat	0	2	0	0	0	0
eats	2	0	3	1	0	0
a			0			
Dog				0		
Bone					0	
Mouse						0

Here k is that for a current word we need to check 1 neighbour towards left and right.

Cat appeared two times arround the word eat in a coindow of 1 word arround it.

→ This is still very complex

Solution

- Remove stopwords from column (context)
- Ignore very frequently occuring coords
- Use threshold, $t = 100$

$$x_{ij} = \min\left(\underbrace{\text{count}(w_i, c_j)}_{\text{entry in above matrix}}, t\right)$$

w_i ↓ word, c_j ↓ context

- Use PMI (low if count(w,c) is low) (high if count(w,c) is high)

$$PMI(w,c) = \log \frac{p(c|w)}{p(c)} = \log \left(\frac{count(w,c)}{count(w)} \Big/ \frac{count(c)}{N} \right)$$

($p(c|w)$, $p(c)$)

$$= \log \left(\frac{count(w,c) * N}{count(c) * count(w)} \right)$$

N is total no. of words

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

(1) $PMI_0(w,c) = PMI(w,c)$, if $count(w,c) > 0$
$= 0$, otherwise

(2) $PPMI(w,c) = PMI(w,c)$; if $PMI(w,c) > 0$ (positive)
$= 0$, otherwise

* Using SVD for dimentionality reduction

$$\hat{X}_{m \times n} = U_{m \times k} \; \sigma_{k \times k} \; V_{k \times n}$$

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

Rank k approx we construct it using adding k matrix

Now,

$\hat{X}\hat{X}^T$ = dot product = cosine similarity if we ignore denominator

(Thus each element of XX^T is cosine similarity between (i row of X) & (j column of X^T))

Ishan Modi

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

$\therefore$ $X < \hat{X}\hat{X}^T$ } similarity between various rows & cols increases thus it is able to learn relation

Now,

$X = U \Sigma V^T$

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

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

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

$(\Sigma \times V) = U \Sigma_{m \times k} (U \Sigma)^T$ taking

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

let $U \Sigma = W_{word}$ } $\rightarrow$ dimension of $W_{word} = m \times k$

$\therefore$

$XX^T = W_{word} W_{word}^T$

both have same cosine similarity (dot product) ignoring denominator

Though dimension of W_{word} is very less than X

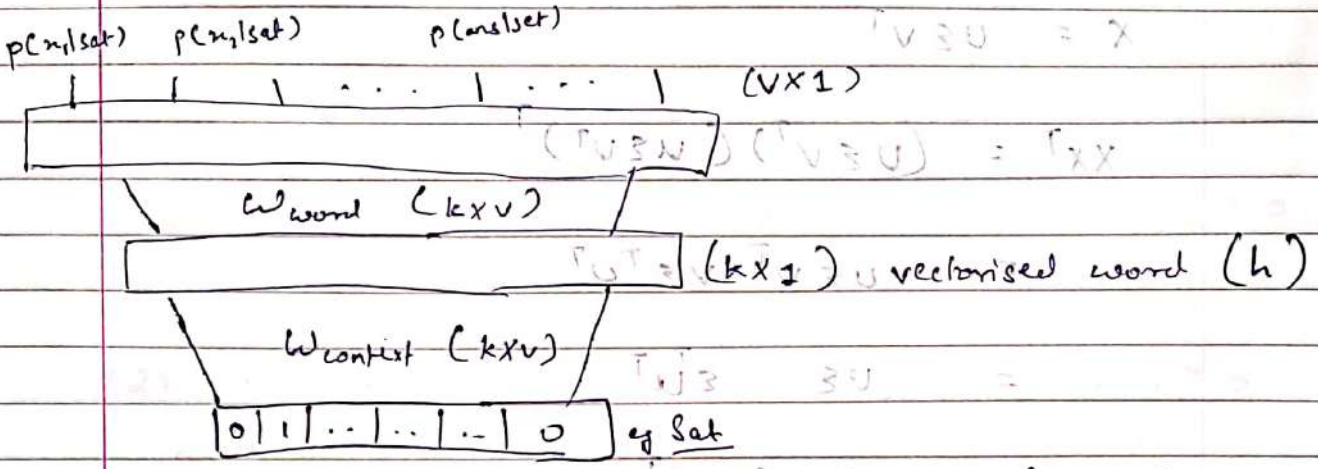
Ishan Modi

Here,

$W_{context} = V_{n \times k}$

it is the reduction for columns or context words

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



one hot encoded word as input (V x 1)

So,
here we have made a Neural Network

→ W_{word} & $W_{context}$ are weight matrix. to be learnt between layers

→ Final activation is Softmax since we need probability

→ loss is cross-Entropy loss.

Working

1 word next word (model)

→ When we move from input x :

$W_{context} \times x_i = h$

eg:- $W_{context} \quad x = h$

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

∴ if i^{th} index of x is 1 then

$h = i^{th}$ column of $W_{context}$.

→ Now,

$\text{softmax}(W_{word} \times h) = \text{output } (v \times 1)$

lets see any one entry from $v \times 1$

$$p(\text{on} \mid \text{sat}) = \frac{e^{(W_{word}\,h[i])}}{\sum_j e^{[(W_{word}\,h)[j]]}}$$

Now,

$W_{word}^T \times h = o/p$

$$W_{word} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

$$W_{word}^T = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 6 \end{bmatrix} \quad j=0 \rightarrow 4$$

$j=0$

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

Ishan Modi

Now, h = i^{th} column of $W_{context}$

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

→

$$L(\theta) = -\log \hat{y}_w = -\log P(w|c)$$

$$h = W_{context} \cdot x_c = u_c$$

(i^{th} col of $W_{context}$)

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

(j^{th} col of W_{word})

Now we want to learn v_w

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

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

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

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

loss funct wrt v_w

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

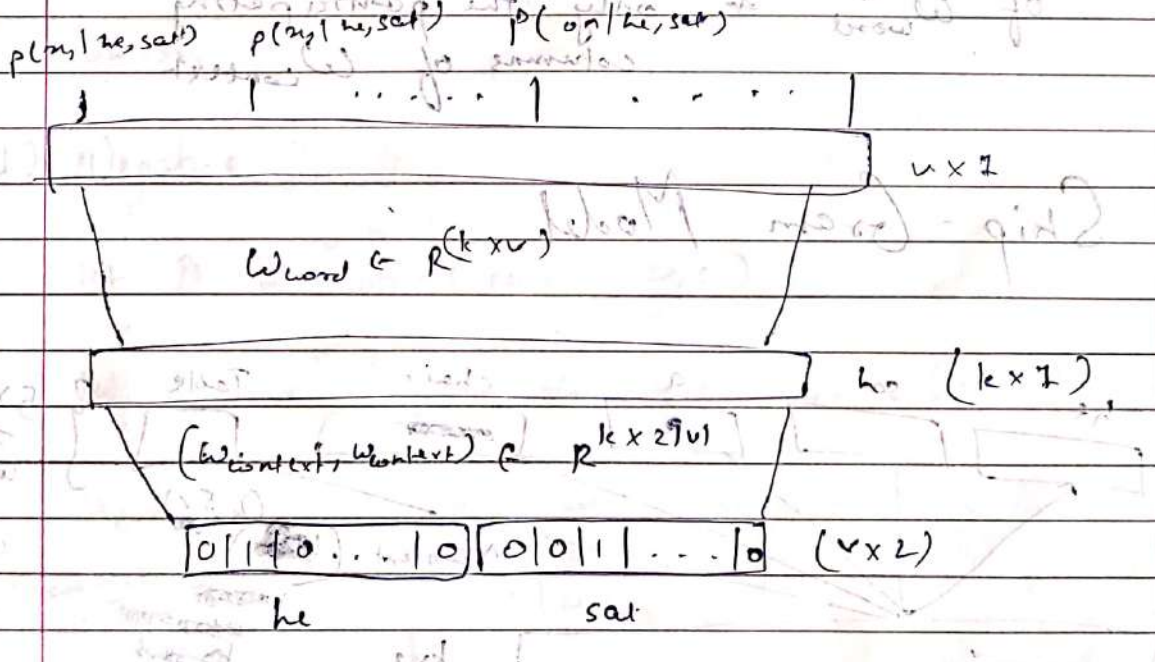
Update rule

$$\& \; V_w = V_w - \eta \nabla_{V_w}$$

$$= V_w + \eta u_c (1 - \hat{y}_w)$$

Note - Current entry of V_w comes closer to entries in u_c

→ More than one word are given

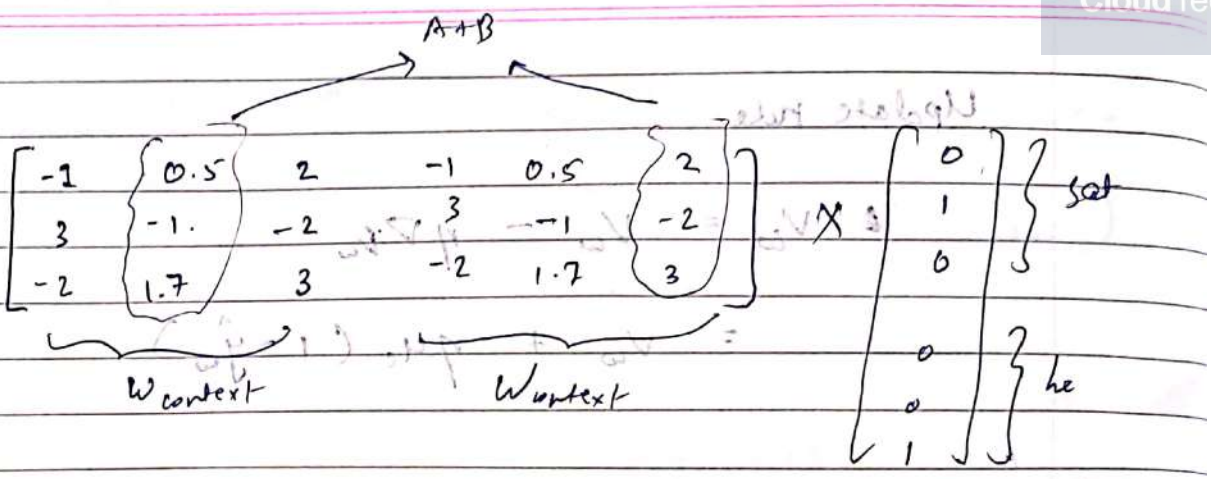


Now,

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

(δ_{-1}: number of words (here 2))

→ sum of columns in $W_{context}$ corresponding to one hot of words given

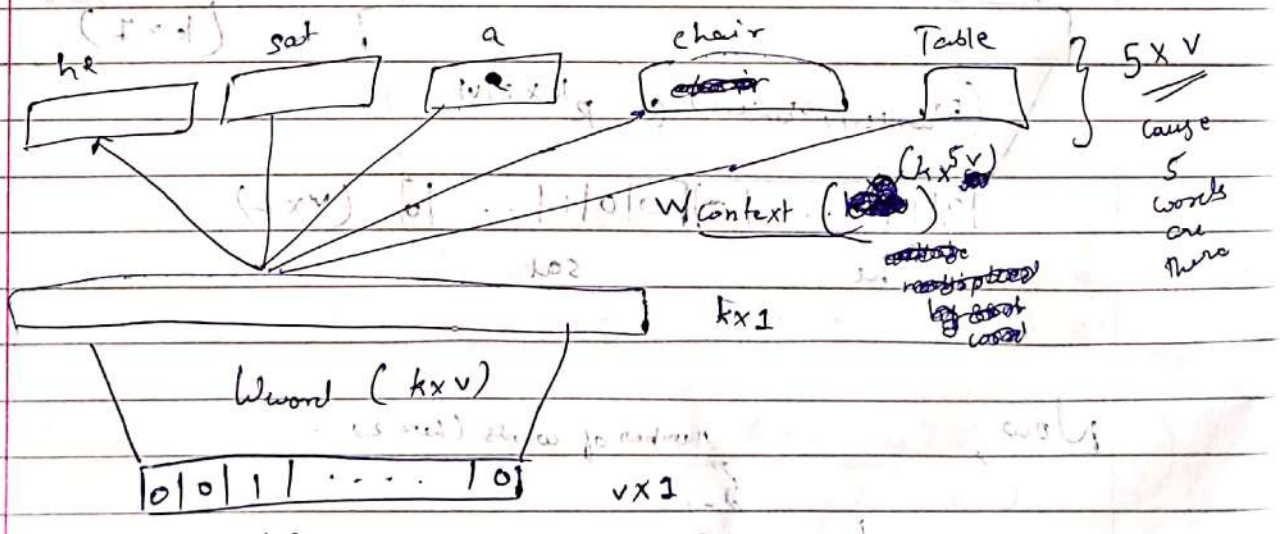


$$= \begin{bmatrix} 2.5 \\ -3 \\ 4.7 \end{bmatrix} = A + B$$

Now,

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

* 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.

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

If there was only 1 context word at output model is same as bag of words.
But here 5 words are there and this loss function is addition of all loss.

Problem

- lot of computation at final layer

Solution

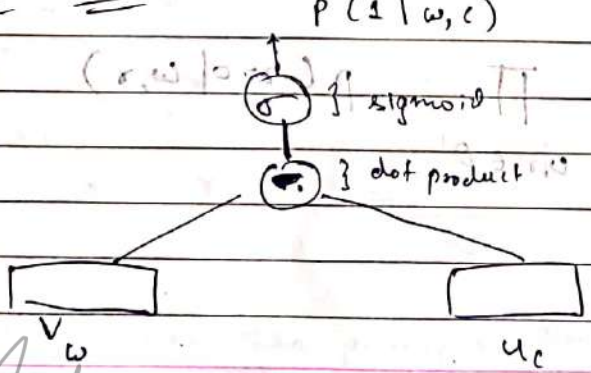
(1) Negative Sampling

let D be set of all correct pair (w, c)

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

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

correct context words



Ishan Modi

$$p(z=1 \mid w, c) = \sigma(u_c^T v_w)$$

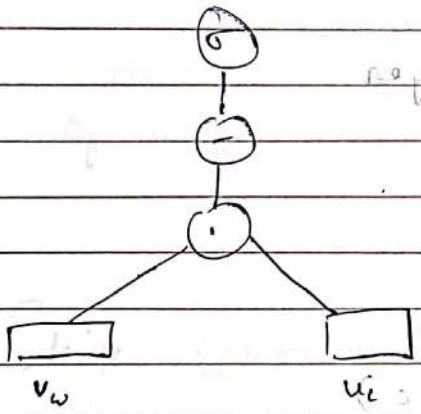
$$= \frac{1}{1 + e^{-u_c^T v_w}}$$

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

$$\text{maximize}_{\theta} \prod_{(w,c) \in D} p(z=1 \mid w, c)$$ → and (product of probabilities)

incorrect context words

Now,



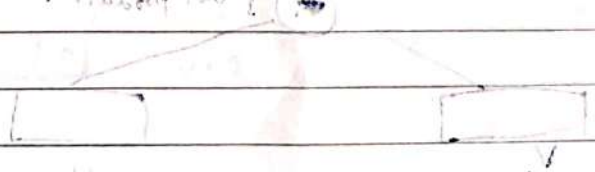
$$p(0 \mid w, r) = 1 - \sigma(u_r^T v_w)$$

$$= 1 - \frac{1}{1 + \exp(u_r^T v_w)}$$

$$= \frac{1}{1 + e^{u_r^T v_w}} = \sigma(-u_r^T v_w)$$

$$\therefore p(z=0 \mid w, r) = \sigma(-u_r^T v_w)$$

$$\therefore \text{maximize}_{\theta'} \prod_{w,r \in D'} p(z=0 \mid w, r)$$



Ishan Modi

Total

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

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

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

$$= \underset{\theta}{\text{maximize}} \sum_{\omega,c\in D} \log \sigma(u_c^T v_\omega) + k \sum_{\omega,r\in D'} \log \sigma(-u_r^T v_\omega)$$

$\log \sigma(u_c^T v_\omega)$ → bringing u_c & v_ω closer

$k \log \sigma(-u_r^T v_\omega)$ → moving u_r & v_ω away

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

→ Selection of (ω, r)

The random context word is drawn from a modified unigram distribution

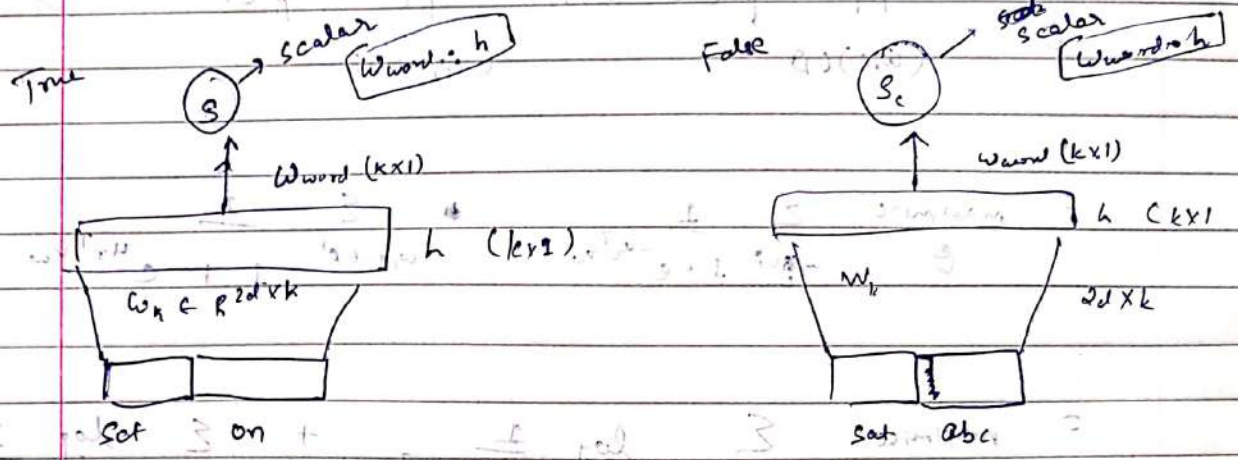
$$p(r)^{\frac{3}{4}}$$

$$r \sim \frac{count(r)^{\frac{3}{4}}}{N}$$

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

Ishan Modi

(2) Contrastive Estimation



We want to maximize

s = s_c (and we want margin between them (large difference))

Thus we maximize

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

Now,

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

$\therefore$

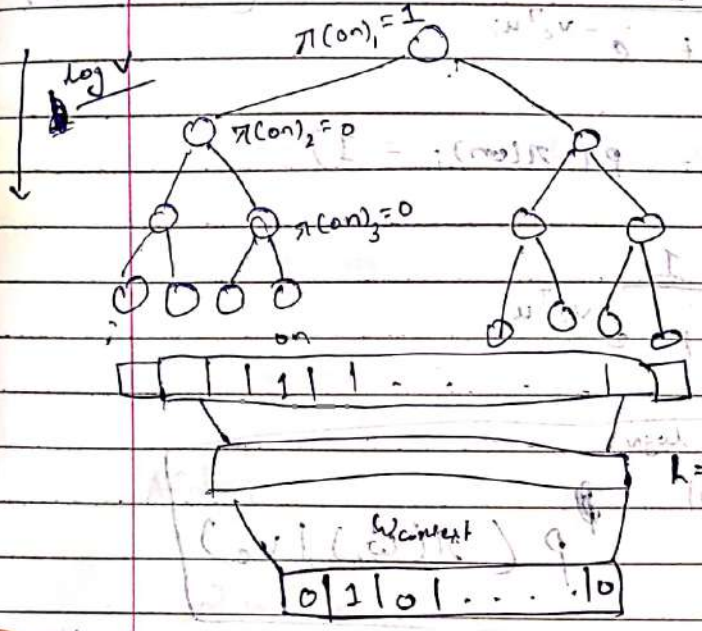
maximize $\left[\min(0, s - (s_c + m))\right]$ } loss function

Here

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

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

(3) Hierarchical Softmax



$\pi(w)_k = 1$ path branches left at node $l(w_k)$

$= 0$ otherwise

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

$W_{context} \triangleq u_1, u_2, \ldots, u_{v-1}$

all the nodes in tree except leaf nodes

every non-leaf node has a vector associated with it

→ Now,

$$p(w|v_c) = \prod_k \left(\pi(w_k) | v_c\right)$$

eg. $p(on | v_{set}) = p(\pi(on)_1 = 1 | v_{set})$
$* p(\pi(on)_2 = 0 | v_{set})$
$* p(\pi(on)_3 = 0 | v_{set})$

Here

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

v_c & u_i are $k \times 1$

$$p(\pi(on)_i = 0) = 1 - p(\pi(on)_i = 1)$$

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

Thus,

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

$|\pi(\omega)|$: $\log V$

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 coocurence matrix

$$p(C_j | i) = \frac{x_{ij}}{\sum_j x_{ij}} = \frac{x_{ij}}{x_i}$$

$$X_{ij} = X_{ji}$$

Now, lets say

$$u_i^T v_j = \log(p(j|i))$$

$$u_i^T v_j = \log(X_{ij}) - \log X_i \quad \text{— (1)}$$

Similarly for $p(i|j)$

$$v_j^T u_i = \log(X_{ij}) - \log(X_j) \quad \text{— (2)}$$

Adding (1) & (2)

$$2\, u_i^T v_j = 2\log(X_{ij}) - \log X_i - \log X_j$$

$$u_i^T v_j = \log(X_{ij}) - \frac{1}{2}\log X_i - \frac{1}{2}\log X_j$$

$\frac{1}{2}\log X_i$ → a, $\frac{1}{2}\log X_j$ → b

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

learnable parameters — known

→ loss function

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

Ishan Modi

* Evaluating Word Representations

→ Semantic Relatedness
Comparing computer predictions with human perceptions

→ Synonym Detection
Term - levied
Candidates - {imposed, believed, requested, correlated}
Synonym = $\underset{c \in C}{\arg\max}\ \text{cosine}(v_{term}, v_c)$

→ Analogy

- Semantic Analogy

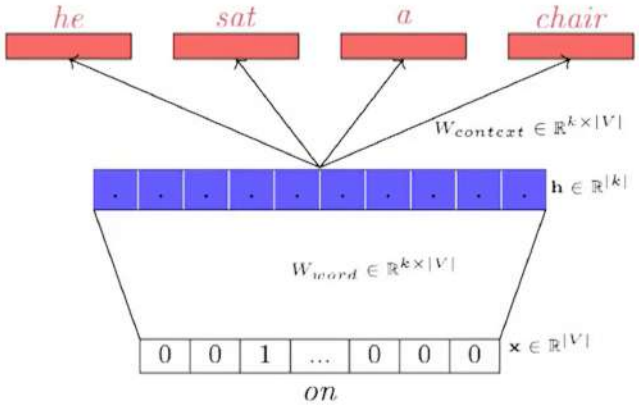
$v_{sister} - v_{brother} + v_{grandson} = v_{granddaughter}$

- Syntactic Analogy

$v_{works} - v_{work} + v_{speak} = v_{speaks}$

* Relation Between SVD & Word2Vec

→



- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization
- What does this mean ?
- Recall that word2vec gives us $W_{context}$ & W_{word} .
- Turns out that we can also show that

$$M = W_{context} * W_{word}$$

where

$$M_{ij} = PMI(w_i, c_i) - log(k)$$

k = number of negative samples

- So essentially, word2vec
rix M which is related t
co-occurrence matrix (ve
SVD does)