Convergence - Scaling Axis

Quadratic eq = $f(x) + (x - x_0) f'(x_0)$

original eq = $E = \frac{1}{2} \vec{w}^T A \vec{w} + b^T \vec{w} + c$

let $S = A^{0.5}$, $\hat{w} = S w$

then $E = \frac{1}{2} w^T S^T S w + b^T S w + c$

equating coefficients, $\boxed{S^T S = A}$

$b^T S = b^T$

$\therefore \boxed{\hat{b} = A^{-0.5} b}$

$E = \frac{1}{2} \hat{w}^T \hat{w} + \hat{b}^T \hat{w} + c$

$\hat{w} = A^{0.5} w$

$w = A^{-1/2} \hat{w}$

$\vec{w} = S w$

$\hat{w} = A^{1/2} w$

$A = A^{-1/2\,T} A$

then, $\frac{\hat{w}}{A^{1/2}} = w$

$\boxed{w = A^{-1/2} \hat{w}}$ — (1)

Using (1) in $E = \frac{1}{2} w^T A w + b^T w + c$

$E = \frac{1}{2} w^T \left(A^{1/2\,T}\right) A \cdot A^{-1/2} \hat{w} + b^T A^{-1/2} \hat{w} + c$

$\boxed{E = \frac{1}{2} \hat{w}^T \hat{w} + b^T \hat{w} + c}$

GD: $\hat{w}^{k+1} = \hat{w}^{k} - \eta \nabla_{\hat{w}} E \, (\hat{w}^{k})^T$

$\hat{w} = A^{1/2} w$

$A^{1/2} w^{k+1} = A^{1/2} w^{k} - \eta A^{1/2} \nabla_w E$

where, $w^{k+1} = w^k - \eta \nabla_w E$ → Newton Method.

$A = \nabla_w^2 E$

$E = \frac{1}{2} w^T A w + w^T b + c$

Forward pass

$y_1 = f_1(w_1 y_0 + b_1)$ $\quad z_1 = w_1 y_0 + b_1$

$y = \text{activation fn}(z) = f(z)$

$y_2 = f_2(z_2)$

$y_N = f_N\left[w_N f_{N-1}\left[\cdots f_2(w_2 f_1(w_1 x + b_1) + b_2)\cdots\right] + b_N\right]$

back prop:-

$\frac{\partial}{\partial y_i^N} Div = \frac{\partial}{\partial y_i} Div(y/d)$

$\frac{\partial}{\partial z_i^N} Div = f'(z_i^N) \cdot \frac{\partial Div}{\partial y_i}$

For $k = N-1, \ldots 0$

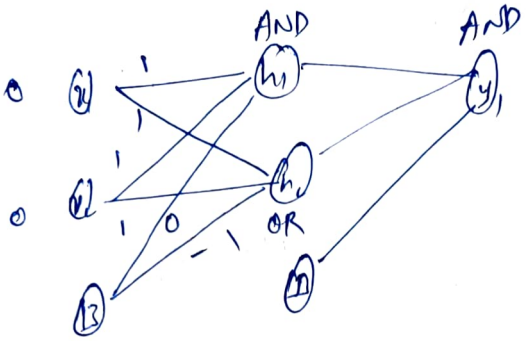
for $i = 1$: layer width

$\frac{\partial}{\partial y_i^k} = \sum_j w_{ij}^{k+1} \frac{\partial Div}{\partial z_j^{k+1}}$ — (y)

and $\frac{\partial Div}{\partial z_i^k} = f_k'(z_i^k) \frac{\partial Div}{\partial y_i^k}$ — (z)

for all, $\frac{\partial Div}{\partial w_{ij}^{k+1}} = y_i^k \frac{\partial Div}{\partial z_j^{k+1}}$ — (w)

XOR



x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$Relu[w_1 x_1 + w_2 x_2 + b]$ for each neuron

$h_1 = 0 \cdot 1 + 0 \cdot 1 + 0 = 0$

$Relu(h_1) = Relu(0) = 0$

Lec 4

Empirical Risk =

$$\text{Loss}\ \omega = \frac{1}{T} \sum div(f(x_i, \omega), d_i)$$

emp error = $\arg\min_{\omega}$ Loss ω

$$div(f(x;\omega), g(x)) = \begin{cases} 0 & \text{if } f(x,\omega) = g(x) \\ > 0 & \text{if } f(x,\omega) \neq g(x) \end{cases}$$

$$div(\ ) = Div(y, d) = \frac{1}{2} \|y - d\|^2$$

$$= \frac{1}{2} \sum (y_i - d_i)^2$$

Squared euclidian distance between two true and desired output

$$\frac{d}{dy_i} Div(y, d) = (y_i - d_i)$$

For binary classification = (as it will be scalar)

$$\frac{d}{dy} Div(y,d) = \begin{cases} \frac{-1}{y} & \text{if } d=1 \\ \frac{1}{1-y} & \text{if } d=0 \end{cases}$$

$$Div(y,d) = -d \log y - (1-d) \log(1-y)$$

minimum when $d=y$

Lecture 7

[34:30] time stamp

Finding best step size

Any quad. eq :- $E(w) = E(w^k) + E'(w^k).(w - w^k) + \frac{1}{2} E''(w^k).(w - w^k)^2$

where derivative (constant)

$$\frac{d}{dw} E(w) = 0 + E'(w^k) \cdot 1 + \frac{1}{2} 2 (w - w^k) E''(w^k) (w - w^0) = 0$$

$$\Rightarrow \boxed{\frac{-E'(w^k)}{E''(w^k)} = w - w^k}$$

Compare this with update rule.

$$\hookrightarrow w^{k+1} = w^k - \eta \frac{d E(w^k)}{dw}$$

$$\boxed{\eta_{opt} = \frac{1}{E''(w^k)} = a^{-1}} = \left[\frac{d^2 E(w^k)}{dw}\right]^{-1}$$

if $\eta < E''(w^k)^{-1}$! $\Rightarrow$ goes before the minima.

if $\eta > E''(w^k)^{-1}$ $\rightarrow$ goes above the minima

$\rightarrow$ $\eta \geq 2\eta_{opt}$ $\rightarrow$ divergence

multivariate Quadratic with Diagonal A

$$E = \frac{1}{2} w^T A w + w^T b + c = \frac{1}{2} \sum [a_{ij} w_i^2 + b_i w_i] + c$$

$\eta < 2 \min \eta_{i,opt}$

issue 1
- but makes slow learning
- will oscillate in all dim, $\eta_{i,opt} < \eta < 2\eta_{i,opt}$
- these estimation are for 2 dimension.

If there are n dimensions the computing and derivatives are complicated

How to fix these problems?
- Scale the vertical axis - 1 step thing

newton method does this
↓
by - multiplying space by [Hessian]$^{-1}$

issue 2 :- learning rate
- for complex models - loss function is not convex

Sol: Start with large η ; $\eta > 2$
then
Gradually reduce it with iterations.

Other Sol

Iterate
- train with fixed η until loss is stagnant
- $\eta = \alpha\eta$; $\alpha < 1$

RProp

- this is based on the sign of the derivative
  - if sign = +ve → move forward
  - if sign = -ve → move backward
- Simple first order algo.
  - much efficient than gradient descent
  - sometimes better than some good 2nd order algorithms
- only makes minimal assumptions about loss function

Quickprop

- employs newtons update with empirically derived derivative
- prone to instability for non convex objective fns.
- Still one of the fastest training algo for many problems.

- keep track of oscillations.

momentum method.

- maintain running avg.
  - if smooth - average will have large value
  - if swings - avg has +ve and -ve values that get ~~cancelled~~ cancelled out

- update avg.

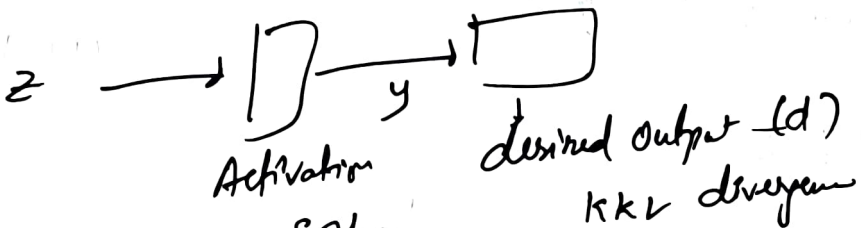
(cont'd) Lec 8

What is a good divergence?

- Smooth and less local optimas
- low slopes far from optima = bad
- high slopes near optima = bad.

L_2 div gradient →

$$\nabla_z \frac{1}{2}\|y-d\|^2 = (y-d)^T$$



classification : Softmax

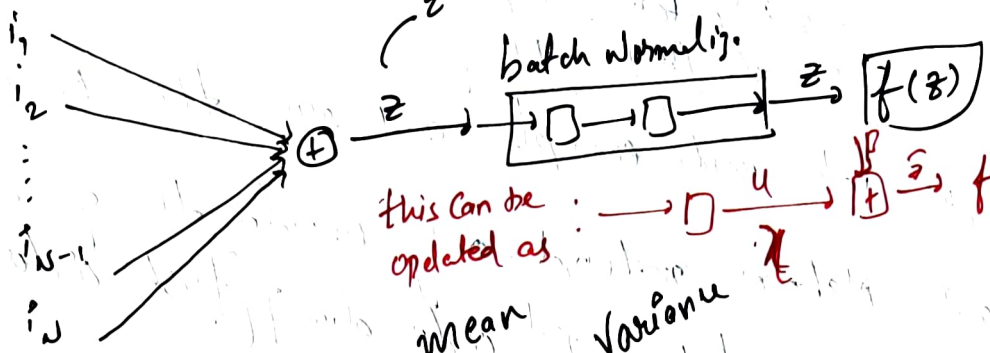
Batch Normalization

has 2 steps.

1. move the entire minibatch to origin and normalize it
   - has 0 mean
2. move it to new location

to move to new location :-

$z = \sum_i w_i x_i + b$



Step 1 $\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$

Step 2 $\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$

Step 3 $u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ } Centering

Step 4 $z_i = \gamma u_i + \beta$

B = mini batch size
μ_B = minibatch mean
ϵ = " Standard deviation

Substract mean from every instance of the minibatch.
Make sure, collection of minibatch have avg of 0.

((new location beta
gamma

- Scaling to the right/left size and to new location β

→ this Step 1, 2, 3, 4 will happen for every neuron.

[mean, Variance, Centering and Normalizing] [scaling + shifting] (also like bias

γ, β : Unit specific for every neuron.

Step 3 :- It is like having everything at center, at origin

β :- It is like Bias of neuron when we computing

Class 8
31/8/23
Thursday

Deep Learning

[Quiz solns on Monday]

notation
- mini batch = mb
- Back Prop = BP

momentum

In BP, we take one example and even the BP algo. momentum and Nesterov method only consider the gradient [other algos take 2nd order diff] [first order]

RMS Prop

not taking momentum of gradient

Compute square of derivative (not 2nd order) each individual element of gradient

$$\left(\partial_w D\right)^2 = \partial_w^2 D$$

$$= E\left(\partial_w^2 D\right)_k = \gamma E\left(\partial_w^2 D\right)_{k-1} + (1-\gamma)\left(\partial_w^2 D\right)_k$$

$$\Rightarrow w_{k+1} = w_k - \frac{\eta}{\sqrt{E\left[\partial_w^2 D\right]_k + \epsilon}} \partial_w D$$

running avg of square of gradient

ADAM : Rms Prop with momentum

Replace gradient with momentum term.

$$m_k = \beta m_{k-1} + (1-\beta)\left(\partial_w D\right)_k$$

$$v_k = \gamma v_{k-1} + (1-\gamma)\left(\partial_w^2 D\right)_k$$

$$w_{k+1} = w_k - \frac{\gamma}{\sqrt{v_k + \epsilon}} m_k$$

$$\hat{v}_k = \frac{v_k}{1-\gamma^k}$$

$$m_k = \frac{m_k}{1-\beta^k}$$

In early iteration, β and γ won't get normalized. So we do that in last layer

Optimization done.

- Bias-Variance tradeoff
- Regularization
- Dropout, Early stopping

} Useful but not much are using

→ model to perform good on unseen data.

- this error can be decomposed as 3 terms:-
  1. Bias ↑ BT VL
  2. Variance ↓
  3. Noise

Polynomial (P)

- As P ↑ (incl) the function becomes complex and tries to Overfit the data
- Test error decrease earlier and as P increases, the test error starts increasing test error

Too much simplicity of model is called BIAS.

Variance: In prediction model will be high, if model complexity is high. ~~test~~ Polynomial, P middle part has perfect balance between Bias and Variance

Underfit: bias
Overfit: Variance

- High bias ∨ cause high test errors or high variance

function we learned is this:- model built from dataset-

$$MSE = E_{P(x,y,D)}\left[(y - \underbrace{f(x:D)}_{\text{with constant } |D| = N})^2\right]$$

↓

effectiveness of model using mean Squared Error

$\mathbb{E}$ = expected value. * board

$$NSE = \mathbb{E}_{(x,y)}\left[(f(x) - t)^2\right]$$

$$= \int (f(x) - t)^2 p(x,t)\, dx\, dy$$

$$NSE = \int (y(x) - t)^2 p(x,t)\, dx\, dt$$

need to minimize this, then take the gradient, equate it to 0

$p(x,t)$ = Joint distribution on x and t

derivation taken from:- Book "Pattern Recognition and Machine Learning" by Christopher Bishop. read chapter 4

t = some $h(x) + E$ (errors, random variable $N(0, \dots)$)

Optimal = $y^*(x) \rightarrow$ [minimizer that minimizes NSE]

$$y^*(x) = \int \frac{t\, p(x,t)}{p(x)}\, dt = \int t\, p(t|x)\, dt$$

conditional prob.

$$y^*(x) = \mathbb{E}_{t|x}(t)$$

$E(t|x)$ = conditional Expectation

$$(y(x) - t)^2 = \{ y(x) - E[t|x] + E(t|x) - t \}^2$$

$$= [y(x) - E(t|x)]^2 + 2[y(x) - E(t|x)][E(t|x) - t] + (E(t|x) - t)^2$$

Noise Variance :- this is the minimum error, no matter how good model we take. (not in our hands)

2nd Term :- $\mathbb{E}_{x,t}(y(x) - E(t|x))(E(t|x) - t)$

$$= \mathbb{E}_x\left[\mathbb{E}_{t|x}\left[\{y(x) - E(t|x)\}\{E(t|x) - t\} \,|\, x\right]\right]$$

does not depend on t, so it will come out

$$= E_x \left[y(x) - E(t|x) \right] \underbrace{E_{t|x} \left[E(t|x) - t \right]}_{\text{this will be } 0} x$$

$$= E_x \left[(y(x) - E(t|x) \right] \underbrace{\left[E(t|x) - E(t|x) \right]}_{0}$$

$$= 0$$

Hence, the 2nd term is 0 (vanishing)

Hence, MSE will depend on 2 terms.

$$E(L) = \int \left[y(x) - E(t|x) \right]^2 p(x)\, dx + \int \text{Var}(t|x)\, p(x)\, dx$$

1st term

$$\text{MSE} = E\left[\left(y(x) - E(t|x) \right)^2 \right]$$

Decomposition

$$= E_x E_D \left[\left[y(x;D) - E(t|x) \right]^2 \right]$$

$$E\left(y(x) - E(t|x) \right)^2 = E_D \left[\left(y(x;D) - E(t|x) \right)^2 \right]$$

$$\text{MSE} = \underbrace{E_D \left[y(x;D) - h(x) \right]^2}_{(\text{Bias})^2} - \underbrace{E_D \left[\left(y(x;D) - E_D \left[y(x;D) \right]^2 \right]}_{\text{Variance}}$$

(Bias)^2

how much your Predictor differ from the optimal.

[depends on choice of complexity]

y = linear function.

Variance

L_2 regularization :-

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

↓ MSE

- If α ↓'s, then the algo focus on 2nd part
- If α ↑ → high Regularization

class 9
4/9/23

Solving Quiz 1 Questions

① tanh is odd fn

$$f(x) = \sum_{j=1}^{n_{hd}} a_j \tanh\left(w_j^{[1]} x + b_j\right)$$

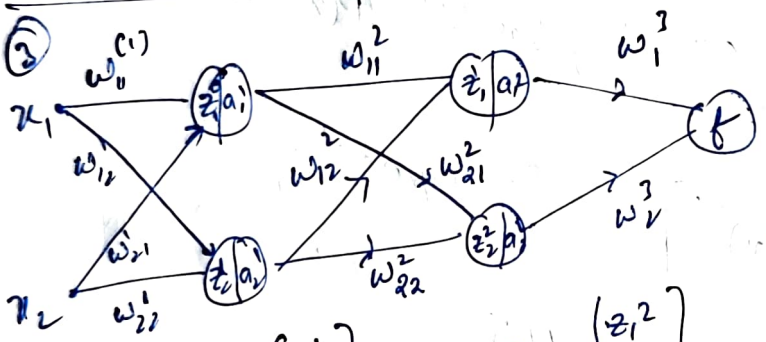
$$\bar{w}_1 = \begin{pmatrix} w_{11} \\ w_{12} \\ \\ \\ w_{1n} \end{pmatrix}$$

$$= \sum -a_j \tanh(-wx - b)$$

$$= f(x)$$

---

②



$$z^1 = \begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix} \qquad z^2 = \begin{bmatrix} z_1^2 \\ z_2^2 \end{bmatrix}$$

$$A^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix} \qquad A^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}$$

$$f = w_1^{[2]} a_1^2 + w_2^{[2]} a_2^2$$

$$\delta_1 = \frac{\partial f}{\partial z_1^2}$$

$$f = g(z_1^2) w_1^{[2]} + g(z_2^2) w_2^{[2]}$$

(activation fn)

$$\frac{\partial f}{\partial z_1^2} = w_1^{[2]} g'(z_1^2)(1)$$

$$\frac{\partial f}{\partial z_1^2} = w_1^{[2]} g(z_1^2)\left(1 - g(z_1^2)\right)$$

$$\delta_2 = \frac{\partial f}{\partial z^2} = \frac{\partial f}{\partial A^2} \cdot \frac{\partial A^2}{\partial z^2}$$

$$= \begin{bmatrix} \frac{\partial f}{\partial z_1^2} & \frac{\partial f}{\partial z_2^2} \end{bmatrix} \longrightarrow \text{Row Vector.}$$

$$\delta_1 = \frac{\partial f}{\partial z^1} = \frac{\partial f}{\partial z^2} \cdot \frac{\partial z^2}{\partial z^1} = \frac{\partial f}{\partial z^2} \cdot \frac{\partial z^2}{\partial A^1} \cdot \frac{\partial A^1}{\partial z^1}$$

$$= \delta_2 \cdot \frac{\partial z^2}{\partial z^1} = \delta_2 \frac{\partial \left[A^1 w^{2T} \right]}{\partial z^1}$$

$$= \delta_2 \left(w^{2T} \right) \frac{\partial (A^1)}{\partial z^1}$$

$$= \delta_2 w^{2T} \left[g'(z^1) \right]$$ — activation fn

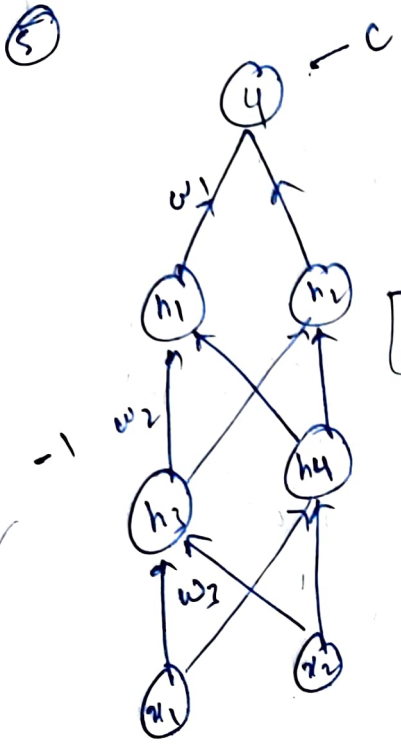
$$z_1^2 = w_{11}^2 a_1^1 + w_{12}^2 a_2^1$$

$$z_2^2 = w_{21}^2 a_1^1 + w_{22}^2 a_2^1$$

$$\frac{\partial z^2}{\partial A^1} = \begin{bmatrix} \frac{\partial z_1^2}{\partial a_1^1} & \frac{\partial z_1^2}{\partial a_2^1} \\ \frac{\partial z_2^2}{\partial a_1^1} & \frac{\partial z_2^2}{\partial a_2^1} \end{bmatrix}$$

$$= \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{bmatrix}$$

(5)



$h_i = \max(0, z_i)$

z_i = preactivation value

(a) which of these 2 will be 0

(A) $\frac{\partial C}{\partial w_1} = 0$ (B) $\frac{\partial C}{\partial w_2}$

(A) $\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial w_1}$

(B) $\frac{\partial C}{\partial w_2} \neq 0$? How → $h_2 w_2 + h_4 w_3$

[need to find the gradient]

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial h_1} \cdot \underbrace{\frac{\partial h_1}{\partial z_1}}_{=0} \cdot \frac{\partial z_1}{\partial w_2}$$

Combined output = -1

for this $z_1 = -1$

$z_1 = -1$

then $h_1 = (0, -1)_{(z_1)} \max$

$h_1 = 0$

$z_1 = w_2 h_3 = -1$

$$\frac{\partial h_1}{\partial z_1} = \begin{cases} 0 & \text{if } z_1 < 0 \\ 1 & \text{if } z_1 > 0 \end{cases}$$

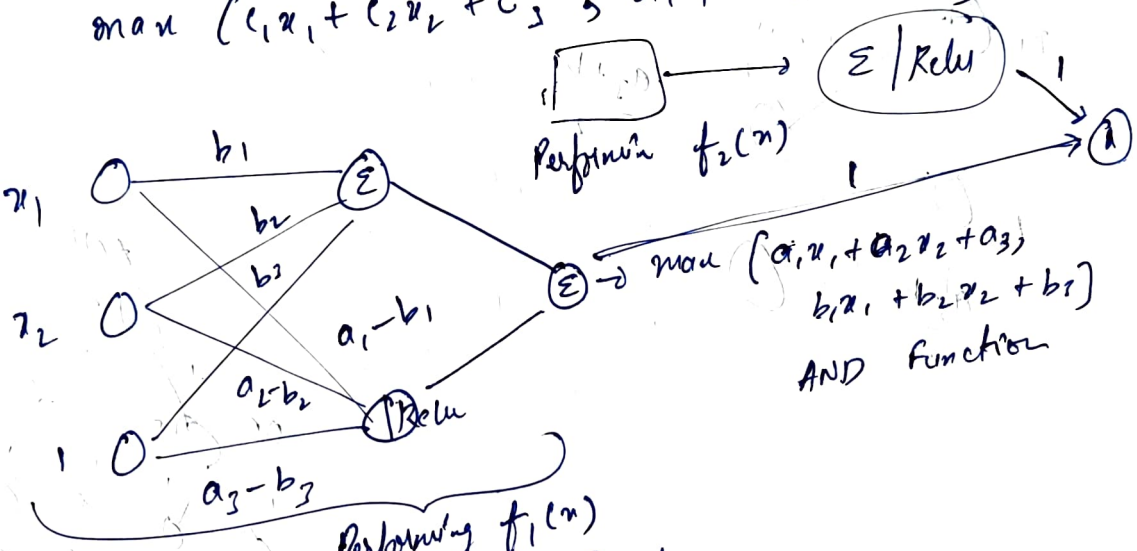
① implement max for of 2 linear, for using Relu method. (Need to use ReLu)

$\max(a, b)$

$= \max(a, b) + b - b$ → { if $a > b \rightarrow a$; if $a < b \rightarrow b$

$= \max(a - b, 0) + b$

$\min[\max(a_1x_1 + a_2x_2 + a_3, b_1x_1 + b_2x_2 + b_3),$
$\max(c_1x_1 + c_2x_2 + c_3, d_1x_1 + d_2x_2 + d_3)]$



$\min[f_1(x), f_2(x)]$ → Question.

decision boundary — this is like AND function

Q. we need to find out if this $\min[f_1(x), f_2(x)]$ is >0 or not.

$$\max(a, b) = \frac{a + b + |a - b|}{2}$$

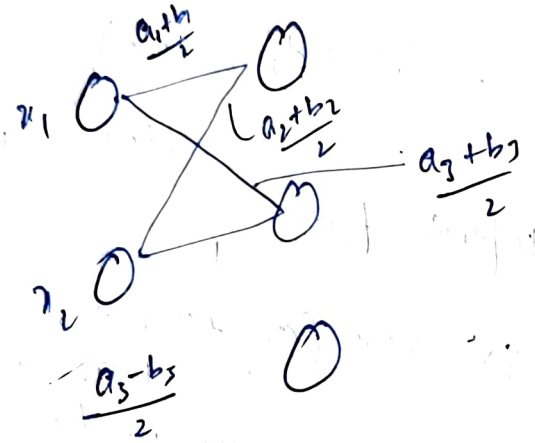
#

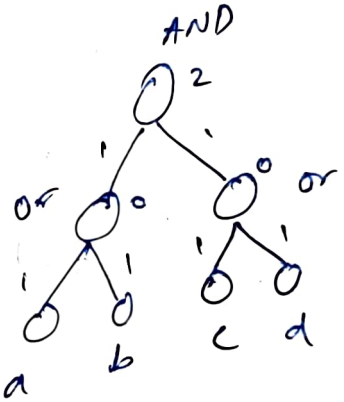
→ Implement max(a, b) Using 2nd method

$$\max(a_1x_1 + a_2x_2 + a_3,\ b_1x_1 + b_2x_2 + b_3)$$

$$= \frac{1}{2}\left[(a_1+b_1)x_1 + (a_2+b_2)x_2 + (c_1+c_2) + |(a_1-a_2)x_1 + (b_1-b_2)x_2 + (c_1-c_2)|\right]$$

(a_3+b_3; b_1; a_2-b_2; a_3-b_3)





$a = a_1x_1 + a_2x_2 + a_3$

$b = b_1x_1 + b_2x_2 + b_3$

generally, $\min(a,b) = -\max(-a,-b)$

$= -\max(-a,-b) + b - b$

$= -[\max(-a,-b) + b] + b$

$= -\max[b-a, 0] + b$

l_2 Regularization

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

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

$H =$ Symmetric & +ve Semi definite

$\nabla \tilde{L}(w) =$

$= \nabla L(w) + \alpha w$ $\quad = $ Hessian at H^*

$= H(w - w^*) + \alpha w$ $\quad = Q \Lambda Q^T$

where $QQ^T = Q^TQ = I$

$\Lambda =$ diagonal matrix with eigen values of H

If $\nabla \tilde{L}(w) = 0$

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

$H\bar{w} + \alpha\bar{w} = H\bar{w}^*$

$(H + \alpha I)\,\bar{w} = H\bar{w}^*$

$w^* = (H + \alpha I)^{-1} H \bar{w}^*$

$\alpha = 0 \rightarrow$ home no regularization

$\alpha \neq 0 \rightarrow$ Q is rotated back to it's original dimension

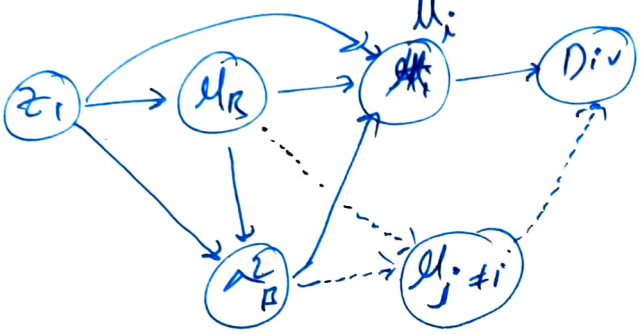
$\left[\frac{\lambda_i}{\lambda_i + \alpha}\right] = \begin{cases} \text{close to } 1 & \text{for larger eigen Values} \\ \text{close to } 0 & \text{for smaller eigen Values} \end{cases}$

Class 11
11/09/23

(Refer to before class 8 notes) [Nothing on 21]

Batch Normalization

- Applied before Activation
- input and output are different instances



$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum (z-\mu_B)^2$$

$$\mu_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma \mu_i + \beta$$

Div = function $(\mu_i, \mu_B, \sigma_B^2)$

$$\frac{dDiv}{d\beta} = \frac{dDiv}{d\hat{z}_i}\underbrace{\frac{d\hat{z}_i}{d\beta}}_{1}$$

$$\Rightarrow \frac{dDiv}{d\hat{z}_i}$$

$$\frac{dDiv}{d\gamma} = \frac{dDiv}{d\hat{z}_i}\underbrace{\frac{d\hat{z}_i}{d\gamma}}_{\mu_i} = \mu_i \frac{dDiv}{d\hat{z}_i}$$

$$\frac{dDiv}{d\mu_i} = \frac{dDiv}{d\hat{z}_i}\cdot\frac{d\hat{z}_i}{d\mu_i} = \frac{dDiv}{d\hat{z}_i}\gamma$$

$$\frac{\partial Div}{\partial z_i} = \frac{\partial Div}{\partial \mu_i}\cdot\frac{\partial \mu_i}{\partial z_i} + \frac{\partial Div}{\partial \sigma_B^2}\frac{\partial \sigma_B^2}{\partial z_i} + \frac{\partial Div}{\partial \mu_B}\cdot\frac{\partial \mu_B}{\partial z_i}$$

$$\frac{\partial Div}{\partial \sigma_B^2} = \frac{-1}{2}(\sigma_B^2+\epsilon)^{-3/2}\sum_{i=1}^{B}\frac{\partial Div}{\partial \mu_i}(z_i-\mu_B)$$

$$= \sum \frac{\partial Div}{\partial \mu_i}(z_i-\mu_B)\left[-\frac{1}{2}\right](\sigma_B^2+\epsilon)^{-3/2}$$

$$\boxed{\frac{\partial Div}{\partial \sigma_B^2} = \frac{-1}{2[\sigma_B^2+\epsilon]^{3/2}}\sum_{i=1}^{B}(z_i-\mu_i)\frac{\partial Div}{\partial \mu_i}}$$

$$\frac{\partial Div}{\partial \mu_B} = \left[\sum \frac{\partial Div}{\partial \mu_i}\cdot\frac{-1}{\sqrt{\sigma_B^2+\epsilon}}\right] + \frac{\partial Div}{\partial \sigma_B^2}\cdot\frac{\sum^{B} -2(z_i-\mu_B)}{B}$$

How? ↗

Sol:- $\frac{\partial Div}{\partial \mu_B} = \sum_1^B \frac{\partial Div}{\partial \mu_i}\cdot\frac{\partial \mu_i}{\partial \mu_B} + \frac{\partial Div}{\partial \sigma_B^2}\cdot\frac{\partial \sigma_B^2}{\partial \mu_B}$

$$\boxed{\frac{\partial Div}{\partial \mu_B} = \sum \frac{\partial Div}{\partial \mu_i}\cdot\frac{-1}{\sqrt{\sigma_B^2+\epsilon}} + \frac{\partial Div}{\partial \sigma_B^2}\cdot\frac{\sum -2[z_i-\mu_B]}{B}}$$

this whole term = 0

$$\boxed{\frac{\partial Div}{\partial \mu_B} = \sum \frac{\partial Div}{\partial \mu_i}\cdot\frac{-1}{\sqrt{\sigma_B^2+\epsilon}}}$$

→ for each batch, there exists different Mean.

So at the time of Inference:- B, B-1 can be ignored

μ_{B_1}	σ_{B_1}
μ_{B_2}	σ_{B_2}
⋮	⋮
μ_{B_k}	σ_{B_k}

$$\mu_B = \frac{\sum_1^k \mu_{B_i}}{k} \qquad \sigma_B^2 = \frac{1}{k}\sum_{i=1}^{k}\cdot\frac{B-1}{B}\sigma_{B_i}^2$$

$\hat{\mu}$ = unbiased estimator of μ

$x_1, x_2 \dots x_n$ are iid rv

$x_i \sim (\mu, \sigma^2)$

$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$

$$E(\hat{\mu}) = E\left[\frac{1}{n} \sum_{i=1}^{n} x_i\right]$$

$$= \frac{1}{n} [\Sigma E(x_i)]$$

$$= \frac{1}{n} \Sigma \mu$$

$$E(\hat{\mu}) = \mu$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$$

$\boxed{E(\hat{\sigma}^2) \neq \sigma^2}$

$$E\left[\frac{n}{n-1} \hat{\sigma}^2\right] = \sigma^2$$

$$\tilde{\sigma}^2 = \frac{n}{n-1} \hat{\sigma}^2 = \frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \mu)^2$$

→ BN is applied to only some layers [or selected neurons only]
→ BN improves convergence rate and NN performance
→ BN takes care of step size also.

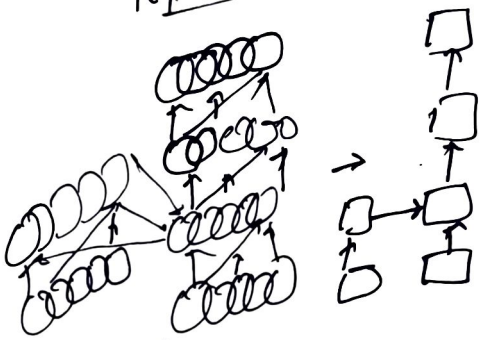
Modelling Series

RNN

- most models until now consider data was coming from constant series.
- In this data is a non constant like weather data, sequential data, video data, speech data, music.

Input = Sequence of Vector
Output = Classification Result

Representation of NN



Input = Vector
Each layer has many neurons

→ output of hidden layer at $t-1$ is the input of the hidden layer at t

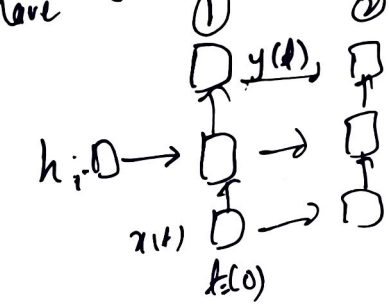
finite - response model
→ Something that happen today only affects the output of the systems for N days into the future

infinite response systems :-
long term affect on output into the future

$$y_t = f(x_t, y_{t-1})$$

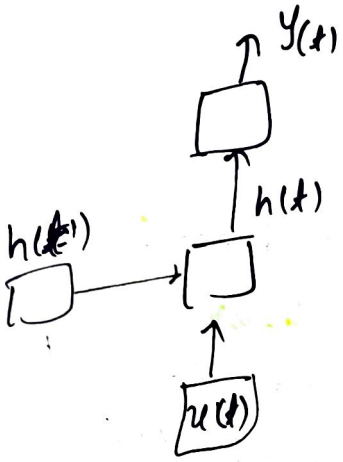
that why Recurrent NN

State Space Model
Each layer represented by a box
have same network replicating each time stamp, ① = ② ③ with only change in the values.



→ While Backprop in RNN, if at $t=0$, gradients are also going to $(t-1)$, back in time. This is called backprop through time

given - collection of Sequence Inputs



Lec 8 Video

Batch Normalization

Convergence can be improved using smoothed updates

$$\underset{w_1, w_2, \ldots w_k}{\operatorname{argmin}} \; L(w_1, w_2, \ldots w_k)$$

Dropout

→ To understand this we need to understand **Bagging**

- turn off some neurons with prob $(1-\alpha)$
- Pattern of dropped nodes change for each input
- Backprop is only performed over remaining part
- 2^N = possible networks [for network with N neurons]
- Obtains by choosing different subsets of nodes
- Mechanism to increase pattern density
- Helps to learn
  - → Rich pattern
  - → Redundant patterns
  - → learns denser patterns + redundance

Training

- Trains 2^N networks
- Bagged output = Avg over all 2^N networks = Statistical expectation of output over all network

$$\text{Bagged output} \Rightarrow Y = E\left[\text{network}\left[y_j^k, j = 1 \ldots D_k, k = 1 \ldots K\right]\right]$$

Can't compute this Expectation
So we use approximation

$$E\left[\text{network}\left[y_j^k, \forall k j\right]\right] = \text{network}\left[E\left[y_j^k\right] \forall k, j\right]$$

Activation of each Neuron $= y_j^k = D\alpha\left[\sum w_{ji}^k y_j^{k-1} + b_i^k\right] = \alpha\, a\left[z_j^k\right]$

D = Bernoulli Variable

D = 1 with probability α

thus

$$E\left[y_i^k\right] = \alpha\, a\left[\sum w_{ji}^k y_j^{k-1} + b_j^k\right]$$

$w_{test} = \alpha\, w_{trained}$

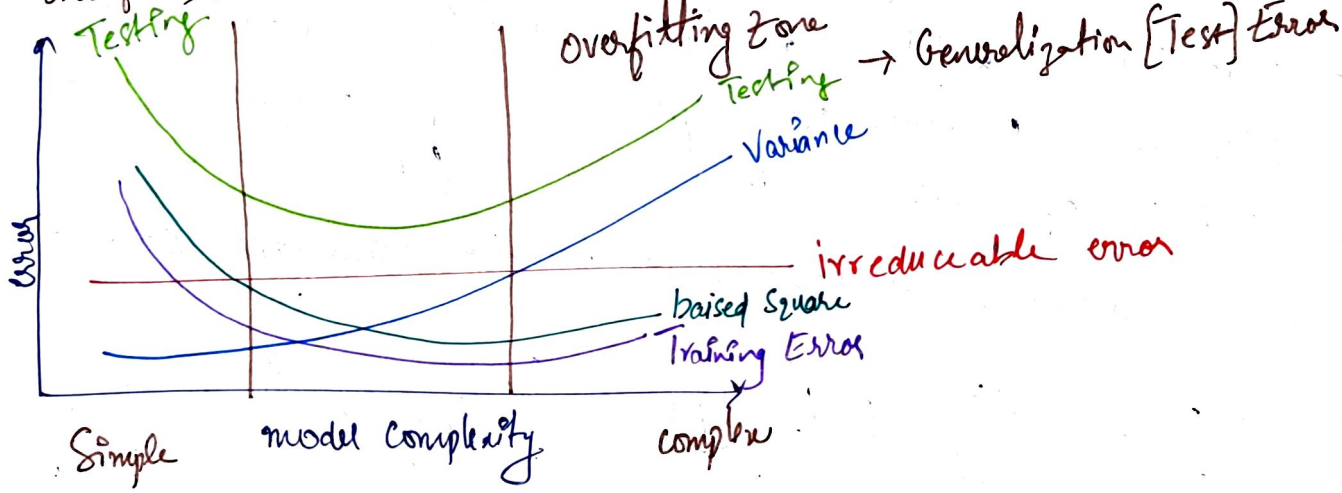
NPTEL IIT Guwahati

Bias - Variance Tradeoff

High Variance → Overfitting

High Bias → Underfitting

Test error = Bias Error + Variance + irreducible error



Problem Definition

x = Independent Variable

y = Dependent Variable

$y = f(x) + \epsilon$ → Noise

ϵ = Noise, Zero mean Variance σ_ϵ^2

$\therefore E[\epsilon] = 0$

$var(\epsilon) = E\left[\epsilon^2\right] = \sigma_\epsilon^2$

Q → to determine - $\hat{f}$

Q. to find a function $\hat{f}$ such that it is as close to the true function f (can learn from training data)

$\hat{f}$ is learned by minimizing a loss function

Goal → to bring predictions from training data as close as possible to the observed value.

i.e., $y \approx \hat{f}(x)$

MSE → loss function

MSE = $\mathbb{E}\left[\left[y - \hat{f}(x)\right]^2\right]$ Squared differen of prediction from the true value y.

Bias = $\text{bias}\left[\hat{f}(x)\right] = \mathbb{E}\left[\hat{f}(x)\right] - f(x)$ → it is the differen of the average value of prediction to the true underlying function $f(x)$ for a given unseen test Point x

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

↓

→ mean squared deviation of $\hat{f}(x)$ from its expected value $\mathbb{E}[\hat{f}(x)]$ over different realizations of training data.

mse + bias + variance + irreducible error. → combined formula

$$\mathbb{E}\left[\mathbb{E}\left[y - \hat{f}(x)\right]^2\right] = \mathbb{E}\left[\text{bias}\left[\hat{f}(x)\right]^2 + \mathbb{E}\left[\text{Var}\left(\hat{f}(x)\right] + \sigma_\varepsilon^2\right.\right.$$

MSE — bias — Variance — irreducible error

decomposing MSE to bias, variance and irreducible error

[Lec 7 youtube CMU 11785]
minibatches, RMS Prop, Quick Prop ....

[50:00 minutes] SGD uses samples of only one samples gradient | Provides Sparse updates

SGD converges faster and has large variance.

So need a better one :-

① Mini batch update (MB)

- adjust function at small, randomly chosen subset of points
- Randomly Sample (Shuffle the data randomly)

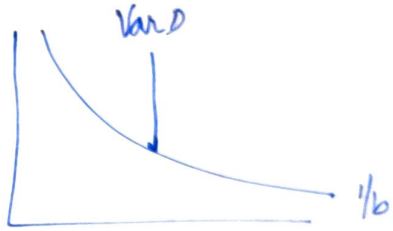
How this works :-

$$M.B.\text{loss} = \frac{1}{b} \sum div(f(x_i : w), d_i)$$

MB loss = Unbiased estimate of expected loss

$$E[\text{MB loss}] = E\left[div(f(x:w), g(x))\right]$$

$$\text{Var}(\text{batch loss}) = \frac{1}{b}$$



→ Variance is much smaller than SGD
→ Converges faster [overall training loss]
→ loss is computed on entire training set, but then need to process whole data once, Hence, it will be complicated / costly

Mini Batch Size = largest that the Hardware supports

large MB = Less Variance
= fewer updates per epoch

SGD Variance → ↑ than Batch

Online → As the data come in.

MB
- Has low variance than SGD
- Converges ~~Less~~ worse than SGD
- Compensate ↑ by being able to perform batch processing.

Momentum Update

:- maintains running average of all gradients until current step

$$\Delta w^k = \beta \Delta w^{k-1} - \eta \nabla_w \text{loss}[w^{k-1}]^T \qquad \beta = 0.9$$

$$w^k = w^{k-1} + \Delta w^k$$

at any iteration, to get current step :-

(1) compute gradient step at current location
(2) Add scaled previous step
(2.1) this is actually running average

To get final step:-

→ Updates are computed in 2 stages:-

(1) take step against gradient at current loc.
(2) Add scaled version of previous step

↑ this can be more optimal by Reversing the order of operations

→ Nesterov's method

Change the order of steps

(1) extent previous step
(2) compute gradient at resultant position
(3) add the 2 to obtained final step

Momentum is smoothing out the variations

These methods will improve convergence by Normalizing the mean of the derivatives

Other methods also consider their variance

- RMS prop
- Adagrad
- Ada delta
- ADAM (popular)

RMS Prop

Considers total Movement of the gradient.
Variant based on Mini Batch SGD algorithm

[Squares of the derivative] $\rightarrow [\partial_w D]^2$

$$\text{running average} = \frac{1}{\sqrt{E[\partial_w^2 D]_k + \epsilon}}$$

} Scaling down the running average.

- Maintain running estimate of MS value of derivatives for each parameter
- Scale update of parameter by Inverse of root [mean Squared Derivative]

$$E[\partial_w^2 D_k]_k = \gamma E[\partial_w^2 D]_k + [1-\gamma][\partial_w^2 D]_k$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{E[\partial_w^2 D]_k + \epsilon}} \partial_w D$$

this is similar to RProp.
Thing that is not happening here are:- Running average of derivatives is not being changed, and Momentum.

ADAM

Combination of RMS Prop + Momentum
Utilises Smoothed Version of Momentum-Augmented gradient [Considers first and Second Momentums]

M_k = Running Average of derivative
V_k = Squared derivative
γ^k, δ^k = to make sure these terms do not dominate in early iterations.

RMS Prop	ADAM
$\eta = 0.001$	$\eta = 0.001$
$\gamma = 0.9$	$\delta = 0.9$ & $\gamma = 0.999$

} Typical Params

CMU Lecture 13 11-785 — RNN — Modelling Network

- All columns are identical (even incoming edges)

current weight Recurrent weights

$$h^1(t) = f_1\left[w^1 x(t) + w^{11} h^1(t-1) + b_i^1\right]$$

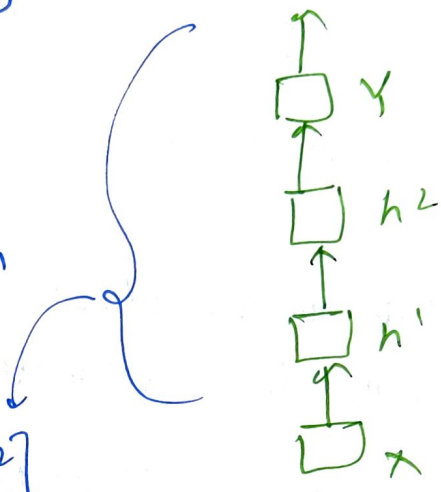
$$y(t) = f_2\left[w^2 h_j^1(t) + b_k^2 \;,\; k = 1, \ldots, M\right]$$

Notations

w^1 → weights of layer 1

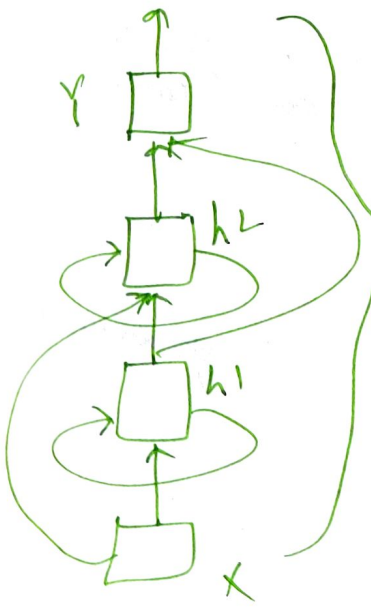
w^{11} → weights of layer 1 to layer 1

w^2 → weights of layer 2



$$h^2(t) = f_2\left[w^2 x(t) + w^{22} h^2(t-1) + b_i^2\right]$$

Recurrent State Activation = tanh() — typically



$$h^1(t) = f_1\left[w^{0,1} x(t) + w^{11} h^1(t-1) + b^1\right]$$

$$h^2(t) = f_2\left[w^{12} h^1(t) + w^{0,2} x(t) + w^{22} h^2(t-1) + b^2\right]$$

$$y(t) = f_3\left[w^3 h^2(t) + w^{13} h(t) + b^3\right]$$

Backprop through time: focus on one training instance

Divergence computed is not just the error, it is btw the sequence of outputs by network and desired seq of outputs

and this is the scalar function of series of vector

Div is function of all outputs $y(0) \ldots y(T)$

global Div = $\sum$ Div(t)

Backprop: Should start at end

once we have gradients at $y(T)$

going backward assuming I have $\frac{dy(T)}{dt}$ what we need to do to get derivative of affine value?

we should do Chain Rule
Should do Jacobian of the activation - will give derivation of divergen
Now hidden layer → multiply derivative z_2 by $\frac{dz_2}{dh}$ (weight matrix)

Class 12
14/9/23

Training the Network

(x_i, D_i) = Inputs

$x_i = x_{i0}, x_{i1}, \ldots x_{iT}$ (given)

$D_i = D_{i0}, D_{i2}, \ldots D_{iT}$ → Targeted Output

$Y_i = Y_{i0}, Y_{i1} \ldots Y_{iT}$ → Output of the Network (predicted Output)

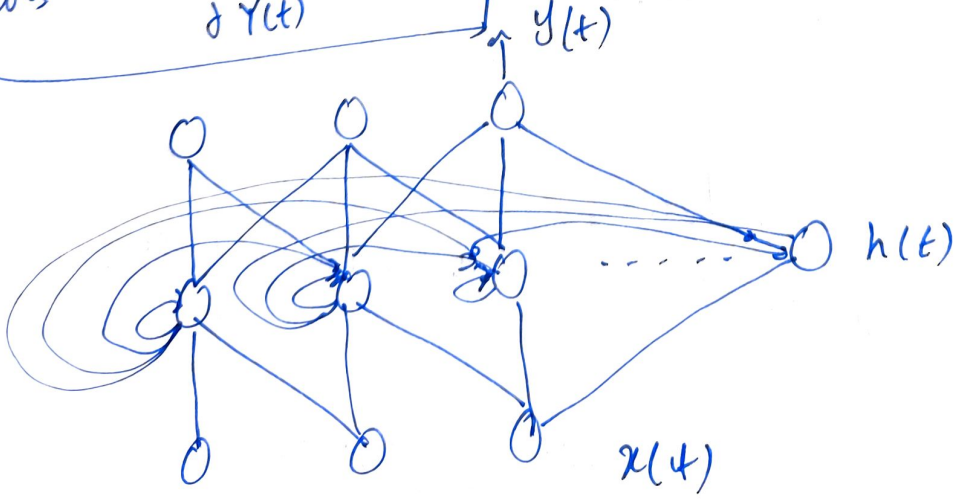
$loss = Div\left[[Y(0), Y_1 \ldots Y_T] - [D_0, D_1 \ldots, D_T]\right]$

Y_T compared to D_T only.

$= \sum Div(Y(t) . D(t))$

$= \frac{\partial Div\left[[Y_0, Y_1 \ldots Y_T] - [D(0), D_1 \ldots D_T]\right]}{\partial Y(t)}$

$$loss = \frac{\partial Div[Y(t) . D(t)]}{\partial Y(t)}$$



$$z_i^2(t) = \sum_{j=1}^{N_i} w_{ji}^2 h_j(t) + b_j^2 \qquad i = 1, \dots M$$

$f_i^2(z_i^2(t))$

activation function

Backprop

→ will only focus on one training instance

→ using forward pass, we computed $Y(t)$

→ using backprop we need to compute gradient from output layer starting from $y(t)$

Step 1 - finding gradients wrt $y(t)$ → of the t^{th} output

$y(t) \in \mathbb{R}^{th}$

$\frac{\partial Div}{\partial y(t)}$

Step 2: gradients wrt $z_2(T)$

$\nabla_{z^2(T)} Div = \nabla_{y(T)} Div \; \nabla_{z^2_T} Y(t)$

activation can be soft-max also

Step 3: gradient wrt $z_i^2(T)$ i^{th} component of z^2_T

Scalar — Vector

$$\frac{d\,Div}{\partial z_i^2(T)} = \frac{\partial Div}{\partial y_i T} \cdot \frac{\partial y_i T}{\partial z_i^2(T)} \qquad \frac{\partial Div}{\partial z_{i(T)}^2} = \sum$$

derivations → Pg 5 | 26

If we need to find gradient with all components

then

$\nabla_{z^2(T)} Y(T)$ (Jacobian) (Pg 6)

→ each $z_{i,T}$ is formed by applying activation for y $h_{i,T}$

→ $z_{1,T}$ has N_1 components.

→ w_{ij} → weights connecting to input layer

$$z'_{j,T} = \sum w'_{ij} x_{i(T)} + \sum w^{2,2}_{ij} h_{2(T-1)} + b'_j$$

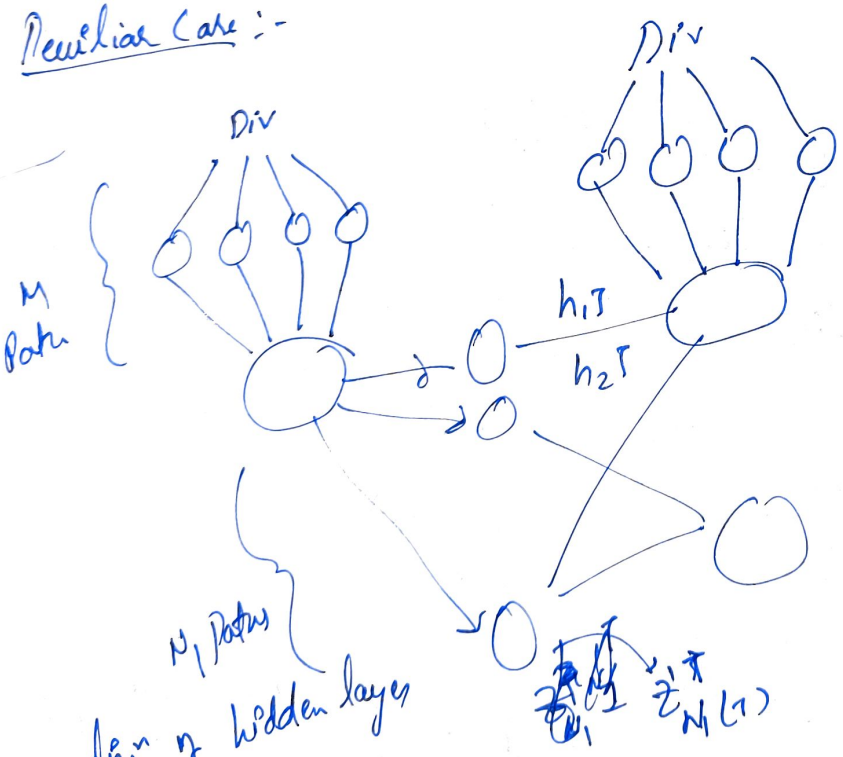
$$\nabla_{w'} Div = X_T \nabla_{z'_T} Div$$

$$\nabla_{w''} Div = h_{(T-1)} \nabla^{Div}_{z'_T}$$

$N_1 \times M_1$

computed one set of gradients at T, should do the same for time $(T-1)$

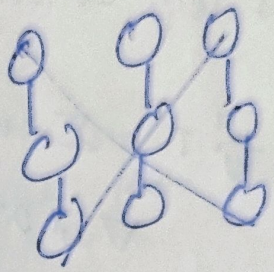
Peculiar Case :-

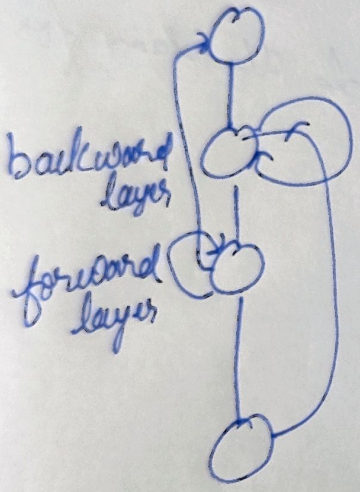


$$\frac{\partial Div}{\partial h_i(T-1)} = \sum_{j=1}^{M} \frac{\partial Div}{\partial z_j^2(T-1)} \cdot \underbrace{\frac{\partial z_j^2(T-1)}{\partial h_i(T-1)}}_{w_{ij}^2} + \sum_{j=1}^{N_1} \underbrace{\frac{\partial Div}{\partial h_j(T)} \cdot \frac{\partial h_j(T)}{\partial z_j^1(T)}}_{\delta_j^1(T)} \cdot \underbrace{\frac{\partial z_j^1(T)}{\partial h_i(T-1)}}_{w_{ij}^{(1)}}$$

Bidirectional RNN

has 2 hidden layers
$H(T)$ depends on $H(T-1)$
$H(T-1)$ will depend on $H(T)$





these both are added directly.

BIBO : bound Input Bounded Output

1. assume activation fn as linear
2. Analyse BIBO stability

$h_k = z_k$

Linear Recursion

stability depends on this

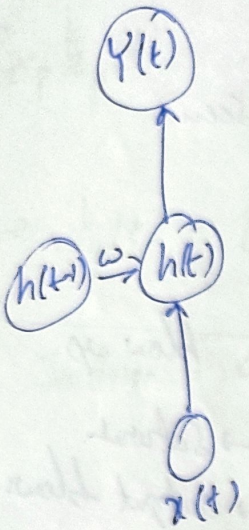
if $w < 1$ → Vanishing Gradient
$w > 1$ else exploding Gradients

$$h(t) = w(h(t-1)) + Cx(t) + b$$

$$h_0(t) = w^t C x(0)$$

only supplying Input at $t = 0$ from $t = 1$ there no Input

w : matrix of feedback loop



$y(t) \in R^m$

$h(t) = w\,h(t-1) + c\,x(t)$

$= w\left[w\,h(t-2) + c\,x(t-1)\right] + c\,x(t)$

$= w^2 h(t-2) + cw\,x(t-1) + c\,x(t)$

$\vdots$

$= w^t h(0) + c\left[\sum_{k=0}^{t} w^k x(t-k)\right]$

$= w^t h(0) + c\,w^t x(0) + c\underbrace{\sum_{k=1}^{t} w^k x(t-k)}_{0}$

$x(i) = 0 \quad i = 1 \ldots t$

$$\boxed{h(t) = cw^t x(0)}$$

Class 13
18/9/23

BIBO Stability with linear Activation fns. — Seen

Now

Non-linear fns

Sigmoid	Tanh	Relu
Saturates in limited steps, regardless of w	sensitive to w but eventually saturates	sensitive to w, can blow up all values $< 1 \rightarrow$ Saturate at $> 1 \rightarrow$ output blows

multilayer perceptron, Deep NN [pg 48]

$$Div(n) = \underbrace{D}_{\text{loss of}}\Big[\underbrace{f_N\big[w_{N-1} f_{N-1}\big[w_{N-2} f_{N-2}\big[\ldots w_0 x_0\big]\big]\big]}_{\text{output of deep NN}}\Big]$$

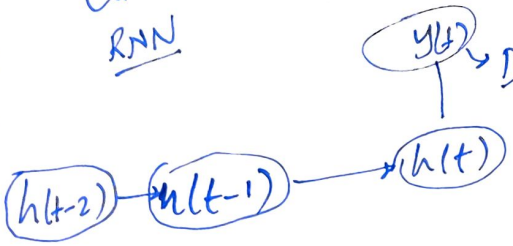
[Bias variance $\longrightarrow$ Cornell]

$\frac{d}{dx}$ [Activation fn] = Always Bounded

Can think of these as a shrinking operator. [pg 52] The Jacobian

because of this, there is vanishing gradient

RNN



all of these are contributing to the loss.

because this is sequential model or infinite

$$\frac{dh_t(w)}{dw} = \frac{dh(n)}{dw}\underbrace{h(h-1)}_{\text{constant wrt } \theta}$$

$$\Rightarrow z_p(t) = \sum_{j=1}^{N_1} w_{jp}^{(1,1)} h_j(t-1) + \sum_{j=1}^{N} w_{jp} x(t) + b_0$$

$$\Rightarrow h_p(t) = f[z_p(t)]$$

Jacobian : derivative that comes after applying an activation fn

$\frac{d}{dw}$ (activation fn) $(\| \|)$ norm = magnitude

$$\frac{\partial h(t)}{\partial z_i(t)} = \text{Jacobian} = f'(z(t))$$

$$|f'(z(t))| \leq \gamma$$

tanh

$$\frac{\partial (h(t))}{\partial h(k)} = \prod_{i=k+1}^{t} W^{11} \nabla_{z'(i)} h(t)$$

$$\left\| \frac{\partial h(t)}{\partial h(k)} \right\| = \prod_{i=k+1}^{t} \| W^{11} \| \, \| \nabla_{z(i)} (h(i)) \|$$

$\leq 1/\gamma$ $\leq \gamma$

multiplication

if thru, we see Vanishing gradi

if greater than, will not be exploding gradients

Vanishing / Exploding gradi 2
will depend on Eigen Value (Spectral) on W^{11} Properties

LSTM

(CEF)

Uses concept of Constant Error Flow for RNN to Create constant Error Carousel (CEC), to ensure gradients don't decay

→ State (S_i) of RNN records info from all previous time step
→ at each time step, old info is morphed by current input.
This has limited capacity, can't store the entire memory. Some old info has to go down to accomodate new info.
→ Similar thing happen when we do backprop in RNN
→ if we make wrong prediction in previous step, the the current step will also be wrong. Can't rectify this in the backprop.

writing on board - Analogy

(Q) How to convert these intutions for RNN and write mathematically?

$$S(t) = f\left[\underbrace{W S(t-1)}_{\text{this should be selective}} + C\, x(t) + b\right)$$

Inp
Selective read
" write
" forget

there exists one output gate that selects the value that's important for us to use in the next step O_{t-1}

O_{t-1} = selector = restricted between 0 and 1
also called as Output gate

$$O_{t-1} = \sigma\left(W_o h_{t-2} + C_o x_{t-1} + b_o\right)$$

$$h_{t-1} = \underbrace{O_{t-1}}_{\text{Vector}} \odot \sigma(S_{t-1})$$

element wise product

Vector

$$S_t = \sigma\left(\underbrace{W h_{t-1}}_{\text{selective portion of } h_{t-1}} + C x_t + b\right)$$

$$i_t = \sigma(\omega h_{t-1} + C_i u_i + b_i)$$

$$i_t \odot \tilde{S}_t$$

Previous State — S_{t-1}

Output gate — $O_{t-1} =$

Selective write — $h_{t-1} = O_{t-1} \odot \sigma(S_{t-1})$

current (temporary) State —

Input State —

Selective Forget — $S_t = S_{t-1} + f_t \odot \tilde{S}_t$

Forget Gate — to forget some parts of S_{t-1}

$$f_t = \sigma(\omega_f h_{t-1} + C_f u_t + b_f)$$

$$S_t = f_t \odot S_{t-1} +$$

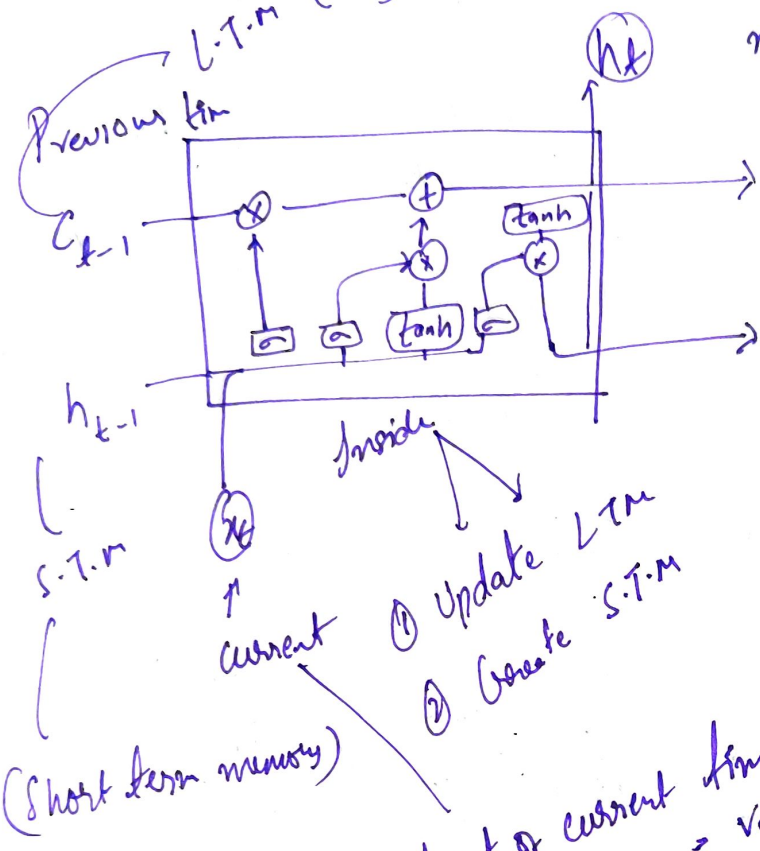
temporary state
Input State
Output State

final output

$$h_t = O_t \odot \sigma(S_t)$$

LSTM

L.T.M (long term memory)



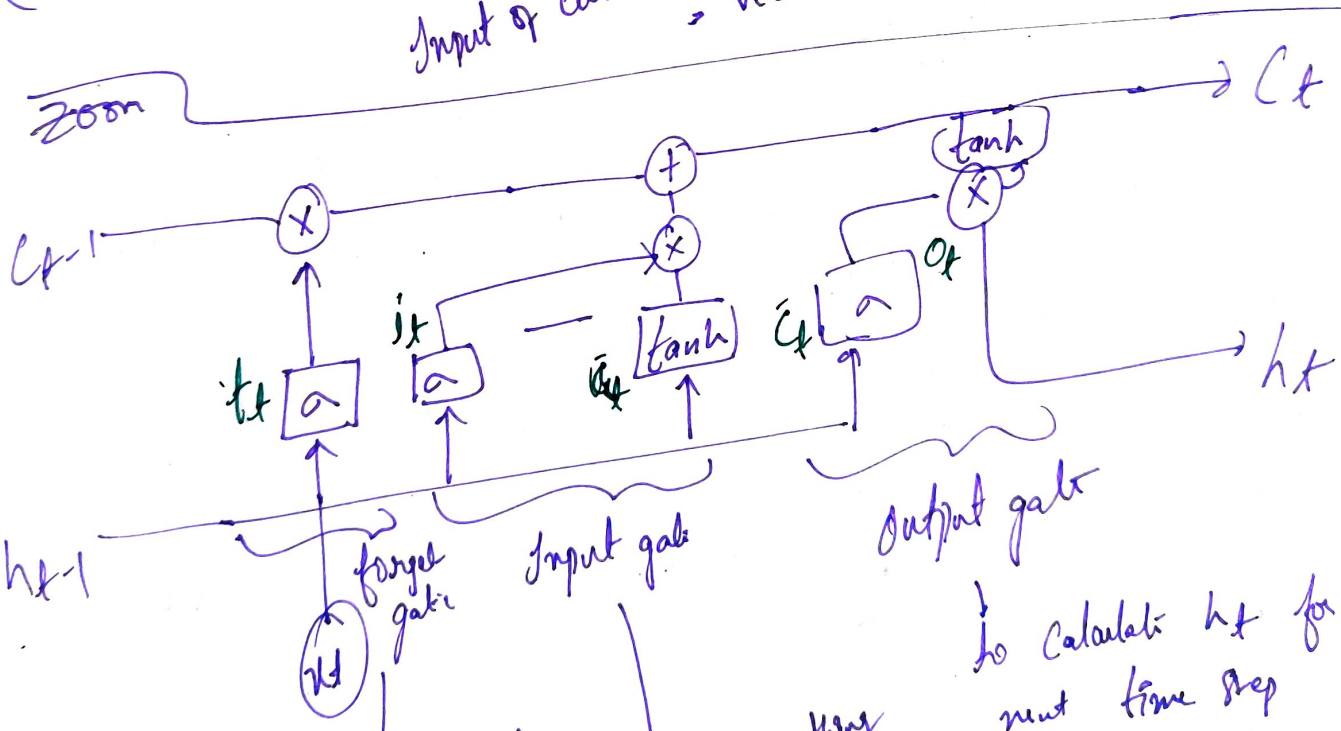
these both are vectors [3D]

dim(h_t) = dim(C_t)

S.T.M

(Short term memory)

Zoom



the gates:-

② Input
① Forget
③ output

Forget Gate

multilist encoded.

f_t it $\bar{C}_t$ → candidate cell state
forget input O_t output } vector

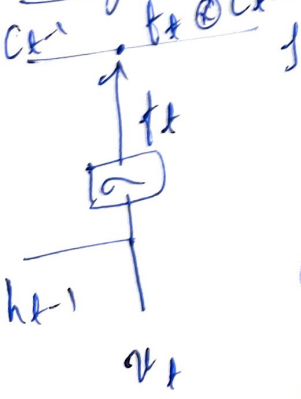
$\dim(f_t) = \dim(\bar{C}_t) = \dim(f_t) = \dim(o_t)$

Pointwise operation = ⊕, ⊗, tanh

↳ output = vector

a ,tanh : neural networks
collection of multiple nodes with
Sigmoid. sigmoid and tanh activations.

Forget Gate

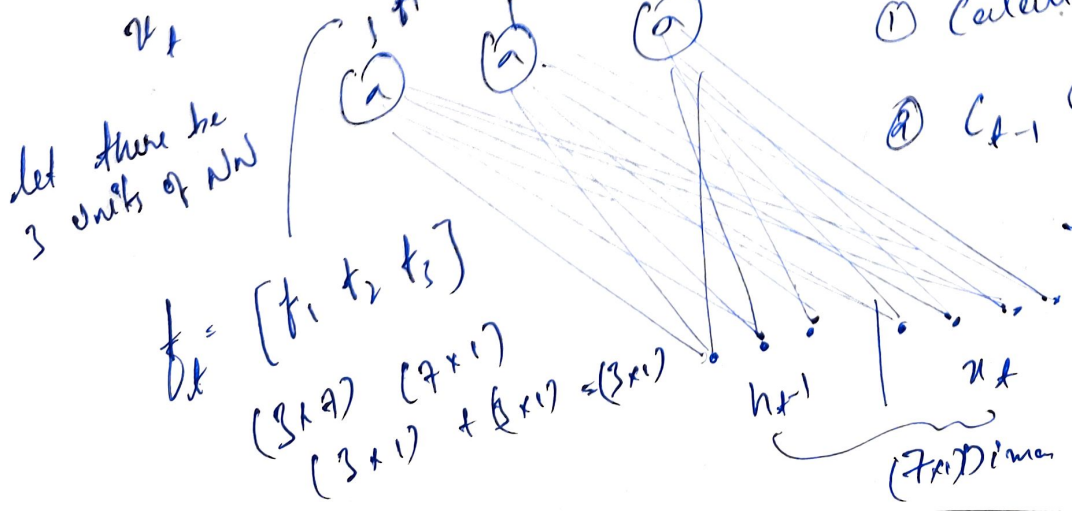


Input = x_t : current stat
h_{t-1} : previous hidden stat

Output : Can remove something from
f_t Cell state.

Stages:
① Calculate f_t
② $C_{t-1} \otimes f_t$

let there be 3 units of NN



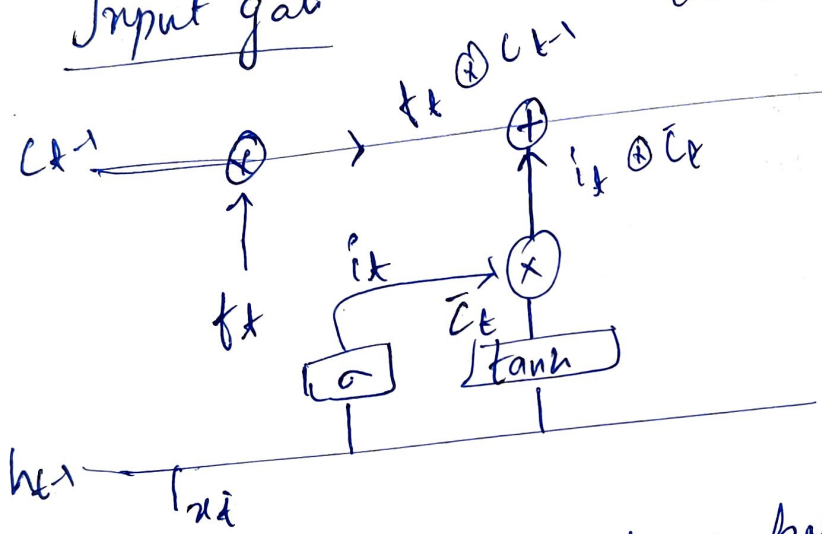
$$f_t = \sigma\left[W_f \underbrace{[h_{t-1}, x_t]}_{4\times 1} + b_f \right] \rightarrow (3\times 1)$$

(1×4) $\quad (3\times 1)$

$[h_{t-1}, x_t]$ → concatenation of 2 vecto

$f_t \circledast C_{t-1} \rightarrow 3\times 1$

Input gate — adds new info to the data



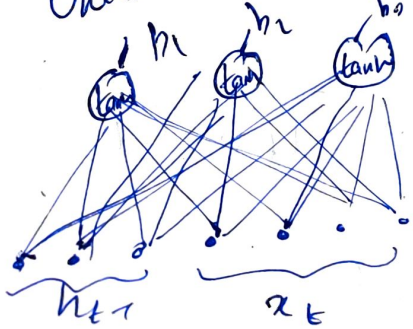
Stages:

① $\tilde{C}_t$: Candidate cell state → based on current input and previous cell state, we add new info that's needed to be added.

② i_t : decide what item needed to be added to final current

③ C_t : Calculate the final cell state.

Units from NN ⇒ $\tilde{C}_t$



$\rightarrow W_R \quad (3\times 4)$

$\tanh\left[W_C (h_{t-1}, x_t) + b_C\right]$

$$\bar{C}_t = \tanh\left[W_c (h_{t-1}, x_t) + b_c \right]$$

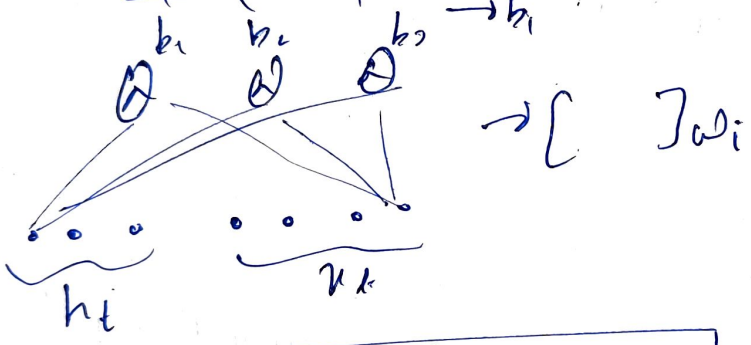
(3×4) (4×1) (3×1)

(3×1)

Potential important info $= \bar{C}_t$

i_t filter from $\bar{C}_t$, next step, calculating i_t

activation – Sigmoid $\rightarrow i_t$



$\rightarrow [\ ] w_i$

$$i_t = \sigma\left[W_i [h_{t-1}, x_t) + b_i \right]$$

(3×4) (4×1) (3×1)

(3×1)

3×1

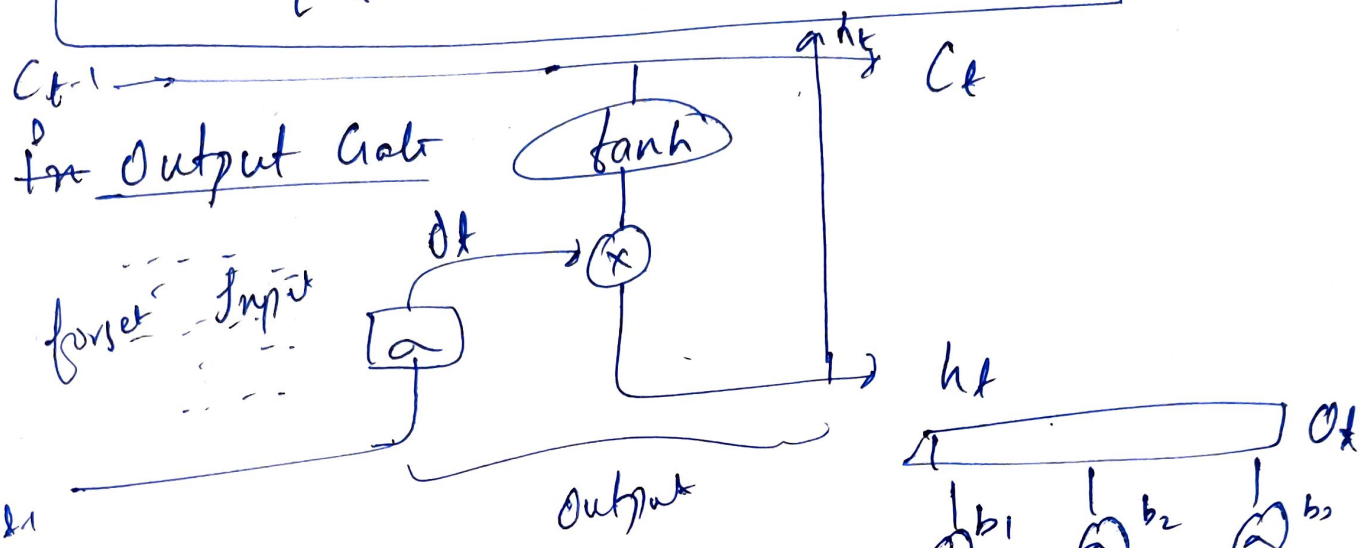
$i_t \circledast \bar{C}_t \rightarrow \bar{C}_t^*$ → filtered Candidate cell State

(3×1) (3×1)

Now, finally $C_t = \left[f_t \circledast C_{t-1} \right] \oplus \left(i_t \circledast \bar{C}_t \right)$

$= C_t$

$$C_t = [f_t \otimes C_{t-1}] \oplus [i_t \otimes \tilde{C}_t]$$



For Output Gate

$$o_t \otimes \tanh(C_t)$$

$$o_t = \sigma[W_0 [h_{t-1}, x_t] + b_0]$$

3×7 · 7×1 → 3×1 → 3×1

(3×1)

σ (3×1) = (3×1)

$$h_t = o_t \otimes \tanh(C_t)$$

3×1 → 3×1

3×1 → 3×1

Class 14
25/9/23

mid term

(Q1) Cross entropy is not a very stable loss to use it

$$H(P,Q) = -\sum P(x) \log Q(x)$$

(Q4) RNN only forward pass

[Solutions on Monday]

Module 15-3

LSTM Can't handle exploding gradients. But can handle Vanishing gradients.

(Q) How they control Vanishing gradients?

Ans)

$$S_t = i_t \odot \tilde{S}_t + f_t \odot S_{t-1}$$

$$h_t = S_t \odot O_t$$

$\odot$: element wise Product

backprop is very complicated to flow, thus blocks the gradient

* If S_{t-1} is not contributing means f_t might be 0 ! so in this case S_{t-1} is not responsible for S_t's Vanishing gradients

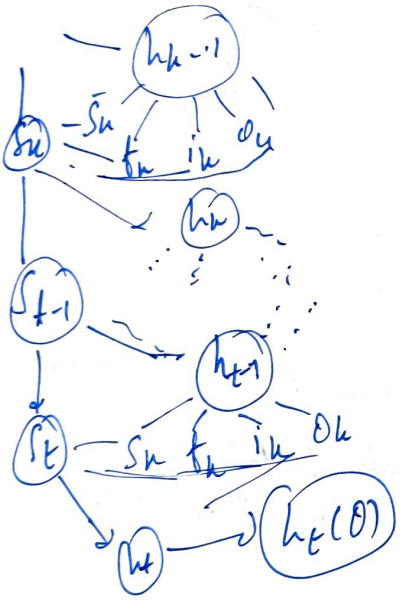
→ V.G happens in RNN when gradients flow in odd Paths is 0, Cause we are multiplying with power of w

Gradient will explode depending on the power of w.

→ In LSTM there exists one path where gradient won't vanish or explode.

(Q) is $\frac{\partial L_t}{\partial S_k}$ alive or Vanished?

Ans) There are many paths to L_t from S_k, so should calculate the perfect path where gradients doesn't vanish.



~~$h_t = a(S_t) \odot O_t$~~

$$h_t = a(S_t) \odot O_t$$

$$\frac{\partial h_t}{\partial S_{t-1}} = a'(S_{t-1}) \times (O_t)$$

|

this will be a Jacobian Matrix

$$\frac{\partial h_k}{\partial S_{t-1}} = \begin{bmatrix} \frac{\partial h_{1,1}}{\partial S_{t-1}} & 0 & \cdots & 0 \\ 0 & \frac{\partial h_{2,2}}{\partial S_{1,2}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \frac{\partial h_{(t,t)}}{\partial S_{(t,t)}} \end{bmatrix}$$

$$= D\left[O_t \odot a'(S_t)\right]$$

diagonal

$$\frac{\partial S_t}{\partial S_{t-1}} = ?$$

$$S_t = f_t \odot S_{t-1} + i_t \odot \tilde{S}_t$$

$$\tilde{S}_t = \left[W h_{t-1} + V x_t + b \right]$$

$$h_{t-1} = \sigma(S_{t-1}) \odot O_{t-1}$$

Encoder / Decoder Model, Attention Mechanism

reading entire sentence — encoder part
generating entire sentence in another lang — decoder
needs 2 RNN's.

Image Captioning — needs 2 blocks.

attention mechanism :- when encoding info in one language,
do we need entire info to translate ? No we can
translate as on going process. A.m does this work
give priority for which part to give

Class 15
5/10/23

Mid Sem paper discussion

(Q1) multilabel problem classification
multihot vector representation.

4 classes
if class 1, 3 → true for some, then its multihot
vector representation = [1, 0, 1, 0]

$l_{cross\ ent} \geq l \log l$

S.T:- not a good loss fn.

k: total no of classes.
L = example labels are true

$l \leq k$

$l_{ce}(y, \hat{y}) \geq l \log l$

$y = [y_1, \dots y_l]$ $y_i \in (0, 1)$

$l_{ce} = -\sum_{i, y=1} \log f_{y_i}$

$l_{ce} = -\frac{l}{l} \sum_{i, y=1} \frac{1}{l} \log f_{y_i}$

convex fn

$$\sum_{y_i=1}^{L} \frac{1}{L} \log f_{y_i} \leq \log \left[\sum_{y_i=1}^{L} \frac{1}{L} f_{y_i} \right]$$

$$\sum \lambda_i f(x_i) \leq f\left(\sum \lambda_i x_i\right)$$

$$\geq -L \log \left[\sum_{y_i=1}^{L} \frac{1}{L} f_{y_i} \right]$$

$$= -L \log \frac{1}{L} \sum_{y_i=1}^{L} f_{y_i}$$

$$\geq -L \log \frac{1}{L}$$

$$= L \log L$$

---

② $w(n) = w(n-1) - \eta \nabla_w J(w(n-1))$

weight decay - multiply each parameter w by $(1-\epsilon)$

Should show this is same as L_2 Reg

$w'(n) = (1-\epsilon)\, w(n)$

$w'(n) = (1-\epsilon)\left[w(n-1) - \eta \nabla_w J[w(n-1)] \right]$

$= w(n-1) - \eta \left[\underbrace{(1-\epsilon) \nabla_w J(w(n-1))}_{\text{gradient of some (new) objective function}} + \frac{\epsilon}{n} w(n-1) \right]$

to get objective fn → we need to integrate it

$$J_{new}(w) = (1-\epsilon)\, J(w) + \frac{\epsilon}{2n} \|w\|^2$$

[Can we take 2nd order of taylor series expansion of w ? No
because w is parameter]

(Q6) curve fitting problem

(x_i, y_i) ⟶ given

$x_i, y_i \in R$

linear mapping $f: x \to y$

$$\boxed{y = w_0 + w_1 x}$$

<u>without noise :-</u>

objective fn :-

$$L = \frac{1}{N} \sum_{i=1}^{N} [y_i - w_0 - w_1 x_i]^2$$

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

$\epsilon \sim N[0, \sigma^2]$

$$\Sigma = \frac{1}{N} \sum_{i=1}^{N} (y_i - w_0 - w_1 \tilde{x}_i)^2$$

$$= \frac{1}{N} \Sigma [y_i - w_0 - w_1 x_i - w_1 \epsilon_i]^2$$

$$= \frac{1}{N} \Sigma (y_i - w_0 - w_1 x_i)^2 + \frac{1}{N} w_1^2 \Sigma \epsilon_i^2 - \frac{2}{N} w_1 \Sigma (y_i - w_0 - w_1 x_i)$$

$$\tilde{L} = L + \frac{1}{N} w_1^2 \sum \epsilon_i^2 - \frac{2}{N} w_1 \sum \epsilon_i (y_i - w_0 - w_1 x_i)$$

$$E[\tilde{L}] = L + \frac{1}{N} w_1^2 \sum_{i=1}^{N} \underbrace{E[\epsilon_i^2]}_{\sigma^2} - \frac{2}{N} w_1 \underbrace{\sum E[\epsilon_i]}_{0} (y_i - w_0 - w_1 x_i)$$

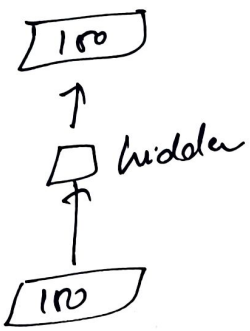
$$= L + \frac{w_1^2}{N} \sum_{i=1}^{N} \sigma^2$$

$$= L + w_1^2 \sigma^2 \longrightarrow L2 \text{ regularization}$$

---

(8)

A



B

Advantages of

A over B

B over A

A over B

① number of weights are more [disadvantage]
→ 16000

② nodes are lesser

B over A

1000 + 1000

① less number of weights. [adv]

(5)

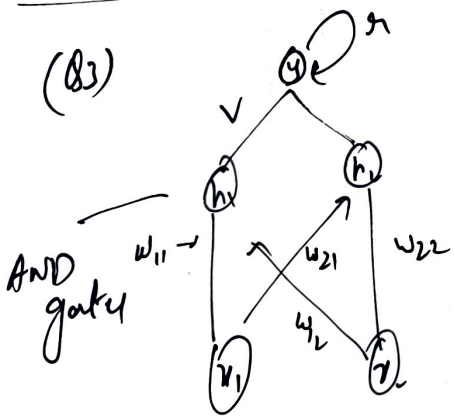
$$h_t = x_t - h_{t-1}$$
$$= x_t - (x_{t-1} - h_{t-2})$$
$$= x_t - x_{t-1} + h_{t-2}$$
$$= x_t - x_{t-1} + x_{t-2} - h_{t-3}$$
$$\vdots$$

What does it do?

$$h_t = \begin{cases} \sum_{j=1}^{t/2} x_{2j} - \sum_{j=1}^{t/2} x_{2j-1} & \text{if } t = \text{even} \\ \sum_{j=1}^{(t+1)/2} x_{2j-1} - \sum_{j=1}^{(t-1)/2} x_{2j} & \text{if } t = \text{odd} \end{cases}$$

$$y_t = \sigma[10\, h_t]$$

(Q3)



find parameters of RNN that decides if the 2 inputs are same or not.

$i_1 = x_1(1)\ x_2(2) \cdots$

$i_2 = x_2(1)\ x_2(2) \cdots$

are $i_1 = i_2$

same using RNN?

both func f they are same if $y(t-1)=1$ and their current bits are same.

$$h^{(t)} = \phi(w x^{-(t)} + b)$$

$$y^t = \begin{cases} \phi(v^T \cdot h^{(t)} + r y^{t-1} + c) & t > 1 \\ \phi(v^T \cdot h^{(t)} + c_0) & t = 1 \end{cases}$$

find v, r, c, w, b

$h_1(t) = x_1(t) \cdot x_2(t)$

$h_2(t) = \bar{x}_1(t) \cdot \bar{x}_2(t)$

$$w = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

at $t > 1$

$y(t) = h_1(t) + h_2(t)$

$v = [1, 1]$

$c_0 =$ threshold $= 1$

at $t > 0$

$y(t) = y(t-1)\,(h_1(t) + h_2(t))$

$$r = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$r = 1$

$c = -2$

(Q7) design RNN architecture that starts emitting 0 once it sees the first 0

input : 1110101

output : 1110000

h_t → should record if it received 0 or not

Truth table :-

h_{t-1}	x_t	h_t
0	0	1
0	1	0
1	0	1
1	1	1

Todays class
- encoder
- decoder
- attention mechanism

12/10/23

Variational Autoencoders

$\hat{p}_j = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_j$ — l = neuron → Sparse

$\Omega(\theta) = \sum p \log \frac{p}{\hat{p}_j}$ +

$L(\theta)$ = Squared error diff between x_i and output

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

$\Omega \rightarrow$ vector

$$\frac{\partial \hat{p}}{\partial w} = \left[\frac{\partial \delta_1}{\partial w} \quad \frac{\partial \delta_2}{\partial w} \quad \cdots \quad \frac{\partial \delta_n}{\partial w} \right]$$

$$\frac{\partial \hat{p}_1}{\partial w_{ij}} = \frac{\partial \left[\frac{1}{m} \sum^{m} g(w^T x_i + b_j) \right]}{\partial w}$$

$$= \frac{1}{m} \sum \frac{\partial [g(w^T x_i + b_j)]}{\partial w_{ij}} = \frac{1}{m} \sum g'(w_{ij}^T x_i + b_j) x_{ij}$$

$$\frac{\partial \hat{p}_j}{\partial w} = x_i [g'(w^T x_i + b)]^T$$

281 deep learning Stanford - Justin Johnson

lg may 2 202-

Sequence to Sequence with RNNs Self attention

Input → Seq : $x_1, \dots x_T$

Output → Seq : $y_1, \dots y_T$

encoder = $h_t = f_w [x_t, h_{t-1}]$

Initial decoder state → s_0

context vector → C [often $C = h_T$]

→ Cant achive parallism using LSTM, due to feedback going back to the model.

→ e_{ij} capturing contribution of j^{th} input in predicting i^{th} output

h_is → encoder states. (hidden state)

s_t → hidden state of decoder

context vector = $C_t = \sum_{i=1}^{T} a_{t-1} h_i$

Attention is all you need
- Vikas Sindhwani

class 18

16/10/23

Paper's

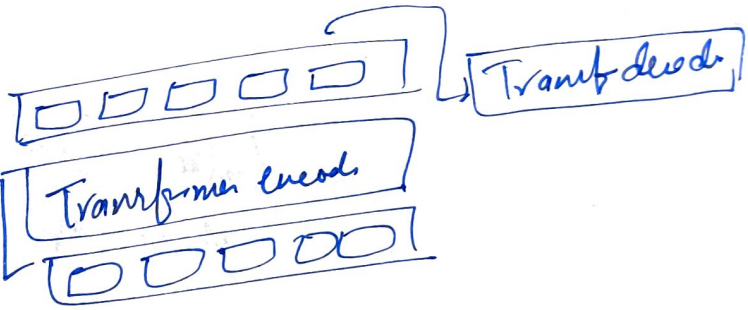
→ Attention is all you need [By google Researcher] 2017

→ Opportunities and Risks of Foundation models.
Survey model.

" Image Captioning using Transformer "

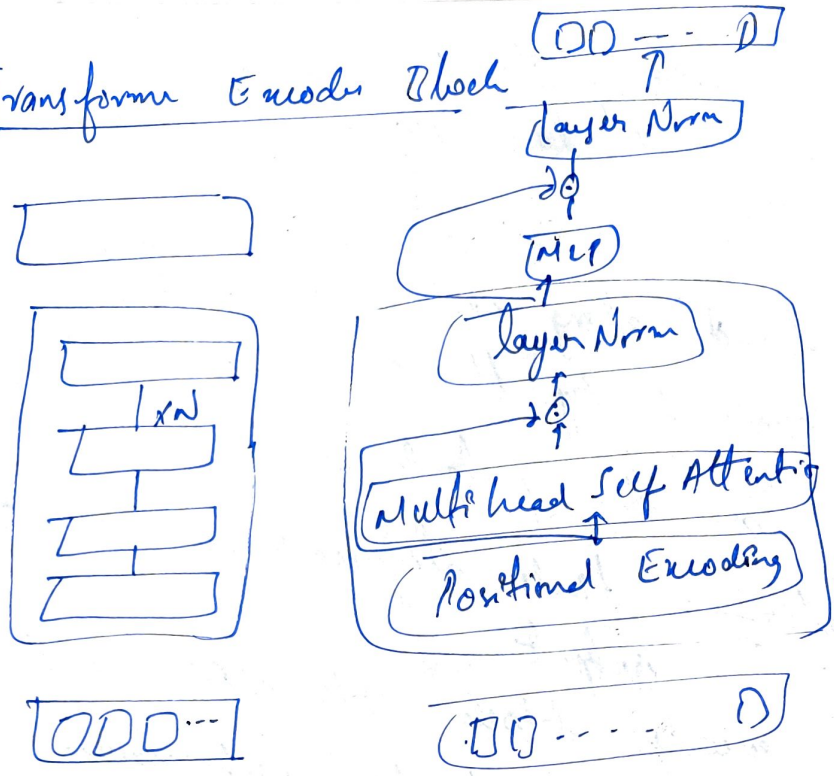
Input → Image
Output → words seq

Image → CNN → features



decoder → $y_t = T$

Transformer Encoder Block



→ batch Norm → helps in invariant that

→ layer Norm → helps in normalizing output of all layers, take mean, s.d of hidden layer

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} -a_1(m) \\ -a_2(m) \\ \vdots \\ -a_n(m) \end{matrix} \qquad \mu = \frac{1}{n}\sum_{i=1}^{n} a_i(m)$$

$$\sigma^2 = \frac{1}{n}\Sigma\,[a_i(m) - \mu]^2$$

no seperate parameters like in batch Norm here, output normalization is directly done and the forward work is done

Main Idea is self attention here.

even though we are doing positional encoding, we still do self attention and layer Norm

decoder block

alignment is found along with each output.

Violet box is produced by the encoder.

x_1 x_2 x_3 ... are the outputs.

multi-head attention → finds alignment between the output

kinda similar to encoder block with violet addition

This works better than ResNet / matches performance

Vision Transformers → Skipped #

GPT's are the decoders of these transformers.

Latent Semantic Indexing (LSI) / (LDA)

Idea for giving vector representation for words

frequency of occurance

words ==

words	D1	D2	...	Dn

D = document

"Document Frequency Matrix"

do singular value decomposition → we get singular value → using these singular values we get matrices / vectors of words

(Q) what is one hot Representation? — Search on GPT

SVD doesn't capture relation b/w words well
computational cost is high — $O(nm^2)$

Word 2Vec

associate words to point in the space
NN — Single hidden layer network
Supervised model.

SKIP graph — another architecture

Auto encoder — done #

Remaining
CNN, Variants
ResNets.
GANS
Diffusion models

CNN

→ learning filters that detect different things.
→ CNN finds them automatically and finds out and learns
outputs.
→ when a neuron maximum fires, when input that passed is highly correlated with the output neuron.

$x = (x_1 \dots x_w)$

Kernal: $w = (w_1 \dots w_\omega)$

$$(x * w)_i = \sum_{j=1}^{\omega} x_{i-1+j}\, w_j = \left[x_1 \dots x_{1+\omega-1}\right] w$$

$(1,2,3,4) * [3,2] = [3+4, 6+6, 9+8] = (7, 12, 17)$

26/10/23

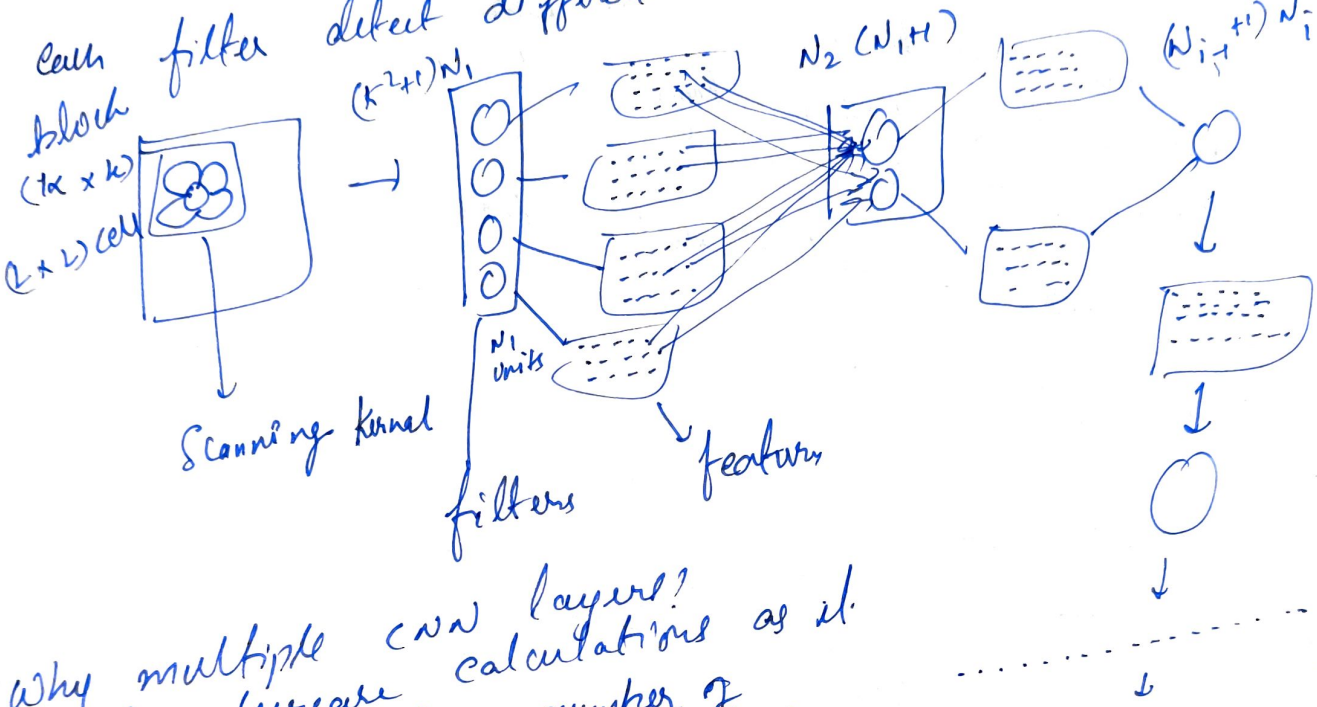
CNN's

→ Neurons fire when a certain pattern is detected

→ Corelation operation is done

fire → yes if corelation between weight and input exceeds threshold

Each filter detect different kind of pattern

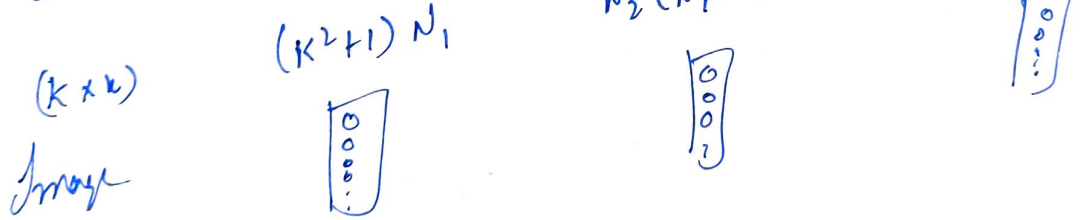


Q) Why multiple CNN layers?

A) to decrease calculations as it will have lesser number of parameters. to detect same task

Q) Why distribution?

A)



Weights

1st layer = $N_1 (k^2+1)$ weights.

2nd layer = $\left[\left(\frac{k}{L}\right)^2 N_1 + 1\right] N_2$ weights

total parameters = $O\left[L^2 N_1 + \left(\frac{k}{L}\right)^2 N_1 N_2 + N_2 N_3 + \cdots\right]$

receptive field = the input that makes neuron fire

2/11/23 **Diffusion Models**

generative model.
trying to learn distribution of some data
→ has 2 components - Encoder / Decoder
→ takes sample data and maps it to series of intermediate latent variables

$x \longrightarrow z_1 \dots z_T$

Sample data — latent variable → noise

Adds noise to image and reconstruct original image

Encoder :-

$z_1 = \sqrt{1-\beta_1}\, x + \sqrt{\beta_1}\, \epsilon_1$

noise / Standard Normal distribution

$z_t = \sqrt{1-\beta_t}\, z_{t-1} + \sqrt{\beta_t}\, \epsilon_t$

If $\sqrt{\beta_t}$ > high then,

$\epsilon_i = 0 \Rightarrow$ mean is 0

mean of z_t is more closer to 0 as we are increasing
∴ the mean is getting closer to 0

Mean of z_1 = ? = If x is fixed the distribution is gaussian.

$(\sqrt{1-\beta_1})\, x$ = mean

Covariance = $(\beta_1 \cdot$ Identity Matrix$)$

$\mathcal{N}$ → Normal distribution

Forward eqn:-

$$q(z_1|x) = \mathcal{N}(\sqrt{1-\beta_1}\, x, \beta_1 I)$$

$$q(z_t|z_{t-1}) = \mathcal{N}[\sqrt{1-\beta_t}\, z_{t-1}, \beta_t I)$$

thus forms markov chain because $P(z_t)$ depends only on value of immediate preceding variable z_{t-1} with sufficient steps in T; $q(z_T|x) = q(z_T)$

becomes standard Normal distribution

Joint distribution of $z_1 \cdots z_T$ =

$$q(z_1, z_2, \cdots z_t|x) = q[$$

$$z_1 = \sqrt{1-\beta_1}\, x + \sqrt{\beta_1}\, \epsilon_1 \quad \text{—(1)}$$

$$z_2 = \sqrt{1-\beta_2}\, z_1 + \sqrt{\beta_2}\, \epsilon_2 \quad \text{—(2)}$$

= Replace (1) in (2)

$$z_2 = \sqrt{1-\beta_2}\left[\sqrt{1-\beta_1}\, x + \sqrt{\beta_1}\, \epsilon_1\right] + \sqrt{\beta_2}\, \epsilon_2$$

$$z_2 = \sqrt{(1-\beta_2)(1-\beta_1)}\, x + \sqrt{\beta_1(1-\beta_2)}\, \epsilon_1 + \sqrt{\beta_2}\, \epsilon_2$$

$$E[z_2] = \sqrt{(1-\beta_2)}\sqrt{(1-\beta_1)}\, x = \boxed{\sqrt{\alpha_2}\, x = E(z_2)}$$

$$\text{Var}(z_2) = \beta_1[1-\beta_2] + \beta_2 = \beta_1 - \beta_1\beta_2 + \beta_2$$

$$\text{Var}(z_2) = \beta_1 + \beta_2 - \beta_1\beta_2 = 1 - (1-\beta_1)(1-\beta_2)$$

$$\boxed{\text{Var}(z_2) = 1 - \alpha_2}$$

Marginal and conditional Distribution :

$q_\phi(z_t) = \int q(z_t|x)\, q(x)\, dx$: Marginal

$q(z_{t-1}|z_t) : q(z_{t-1}|z_t) = \dfrac{q(z_t|z_{t-1})\, q(z_{t-1})}{q(z_t)}$

↓

conditional — Intractable

done by sampling ↳ as we can't compute marginal distribution of $q(z_{t-1})$

Useful when training decoder.

couldn't find reference for page 12 → 13.

Page 13, ② conditional distribution is used in marginal.

Pg 17

Simplification of log term :-

$$\log\left[\frac{p(x, z_{1:T}|\phi_{1:T})}{q(z_{1:T}|x)}\right]$$

$$= \log\left[\frac{p(x|z_1,\phi_1)\prod p(z_{t-1}|z_t,\phi_t)\, p(z_T)}{q(z_1|x)\prod_{t=2}^{T} q(z_t|z_{t-1})}\right]$$

$$= \log\left[\frac{p(x|z_1,\phi_1)}{q(z_1|x)}\right] + \log\left[\frac{\prod p(z_{t-1}|z_t,\phi_t)}{\prod q(z_t|z_{t-1})}\right] + \log p(z_t)$$

(1)

Reverse probability.

We observe that:

$$q(z_t|z_{t-1}) = q(z_t|z_{t-1}, x)$$

$$q(z_t|z_{t-1}) = \frac{q[z_{t-1}|z_t, x]\, q(z_t|x)}{q(z_{t-1}, x)} \quad \text{(*)}$$

Using (*) in (1):

$$\log\left[\frac{p(x, z_{1\cdots T})|\phi_{1\cdots T}}{q(z_{1\cdots T}|x)}\right] = \log\left[\frac{p(x|z_1, \phi_1)}{q(z_1|x)}\right] + \log\left[\frac{q\, p(z_{t-1}|z_t)}{\pi[p(z_{t-1}|z_t, x)]}\right] + \frac{q(z_{t-1}|x)}{q(z_t|x)}\Bigg] + \log P(z_t)$$

$$\leq \log p[x|z_1, \phi_1] + \log \frac{\pi p[z_{t-1}|z_t, \phi)}{q[z_{t-1}|z_t, x]} + \log \frac{P(z_T)}{q[z_T|x]}$$

(keep this out of optimization as it is normal distribution)

$$ELBO(\phi_{1\cdots T}) = \int q[z_{1\cdots T}|x] \log \frac{p(x, z_{1\cdots T}|\phi_{1\cdots T})}{q(z_{1\cdots T}|x)} \cdot dz$$

$$= \int q(z_{1\cdots T}|x) \left[\log p(x|z_1, \phi_1) + \log \frac{\pi_t p[z_{t-1}|z_t, \phi_t)}{\pi[q][z_{t-1}|z_t, x, \phi_t]}\right] dz$$

(this is conditional prob. for q, z_1)

$$= \mathop{E}_{q(z_{1\cdots T}|x)}\left[\log p(x|z_1, \phi_t)\right] + \sum_t \log \frac{p(z_{t-1}|z_t|\phi_t)}{q[z_{t-1}|z_t, x)} dz$$

$$= E_{q(z_1|x)}\left[\log p(x|z_1,\phi)\right] + \sum \int q(z_{1\ldots T}|x)$$

$$\log \frac{p(z_{t-1}|z_t,\phi_t)}{q(z_{t-1}|z_t,x)} dz_{1\ldots T}$$

$$= E_{q(z_1|x)}\left[\log \frac{p(x|z_1,\phi_1)}{} \right] - \sum \int q(z_{1\ldots T}|x)$$

$$\log \frac{q(z_{t-1}|z_t,x)}{p(z_{t-1}|z_t,\phi)} dz_{1\ldots T}$$

$$= E_{q(z_1|x)}\left[\log p(x|z_1,\phi_1)\right] + \sum \int q(z_{t-1}|z_t,x)$$

$$\log \frac{q(z_{t-1}|z_t,x)}{p(z_{t-1}|z_t,\phi)} dz_{t\ldots T}$$

this is KL divergence between 2 elements

$$= E_{q(z_1|x)} \log p(x|z_1,\phi_1) - \sum KL\left(q(z_{t-1}|z_t,x)\right.$$
$$\left.\| p(z_{t-1}|z_t,\phi_t)\right)$$

16/11/23 V.A-E

	Encoder	Decoder
Auto encoder	$x \rightarrow z$	$z \rightarrow x$
Variational Auto encoder	$x \rightarrow P[z\|x]$	$z \rightarrow P(x\|z)$

" Skipping relationships pg 12 · lecture 21 · pdf "

$\hat{x}$ [reconstruction]

↑

Decoder $P_\phi(x|z)$

↑

Encoder $Q_\theta(z|x)$

↑

x (data)

→ latent variable are distributed as standard Normal distribution.

$z \sim N[\mu(\theta, x), \Sigma[\theta, x])$

Parameters of multivariate Normal distribution.

try to enforce this the same as standard Normal distribution by minimizing KL divergence.

~~KL divergence btw~~

$\hat{x}_t$

$P_\phi(x|z)$

Sample z

Sample

μ Σ

$Q_\theta(z|x)$

x_t

θ are the parameters of encoder network

ϕ parameters of decoder network

encoder and decoder both are neural networks.

objective fn of decoder → maximize $p(x_i)$ → likelihood Maximization

$$p(x_i) = \int p(z)\, p(x_i|z)\, dz$$

$$= + \mathbb{E}_{z \sim q_\phi(z|x_i)} \left[\log p_\theta(x_i|z) \right]$$

(Q) Find KL div b/w 2 distributions Pg 29 Lec 21 (in exam)

VAE Approximate it using a single sample.

using $\frac{1}{N} \sum_{i=1}^{N} f(x)$ → generally

$$\mathbb{E}[f(x)] \approx f(x)$$ → in VAE

finding expectation is not tractable.

distribution that is outputted by encoder.

decoder network predicts mean with Identity covariance matrix.

find log of it

no need to find expectation as we are Sampling with single fn. → $f(x)$

$$\log [P(x = x_i|z)] = C - \frac{1}{2} \| x_i - \mu(z) \|^2$$

(gaussian distribution) (C can be ignored) (output of decoder network (mean of distribution))

Overall obj of VAE = pg 31 lec 21

Can update parameters using backprop.

(wrt θ, ϕ)

encoder — decoder.

Problem is not end to end differentiable

output of decoder is not

difficult to implement backprop. as chain rule is hard in this case.

Hence $f_\phi(z)$ is $\begin{cases} \text{non deterministic.} \\ \text{Random fn of } x \\ \text{not continuous fn} \end{cases}$

Reparameterization trick

q_ϕ is also multivariate gaussian distribution - If we want to generate any sample

[multiply of Standard deviation] × [ ]

Inverse × Shift

$z \sim N(\mu, \sigma)$

$x \rightarrow N(\mu, \sigma)$

$x = \mu + \sigma \epsilon$

Inputs → data points.

Sampling Step → ϵ (Sampled z, that can be fed to decoder)

$$\mu(x) + \Sigma(x)\cdot\epsilon$$

"This can be done at Sampling".

- $\epsilon \sim N(0, I)$
- $x \sim N(\mu, \Sigma)$

$\Longrightarrow$ $x = \mu + \Sigma^{1/2}\epsilon$

→ after updating Sampled z, the back prop will be done easily.

→ Randomness is now associated to ϵ Not X that how the problem is taken care of.

Classes done