Camlin Page
Date / /

Improving Deep Neural Networks:

Hyperparameter Tuning

Regularization & Optimization

Train set | "Dev set" validation set | "Test" set.

Train Set: Keep on training on train set use "Dev set" to tune parameter to find which combination work "best".

Test Set: Evaluate your best model on test set & report that accuracy.

Train	Test
70	30

$\mapsto$

Train	Dev	Test
Train	Do	Do

These ratio are pretty hold valid if your data set holds $\approx$ 10 K samples.

In modern big data if you have $\approx$ 10,000,00 then

So in case $\approx$ 10 K dev test is valid enough

$\approx$ 10 K test data set is valid enough

So in case Ratio =

98 %	1 %	1 %
Train	dev	Test

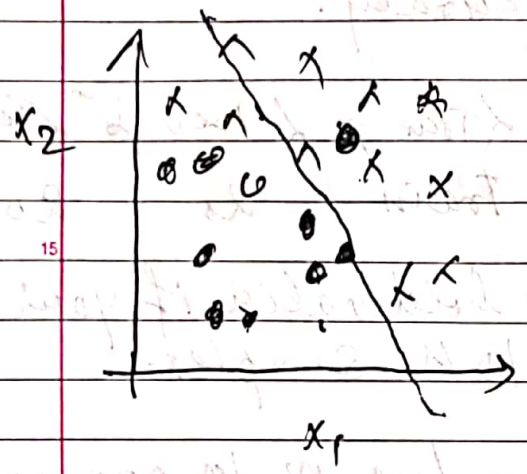
Camlin Page
Date / /

{ Mismatches Train/Test distribution }

(*) Make sure "dev test" & "test" come from same distribution.

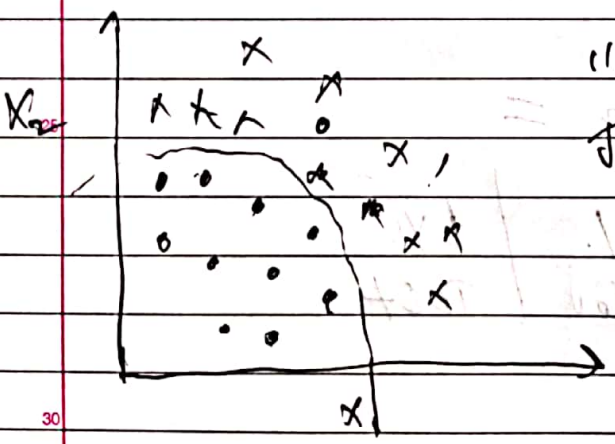
(*) Not having test set is ok, We can report "dev" set accuracy.

{ Bias & Variance }

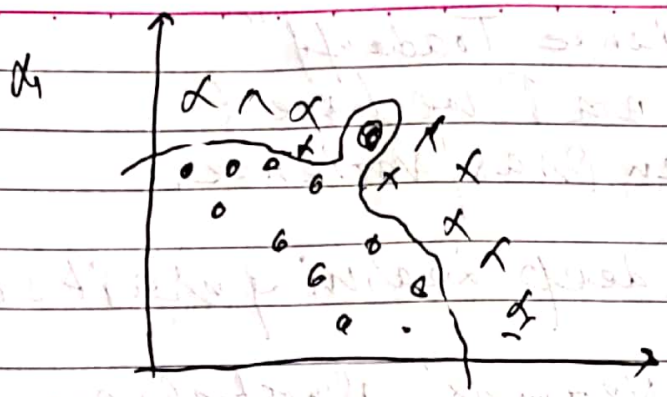


"High Bias"

"Underfitting data"



"Just Right" ✓



"Overfitting"

"High Variance"

"High Variance" x_2

(Cat vs Dog) Base error / Human err / ≈1%.

Train Set Error	1%	15%	15%	0.5%
Dev Set Error	11%	16%	30%	1%
	"High Variance"	"High Bias"	High Bias & High Variance	Low Bias Low Variance →

The above analysis based on Bay Optimal err ≈1%.

High bias? —Y→ Bigger Network, Train longer "(NN Architecture search)"

N

"Training Bigger Network almost always help" to get rid of "bias" problem.

High Variance? —Y→ More data, Regularizat

N

"DONE"

"Bias Variance Tradeoff"

"Previous ML era I we have to choose between Bias / variance.

But in moder deep learning era it's not

Getting "Bigger Network" almost always help "Bias" without hurting variance a lot.

Regularization

Logistic regression: $w \in \mathbb{R}^{n_x}$ $b \in \mathbb{R}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

$$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \quad = L_2 \text{ Regularization}$$

L_1 Regularization $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$

λ = lambd

Neural Network

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|^2$$

$$\|w^{[l]}\|^2 = \sum_{i}^{n^{[l-1]}}\sum_{j}^{n^{[l]}}\left(w_{ij}^{[l]}\right)^2 \quad \leftarrow \quad w: (n^{[l]}, n^{[l-1]}) \text{ (dimension)}$$

"Frobenius Norm" of a matrix.

$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m}w^{[l]}$

$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$

L_2 regularization = Weight Decay

$$W^{[l]} = W^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} W^{[l]} \right]$$

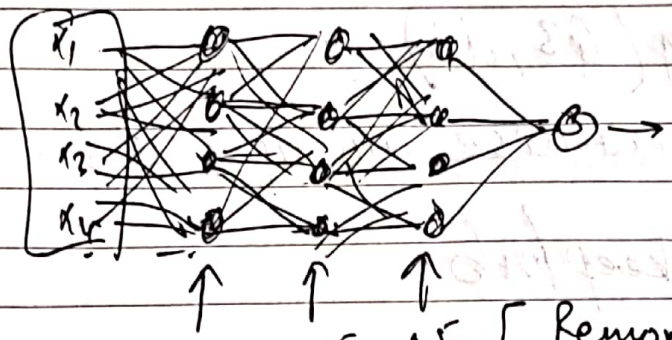
$$= W^{[l]} - \frac{\alpha\lambda}{m} W^{[l]} - \alpha (\text{from backprop})$$

$$= \left(1 - \frac{\alpha\lambda}{m}\right) W^{[l]} - \alpha (\text{from backprop})$$

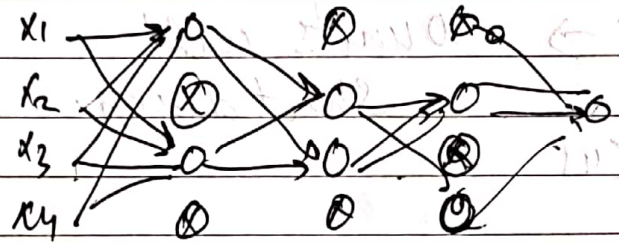
↓

It is like taking the making the $W^{[l]}$ matrix little smaller.

Drop Out Regularization



[Remove the ingoing & out going nodes decided randomly]



{ Diminish Spotted Network } =

We do this for each Traing example.

Implementing Dropout ("Inverted Dropout")

Illustrate with layer $l=3$;

d3 = np.random.rand(a3.shape[0], a3.shape[1])

threshold

random_arr = np.random.rand(3,1)

d3 = random_arr < keepProb | keepProb = 0.6

random_arr =	d3 =
[[0.976060]	[False]
[0.617575]	[True]
[0.44940]	[True]
]	]

Camlin Page
Date / /

~~d3~~ = a3

a3 = np.multiply(a3, d3)
↳ elementwise multiplication

a3 /= a3 / 0.8 ⇒ keep/prob.

Example

50 units of neuron → 10 units will shut down.

$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$
↑ reduced by 20%.

So wt to reduce it a3 /= a3 / 0.8
(Pumpkin)

At test time: given X:

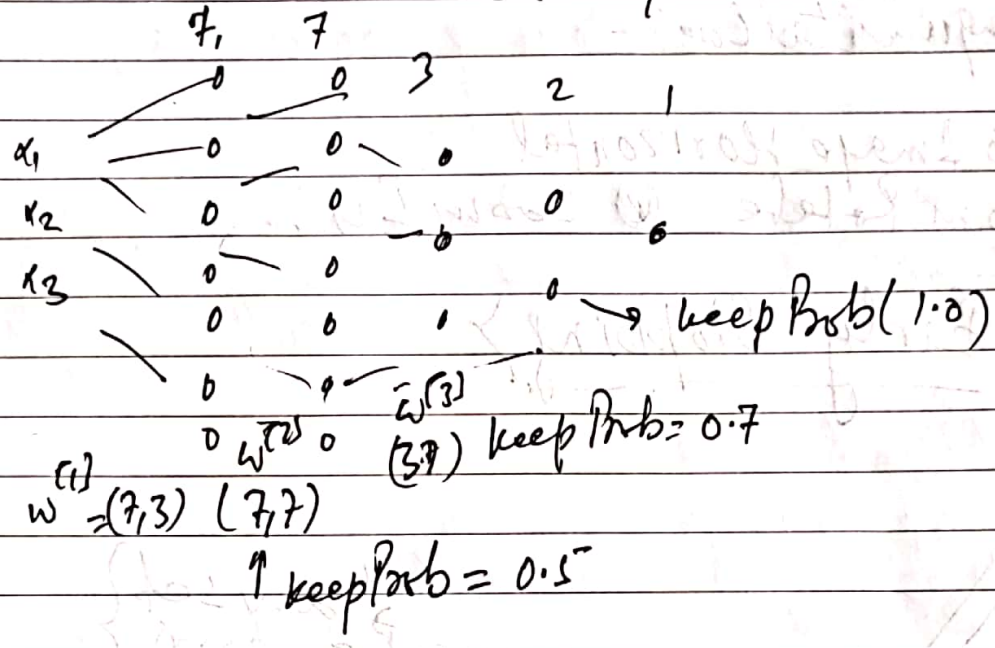
$a^{[0]} = X$ [No DropOut Test Time]

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$
$a^{[1]} = g^{[0]}(z^{[1]})$
$z^{2} = w^{[2]} a^{[1]} + b^{[2]}$

$\hat{y}$ = till

Why Does Drop Out Work?

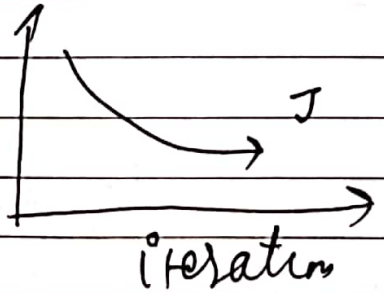
Intuition: Can't rely on any one feature, so have to spread out weights.



In practice we don't apply dropout at input layer hence for that layer keepProb = 1.0 or ~~0.9~~

Downside: Cost function is not well define.

So inorder to check the code validity we FIRST keep keepProb = 1 plot the



then enable dropout.

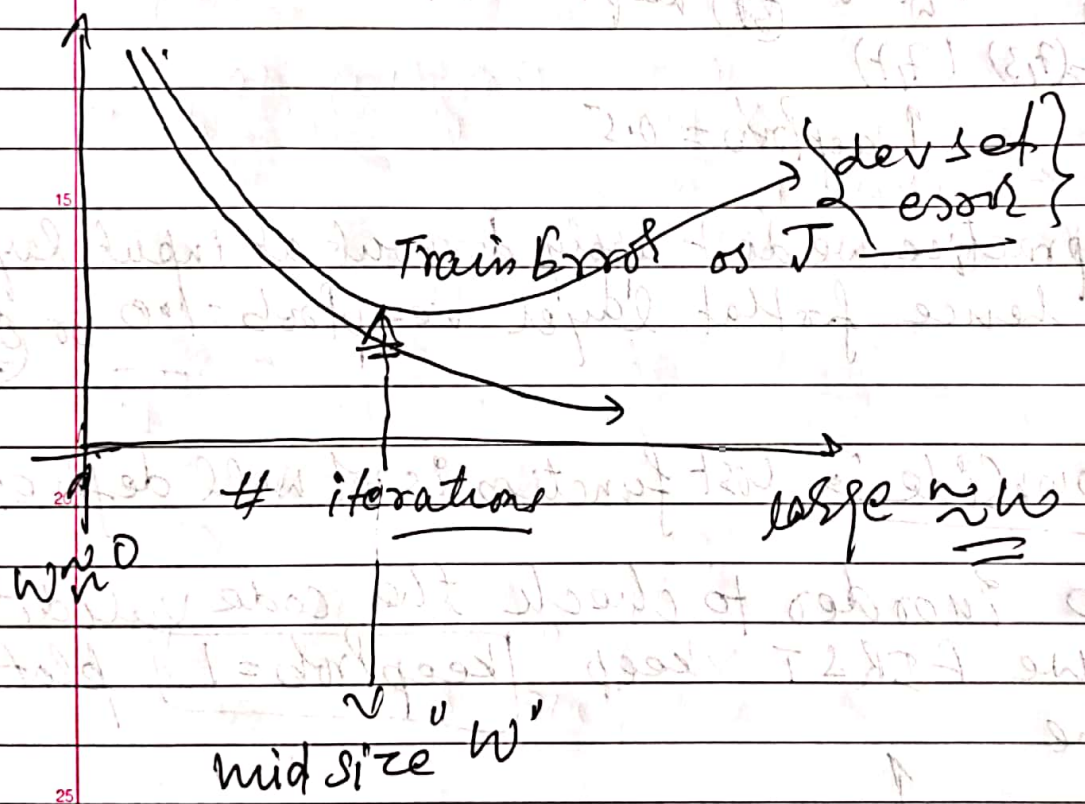
Other Regularization

In order to get more training data :-

Data Augmentation :-

(i) Flip Image Horizontal
(ii) Random Rotate (iii) Zoom In :-

(iv) Early Stopping



Camlin | Page
Date / /

Orthogonalization :- { Segregating Task }

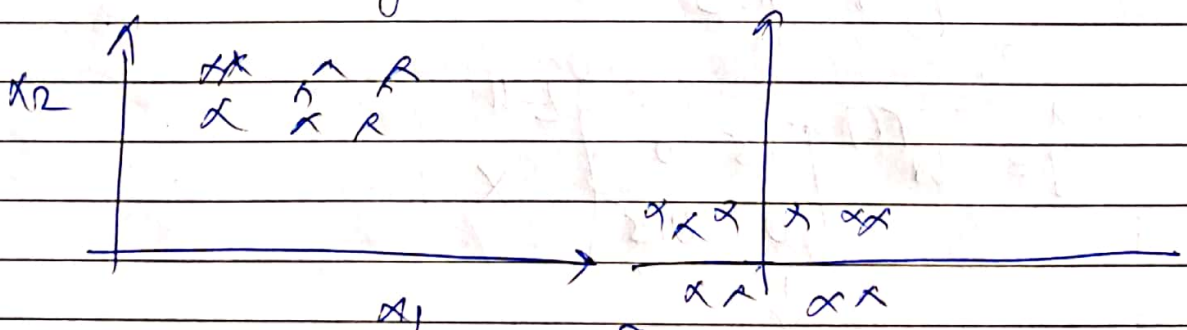
= Optimize Costfunctn J
- B GradientDesc, RM
} J(W, b)

= No Overfitting
- Regularizatn

Early Stopping couple these two hence not encouraging.

{ Normalizing Input }

1. Subtracting mean



$X = X - \mu$

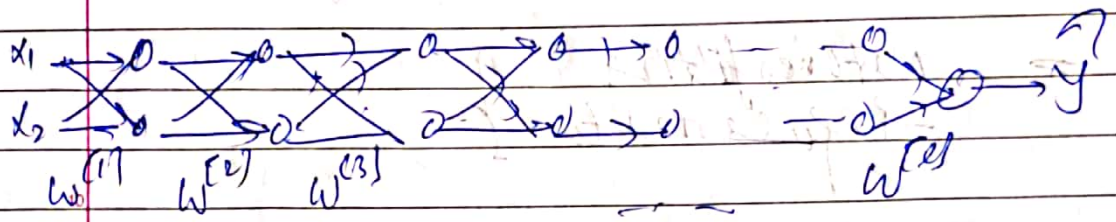
Normalize variance

$\sigma = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} * x^{(i)}$ | $X /= \sigma^2$

(mean) (variance)

use same μ & σ^2 to normalize the TRAIN & TEST Data

Vanishing & Exploding Gradients



$g(z) = z$

$\hat{y} = W^{[L]} W^{[L-1]} W^{[L-2]} \cdots W^{[1]} X$

$a^{[1]} z^{[1]} = W^{[1]}x + b^{[1]} \Rightarrow 0$

$a^{[2]} \quad a^{[1]} = g(z^{[1]}) = z^{[1]}$

$$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

$$\hat{y} = W^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$$

$\downarrow$

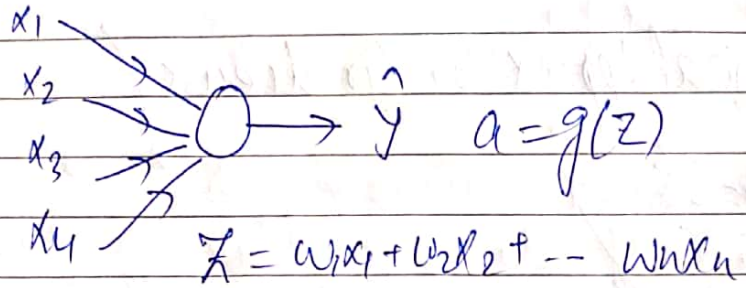
$(1.5)^{L-1}$ if $L >> 1$

So if very deep network $\hat{y}$ will explode. conversely

$(.5)^{L-1}$ is very less.

Weight Initialization for Deep Networks

Partial solution for Vanishing & Exploding problem in Deep Neural Networks is careful init of random weight.



We need z not to be very small nor to be very big. What you need is -

large $n \longrightarrow$ smaller w_i

Set Variance$(w_i) = \frac{1}{n}$

$w^{[l]} =$ np.random.randn(shape) * np.sqrt$\left(\frac{1}{n^{[l]}}\right)$

for Relu activation

Variance$(w_i) = \frac{2}{n}$ ↓ To do this

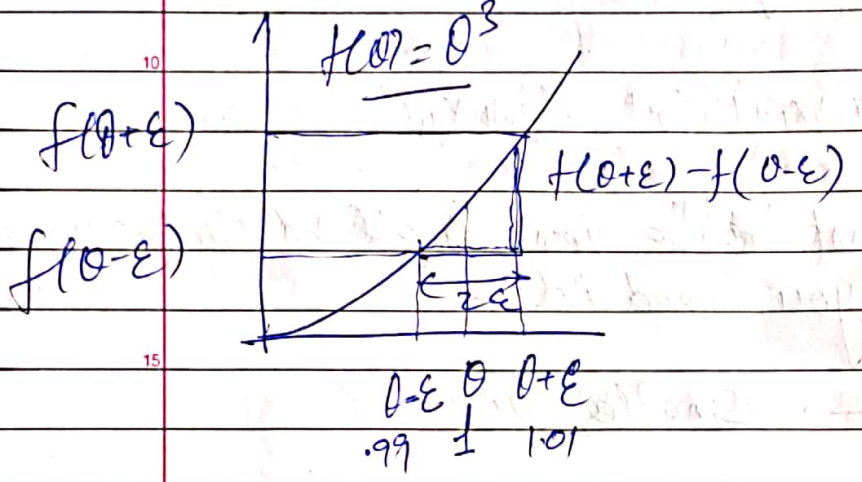
$w^{[l]} =$ np.random.randn(shape) * np.sqrt$\left(\frac{2}{n^{[l-1]}}\right)$

Other variance

tanh activation $\sqrt{\frac{1}{n^{[l-1]}}}$

These are not the first parameters to tune.

[Numerical Approx for Gradients]



$$\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\frac{(1.01)^3-(.99)^3}{2(0.01)} = 3.0001$$

$g(\theta) = 3\theta^2 = 3$ very close

$f'(\theta) = \lim_{\varepsilon\to 0} \frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon}$ | $\lim_{\varepsilon\to 0} \frac{f(\theta+\varepsilon)-f(\theta)}{\varepsilon}$

error dimen | error Order

$O(\varepsilon^2)$ | $O(\varepsilon)$

4

Camlin Page
Date / /

Gradient Check for Neural Network

Take $W^{[1]}, b^{[1]}, \dots W^{[l]}, b^{[l]}$ and reshape into big vector θ

Take $dW^{[1]}, db^{[1]} \dots dW^{[l]}, db^{[l]}$ reshape big vector $d\theta$

$J(\theta) = J(\theta_1, \theta_2, \dots \theta_n)$

for each i:

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2 \dots \theta_i + \varepsilon \dots) - J(\theta_1, \theta_2 \dots \theta_i - \varepsilon \dots)}{2\varepsilon} \approx \frac{\partial J}{\partial \theta_i}$$

$\downarrow$

$d\theta_{approx} \approx d\theta$

check $\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2} \approx 10^{-7}$ or smaller

if $\varepsilon = 10^{-7}$ [Great]

→ 10^{-5} It's ok you should check

→ 10^{-3} Wrong

Grading Checking Implementation

(*) Grad check doesn't work with drop out

(*) If algorithm fails grad check, look at the components to try to identify bug.

Ex. for some layer values of db are very far off 0. If this is the case then the bug is in how you find out "db"

Optimization Algorithm

Batch Vs Mini-batch gradient descent :-

$X = [x^{(1)}, x^{(2)} \dots x^{(m)}]$
(n_x, m)
$Y = [y^{(1)}, y^{(2)} \dots y^{(m)}]$

$m = 5000000$; ← What if m is too large.

We create mini-batch let say of 1000 each

$X = \{ \underbrace{x^{(1)} x^{(2)} \dots x^{1000}}_{X^{\{1\}}} \mid \underbrace{x^{1001} \dots x^{2000}}_{X^{\{2\}}} \mid \dots \underbrace{\dots x^{m}}_{X^{\{5000\}}} \}$

$Y = [\underbrace{y^{(1)} y^{(2)} \dots y^{1000}}_{Y^{\{1\}}}, \underbrace{y^{1001} \dots y^{2000}}_{Y^{\{2\}}} \dots y^{5000}]$

Mini batch t: $X^{\{t\}}, Y^{\{t\}}$

$X^{\{1\}} = (n_x, 1000)$ $Y^{\{1\}} = \{1, 1000\}$

for $t = 1 \longrightarrow 5000$
Forward prop $X^{\{t\}}$

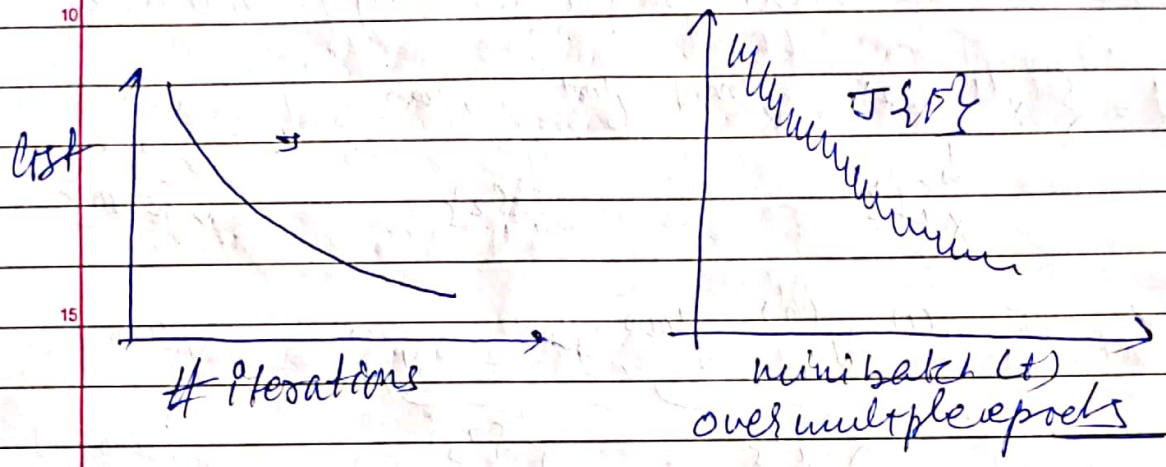
$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$
$A^{[1]} = g^{[1]}(Z^{[1]})$
!
$A^{[L]} = g^{[L]}(Z^{[L]})$
Compute cost :- $J^{\{t\}}$

$$J = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, \hat{y}) + \frac{\lambda}{2 \times 1000} \sum_{t} \| W^{[t]} \|_F^2$$

Backprop to compute gradients w.r.t $J^{\{t\}}$ using $x^{\{t\}}, y^{\{t\}}$

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

This one pass to training set this is called "epoch"



if mini-batch size = m = Gradient descent
if mini-batch size = 1 = Stochastic gradient descent

if small train set $m \leq 2000$

else mini batch size is anything of {64, 128, 256, 512}

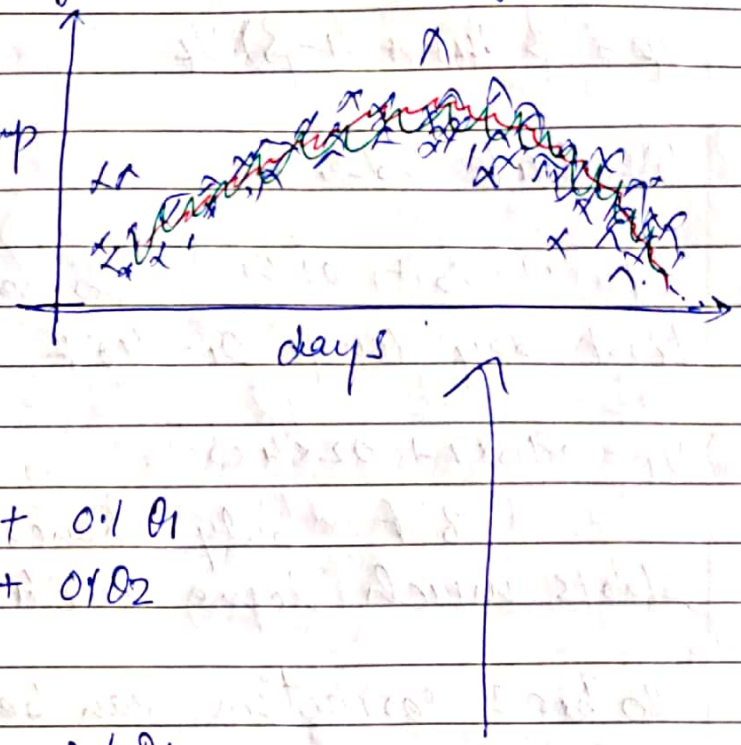
[Exponentially Weighted Averages]

$\theta_1 = 40°F$ $4°C$

$\theta_2 = 49°F$:

:

$\theta_{181} = 56°F$ $15°C$



$v_0 = 0$

$v_1 = 0.9 v_0 + 0.1 \theta_1$

$v_2 = 0.9 v_1 + 0.1 \theta_2$

:

$v_t = 0.9 v_{t-1} + 0.1 \theta_t$

If plot v_t we will get the RED curve

$v_t = \beta v_{t-1} + (1-\beta)\theta_t \qquad \beta = 0.9$

v_t can be considered as average over

$\approx \frac{1}{1-\beta}$ days temprature if $\beta = 0.9$ days

So if is average over $\approx \frac{1}{1-.9}$. Las 10 days temp.

if $\beta = 0.5$ $\approx$ 2 days avg. we get GREEN curve.

Bias Correction in Weighted Avg

$V_t = \beta V_{t-1} + (1-\beta) V_t$

we init $V_0 = 0$

$V_1 = .98 V_0 + .02 \theta_1$ so we even if first day temp $\theta_1 = 40°F$ we get

$V_1 = .98 \times 0 + .02 \times 40$

$= 8°F$ only hence curve starts much lower initially.

so bias correction can be:-

$V_t = \beta V_{t-1} + (1-\beta) V_t$

$V_t = \frac{V_t}{(1-\beta^t)}$

$t \to 2 \rightarrow 1-\beta^t \Rightarrow 1-(.98)^2 = 0.0396$

so estimate

$\frac{V_2}{.0396} = \frac{.0196\theta_1 + .02\theta_2}{0.0396}$

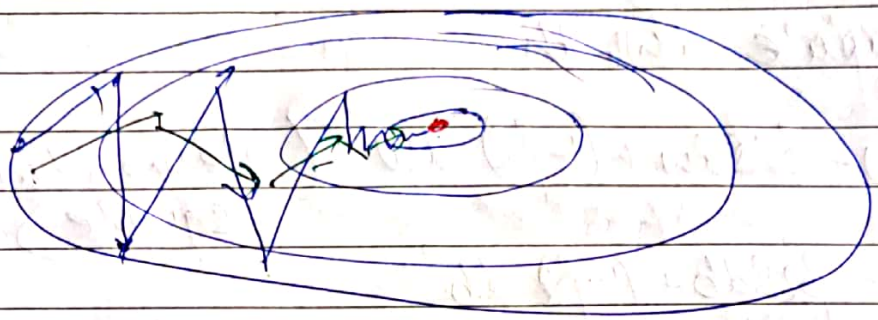
$\approx$

$(1-\beta^t) \rightarrow$ in a long term get vanish

$1-(.98)^{(\text{lots})} \rightarrow \approx 1$

[Gradient Descent With Momentum]

Gradient Descent Example



What we want ↓ vertical axis slower & horizontal axis faster :-

Momentum :-

→ Compute dW, db on current mini-batch.

→ compute [$V_{dw} = \beta V_{dw} + (1-\beta) dw$ (Similar & analogy to $V_\theta = \beta V_{\theta t} + (1-\beta)\theta_t$. -

$V_{db} = \beta V_{db} + (1-\beta) db$

$W = W - \alpha V_{dw}$ / $b = b - \alpha V_{db}$

by doing this we will find "GREEN" arrow. most of the time $\beta = 0.9$ works very well.

↓ In Adam optimizer this is $\beta_1 = .9$

RMSprop

On iteration t:-

Compute dW, db

$$sdW = \beta_2 sdw + (1-\beta_2)\, dW^2 \quad (\text{element wise square})$$

$$sdb = \beta_2 sdb + (1-\beta_2)\, db^2$$

$$W = W - \alpha \frac{dW}{\sqrt{sdW}}$$

$$b = b - \alpha \frac{db}{\sqrt{sdb}}$$

Intuition: sdW will be very small, sdb very large. inspective to remove divide by zero we do

$$W = W - \alpha \frac{dW}{\sqrt{sdW} + \varepsilon}$$

$$b = b - \alpha \frac{db}{\sqrt{sdb} + \varepsilon}$$

ε very very small number

$\varepsilon \approx 10^{-8}$

RMS prop given by Geoffrey Hinton in course 6a

ADAM Optimization [RMSP + Momentum]

$V_{dw} = 0$, $S_{dw} = 0$ | $V_{db} = 0$ | $S_{db} = 0$

On iteration t:

Compute dw, & db with mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dw \quad \left.\right\} \text{Momentum} \leftarrow \beta_1$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db \quad \left.\right\}$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dw^2 \quad \left.\right\} \text{RMS} \leftarrow \beta_2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2 \quad \left.\right\}$$

Bias Correction

$$V_{dw}^{\text{correct}} = \frac{V_{dw}}{1-\beta_1^t} \quad | \quad V_{db}^{\text{correct}} = \frac{V_{db}}{1-\beta_1^t}$$

$$S_{dw}^{\text{correct}} = \frac{S_{dw}}{1-\beta_2^t} \quad | \quad S_{db}^{\text{correct}} = \frac{S_{db}}{1-\beta_2^t}$$

$$W = W - \alpha \left(\frac{V_{dw}^{\text{Correcte}}}{\sqrt{S_{dw}^{\text{Correct}}} + \varepsilon} \right)$$

$$b = b - \alpha \left(\frac{V_{dw}^{\text{Corrected}}}{\sqrt{S_{dw}^{\text{Correcte}}} + \varepsilon} \right)$$

α = needs to tune | β_2 : 0.999 | $\varepsilon \approx 10^{-8}$ | } Common Values

$\beta_1 = 0.9$

Learning Rate Decay.

$x^{\{1\}}$	$x^{\{2\}}$	...	$x^{\{1000\}}$

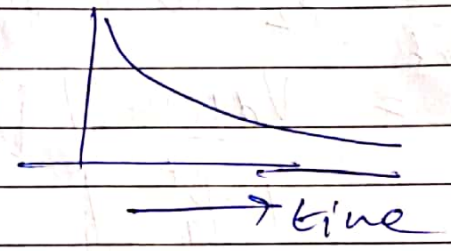
→ 1 epoch

1 epoch = 1 pass to Data

$$\alpha = \frac{\alpha_0}{1 + (\text{decay\_rate}) * \text{epoch\_num}}$$

$\alpha_0 = 0.2$

epoch	α	
1	0.1	$\frac{.2}{1+1} = .1$
2	0.0733	$\frac{.2}{1+2} = \frac{.2}{3} = 0.0\cancel{6}67$



$$\alpha = (0.95)^{\text{epoch\_num}} * \alpha_0 \quad [\text{exp\_decay}]$$

"Hyperparameter Tunning Process"

Hyperparameters are :- α (Learning Rate)

β (momentum) (β_1, β_2) Adam paras ε

layer # hidden units

learning rate decay / mini batch size.

α [Alpha Learning Rate]

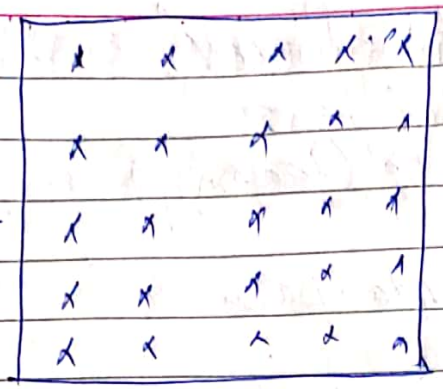
$\beta \approx 0.9$
Hidden Units
mini batch size

layers
learning rate decay

$\beta_1 = .9 \mid \beta_2 = .999 \mid \varepsilon = 10^{-8}$
↑ Adam params

Decreasing Order of Priority to tune hyperparam as per "Andrew Ng"

H2

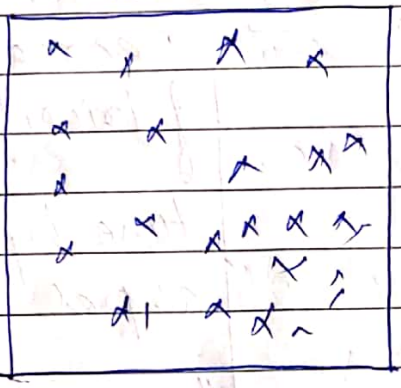


Grid Search Method

Try the combination of H1 & H2. [25 Combo]

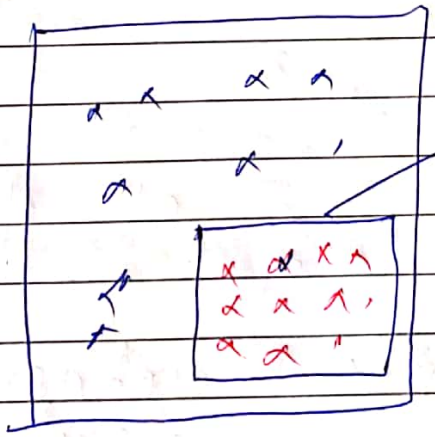
This old method we should not use in "Deep Learning"

H2



Choose the point at random.

(Course to fine)



This area looks better so zoom in at take more dense samples.

Appropriate scale to pick hyperparameters

$n^{[l]}$ = 50 ——— 100

5 50 ——— 100

layer L: 2 – 4 (no do the coarse fine)

linear search is pretty ok here but not true for other.

α = .0001 ——— 1

15 .0001 .1 1

10% 90%

Samples

So taking samples on linear scale we'll be biased hence it is good to take the steps on log "scale".

20 .0001 .001 .01 .1 1

10^a 10^b $(b = \log_{10} 1 = 0)$

$a = \log_{10}(10^{-4})$ $= -4$

25 $r = -4$ # np.random.rand() $\leftarrow r \in (-4, 0)$

$\alpha = 10^r$ $\alpha \in (10^{-4}, 10^0)$

now what we do sample between $r \in [a, b]$ in our case $[-4, 0]$ & put

30 $r \in [a, b]$

$\alpha = 10^r$

Camlin | Page
Date / /

Hyperparameter for exponentially weighted Averages

β = .9 —— .999

.9 —— .999

→ Doesn't make sense to sample the value on linear scale
The way we can think about this is to we have to explore the

$1-\beta$ = .01 —— .0001

0.1 —— .01 —— .0001

10^{-1} 10^{-3}

↑ log scale log scale

$r \in [-3, -1]$

$1-\beta = 10^{r}$

$\beta = 1-10^{r}$

β randomly chosen hyperparameter value:

Mathematical Justification why to sample on "Log" :-

β : $0.9 \rightarrow .9005$ (This doesn't much impact)

$\approx \frac{1}{1-.9} \approx \frac{1}{.1} \approx$ 10 day Avg.

$\frac{1}{1-.9005} = \frac{1}{.0995} \approx 10.05$ ~~days~~ Avg. → data

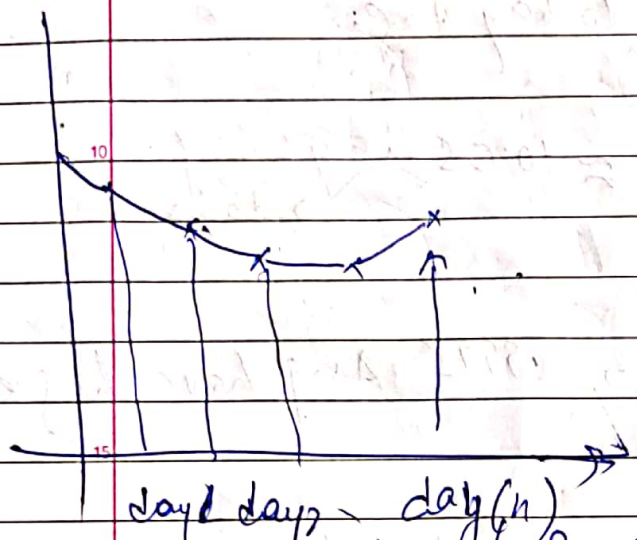
but $\beta = .999 \rightarrow .9995$ Avg have huge impact

$\frac{1}{1-.999} = \frac{1}{0.001} \approx 1000$ ~~days~~ Avg → past data

$\frac{1}{1-.9995} = \frac{1}{1-.0005} \approx 2000$ ~~days~~ Avg → data

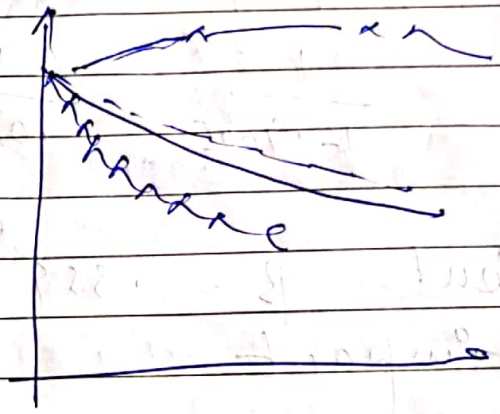
Hyperparameter ~~Tuning~~ Tuning in practice: Panda Vs Caviar

Panda: One model at a time



Patiently Train one model at a time

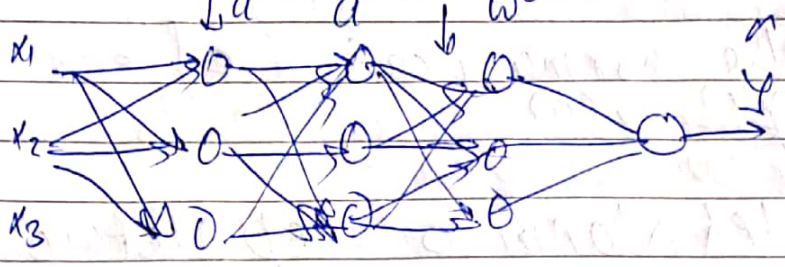
Caviar: Multiple model in parallel



Try multiple model & choose one

(Revert back)

Hyper parameter Tuning: Batch Normalization



Q: For any hidden layer can we normalize the values $a^{[l]}$ to next layer weight $w^{[l+1]}$ $b^{[l+1]}$ faster.

But Technically we normalize $z^{[l]}$ no $A^{[l]}$

Implementing Batch Norm

Given some intermediate values in NN $z^{(1)} \dots z^{(m)}$

$z^{[l](i)}$ → i^{th} unit, hidden layer No:-

$$\mu = \frac{1}{m} \sum z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum (z^{(i)} - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$ → Just in case $\sigma^2 = 0$, $\varepsilon = 10^{-6}$

but we don't want my hidden unit to have $\sigma^2 = 1$ always.

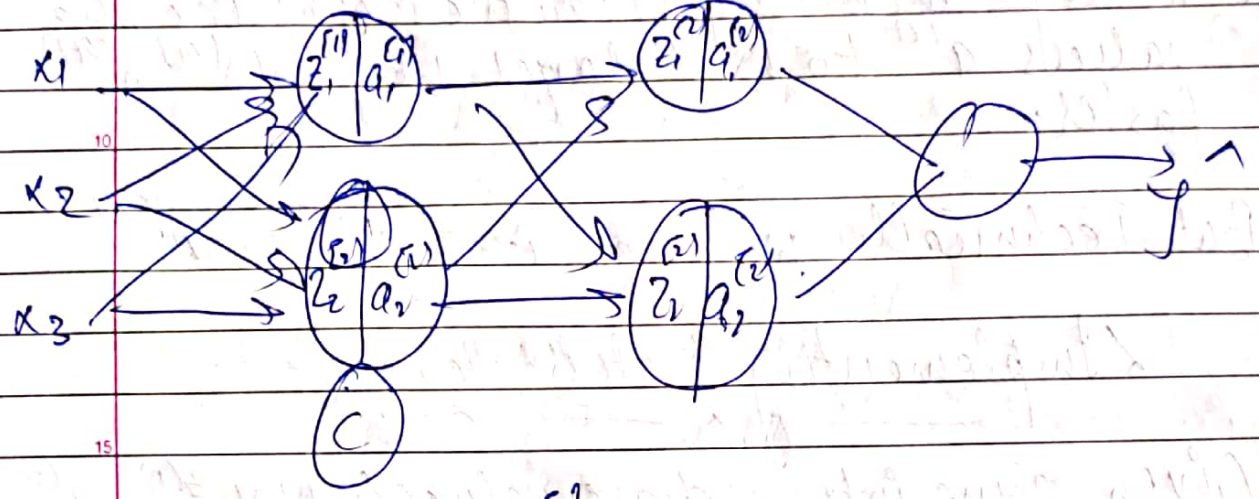
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

γ Gama, β Beta

γ, β is learnable param using Gradient Descent / RMS / ADA / Momentum etc

if $\gamma = (\sigma^2 + \varepsilon)$ | $\beta = \mu$ it will nullify the normalization effect.
so $\tilde{z}^{(i)} = z^{(i)}$

Applying Batch Norm to Deep Networks



$$X \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[\text{BatchNorm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \rightarrow a^{[1]} = g(\tilde{Z}^{[1]})$$

$$a^{[1]} \xrightarrow{W^{[2]}, b^{[2]}} Z^{[2]} \xrightarrow[BN]{\beta^{[2]}, \gamma^{[2]}} \tilde{Z}^{[2]} \rightarrow a^{[2]}$$

Do params are now to be learnt:-

$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]} \dots W^{[l]}, b^{[l]}$

$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]} \dots \beta^{[l]}, \gamma^{[l]}$

$d\beta^{[l]}$

$\beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$

{ BatchNorm + MiniBatches }

$X^{\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[BN]{} \tilde{Z}^{[1]} \rightarrow g^{[1]}(\tilde{Z}^{[1]}) = a^{[1]} \cdots$

$X^{\{2\}} \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[BN]{} \tilde{Z}^{[1]} \rightarrow \cdots$

Params: $W^{[l]}, b^{[l]}, \beta^{[l]}, \gamma^{[l]}$

$Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

↓ while BN $b^{[l]}$ doesn't change anything while taking mean so $b^{[l]}$ can be scrapped so the params are now:-

Param:- $W^{[l]}, \beta^{[l]}, \gamma^{[l]}$

$Z^{[l]} = W^{[l]} a^{[l-1]}$

$\tilde{Z}^{[l]}_{norm} = \gamma^{[l]} Z^{[l]}_{norm} + \beta^{[l]}$

$Z^{[l]} = (n^{[l]}, 1)$ | $\beta = (n^{[l]}, 1)$ } Dimensions

$\gamma = \nearrow$

for t=1 — num mini Batch
compute forward prop on $X^{\{t\}}$
In each hidden layer, use BN to
replace $Z^{[l]} \rightarrow \tilde{Z}^{[l]}$

Use backprop to compute
$dw^{[l]}, d\beta^{[l]}, dr^{[l]}$

Update Params:-

$$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$
$$d\beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$$
$$dr^{[l]} = r^{[l]} - \alpha\, dr^{[l]}$$

This works with ADAM / momentum / RMS ...

[BATCH NORM TEST TIME]

At a test to predict μ, σ^2 is used exponentially weighted avg of mini batches:-

$X^{\{1\}}$ $X^{\{2\}}$ ...

$\mu^{\{1\}[l]}$ $\mu^{\{2\}[l]}$ ... $\mu^{\{3\}[l]}$ $\xrightarrow[\text{avg.}]{\text{exp wt.}}$ $\mu_{\text{expwt avg.}}$

Same way

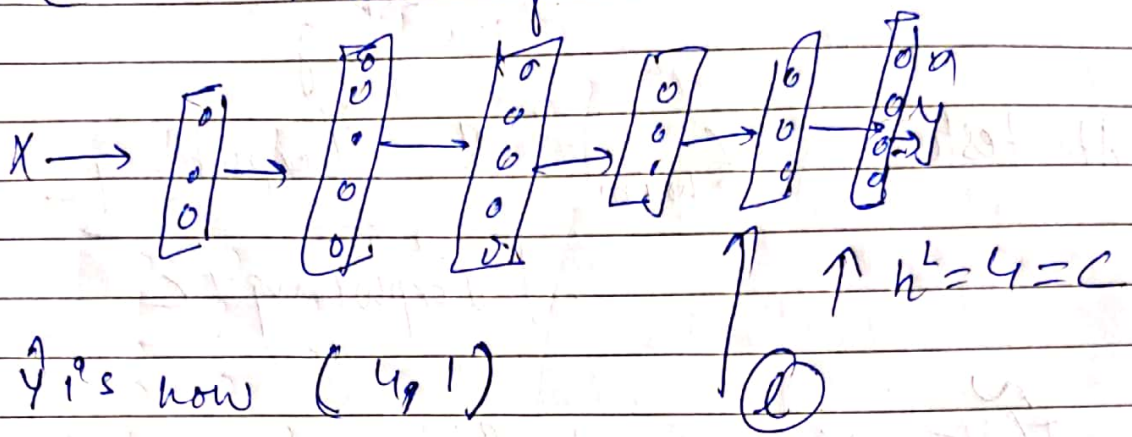
$\sigma^2_{[1]}[0]$, $\sigma^2_{[2]}[0]$ ... $\xrightarrow[\text{avg}]{\text{exp wt}}$ $\sigma^2_{\text{exp wt avg}}$

At testime $Z_{norm} = \dfrac{Z - \mu_{\text{exp wt avg}}}{\sqrt{\sigma^2_{\text{exp wt avg}} + \varepsilon}}$

$\tilde{Z} = \gamma Z_{norm} + \beta$

Softmax Regression

C = 4 = Number of Classes :-



$\hat{y}$ is now (4, 1)

$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$$

Activation Softmax :-

$$t = e^{z^{[L]}} = (4,1) \text{ vector}$$

$$a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{i=1}^{C=4} t_i} = (4,1) \text{ vector.}$$

Ex :- $z = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 0.4 \\ 8.1 \\ 20.1 \end{bmatrix}, \quad \sum_{i=1}^{4} t_i = 176.3$$

$$a^{[L]} = \frac{t}{176.3} = \begin{bmatrix} e^5/176.3 \\ e^2/176.3 \\ e^{-1}/176.3 \\ e^3/176.3 \end{bmatrix}$$

Training Softmax Classifier

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(\quad " \quad) \\ e^{-1}/(\quad " \quad) \\ e^3/(\quad " \quad) \end{bmatrix}$$

Softmax

$$= \begin{bmatrix} .842 \\ .042 \\ .002 \\ .114 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

"hard max"

hard max.

Loss function :- **Predicted** $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ → GT

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C=4} y_j \log(\hat{y}_j)$$

$$= -y_2 \log \hat{y}_2 = -\log \hat{y}_2$$

make it small is to make small and to make we have to make $\hat{y}_2$ as big as possible

This Loss on single train example

So on loss on entire train set:

$$J(w^{[1]}, b^{[1]}, \ldots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$Y = [y^{(1)} \; y^{(2)} \; \ldots \; y^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \ldots$$

$(4, m)$ dimen

$$\hat{Y} \rightarrow [\hat{y}^{(1)}, \ldots \hat{y}^{(m)}]$$

$$\hat{Y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \quad \ldots$$

$(4, m)$ dimension matrix

{Gradient descent with Softmax}

Backward prop: $dz^{[1]} = \hat{y} - y$

$(4,1) = (4,1) - (4,1)$

$\frac{\partial J}{\partial z^{[1]}} =$