Course II

* Train | dev | test sets

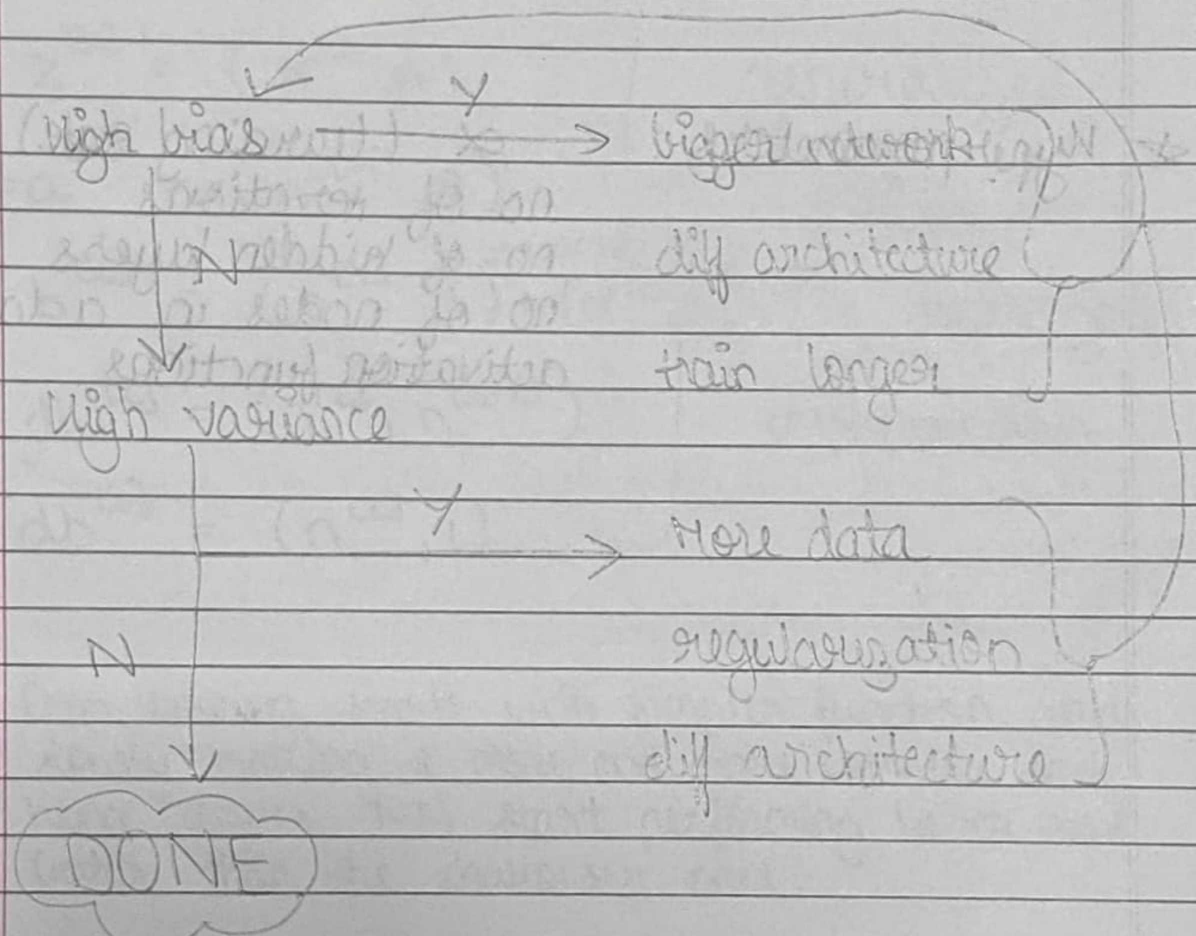
smaller dataset : 60/20/20

larger dataset : 98|1|1

Test and dev datasets should be from same distribution.

→ Train set error → bias

Dev set error → variance



★ Regularisation. (Logistic Regression)

$$J(\omega, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \underbrace{\frac{\lambda}{2m}\|\omega\|^2}_{L2 \text{ regularisation}}$$

$$\|\omega\|^2 = \omega^T\omega$$

★ Regularisation of neural network.

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

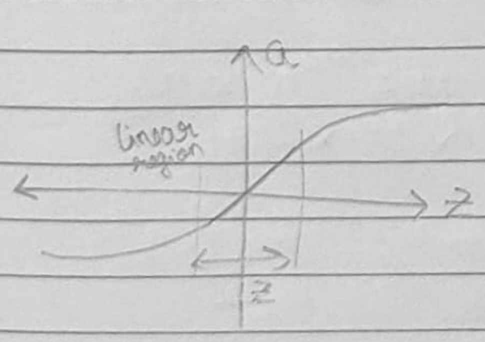
frobenius norm

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}\left(w_{ij}^{[l]}\right)^2$$

$$dW^{[l]} = \left(\begin{matrix}\text{from back} \\ \text{prop}\end{matrix}\right) + \frac{\lambda}{m} w^{[l]}$$

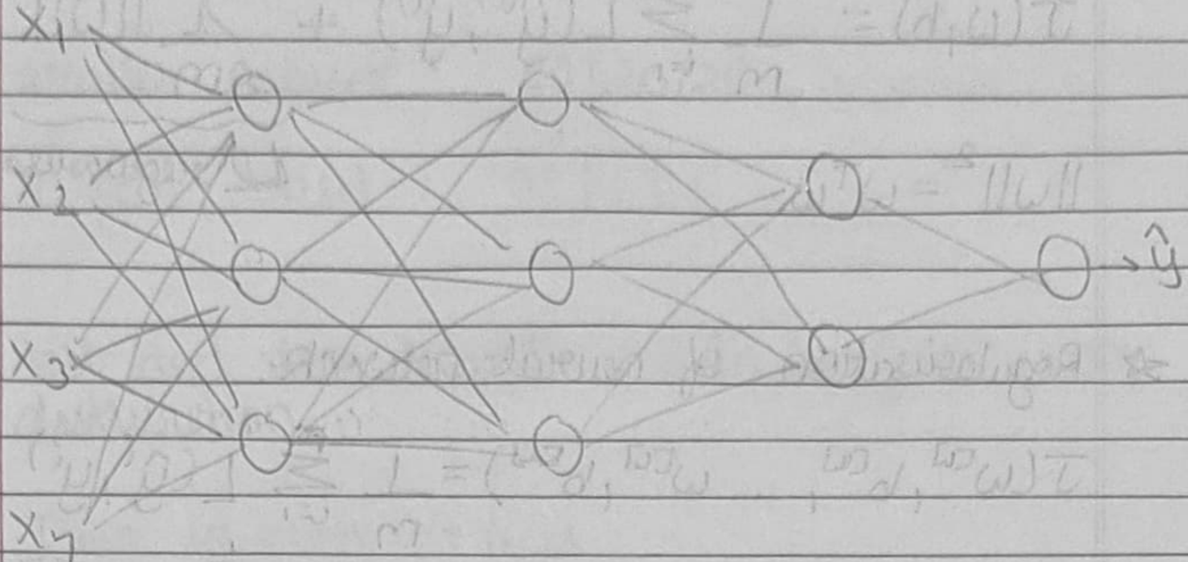
$$W := W - \alpha\, dW^{[l]}$$

Note: $\lambda\uparrow \quad w^{[l]}\downarrow \qquad z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]} \downarrow$



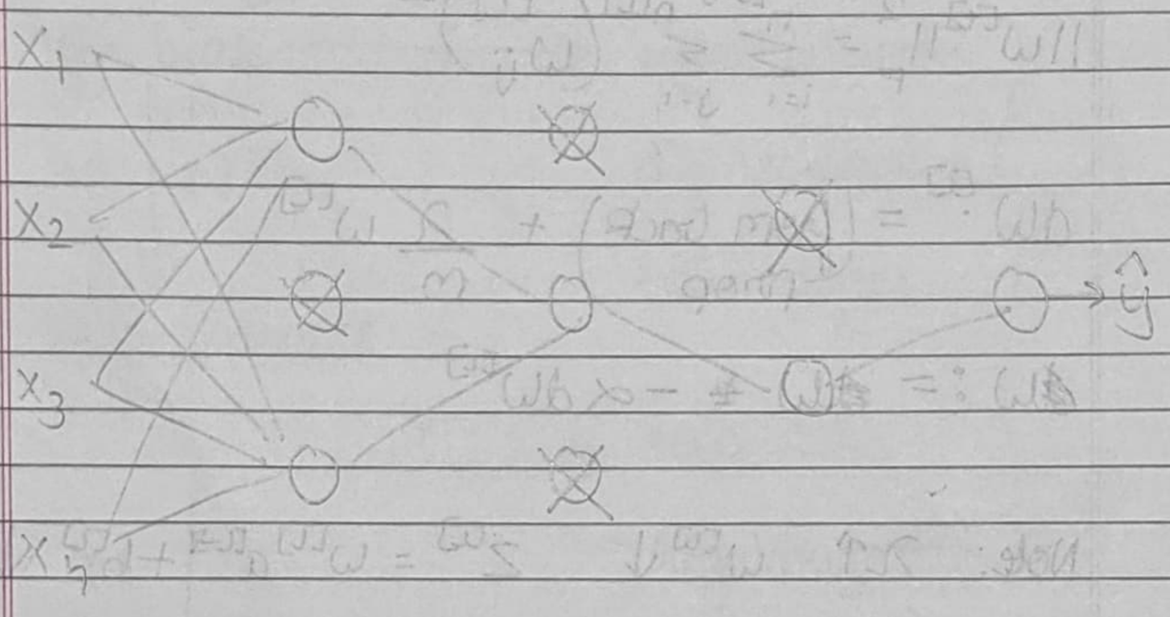
it become linear and hence reduces overfitting of the function.

★ Dropout Regularisation



delete nodes at random

↓ dropout



classmate
Date
Page

* Inverted dropout

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keepprob$ → generates a matrix of given shape with values btw 0 and 1

$a3 = np.multiply(a3, d3)$

$a3 /= keep\_prob$ → to not reduce the expected value of z

keep_prob → is the prob ~~with~~ which decide to keep or eliminate the neuron.

→ higher ~~lower~~ for layers where overfitting isn't a issue

lower ~~higher~~ for layers where overfitting is a issue

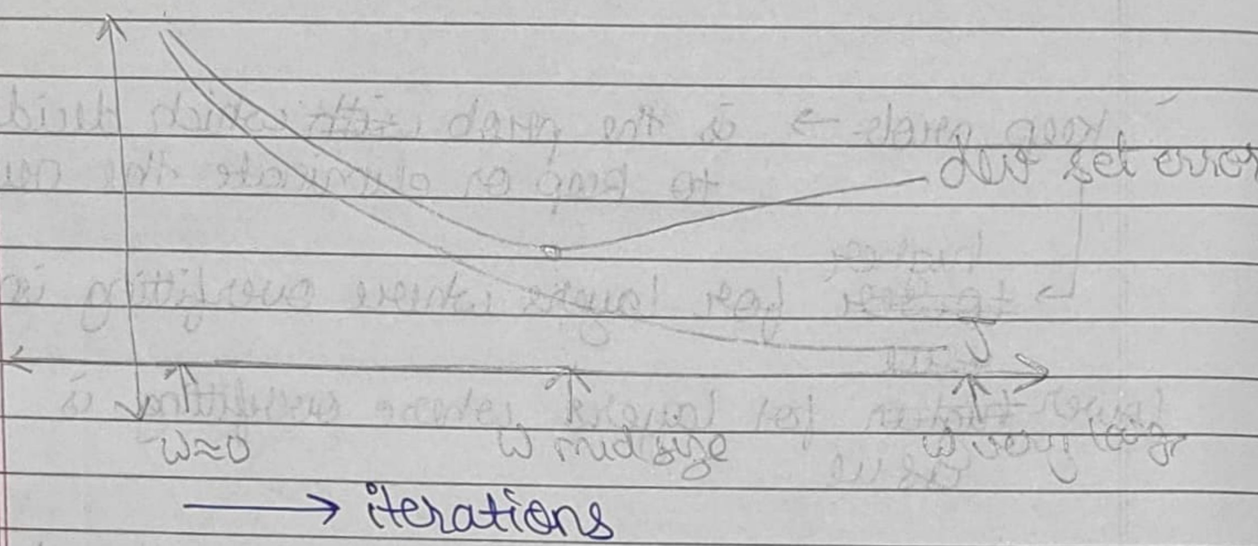
downside is that J is not well defined

★ Data augmentation

→ flip the image horizontally
→ random crops of the image
} don't add too much info and its cheap to do.

This way overfitting can be reduced.

★ Early stopping



orthogonalization → focus on one task at a time.

★ Normalize the data → eg to optimise

Make mean 0 $\mu = 1/m \sum x^i$
$x := x - \mu$

Normalise variance $\sigma^2 = 1/m \sum (x^i)^2$
$x \mathrel{/}= \sigma^2$

* Weight initialization:

ReLU: Var $:= \sqrt{2/n}$

tanh: Var $:= \sqrt{1/n^{[L-1]}}$

* grad check

θ = vector of $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \ldots$

$d\theta$ = vector of $dw^{[1]}, db^{[1]}, dw^{[2]}, db^{[2]} \ldots$

for each i:

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

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

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

10^{-5} okayish

10^{-3} bad

$\varepsilon = 10^{-7}$

use in debug only

NOT WILL TRAINING

Minibatch

↳ split the huge training dataset into smaller parts

$$X = [x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)} \mid x^{(1001)} \dots x^{(2000)} \mid \dots \mid \dots x^{(m)}]$$

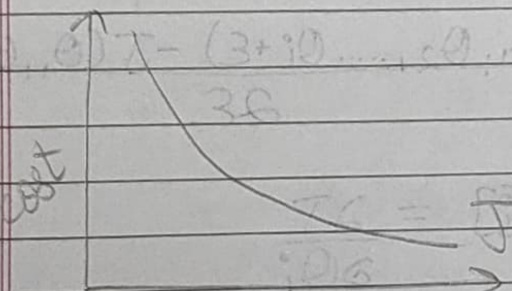
(n_x, m) $\longleftarrow X^{\{1\}} \longrightarrow$

$$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)} \mid y^{(1001)} \dots y^{(2000)} \mid \dots \mid \dots y^{(m)}]$$

$(1, m)$ $\longleftarrow Y^{\{1\}} \longrightarrow$

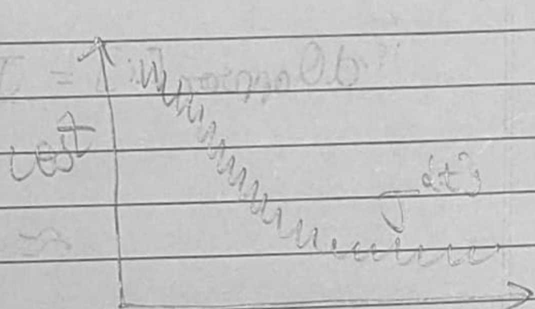
$Y^{\{1\}} : [y^{(1)}, y^{(2)} \dots y^{(1000)}]$

$X^{\{2\}} : [x^{(1)}, x^{(2)} \dots x^{(1000)}]$



iteration

batch gradient descent



mini batch gradient descent

mini batch size $\longrightarrow m \longrightarrow$ batch gradient descent

$\longrightarrow 1 \longrightarrow$ stochastic gradient descent

generally $\longrightarrow$ btw 1 & m

classmate
Date
Page

★ Exponentially weighted average

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

$\frac{1}{1-\beta}$ → average

if $\beta = 0.9 \approx$ 10 day average

if $\beta = 0.5 \approx$ 2 day average

★ Bias Correction

$\frac{V_t}{1-\beta^t}$ → initialize phase me help karta → as "t" gets bigger $1-\beta^t \approx 1$ and hence no effect.

★ gradient descent with momentum

on iteration t:

dW, dB

$$V_{dW} = V_{dW} \times \beta + (1-\beta)\, dW$$
$$V_{dB} = \beta V_{db} + (1-\beta)\, db.$$

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

α, β → hyperparameter

classmate
Date
Page

★ RMS prop

On iteration t:

Example hai generally higher dimensional

complete dw, db

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$ ← small
$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← large

$w := w - \alpha \frac{dw}{\sqrt{S_{dw}+\varepsilon}}$ large making fast

$b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$ small learning slow

→ to ensure it does not blow up as S_{db}, S_{dw} can be 0

★ Adam optimization algorithm.

$V_{dw} = 0, S_{dw} = 0 \quad V_{db} = 0, S_{db} = 0$

On iteration t,

compute dw, db

$\beta_1 \rightarrow$ momentum

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$
$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

$\beta_2 \rightarrow$ RMS prop.

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$
$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$

$$V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta_1^t} \qquad V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t} \qquad S_{db}^{corrected} = \frac{S_{db}}{1-\beta_2^t}$$

$$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$$

$$b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

generally, $\beta_1 = 0.9$ & $\beta_2 = 0.999$ $\varepsilon = 10^{-8}$

* learning rate decay

↳ initial mai bigger step but as the network matures smaller steps.

∴ α is decreased with time.

$$\alpha = \frac{1}{1 + \text{decay rate} \times \text{epoch number}} \alpha_0$$

$$\alpha = 0.95^{\text{epoch num}} \alpha_0$$

$$\alpha = \frac{k}{\sqrt{\text{epoch num}}} \alpha_0 \quad \text{or} \quad \alpha = \frac{k}{\sqrt{t}} \alpha_0$$

use random values for hyperparameter tuning.

* Batch Norm

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

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\gamma = \sqrt{\sigma^2 + \varepsilon}$$

$$\beta = \mu$$

$$\tilde{z}^{[l](i)} = \underbrace{\gamma z_{norm}^{[l](i)} + \beta}_{\text{learnable parameter}}$$

$$X \xrightarrow[w,b]{} z^{[1]} \xrightarrow[\beta,\gamma]{BN} \tilde{z}^{[1]} \longrightarrow a^{[1]} = g(\tilde{z}^{[1]})$$

$$a^{[1]} \xrightarrow{w,b} z^{[2]} \xrightarrow[\beta,\gamma]{BN} \tilde{z}^{[2]}$$

$$\hat{y} \leftarrow \cdots\cdots \leftarrow a^{[2]} = g(\tilde{z}^{[2]}) \leftarrow \tilde{z}^{[2]}$$

* Batch norm has slight regularisation effect.

* Batch norm at test time
  - at test time we may not have enough examples to calculate the μ or σ^2

  so, we estimate it using exponentially weighted moving average.

* Softmax Layers. let $z^{[L]}$ be (4,1)
  - activation func

$$t = e^{(z^{[L]})}$$

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

$$a_i^{[L]} = \frac{e^{z_i^{[L]}}}{\sum_{i=1}^{4} t_i} = \frac{t_i}{\sum t_i}$$

Eg: $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.4 \\ 0.1 \end{bmatrix}$

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

* Plateaus can slow learning

local optima is very rare in higher dimensional space.