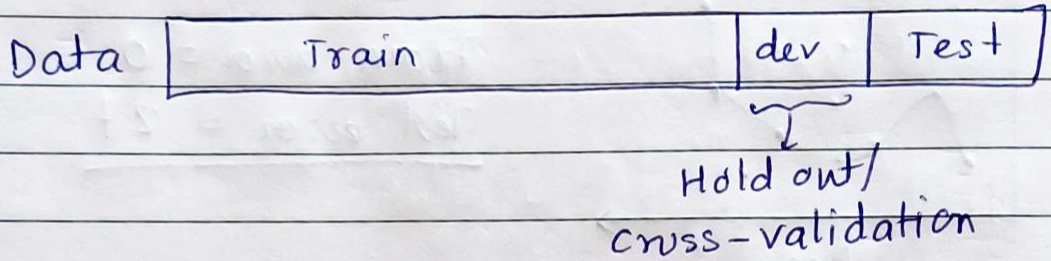
Date : 22/7/24

MON TUE WED THU FRI SAT SUN

☑ ☐ ☐ ☐ ☐ ☐ ☐

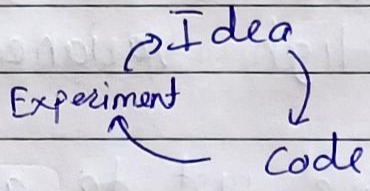
# Course 2:
# Improving Deep Neural Networks

## # Train / dev / test sets



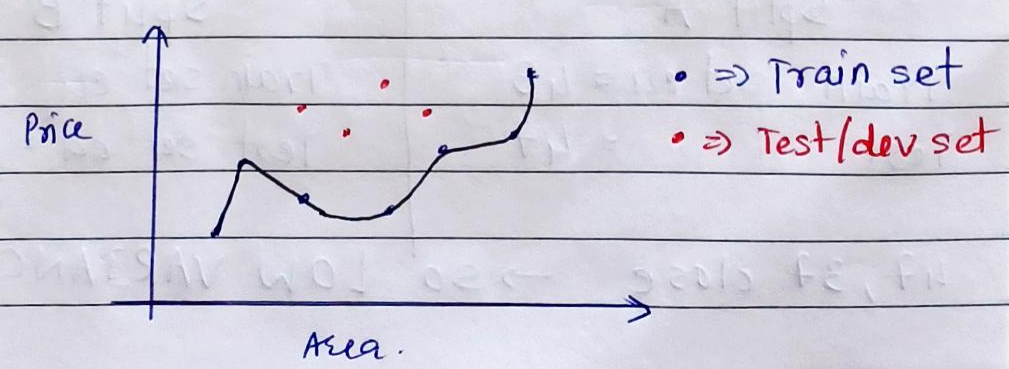

When data very very big: 98% / 1% / 1%
Train dev test

* dev & test sets come from same distribution.




## # Bias and Variance

eg.

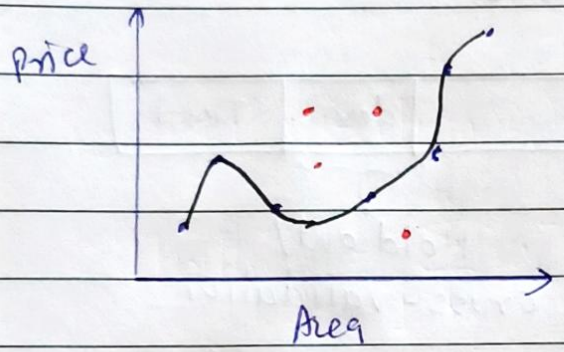


Date :
MON TUE WED THU FRI SAT SUN

Train set error = 0

Test set error = 90 (let)

Now same data but split is different:




Train error = 0

Test error = 27

⇒ Test error varies Greatly based on your selection of train & test data points.

→ This is an OVERFIT Model

→ High variance

If we just do a Linear Regr. in prev. eg.

| Split A | Split B |
|---|---|
| Train set err = 43 | Train set err = 41 |
| Test " " = 47 | Test set err = 37 |

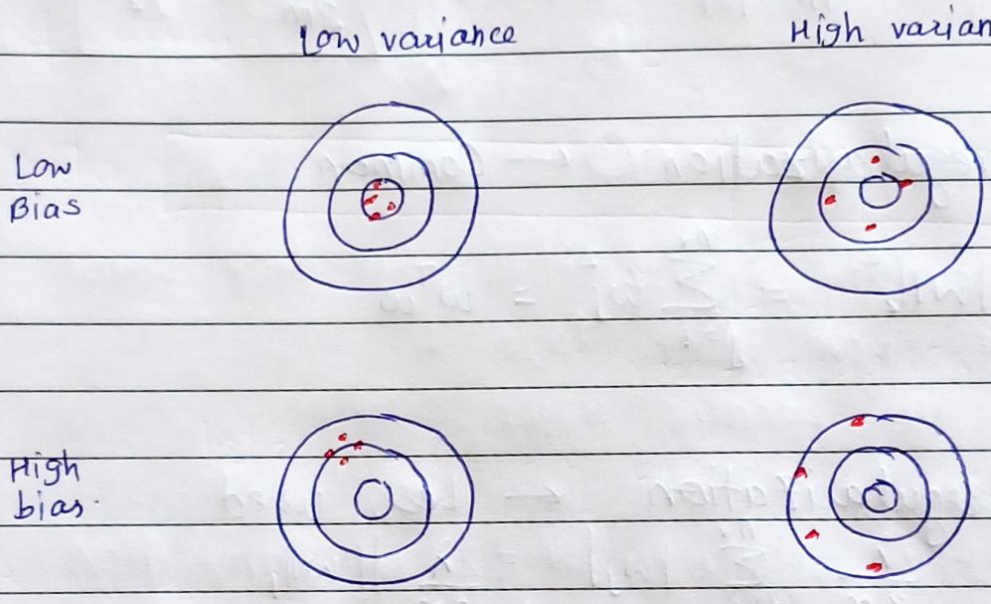
47, 37 close → so LOW VARIANCE

Bayes error = Human's error.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

But huge train err ⇒ HIGH BIAS

BIAS: measurement of how accurately a model can capture a pattern in a training dataset.




# Basic recipe for ML

↑ bias ? —Yes→
- Bigger network
- Train longer
- NN architecture

↓ No

↑ variance? —Yes→
- More data
- Regularisation
- NN archi

↓ NO

✓

$||w|| \rightarrow$ ~~norm~~ Frobenius norm

Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

# Regularization: (for ↑ variance)

eg. In Logistic Regr:

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

$L_2$ regularization: ← Common.

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

$L_1$ regularization ← Less used:

$$\frac{\lambda}{2m}\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}||w||_1$$

$\lambda$ = regularisation parameter.
= hyperparameter.

For NN:

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

Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

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

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

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

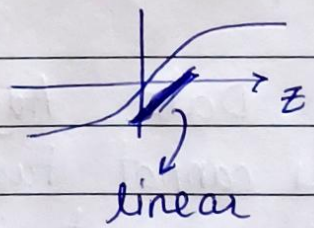
$$\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}$$

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

Explanation:
By adding the L1 or L2 regul. we are adding a penalty to the cost function.

Thus $\lambda \uparrow \rightarrow w^{[l]} \downarrow \rightarrow$ NN becomes simpler

Also, if AF is tanh



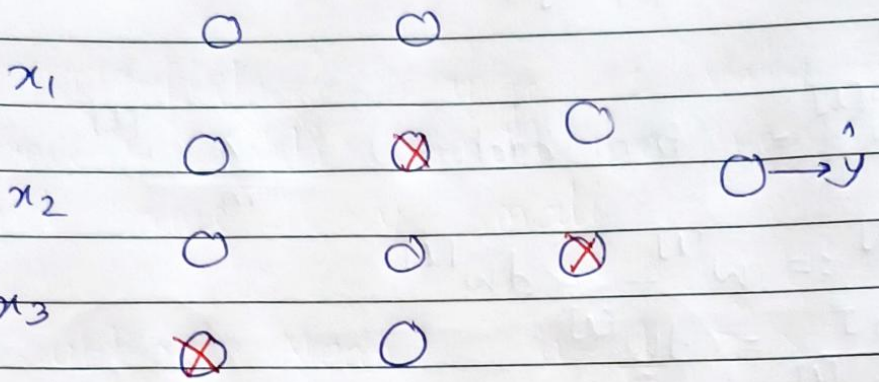

$w^{[l]} \downarrow \Rightarrow z^{[l]} \downarrow$

$\therefore$ As if every layer $\approx$ linear (or linear regression)

$\Rightarrow$ Model will not OVERFIT

Date :
MON TUE WED THU FRI SAT SUN

## → Dropout Regularization ←




By random probabilities, we drop some nodes from each layer & So NN became much simpler.

Dropout efficiency ≈ L2 regularization

* Gen'ly we don't drop input layer.

Method → we keep a threshold probability And generate array of randoms prob. If prob < thresh → drop.

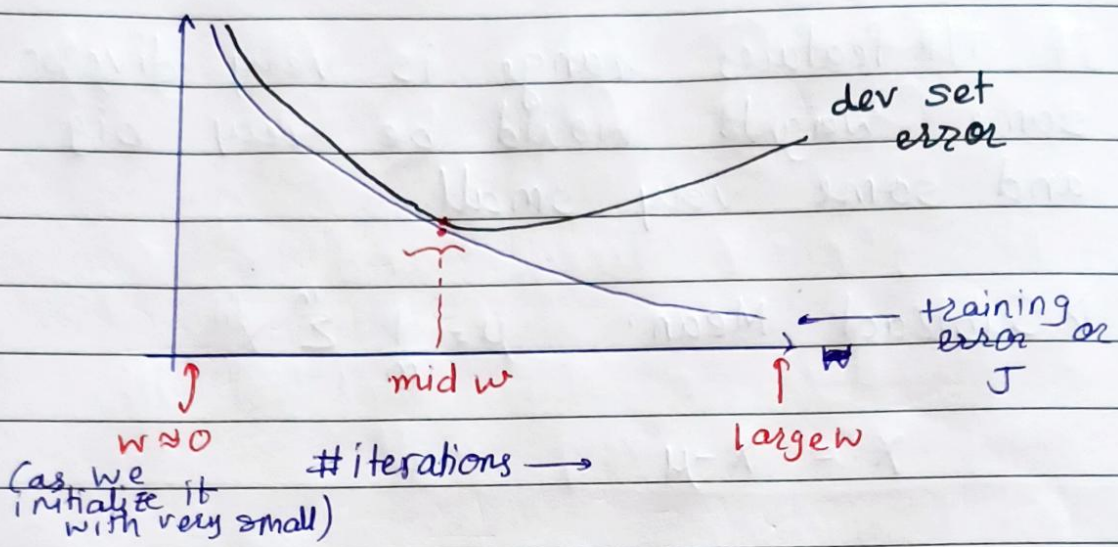
* Dropout also reduces overfitting.

## → Data Augmentation ←

If u cannot further increase your training data, augment it (rotate, flip, crop, skew, images etc)

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

## → Early stopping ←




The point where deflection of dev error starts, we stop training the model there.

But with this method orthogonalization is violated

orthogonalization → do 1 task @ 1 time

- T1 – Optimize J
- T2 – Not overfit

↳ this method does both @ same time.
∴ J is not reduced much.

Date :
MON TUE WED THU FRI SAT SUN

# Normalizing training sets

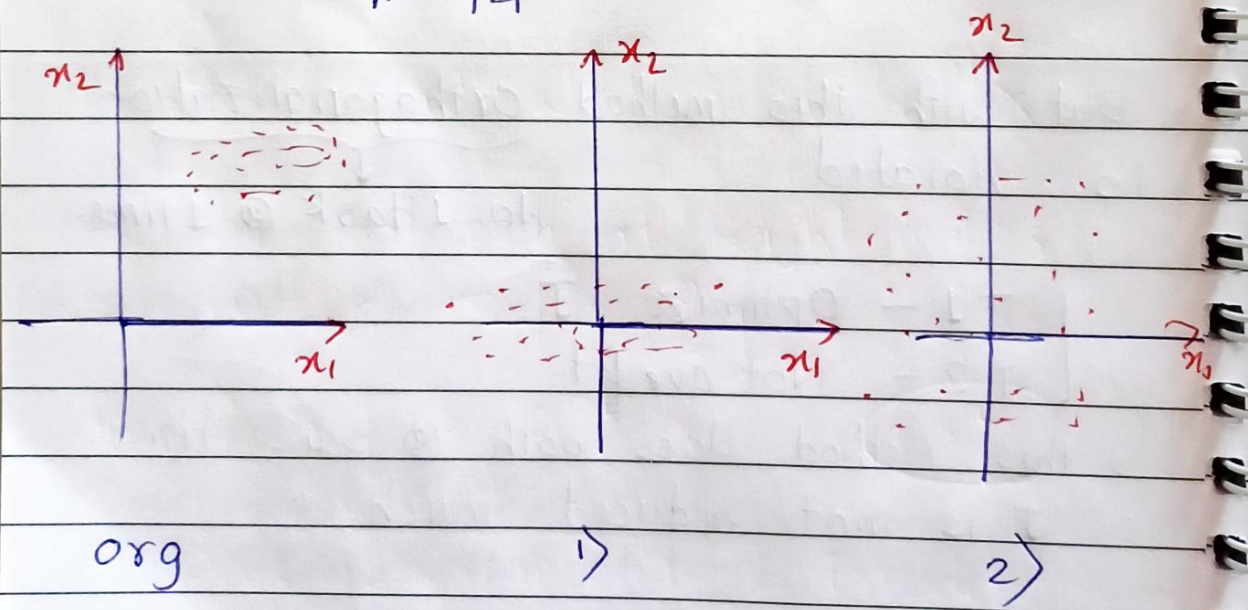
If I/P features range is very diverse: some weights would be very big and some very small.

1) Subtract Mean: $\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$

$$x := x - \mu$$

2) Normalize variances:

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

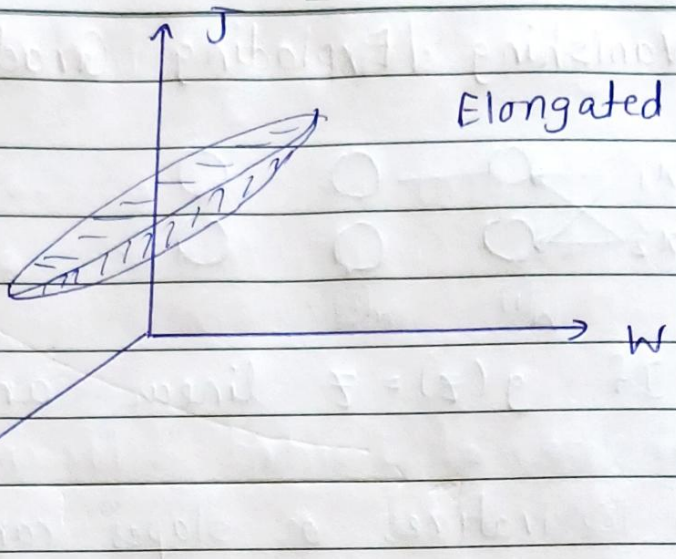



org 1) 2)

Date :.....

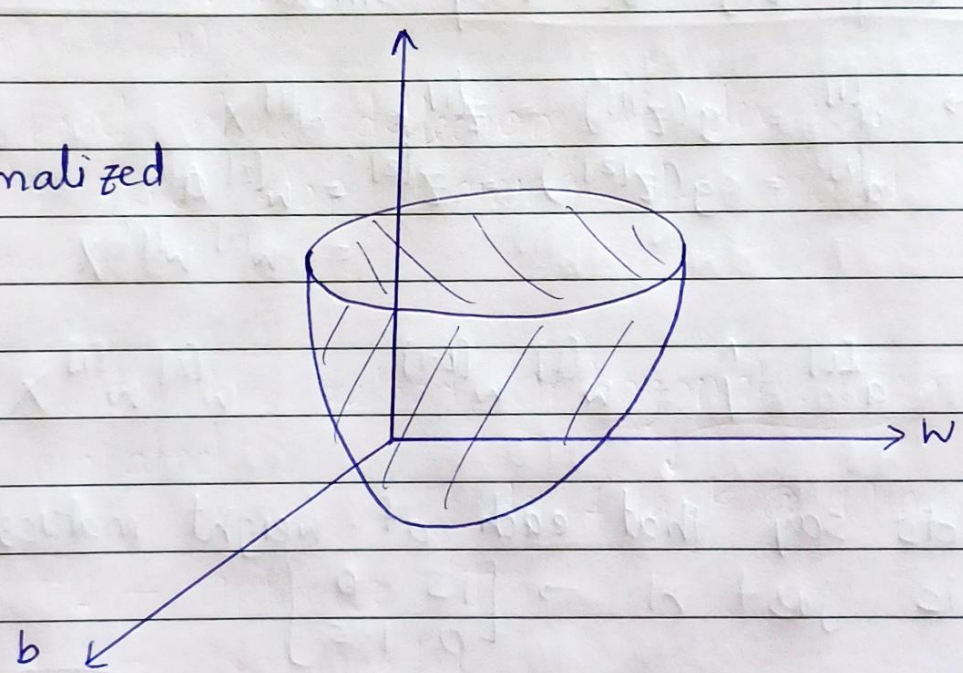
MON TUE WED THU FRI SAT SUN

Unnormalized.



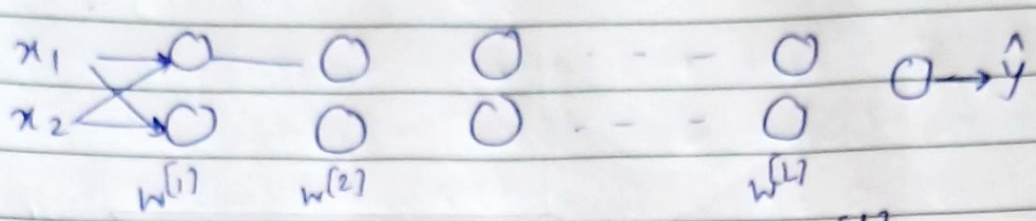

Elongated

Normalized



Date :
MON TUE WED THU FRI SAT SUN

# Vanishing / Exploding Gradients




If $g(z) = z$ linear and $b^{[L]} = 0$

- Derivatives or slopes can get either very big or very small.

$$a^{[1]} = g(z^{[1]}) = z^{[1]} = w^{[1]} x$$
$$a^{[2]} = g(z^{[2]}) = z^{[2]} = w^{[2]} a^{[1]}$$
$$= w^{[2]} w^{[1]} x$$

$$\therefore a^{[L]} = \hat{y} = w^{[L]} w^{[L-1]} \cdots w^{[2]} w^{[1]} x$$

Lets say that each of weight matrics is just $> 1 \rightarrow \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

then $\hat{y} \approx \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$ $\quad (1.5^{L-1})$

$\Rightarrow$ Grows exponentially. (Explode)

similarly $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ $\quad (0.5^{L-1})$

$\Rightarrow$ Vanishes.

Date :
MON TUE WED THU FRI SAT SUN

## → Weight Initialization ←
to prevent vanish/explode

⇒ For single neuron:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b^{0}$$

If $n$ is large → we want small $w_i$ to prevent exploding

$$\therefore \boxed{Var(w_i) = \frac{1}{n}}$$

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

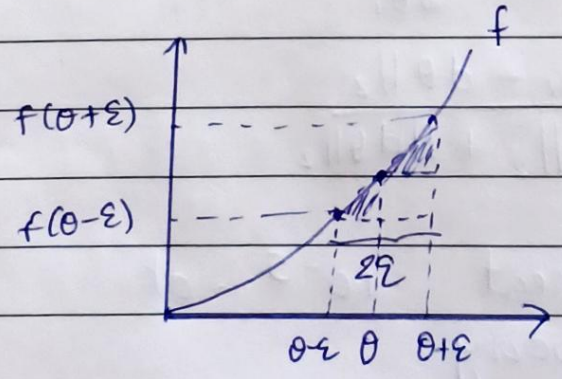
If ReLU then it is $\left(\frac{2}{n}\right)$ & $\sqrt{\frac{2}{n^{[l-1]}}}$

If tanh then $\sqrt{\frac{1}{n^{[l-1]}}}$ (xavier initialisation)

Also used are $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

---

## # Numerical Approx. of Gradients (Gradient checking)




Instead of just moving right, go right then again left so u get a bigger $\delta$

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

Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

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

eg. $f(\theta) = \theta^3$ @ $\theta = 1 \rightarrow 3(1)^2 = 3$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} = 3.0001$$

$\Delta = 0.0001$

In normal 1 sided difference method:

$\Delta = 0.03$

---

Grad. checking:

Reshape $w^{[1]}, b^{[1]}, \cdots w^{[L]}, b^{[L]}$ into a big vector $\theta$

Simly $dw^{[1]}, db^{[1]} \cdots$ into $d\theta$

For each $i$:

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

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

check $$\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$$

$\approx 10^{-7}$ - great $10^{-5}$ - ok

$10^{-3}$ - worry.

Date :
MON TUE WED THU FRI SAT SUN

# optimization Algo.

# Mini-batch gradient descent:

Eg - $m$ = 5 million.

Normally, you process entire dataset then take a small gradient descent step Then again process 5 million.

Solution: Mini-batches.

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

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

Batch Gradient Descent: Algo till now.

Process entire training set at same time

for $t = 1 \dots 5000$:

Forward prop:

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

} vect implem.

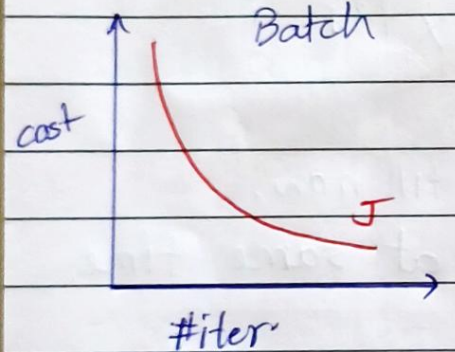
Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

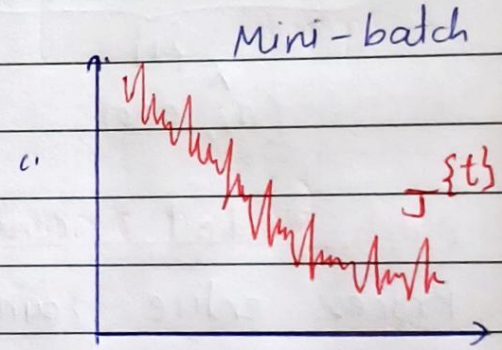
$$\text{Cost } J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 10} \sum_{l} \|w^{[l]}\|_F^2$$

Backprop:

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

This is a single pass (or 1 epoch) thru training set




bcz $X^{\{1\}}, Y^{\{1\}}$ might be easier mini batch but $X^{\{2\}}, Y^{\{2\}}$ might be ~~harder~~.

→ Mini-batch size ←

minibatch size = m : Batch gradient descent

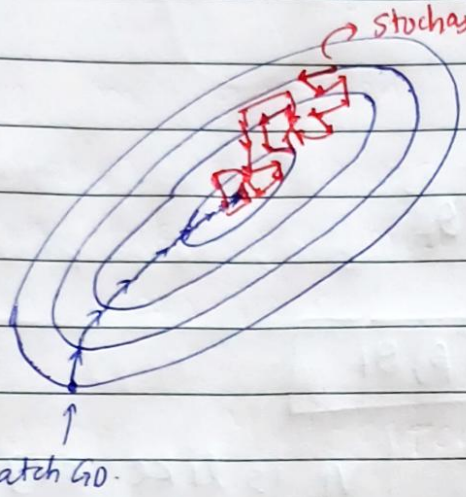
" = 1 : STOCHASTIC GRADIENT DESCENT

Every eg = own mini-batch.

Mini-batch size = Hyperparam.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

So, choose something in betw.



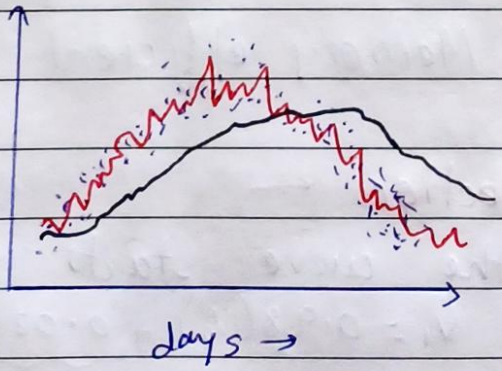

Small train set ($m \leq 2000$) → Batch.

Typical size: 64, 128, 256, 512.

# More optimization Algo other than Grad. Desc.

→ Exponentially Weighted Avg.

eg.




$\theta_1 = 40°F$

$\theta_2 = 49°F$

$\beta$ also a hyperparam.

Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

$V_0 = 0$

$V_1 = 0.9 V_0 + 0.1 \theta_1$

$V_2 = 0.9 V_1 + 0.1 \theta_2$

|

$V_t = 0.9 V_{t-1} + 0.1 \theta_t$

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

$\beta = 0.9$ (Avg over last 10 days)

$\beta = 0.98$ ↑ lot of wt to prev. values

$\beta = 0.5$ $\approx$ 2 days (more noisy).

$V_t$ is like avg over $\approx \frac{1}{1-\beta}$ days temp.

$\Rightarrow V_{100} = 0.1\theta_{100} + 0.9 V_{99}(0.1\theta_{99} + 0.9\theta_{98}) + \cdots$

$= 0.1\theta_{100} + 0.1 \times 0.9 \theta_{99} + 0.1(0.9)^2 \theta_{98}$

$+ \cdots$

Advantage: Memory efficient

→ Bias Correction ←

W/o this, the curve starts very low.

$\because V_0 = 0$ & $V_1 = 0.98(V_0) + 0.02\theta_1$ (0.02 $\theta_1$: very low.)

Date :
MON TUE WED THU FRI SAT SUN

Correction: $v_t = \beta v_{t-1} + (1-\beta)\theta_t$

Instead of $v_t \rightarrow \frac{v_t}{1-\beta_t}$ (t = current day)

eg. $t=2 \quad 1-\beta^t = 1-(0.98)^2 = 0.0396$

$\frac{v_2}{1-\beta t} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$ ↑ sum = $D^r$

# Gradient Descent with Momentum

↳ faster than just G.D.

↳ calc. exponentially weighted avg of gradients & then use it to update wts.

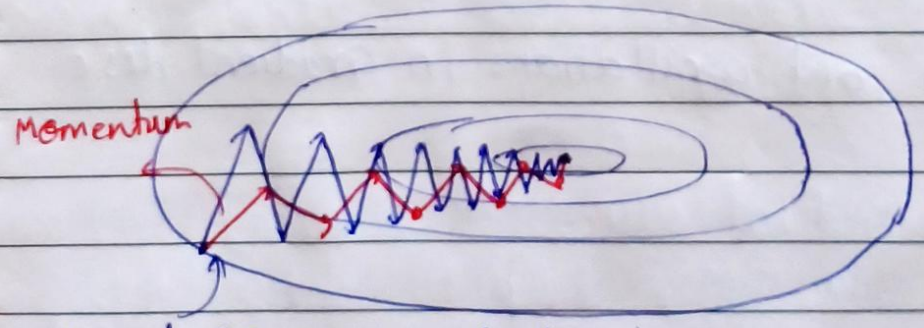
On iteration t:

Compute dW, db on current mini batch/ batch.

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

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

$W := W - \alpha V_{dW} \qquad b := b - \alpha V_{db}$

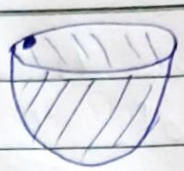



We want
↕ slower
↔ faster learning

Most common $\beta = 0.9$

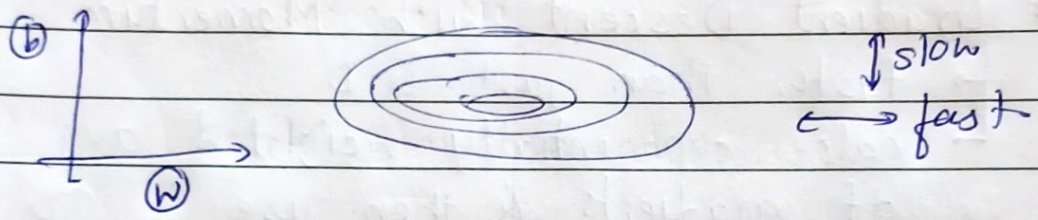
Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

Bowl analogy:
dw, db are like acc. given to ball
$V_{dw}$, $V_{db}$ are velocities & $\beta$ is friction.
Gen'ly we don't do bias corr here.



---

# RMSprop [Root Mean square Prop]




On iter. t:
calc. dw, db on current mini-batch / batch

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

} elem-wise squaring.

$$w := w - \frac{\alpha}{\sqrt{S_{dw}}}$$

$$b := b - \frac{\alpha}{\sqrt{S_{db}}}$$

Now, dw will be small & dw = large.

* damps out oscillations in Gradient desc.

# Adaptive Moment Estimation.

Date : .....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

## # Adam optimization Algo: (RMSProp + Momentum)

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

On iter. t:

Compute $dw, db$ using current mini-batch

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

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

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

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

Bias corr:

$$V_{dw}^{corr} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{corr} = V_{db} / (1-\beta_1^t)$$

$$S_{dw}^{corr} = S_{dw} / (1-\beta_2^t), \quad \text{---}$$

$$w := w - \alpha \cdot \frac{V_{dw}^{corr}}{\sqrt{S_{dw}^{corr}} + \varepsilon} \qquad b := b - \alpha \cdot \frac{V_{db}^{corr}}{\sqrt{S_{db}^{corr}} + \varepsilon}$$

#### Hyperparam. choice:

$\alpha \rightarrow$ to be tuned

$\beta_1 = 0.9$ $\qquad$ $\beta_2 = 0.999$

$\varepsilon = 10^{-8}$ (keep anything doesn't matter)

## # Learning Rate Decay:

Slowly reduce $\alpha$ o/w it will keep oscillating around the minimum.

Date :
MON TUE WED THU FRI SAT SUN

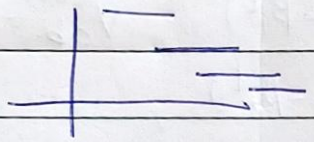
M-1: $\alpha = \frac{1}{1 + \text{decay rate} \cdot \text{epoch No.}} \times \alpha_0$

M-2: Exponential decay:

$\alpha = 0.95^{\text{epoch-No.}} \cdot \alpha_0$

M-3: $\alpha = \frac{\text{const}}{\sqrt{\text{epoch No.}}} \cdot \alpha_0$

M4: Discrete staircase decrease-



M-5: Manually decr. $\alpha$.

# Local optima:

In a very high-dimensional model, u are most likely to get stuck in a saddle point rather than local optimum

Problem = Plateaus → slows down learning very much.
ADAM & RMSprop help to solve this problem.

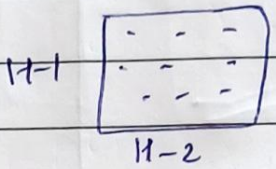
Date : 25/07/24
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☑ ☐ ☐ ☐

# Order of tuning of Hyperparameters

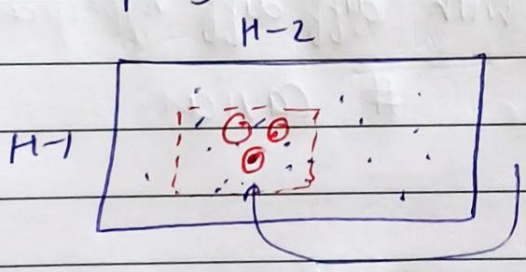
1) → $\alpha$, Most imp.
2) → $\beta$, mini-batch size
3) → # layers, learning rate decay.

# Sampling Hyperparams.

Try Random combinations - don't use grid like



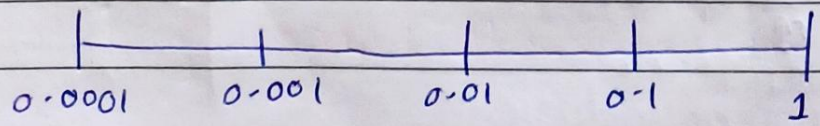

Also, we can COARSE to fine the sampling scheme.




If you find that these points work the best → zoom in to small region & search randomly.

---

Instead of using linear scale to pick random, use log scale.

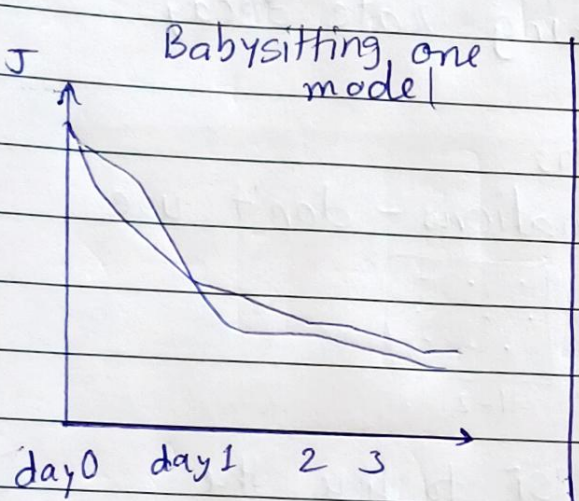


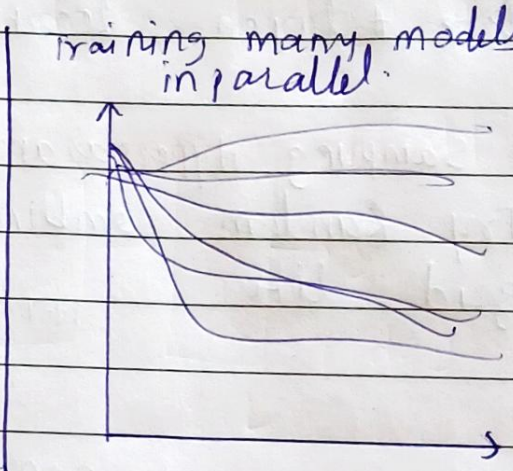
Date :
MON TUE WED THU FRI SAT SUN

$r = -4 * np.random.rand()$ $\Rightarrow r \in [-4, 0]$

$\alpha = 10^r$

Retest hyperparams once every several months.




→ Parallely train models with diff diff hyperparam

# Panda

# Caviar

# Batch Norm

Previously, we only normalized the I/P features. Here we apply this in some hidden layer as well.

If $z^{(1)}$ $z^{(2)}$ ___ $z^{(m)}$ :

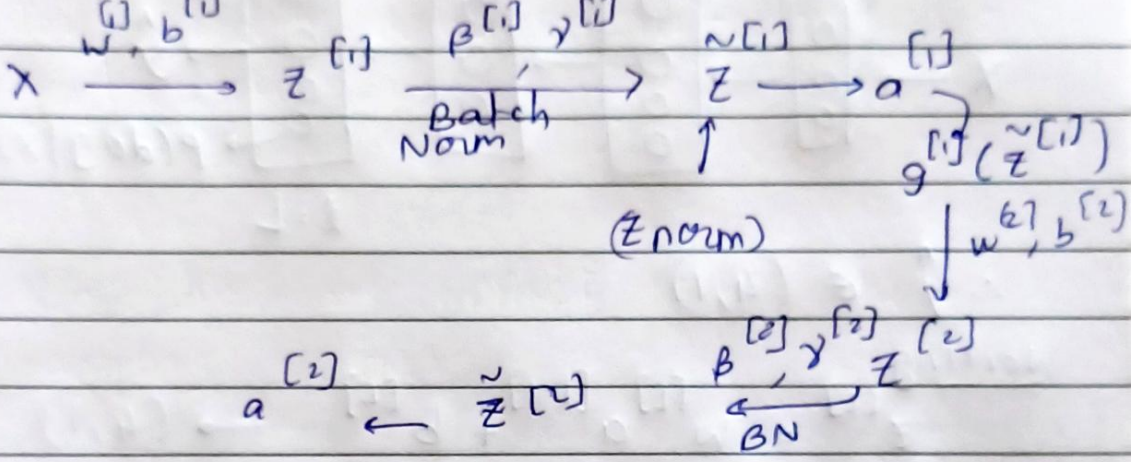
$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

Date :
MON TUE WED THU FRI SAT SUN

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

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$\gamma = \sqrt{\sigma^2 + \varepsilon}$ & $\beta = \mu$ } learnable parameters.



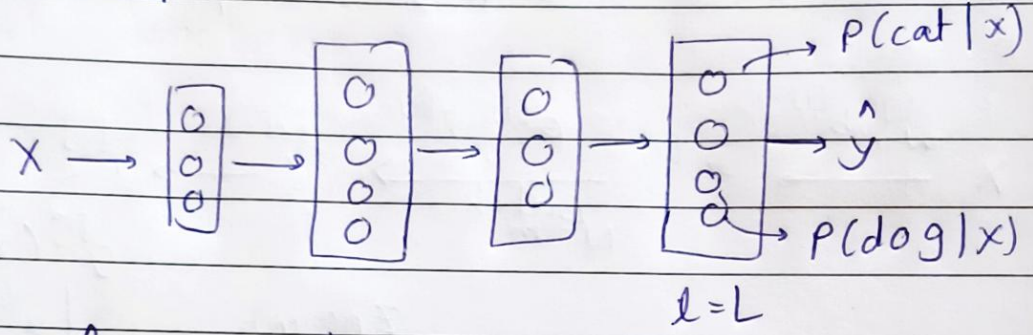

$\beta^{[l]} = \beta^{[l]} - d\beta^{[l]}$ } Updating $\beta$.

Date :
MON TUE WED THU FRI SAT SUN

# Softmax Regression.

- Generalizatn of logistic regr.
- Predict to recognize one of C classes.

eg. Recognizing cats, dogs, chicks, other
$C = 4$




$\therefore \hat{y} \in (4,1)$

Working:

$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]} \rightarrow (4,1)$$

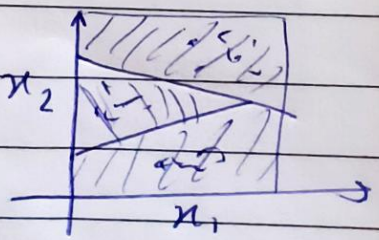
Activation func:

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

$$a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^{4} t_i} \quad , \quad a_i^{[L]} = \frac{t_i}{\Sigma t_i}$$

$(4,1)$

eg. with zero hidden layers
with hidden layers
↓
get a non-linear
decision boundary.



Date :
MON TUE WED THU FRI SAT SUN

[Hard max used to O/P as $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ ]
softmax is more gentle.

If $C = 2$ : softmax = logistic regr.

Loss func. in softmax:

$$y = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \qquad a^{[L]} = \hat{y} = \begin{bmatrix}0.3\\0.2\\0.1\\0.4\end{bmatrix}$$

~~Binary class.~~

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

↳ In this eg. $\mathcal{L}(\hat{y}, y) = -y_2 \log \hat{y}_2 = -\log(\hat{y}_2)$

To make L small, we need $\hat{y}_2$ big.

Vectorized impl:

$$Y = [\, y^{(1)}, y^{(2)} \cdots y^{(m)} \,] = \begin{bmatrix}0 & 1 & \\ 1 & 0 & \cdots \\ 0 & 0 & \end{bmatrix}$$

$$\hat{Y} = [\, \hat{y}^{(1)} \; \hat{y}^{(2)} \cdots \hat{y}^{(m)} \,] = \begin{bmatrix}0.3 & \\ 0.8 & \cdots \\ 0.05 & \\ 0.05 & \end{bmatrix}$$

$(4, m)$

In Tensorflow, we only need to implement forward prop & can use Gradient Tape. for easy backprop.

Date :
MON TUE WED THU FRI SAT SUN

# Grad. Desc. in softmax.

Backprop: $dz^{[L]} = \hat{y} - y$

$(4,1)$

$\partial J / \partial z^{[L]}$

* Usually prog. frameworks take care of backprop.