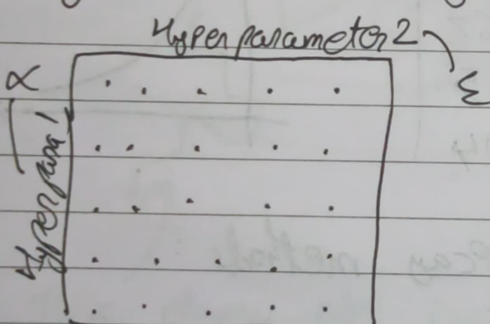
Week 3 :

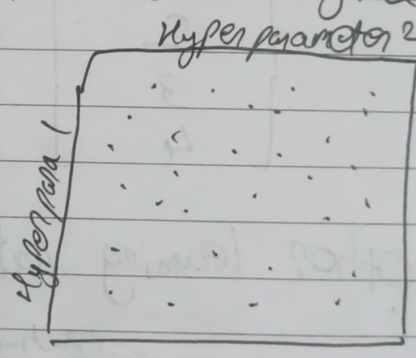
Hyperparameter Tuning :

1) Tuning process

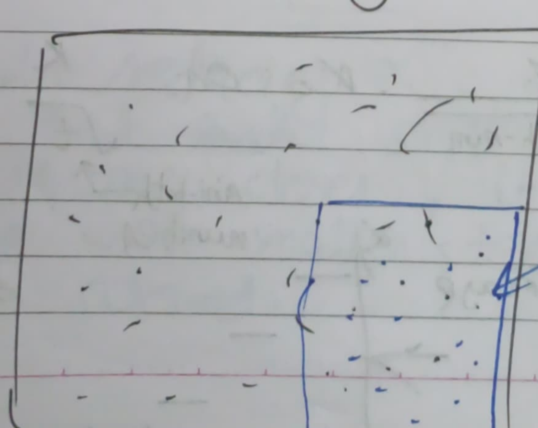
- $\boxed{\alpha}$ - most important (α)
- $\boxed{\beta}$ ~ 0.9 - 2nd
- $\beta_1, \beta_2, \varepsilon \sim 0.9, 0.999, 10^{-8}$
- $\boxed{\text{\# layers}}$ - 3rd
- $\boxed{\text{\# hidden units}}$
- $\boxed{\text{learning rate decay}}$
- $\boxed{\text{mini-batch size}}$

Try random values : Don't use a grid





Coarse to fine sampling scheme



Sample more densely in this smaller square

2) Using an appropriate scale to pick hyperparameters

$n^{[l]} = 50, \ldots, 100$

50 100

x x x x x x x

layers L : 2 - 4

2, 3, 4

$\alpha = 0.0001, \ldots, 1$

x xxx xx x xx → linear

0.0001 1

0.1

10% 90%

$a = -4$

xx x x x x xxx xxx (logarithmic scale)

0.0001 0.001 0.01 0.1 1

$b = 0$

→ this will be ~~a~~ sampling uniformly at random

In python:

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

$\alpha = 10^r \Rightarrow \alpha \in [0.0001, 1]$

$10^a \ldots 10^b$

$r \in [a, b]$ uniformly at random

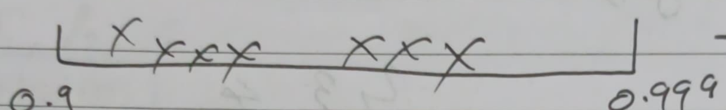
$\alpha = 10^r$

Hyperparameters for exponentially weighted averages

$\frac{1}{1-\beta}$ $\quad \beta = 0.9, \ldots, 0.999$

0.9: average of 10 values 😊

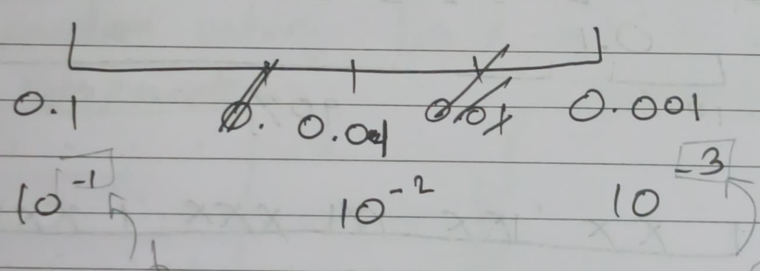
0.999: average of 1000 values 😢



→ linear scale

does not make sense for uniform sampling hence avoid.

Consider, $1-\beta = 0.1, \ldots, 0.001$



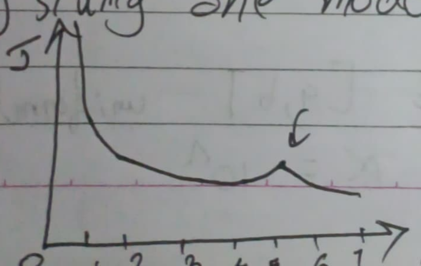
$$r \in [-3, -1]$$

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

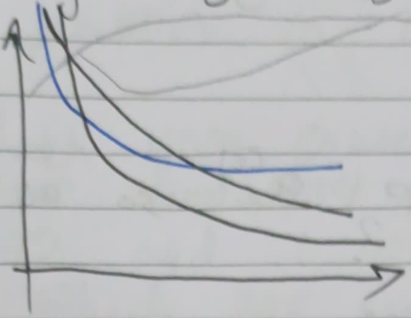
$$\cancel{\beta = 10^{r}}$$

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

3) Baby sitting one model: (panda approach)



Training many models in parallel



→ Pick one that works the best

(Cavier strategy)

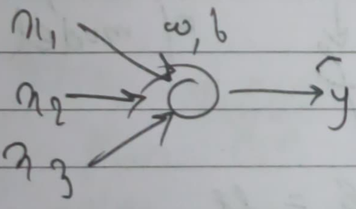
The approach/strategy that we pick depends on the computational resources we have.

If enough computers to train a lot of models in parallel then go with cavier, else if the model is too big which handles a lot of data at the same time then use pandas approach.

Batch Normalization:

1) Normalizing activations in a network:

Normalizing inputs to speed up learning

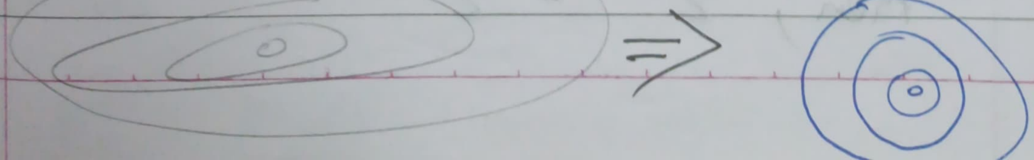


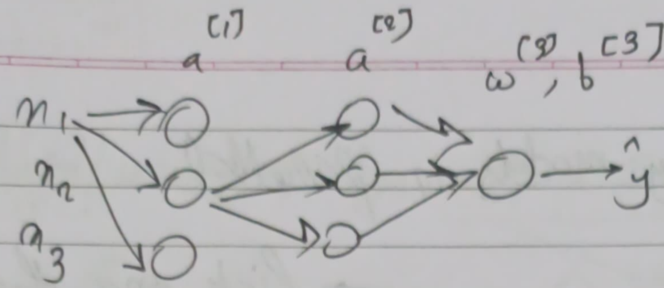
$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu$$

$$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$$

$$X = X / \sigma$$





Can we normalize $a^{[2]}$ so as to train $\omega^{[3]}, b^{[3]}$ faster?

Normalizile $\underline{Z^{[2]}}$

Implimenting Batch Norm

Given some intermediate values in NN

$$\underbrace{z^{(1)}, \dots, z^{(m)}}_{} \xrightarrow[\text{layer } l]{\text{of some}} z^{[l](i)}$$

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

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

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

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

learnable parameters of model

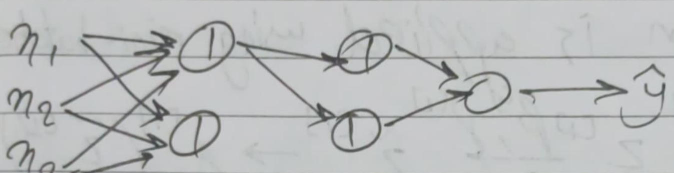
If $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$

then, $\tilde{z}^{(i)} = z^{(i)}$

Use $\tilde{z}^{(i)[l]}$ instead of $z^{(i)[l]}$ for later computations in NN.

Batch norm ensures that the hidden units have standardised mean and variance to 0 and 1 or some other values, which are controlled by γ & β

2) Fitting Batch Norm into a NN



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

$$a^{[1]} \xrightarrow{w^{[2]},\, b^{[2]}} z^{[2]}$$

$$\ldots \longleftarrow a^{[2]} = g^{[2]}(\tilde{z}^{[2]}) \longleftarrow \tilde{z}^{[2]} \xleftarrow[\text{BN}]{\beta^{[2]},\, \gamma^{[2]}} z^{[2]}$$

Parameters : $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \ldots, w^{[L]}, b^{[L]}$

$\beta^{[1]}, \gamma^{[1]}, \ldots, \beta^{[L]}, \gamma^{[L]}$

Note: These values of β are different than the β we had as an hyperparameter in momentum, RMS Prop and Adam algorithms.

$d\beta^{[l]}$ can be computed to update β as:

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

We can use other algos like momentum, RMS prop or Adam also to update β.

Using tensorflow we can implement BN as:

→ tf. nn. batch-normalization

Batch-norm is applied using mini-batches also:

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

$$X^{\{2\}} \longrightarrow z^{[1]} \xrightarrow[BN]{} \tilde{z}^{[1]} \longrightarrow \dots$$

⋮

dim = $(n^{[l]}, 1)$ like $z^{[l]}$

Parameters: $w^{[l]}, \cancel{b^{[l]}}, \beta^{[l]}, \gamma^{[l]}$

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

since it gets cancelled when we rescale $z^{[l]}$ / or setting it to 0 permanently.

Implementing gradient descent

for $t = 1 \dots$ num_Mini-batches
compute forward prop on $X^{\{t\}}$
In each hidden layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$
Use back prop to compute $dw^{[l]}, \cancel{db^{[l]}}, d\beta^{[l]}, d\gamma^{[l]}$

Update parameters :

$$\left.\begin{aligned} w^{[l]} &:= w^{[l]} - \alpha \, dw^{[l]} \\ \beta^{[l]} &:= \beta^{[l]} - \alpha \, d\beta^{[l]} \\ \gamma^{[l]} &:= \gamma^{[l]} - \alpha \, d\gamma^{[l]} \end{aligned}\right\}$$

Note: Also works with other algos like momentum, RMS prop, adam.

~~4) Learning on shifting input distribution~~

5) Batch Norm at test time

Batch norm processed the train data in mini-batches but ~~in test~~ during test phase we will need to process the examples one at a time.

During test time:
Come up with seperate μ, σ^2

μ, σ^2 : estimate using exponentially weighted average (~~using~~ across mini-batches)

$$X^{\{1\}}, \; X^{\{2\}}, \; X^{\{3\}}, \ldots$$

$$\begin{array}{ccccc} \downarrow & \downarrow & \searrow & & \\ \mu^{\{1\}[l]} & \mu^{\{2\}[l]} & \mu^{\{3\}[l]} & \rightsquigarrow & \mu \\ \sigma^{2\{1\}[l]} & \sigma^{2\{2\}[l]} & \ldots & \longrightarrow & \sigma^2 \\ \theta_1 & \theta_2 & \theta_3 & & \end{array}$$

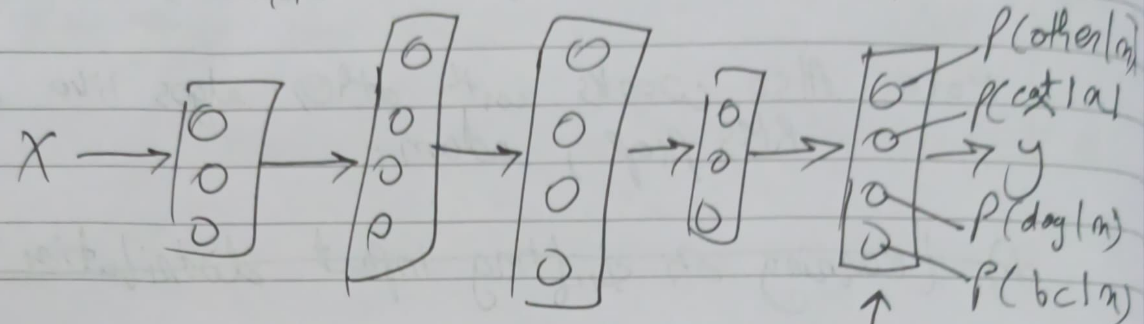
$$z_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}} \;, \quad \tilde{z} = \gamma z_{norm} + \beta$$

Multi-class Classification

1) Softmax Regression

eg: Recognizing cats, dogs, and baby chicks

$C = \#\text{classes} = 4 \quad (0, 1, 2, 3)$



$n^{[L]} = 4 = C$

$\hat{y}$ is $(4,1)$ because it gives us 4 probabilities

Softmax layer:

final layer

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

Activation function:

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

$$\hat{y} = a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^{4} t} \rightarrow (4,1)$$

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

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

$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$, $\sum_{j=1}^{4} t_j = 176.3$

$a^{[L]} = \frac{t}{176.3}$

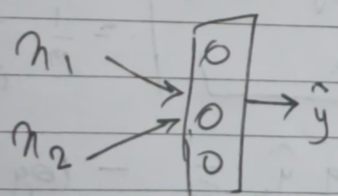
$\therefore a_1^{[L]} = \frac{e^5}{176.3} = 0.842$

$a_2^{[L]} = \frac{e^2}{176.3} = 0.042$

$a_3^{[L]} = 0.002$, $a_4^{[L]} = 0.114$

softmax activation f^n : $\underset{(4,1)}{a^{[L]}} = g^{[L]}(\underset{(4,1)}{z^{[L]}})$

softmax examples



$z^{[1]} = w^{[1]}x + b^{[1]}$

$a^{[1]} = \hat{y} = g(z^{[1]})$

$c = 4 \Rightarrow$ 4 decision boundaries across the plot when there are no hidden layers.

Page No.
Date

2) Training a softmax Classifier

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

\# Softmax regression generalizes logistic regression to C classes.

If $C = 2$, then softmax reduces to logistic reg.

Note: In contrast to softmax, there is hardmax which gives an output matrix generated from $z^{[L]}$. It outputs 1 for the highest element in $z^{[L]}$ and 0 for the rest of the elements

Loss function:

eg: $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $y_2 = 0$, $a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$, $C = 4$

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

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

single example:

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

make $\hat{y}_2$ as big as possible.

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

$$= \begin{bmatrix} 0 & 0 & \cdots & 1 \\ 1 & 0 & & 0 \\ 0 & 1 & & 0 \\ 0 & 0 & & 0 \end{bmatrix} \leftarrow (4, 1)$$

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

$= \begin{bmatrix} 0.3 & & \cdots & \\ 0.2 & & & \\ 0.1 & & & \\ 0.4 & & & \end{bmatrix} \leftarrow (4, m)$

Gradient descent with softmax :

Back prop : $\underset{(4,1)}{dz^{[L]}} = \hat{y} - y^{[L]}$, $\hat{y} = a^{[L]}$

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