* Hyperparameters (Week 3)

≈0.9 0.9 0.999 10^{-8}

α, β, β_1, β_2, ε, layers, hidden unit, learning rate decay, mini-batch size.

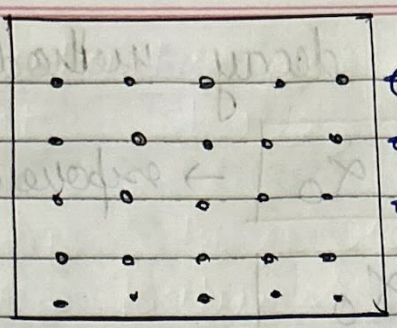
① □
② —
③ □
④ —

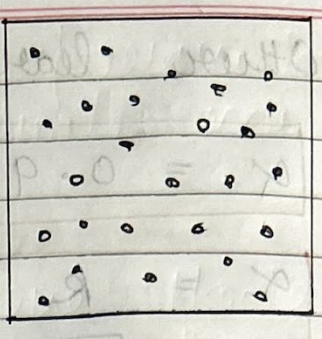
} Priority order of hyperparameters

- It's difficult to know the priority & range of hyperparameters beforehand, thus we'll have to explore.

- By using random over grid, we're ensuring that we're exploring richly as there's more values to try in random.

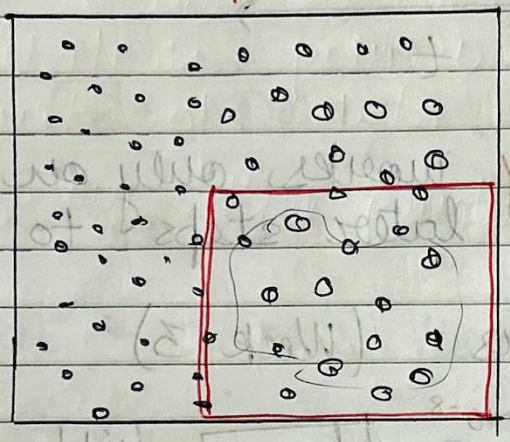
H2 H2





Using Grid < Using Random

- Coarse to fine :

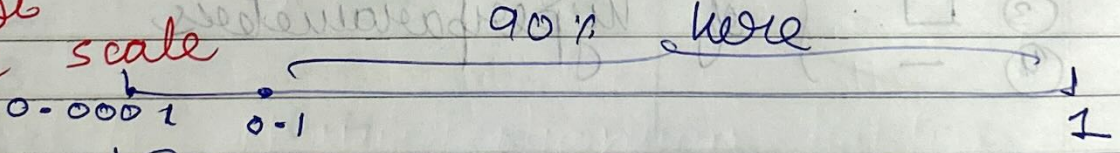


→ after we realise that H1 & H2's optimum values are concised in that area.

* Appropriate Scale for Hyperparameters

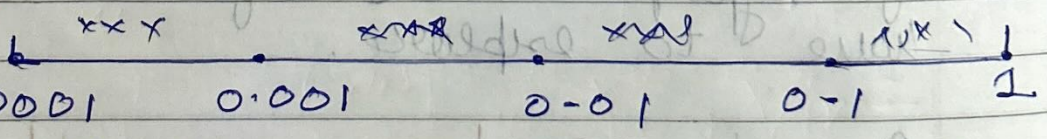
$\alpha = 0.0001 \ldots\ldots 1$

number line scale



∧ only 10% resources here ; unfair.

log scale



thus, dividing resources this way is giving uniform random values

$r = -4 * np.random.rand()$

$r \in [-4, 0]$

$\alpha = 10^r \leftarrow 10^{-4} \ldots 10^0$

Logic:

$10^a \ldots 10^b$

where $a = \log_{10} 0.0001$

$= -4$

$b = \log_{10} 1$

$= 0$.

* Hyperparameters for exponentially weighted averages:

$\beta = 0.9 \ldots 0.999$

$\downarrow$ $\downarrow$

10 1000

0.9 — 0.99 — 0.999

$1 - \beta = 0.1 \ldots 0.001$

allows dense exploring to tune properly

0.1 — 0.01 — 0.001

10^{-1} resources are equally divided 10^{-3}

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

$1 - \beta = 10^r \Rightarrow \beta = 1 - 10^r$.

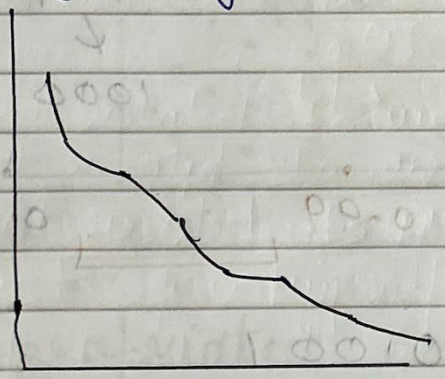
- We're using logarithmic scale over linear scale to distribute resources more efficiently, & because $1-\beta$ is highly sensitive to small changes as well.

- Intuitions for hyperparameters do get stale, re-evaluate occasionally.

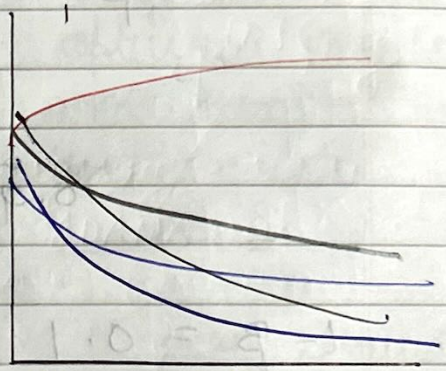
Ways to re-evaluate:
1. Baby-Sitting One Model
→ when you don't have GPUs
high computation

2. Training many models in parallel.

Which one to choose depends on computation power available & the amount of data. Caviar.
huge, huge, then panda.



One model
(Panda)

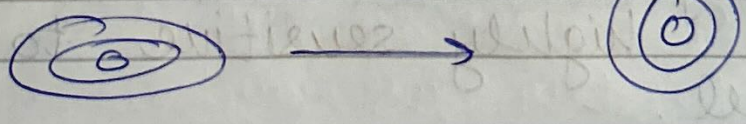


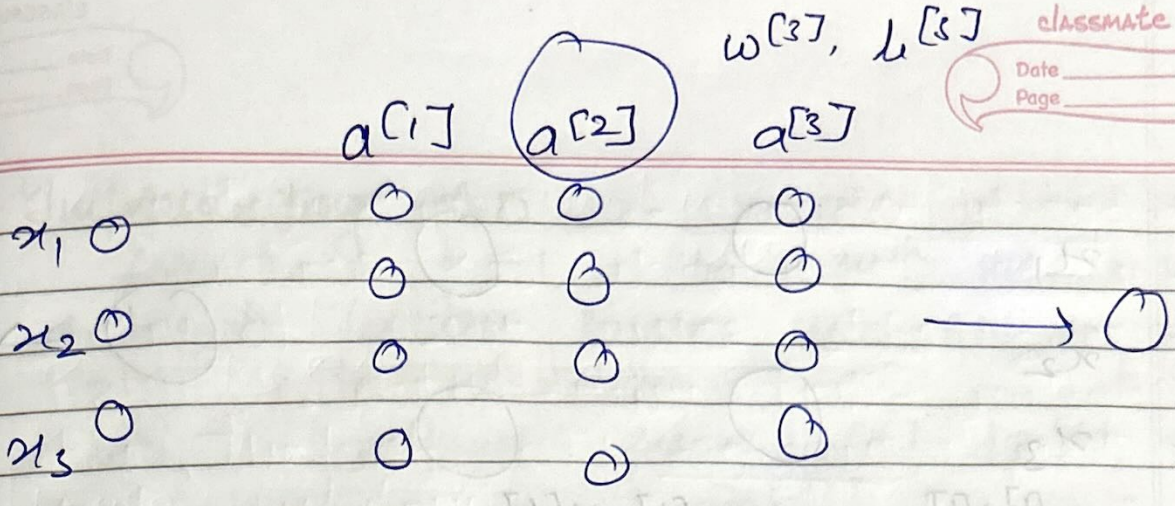
Many Models
(Caviar)

* Batch Normalisation

$$\mu = \frac{1}{m} \sum x^{(i)} \qquad X = X - \mu$$

$$\sigma^2 = \frac{1}{m} \sum (x^{(i)})^2 \qquad X = X / \sigma^2$$





Here, if we normalise $a^{[2]}$ so as to compute $w^{[3]}, b^{[3]}$ faster

Further, normalising $z^{[2]}$ is an even better option.

- Implementing Batch Norm: applies normalisation to deeper units

Given some intermediate value in NN:

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

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

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

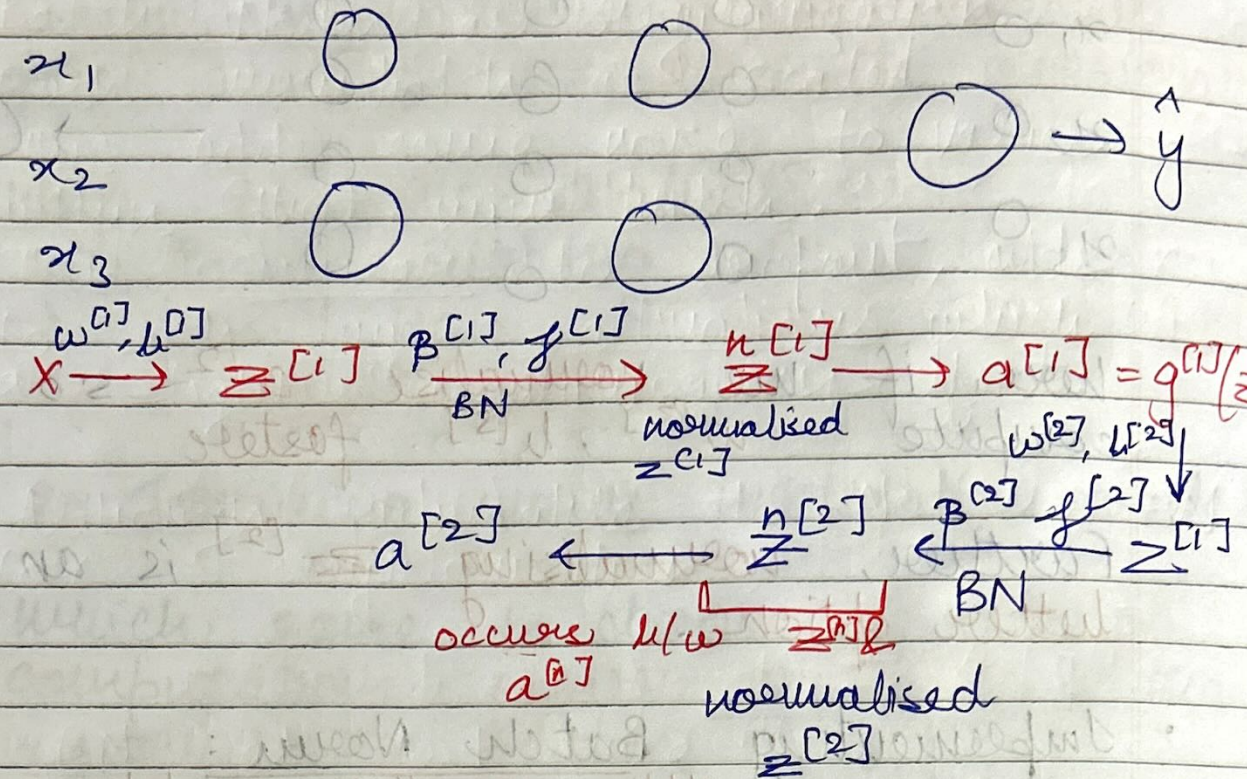
$$\tilde{z}^{(i)} (z \text{ total}(i)) = \gamma z^{(i)}_{norm} + \beta$$

gamma

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

$z^{u[e](i)} >> z^{[e](i)}$

mean & variance, are controlled by γ & β normalised throughout the network.



Parameters : $w^{[1]}, b^{[1]}, \ldots\ldots, w^{[L]}, b^{[L]}$
$\beta^{[1]}, \gamma^{[1]}, \ldots\ldots, \beta^{[n]}, \gamma^{[n]}$

→ not related to momentum
→ not related to weighted averages.
→ not related to RMS prop & ADAMS
→ not a hyperparameter

Now, can apply GD:

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

- BN is applied on mini-batches ($n^{[l]}, m$) ($n^{[l]}, m$) ($n^{[l]}, m$)
- Parameters : $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$

can be eliminated, as in the mean subtraction step, BN zeroes out the mean of $b^{[l]}$

- Implementing GD:
  for $t = 1$ to m^{th} mini batch
  In each hidden layer, use BN to normalise & replace $z^{[l]}$ with $\tilde{z}^{[l]}$
  Use backprop for $dw^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$, ~~$db^{[l]}$~~,
  Update parameters:
  $$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$
  $$\beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$$
  ⋮

* Learning on shifting Input Distribution:

- BN reduces the problem of input batches changing, it lowers the impact of previous layers on the upcoming one. Allows each layer to learn by itself & thus optimise the code.

- Due to this, the previous layers do not get switched arounds as they're constrained to mean & variance.

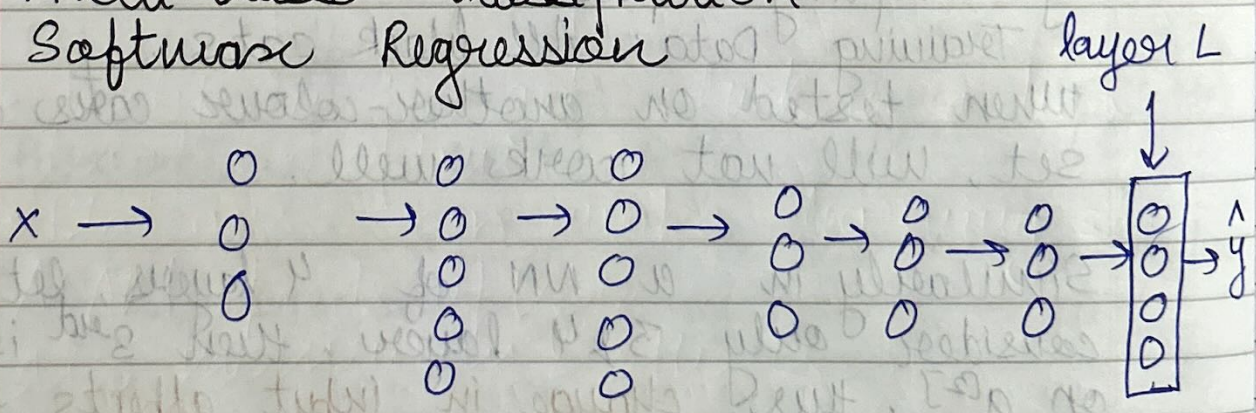
Eg: shifting Input.
Training Data: All black cats.
When tested on another-colour cates as test set, will not work well.

Similarly in a NN of 4 layers, let's consider only 3 & 4 layer, then 3rd is dependent on $a^{[2]}$, thus change in input affects the output

* Batch Norm as regularization :
- Each mini batch is scaled
- This adds some noise to values $z^{[l]}$ within that mini-batch.
- Thus, similar to dropout, adds noise to each layer's activations which has a slight regularization effect.
- Although advisable to Not use BN for regularisation

* BN at test time :
- At test time, we do not take an entire mini-batch, rather every example individually
- Thus μ & σ^2 for single value is useless. Thus people usually implement an exponentially weighted average to keep track of μ & σ^2 obtained from train set.

* Multi Class Classification
Softmax Regression



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

Activation func^n:

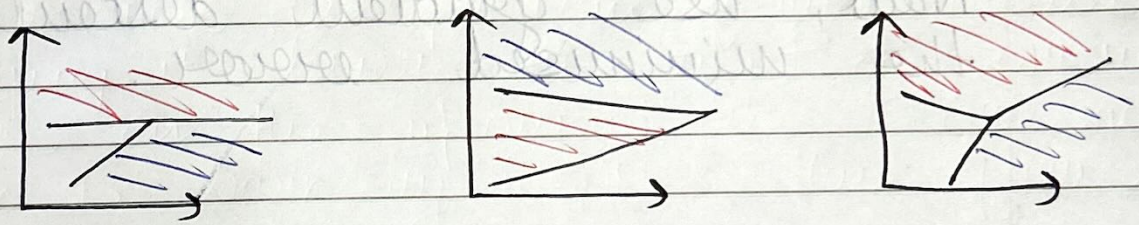
$$t = e^{(z^{[L]})} \Rightarrow \text{temporary variable}$$

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

(4,1)

term to normalise

- Difference from ReLU & tanh as it uses vector input & output.
- More than 2 classes.



- Can give many linear & non linear decision boundary.

Eg: $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $\quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$

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

$$= \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

Softmax Regression generalises logistic regression to C classes.

If C = 2, softmax reduces to logistic regression

* Loss func^n in Softmax:

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

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

Now, use gradient descent calculate the minimised error.

* The problem of local optima:

- GD does not run well.
- Not all points that appear as local optima are minimas, they're actually saddle points
- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow.

↳ ADAM speeds up more also momentum & others