Optimizers & Hyperparameter Tuning

Batch vs. mini-batch gradient descent

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

(n_x, m) $\quad X^{\{1\}}\ (n_x, 1000)\quad X^{\{2\}}\ (n_x, 1000)\ \dots\quad X^{\{5000\}}\ (n_x, 1000)$

M=5,000,000 5000 mini batches of 1000 each

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

$(1, m)$ $\quad Y^{\{1\}}\ (1, 1000)\quad Y^{\{2\}}\ (1, 1000)\quad \dots\quad Y^{\{5000\}}\ (1,1000)$

Mini batch gradient Descent

Size of the mini batch needs to be chosen carefully

If mini batch size = m then it is same as batch gradient descent

If mini batch size =1 then all individual training examples are mini batch in themselves. The training becomes very slow and we can not use vectorization.

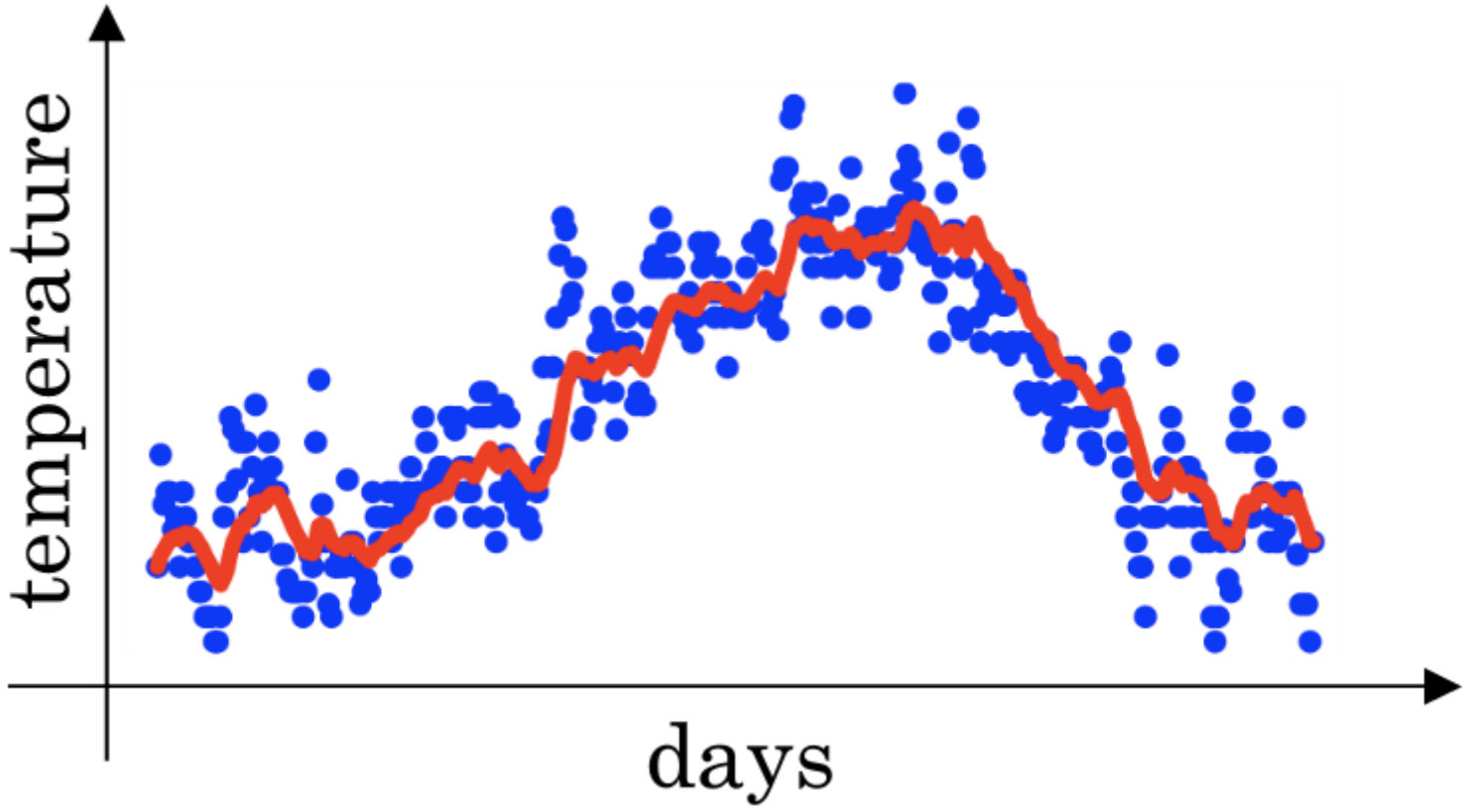
Mini batch GD helps us to make early progress without iterating through the entire data set and results in faster learning. It also takes advantage of vectorization.

So, the size need not be too big or too small. To take advantage of the CPU/GPU Memory the mini batch size may be in context of the available RAM.

Generally the minibatch size is taken as a power of 2. A mini batch of 512 or 1024 depending upon your application can be your starting point.

If the training data set is less than 5000 then there is no need of using mini-batches.

Exponentially weighted averages

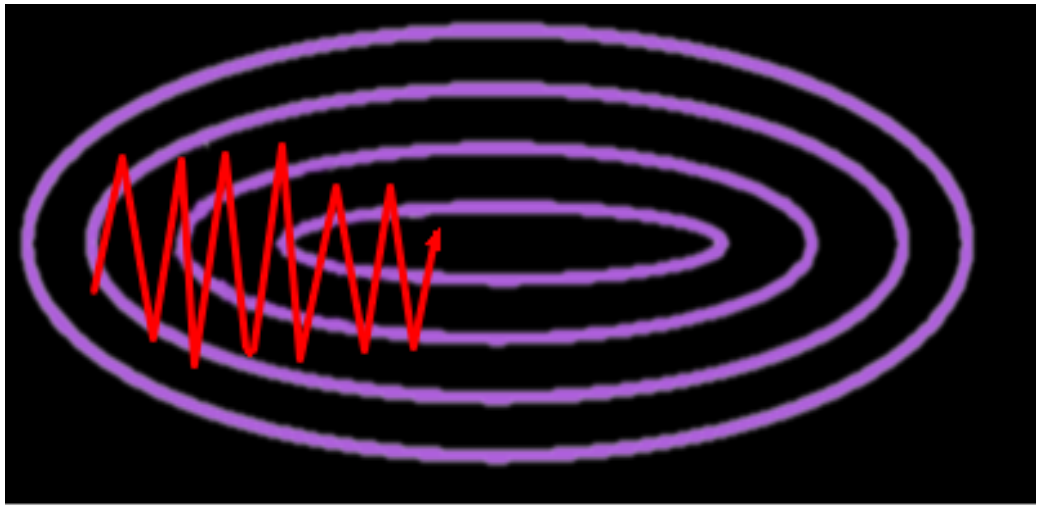


Exponentially weighted averages

Let us understand with the example of every day Temperature for a one year period in Delhi

- $V_{100} = 0.9V_{99} + 0.1\theta_{100}$
- $V_{99} = 0.9V_{98} + 0.1\theta_{99}$
- $V_{98} = 0.9V_{97} + 0.1\theta_{98}$
- $V_t = \beta V_{t-1} + (1-\beta)\theta_t$
- $\beta = 0.9$ V_t is approximately average over $1/1-\beta$ days eg 1/1-0.9 = 10 days

Momentum



- On iteration t
- Compute dw, db on current mini-batch
- $V_{dw} = \beta V_{dw} + (1-\beta) dw$
- $V_{db} = \beta V_{db} + (1-\beta) db$
- $w = w - \alpha V_{dw}$

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

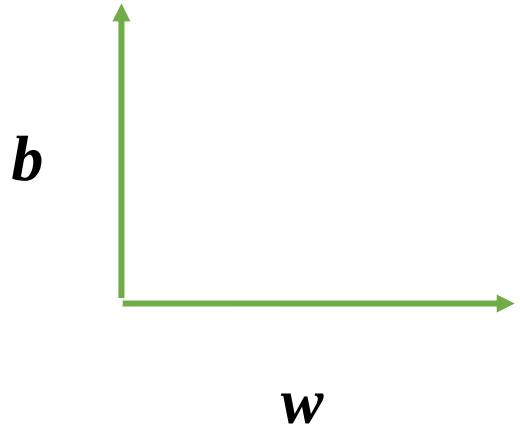
β is new hyperparameter and the recommended value is 0.9

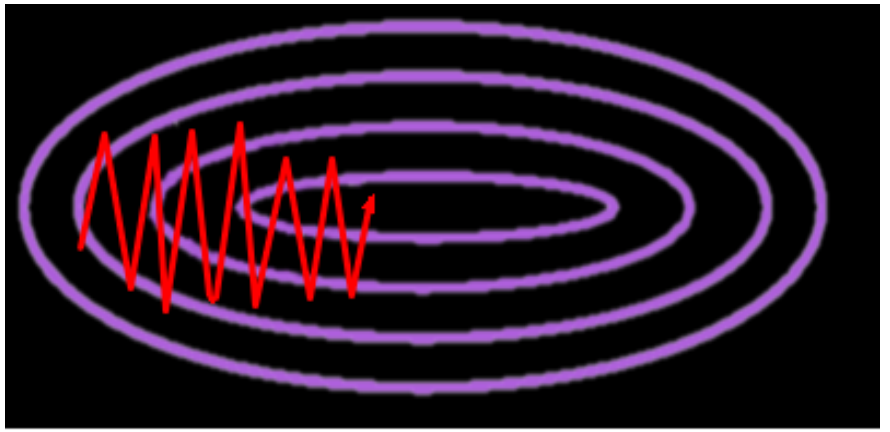
Momentum

- It helps accelerate gradient vectors in the right directions, that helps in faster convergence.
- It will moderate the movements in the vertical direction and will help us move faster in the horizontal direction.
- Instead of calculating gradients independently we use exponentially weighted averages to calculate the gradient.

RMSProp (Root Mean Square Prop)

- On iteration t
- Compute dw, db on current mini-batch
- $S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$
- $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$
- $w = w - \alpha \frac{dw}{\sqrt{S_{dw}} + \epsilon}$
- $b = b - \alpha \frac{db}{\sqrt{S_{db}} + \epsilon}$
- $\epsilon = 10^{-8}$
- b is large and w is small





Adam optimization Algorithm

- On iteration t
- Compute dw, db on current mini-batch
- $V_{dw}= \beta_1 V_{dw}+(1-\beta_1)dw$ $V_{db}= \beta_1 V_{db}+(1-\beta_1)db$
- $S_{dw}= \beta_2 S_{dw}+(1-\beta_2)dw^2$ $S_{db}= \beta_2 S_{db}+(1-\beta_2)db^2$
- $V_{dw}^{corrected} \frac{v_{dw}}{1-\beta_1^t}$ $V_{db}^{corrected} \frac{v_{db}}{1-\beta_1^t}$
- $S_{dw}^{corrected} \frac{S_{dw}}{1-\beta_2^t}$ $S_{db}^{corrected} \frac{S_{db}}{1-\beta_2^t}$
- $w= w-\alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}}+\epsilon}$
- $b=b-\alpha \frac{V_{dB}^{corrected}}{\sqrt{S_{db}}+\epsilon}$

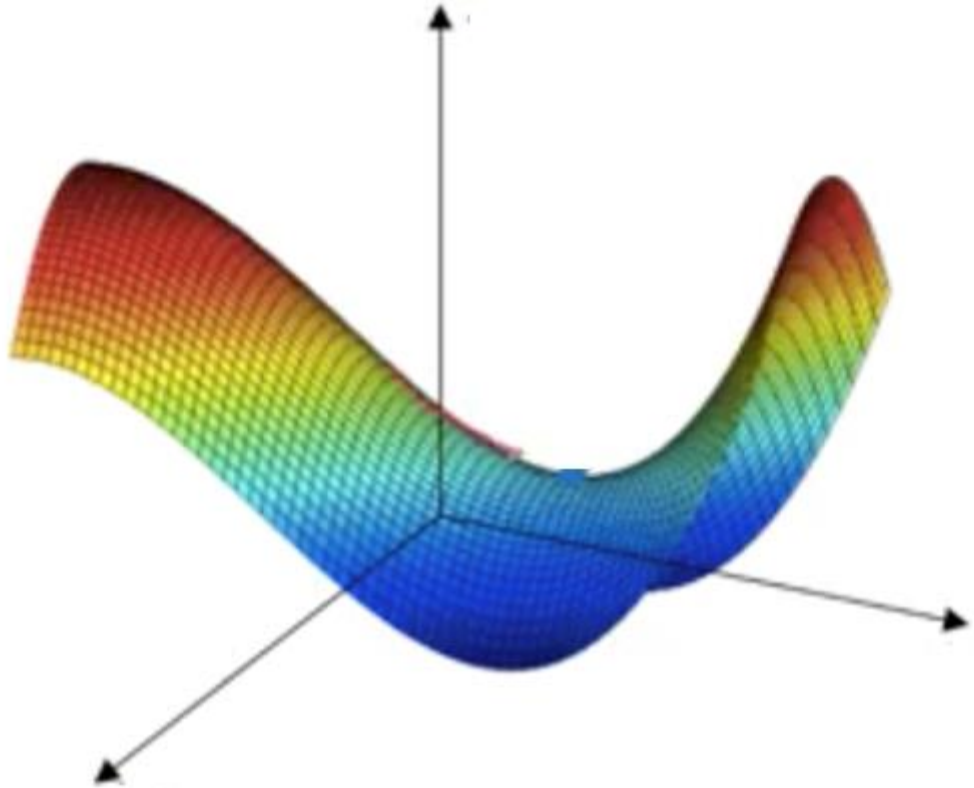
β_1=0.9 (dw) also referred to as moment 1
β_2=0.999 (dw^2) also referred to as moment 2
$\epsilon = 10^{-8}$

Adaptive Moment estimation

Adam Optimization Algorithm

- It is a combination of Momentum and RMS Prop Algorithm.
- It has proved good across different Neural network architectures to improve the learning speed
- It can handle sparse gradients in noisy problem
- Hyper parameters β_1 , β_2 and $\in$ require little tuning and have intuitive interpretation

Problem of plateaus and saddle points



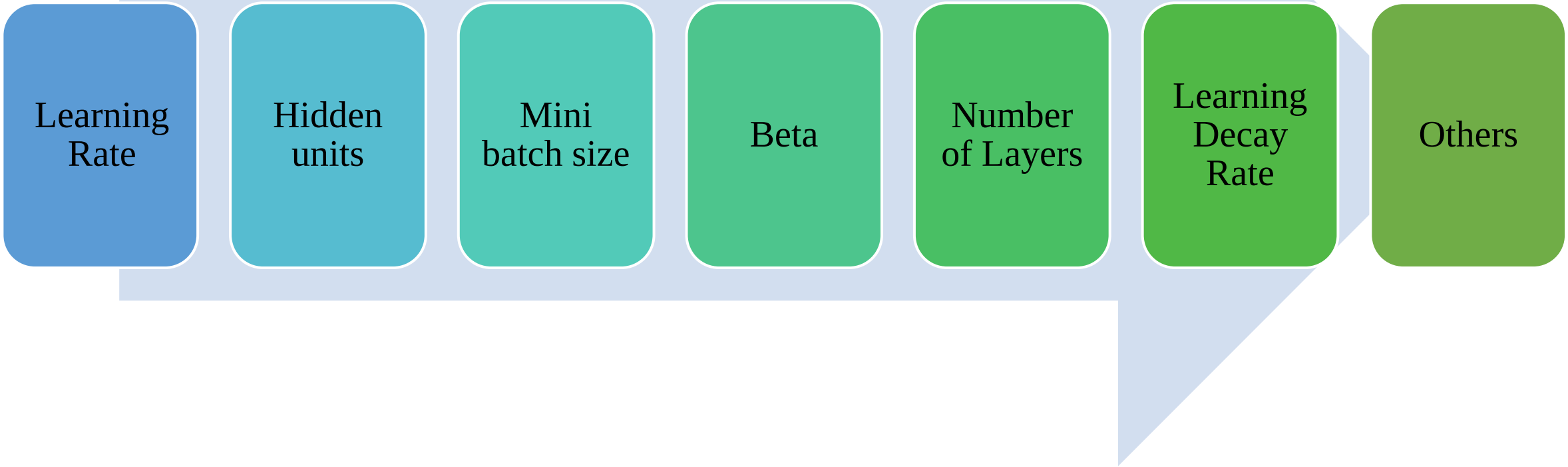
Plateaus can make learning slow
Unlikely to stuck in local minima

Learning rate decay

- 1 epoch = 1 pass through entire data
- $\alpha = \frac{1}{1+Decay\ rate * epoch\_number} \alpha_0$
- α_0=0.2
- $\alpha = \frac{1}{1+1*1} 0.2 = 0.1$
- $\alpha = \frac{1}{1+1*2} 0.2 = 0.67$
- $\alpha = \frac{1}{1+1*3} 0.2 = 0.5$
- $\alpha = \frac{1}{1+1*4} 0.2 = 0.4$

How to try Hyperparameters

First focus on Most important ones and then the lesser ones in the sequence



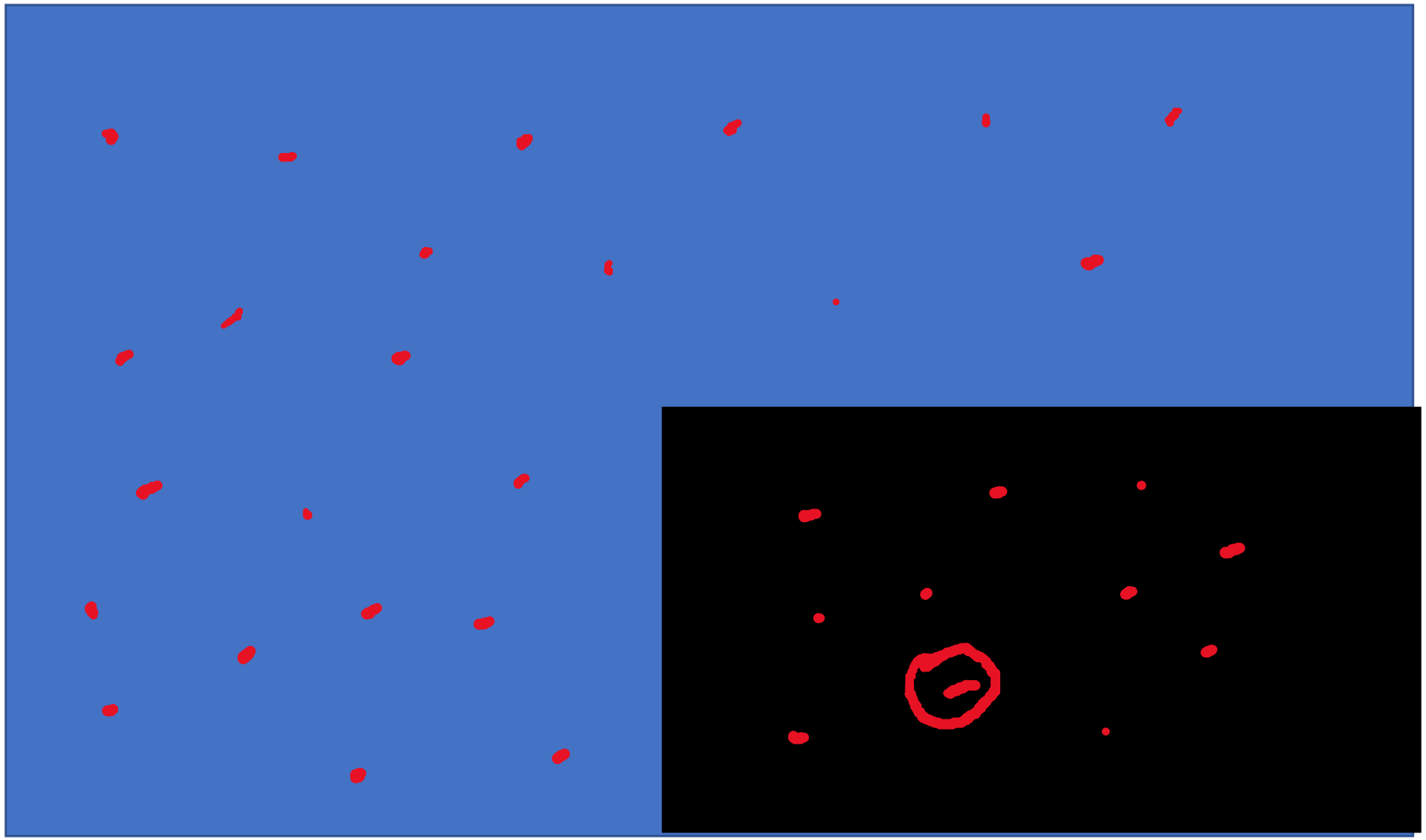
Random is better than a Grid

α

*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*

β

Coarser to finer



Picking hyperparameter at random and as per scale

- Is it ok to use the actual scale for all parameters or in some cases we require the log scale

Babysitting one model Vs Parallel model training

Panda or Caviar Approach

Depends upon the applications, resources and the time you have.

In babysitting we painstakingly try to observe and introduce mid-path corrections.

In Parallel model training we simultaneously try to train with different models