Dataset & Experiments

How to improve the model

- More data may be required
- Data needs to have more diversity
- Algorithm needs longer training
- More hidden layers or hidden units are required
- Add Regularization
- Change the Neural network architecture like activation function etc.
- There are many other considerations you can think of..

Distribution of data for improving the accuracy of the model

Training set	Which you run your learning algorithm on.
Dev (development) set	Which you use to tune parameters, select features, and make other decisions regarding the learning algorithm. Sometimes also called the **hold-out cross validation set** .
Test set	which you use to evaluate the performance of the algorithm, but not to make any decisions regarding what learning algorithm or parameters to use.

Data Distribution Mismatch

It is naturally good to have the data in all the sets from the same distribution.

For example Housing data coming from Mumbai and we are trying to find the house prices in Chandigarh.

Else wasting a lot of time in improving the performance of dev set and then finding out that it is not working well for the test set.

Sometime we have only two partitioning of the data in that case they are called Train/dev or train/test set.

Optimize one parameter and satisfy others

Classifier	Accuracy	Running Time	Safety	False Positive	…	..
A	90	20ms	No			
B	92	80ms	Yes			
C	96	2000ms	Yes			

Maximize ????? Subject to ????? And ????? And…

So few of them can be satisficing metric.
e.g if we say that running time needs to be minimum 100ms that running time is satisficing metric and accuracy can be optimizing metric

Start Quickly Then Optimize

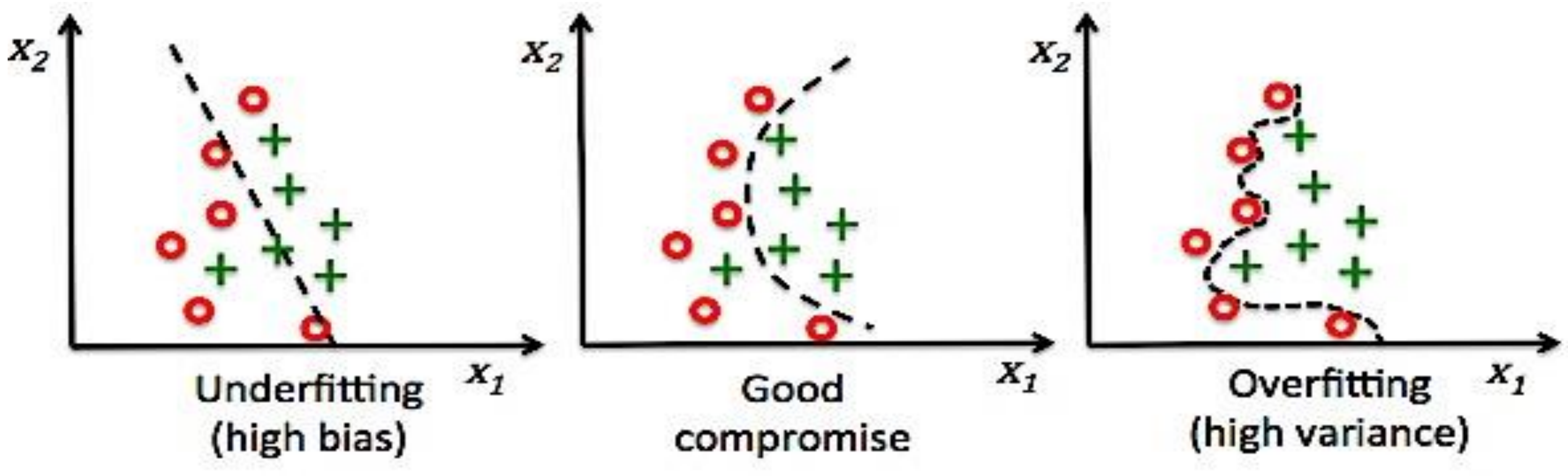
✓Don’t start off trying to design and build the perfect system.

✓Build and train a basic system quickly

✓It is valuable to examine how the basic system functions

✓Find clues that show you the most promising directions in which to invest your time.

Bias/Variance



Bias/Variance

The algorithm’s error rate on the training set is algorithm’s **bias** .

How much worse the algorithm does on the dev (or test) set than the training set is algorithm’s **variance** .

Bias/Variance

Train Set Error	1	12	8	1
Dev Set Error	9	13	16	1.5
	High Variance	High Bias	High Bias, High Variance	Low Bias Low Variance
	Overfitting	Underfitting	Underfitting	Good fit

Multi dimensional system can have high bias in some areas and high variance in some other areas of the system, resulting in High Bias and High Variance issue

High Bias

Increase the model size (such as number of neurons/layers)	It allows to fit the training set better. If you find that this increases variance, then use regularization, which will usually eliminate the increase in variance.
Modify input features based on insights from error analysis	Create additional features that help the algorithm eliminate a particular category of errors. These new features could help with both bias and variance.
Reduce or eliminate regularization (L2, L1 regularization, dropout)	reduces avoidable bias, but increase variance.
Modify model architecture (such as neural network architecture) so that it is more suitable for your problem	This can affect both bias and variance.

High Variance

Add more training data	Simplest and most reliable way to address variance, so long as you have access to significantly more data and enough computational power to process the data.
Add regularization (L2, L1 regularization, dropout)	This technique reduces variance but increases bias.
Add early stopping (stop gradient descent early, based on dev set error)	Reduces variance but increases bias.
Modify model architecture (such as neural network architecture) so that it is more suitable for your problem	This affects both bias and variance.

Regularization

This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.

Any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error

Regularization for a Neural Network

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

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

$w: (n^{[l-1]}, n^{[l]})$

Frobenius Norm

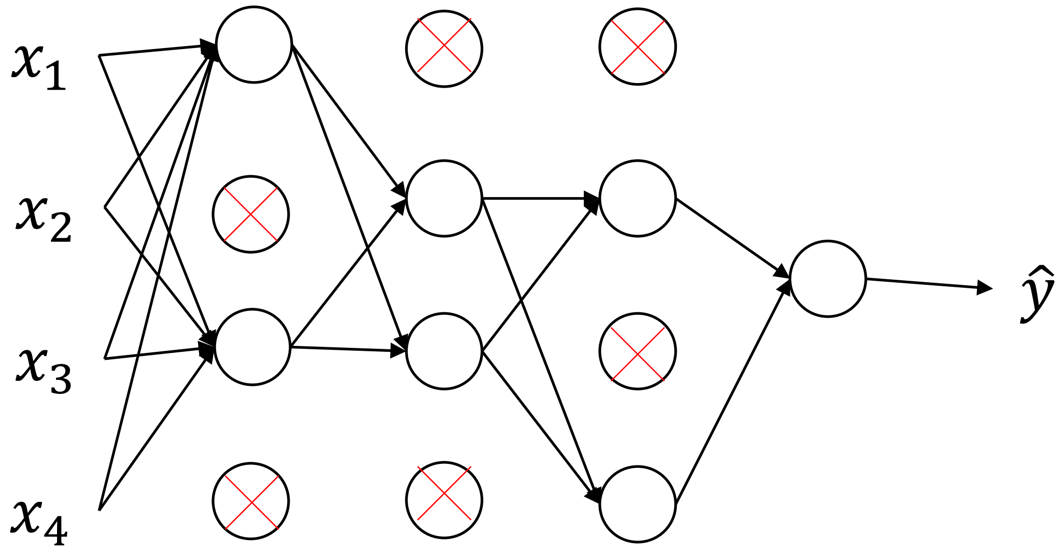
Weight decay $$w^{[l]} = \left(1 - \frac{\alpha\lambda}{m}\right) w^{[l]} - \alpha dw^{[l]}$$

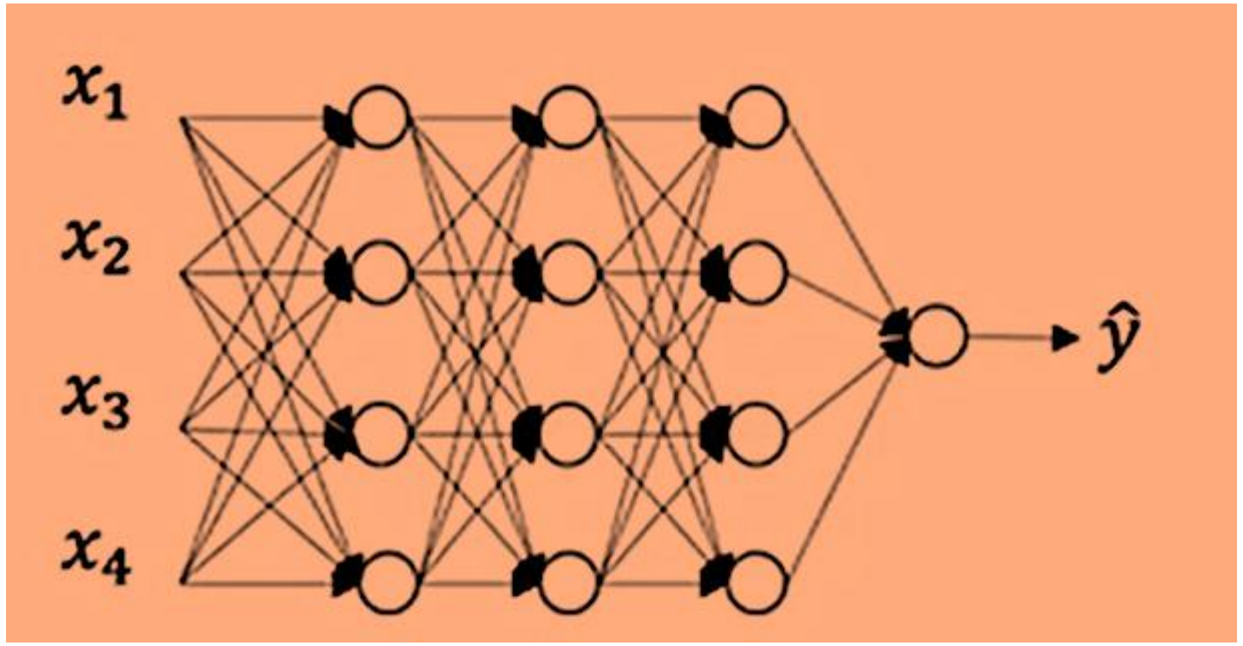
L2 Regularization. Intuition

Mostly we use L2 Regularization instead of L1. It has Direct effect on the weights as we end up in the reduction of weights by a factor of (1-$\propto \lambda/m$)

This is also referred to as weight decay. The larger values of lambda may lead to weights going close to zero and that may than go towards underfitting. The value of λ needs to be balanced. Similarly the small values of λ may not have any significant impact on the weights.λ, becomes another hyperparameter to handle.

Dropout regularization





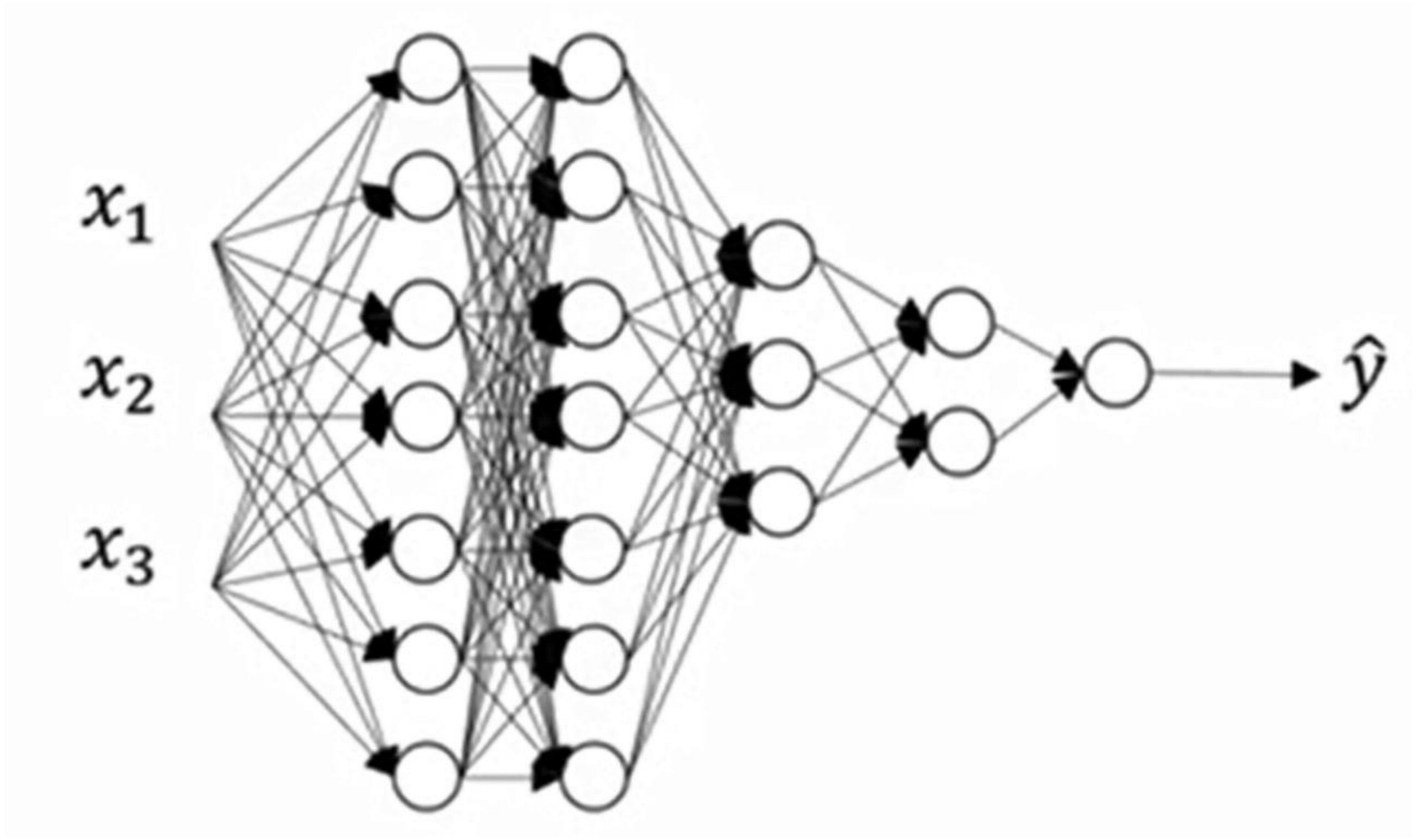
Drop out regularization: Prevents Overfitting

This technique has also become popular recently. We drop out some of the hidden units for specific training examples. Different hidden units may go off for different examples. In different iterations of the optimization the different units may be dropped randomly.

The drop outs can also be different for different layers. So, we can select specific layers which have higher number of units and may be contributing more towards overfitting; thus suitable for higher dropout rates.

For some of the layers drop-out can be 0, that means no dropout

Layer wise drop out



0 0.2 0.2 0 0 0

Drop out

- Drop out also help in spreading out the weights at all layers as the system will be reluctant to put more weight on some specific node. So it help in shrinking weights and has an adaptive effect on the weights.
- Dropout has a similar effect as L2 regularization for overfitting.
- We don’t use dropout for test examples
- We also need to bump up the values at the output of each layer corresponding to the dropout

Data Augmentation

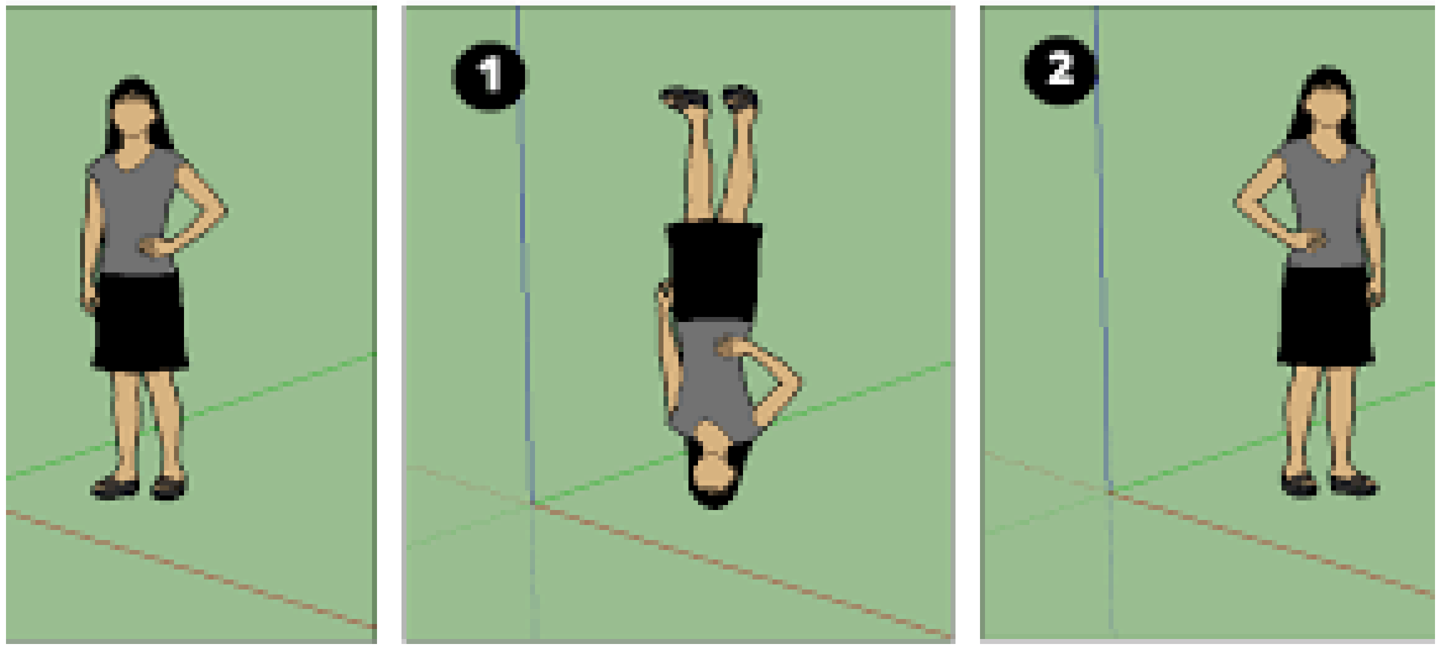
More training data is one more solution for overfitting.

As getting additional data may be expensive and may not be possible

Flipping of all the images can be one of the ways to increase your data.

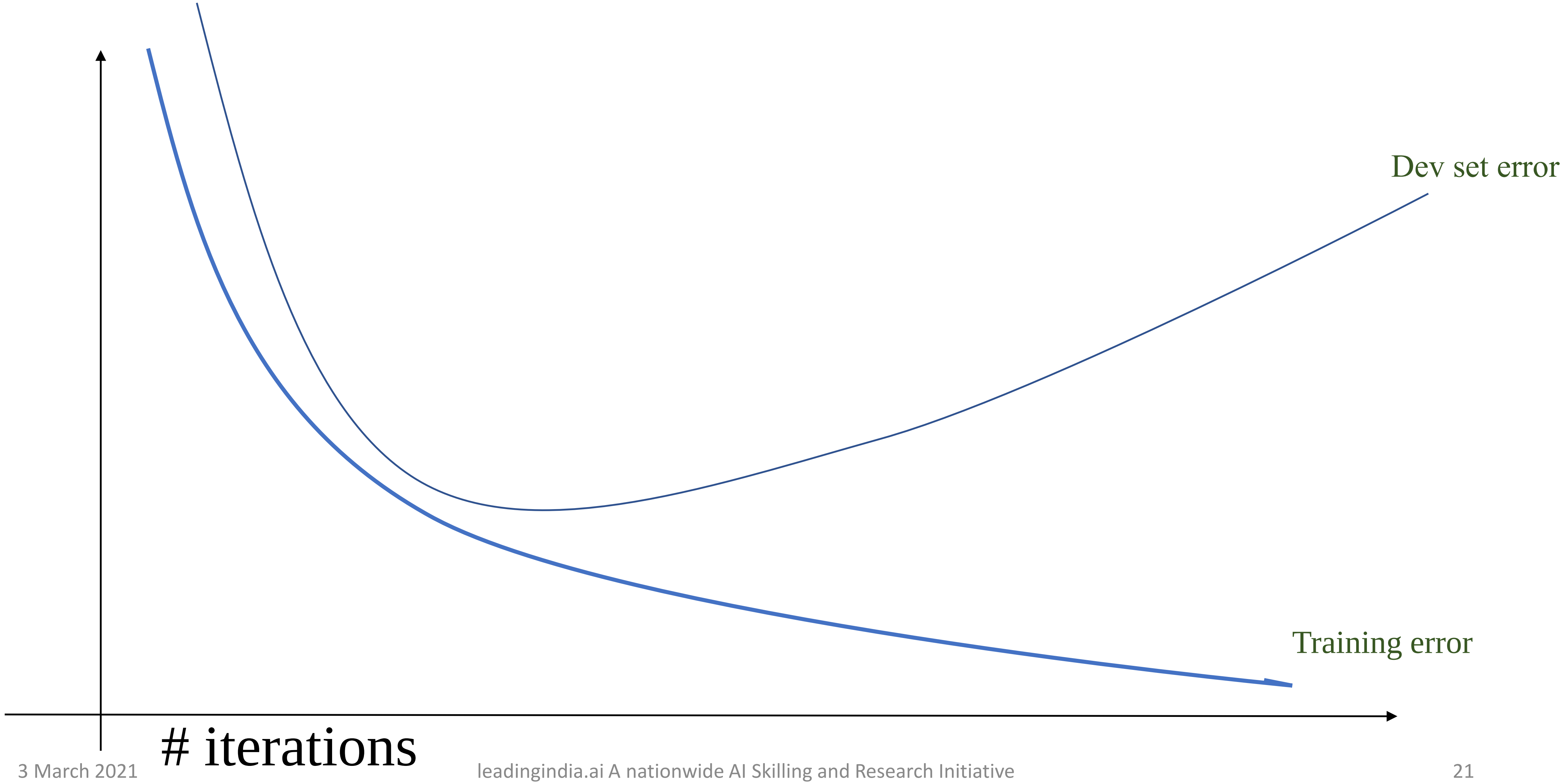
Randomly zooming in and zooming out can be another way

Distorting some of the images based on your application may be another way to increase your data.



Data Augmentation

Early stopping



Early Stopping

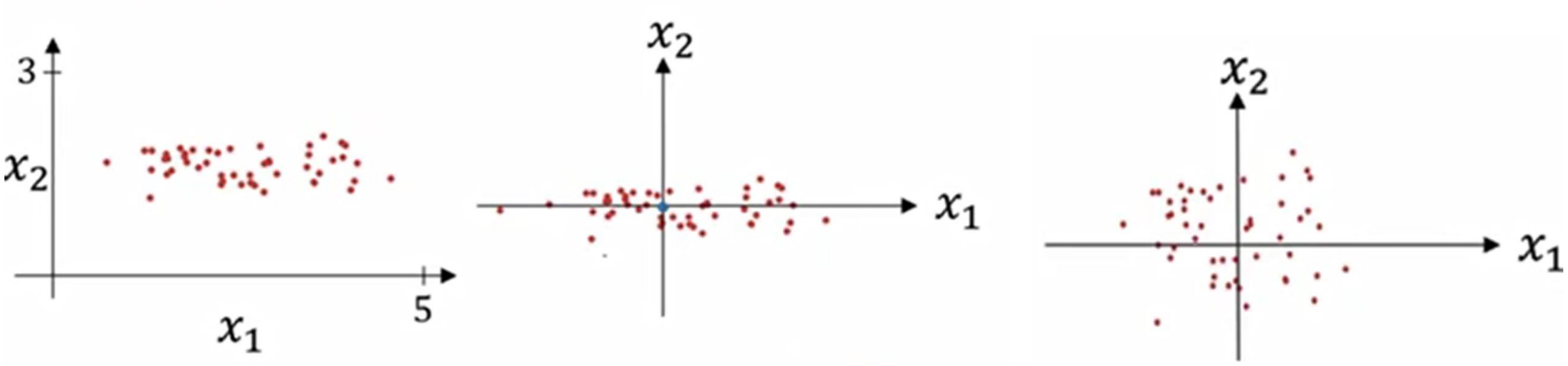
Sometime dev set error goes down and then it start going up. So you may decide to stop where the curve has started taking a different turn.

By stopping halfway we also reduce number of iterations to train and the computation time.

Early stopping does not go fine with orthogonalization because it contradicts with our original objective of optimizing(w,b) to the minimum possible cost function.

We are stopping the process of optimization in between to take care of the overfitting which is a different objective then optimization.

Normalizing Data Sets



Normalizing Training Sets

Subtract Mean

$$\boxed{\begin{gathered}\mu = \frac{1}{m}\Sigma_{l=1}^{m} x^{(i)} \\ x = x - \mu\end{gathered}}$$

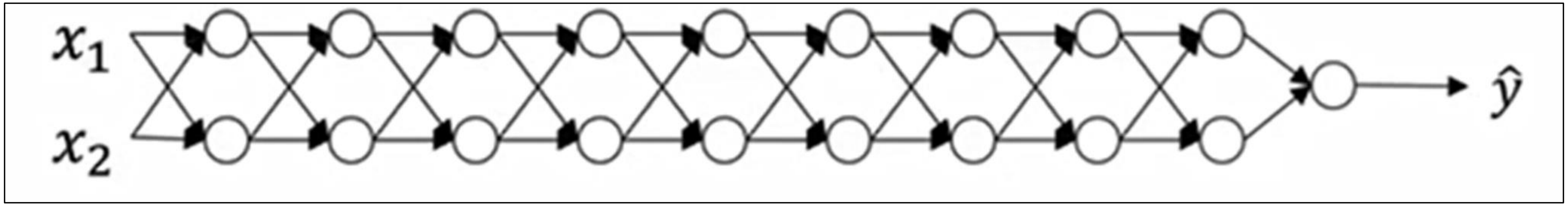
Normalize Variance

$$\boxed{\begin{gathered}\sigma^2 = \frac{1}{m}\Sigma_{i=1}^{m} x^{(i)} ** 2 \\ x /= \sigma^2\end{gathered}}$$

Vanishing/exploding gradients

g(z) = z # A linear function $b^{[l]}=0$

$$\hat{y} = w^{[l]} w^{[l-1]} w^{[l-2]} \dots w^{[3]} w^{[2]} w^{[1]} x$$



1.5	0
0	1.5

.5	0
0	.5

The matrix will be multiplied by l-1 (as $w^{[l]}$ will be different dimension) leading to exploding and vanishing gradients

Exploding/vanishing gradients

Gradients/slope becoming too small or two large

So it is very important to see that how we initialize our weights

If the value of features are large than weights needs to very small

It has been proposed to have the variance between the weights to be 2/n

Batch vs. mini-batch gradient descent

$$\boldsymbol{X} = [\underbrace{\boldsymbol{x}^{[1]}\ \boldsymbol{x}^{[2]}\ \boldsymbol{x}^{[3]} \dots \boldsymbol{x}^{[1000]}}_{\boldsymbol{X}^{\{1\}}\ (\mathbf{n_x}, \mathbf{1000})} | \underbrace{\boldsymbol{x}^{[1001]} \dots \boldsymbol{x}^{[2000]}}_{\boldsymbol{X}^{\{2\}}\ (\mathbf{n_x}, \mathbf{1000})\ \dots} | \dots | \underbrace{\dots \boldsymbol{x}^{[m]}}_{\boldsymbol{X}^{\{5000\}}\ (\mathbf{n_x}, \mathbf{1000})}]$$

(n_x , m)

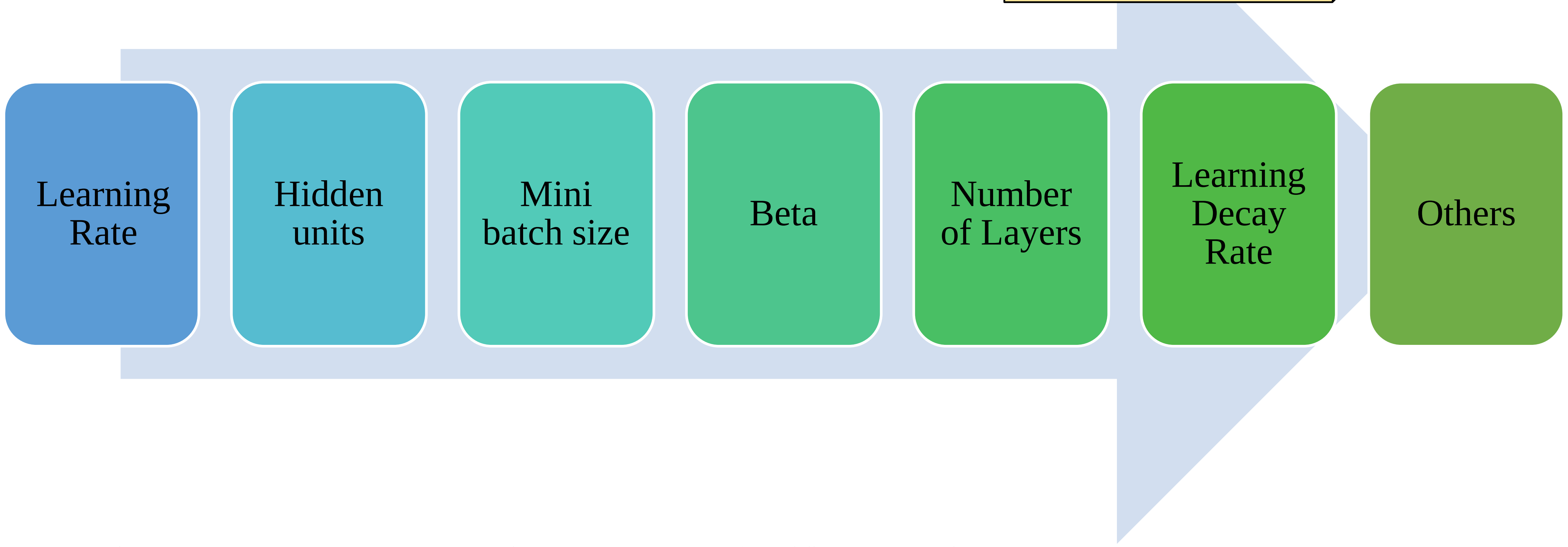
M=5,000,000 5000 mini batches of 1000 each

$$Y = [\underbrace{y^{[1]}\ y^{[2]}\ y^{[3]} \dots y^{[1000]}}_{Y^{\{1\}}\ (1,\ 1000)} | \underbrace{y^{[1001]} \dots y^{[2000]}}_{Y^{\{2\}}\ (1,\ 1000)\ \dots} | \dots | \underbrace{\dots y^{[m]}}_{Y^{\{5000\}}\ (1,1000)}]$$

$(1 , m)$

How to try Hyperparameters

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



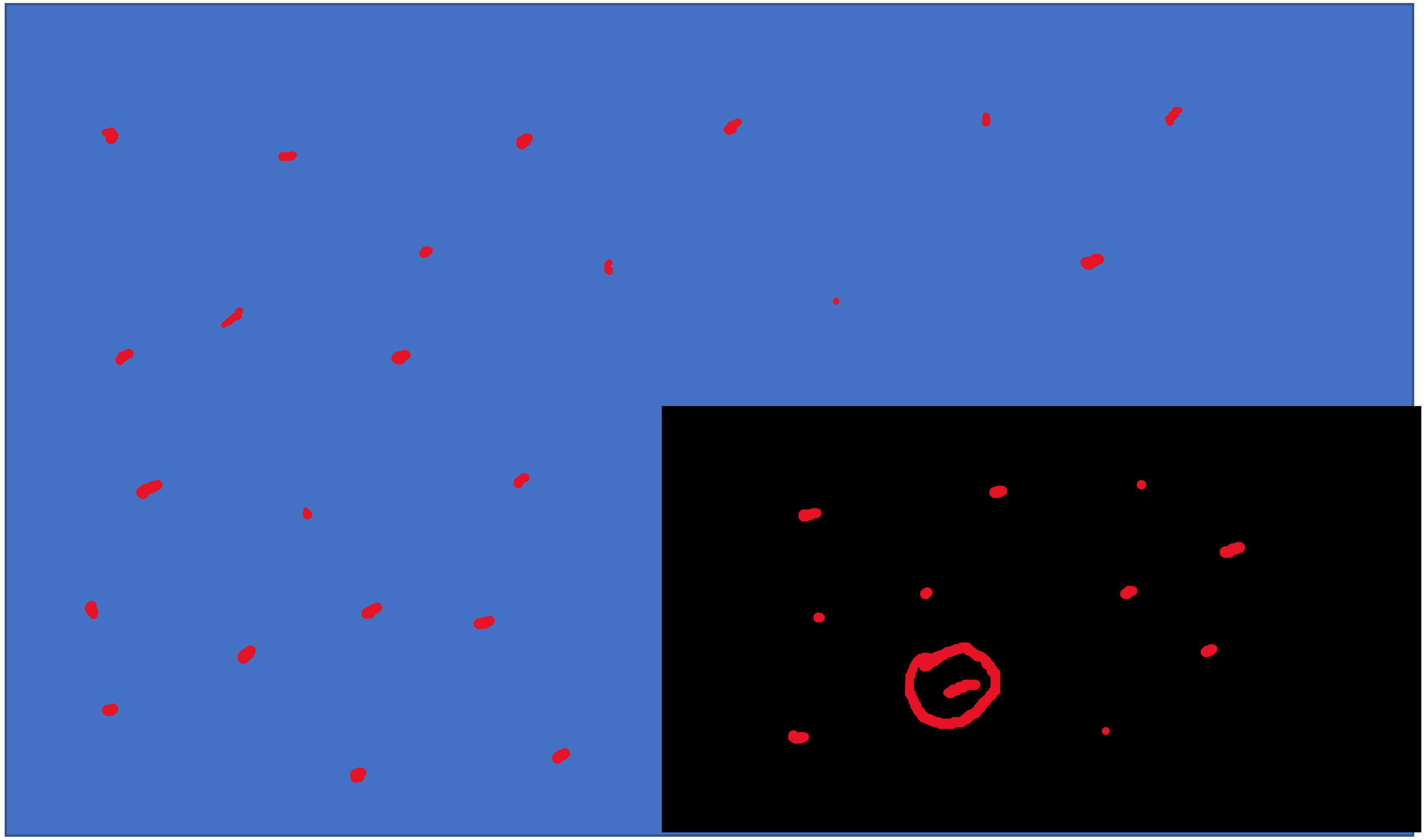
Random is better than a Grid

α

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

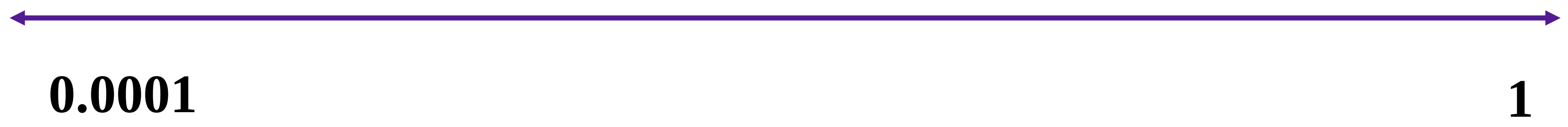
β

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



Public Notice regarding Use of Images/Information

Thank you!