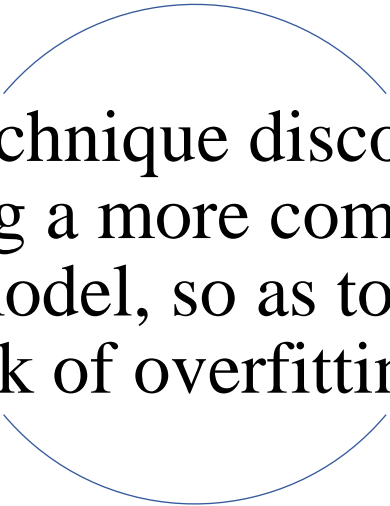
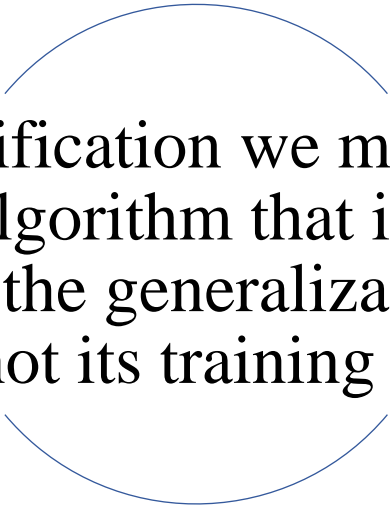


Regularization

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

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

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

Frobenius Norm

Weight decay

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

L2 Regularization. Intuition

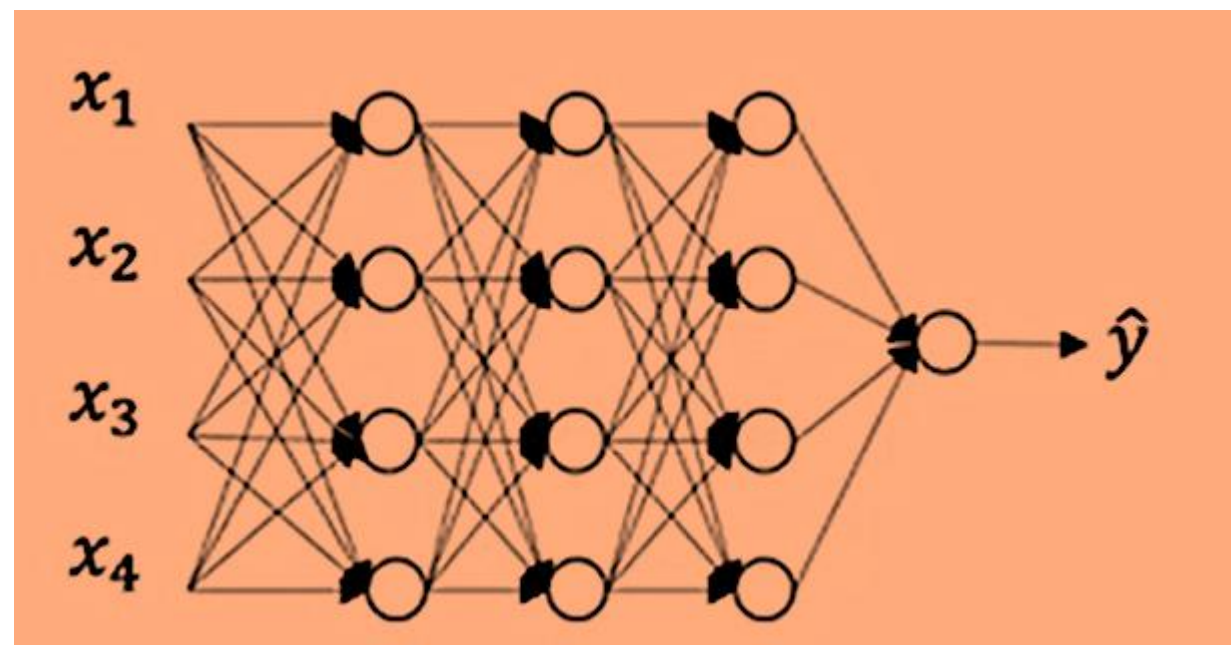
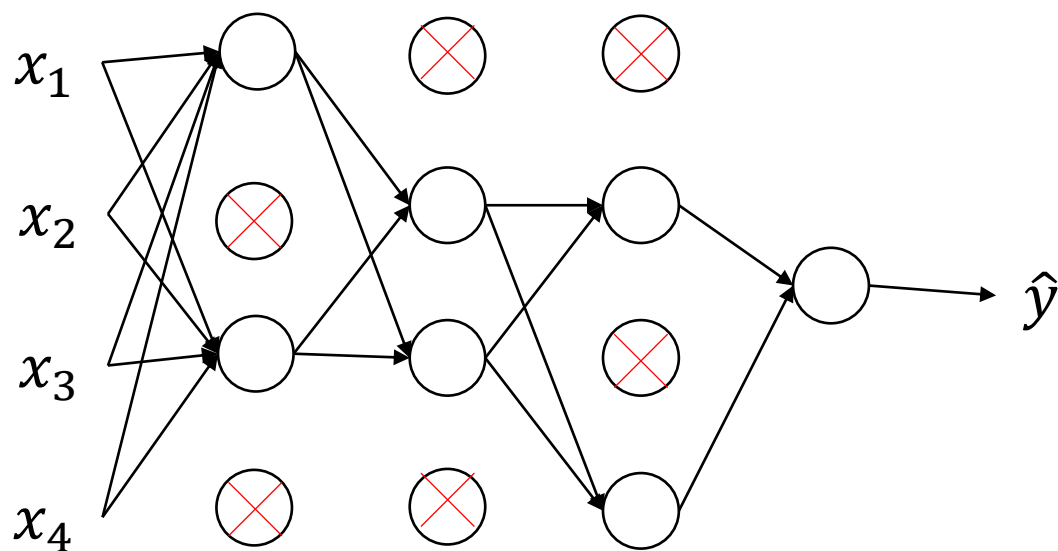
Mostly we use L2 Regularization instead of L1. It has Direct effect on the weights as we end up in the reduction of weights by a factor of $(1 - \alpha \lambda / m)$



This is also referred to as weight decay. The larger values of lambda may lead to weights going close to zero and that may then 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



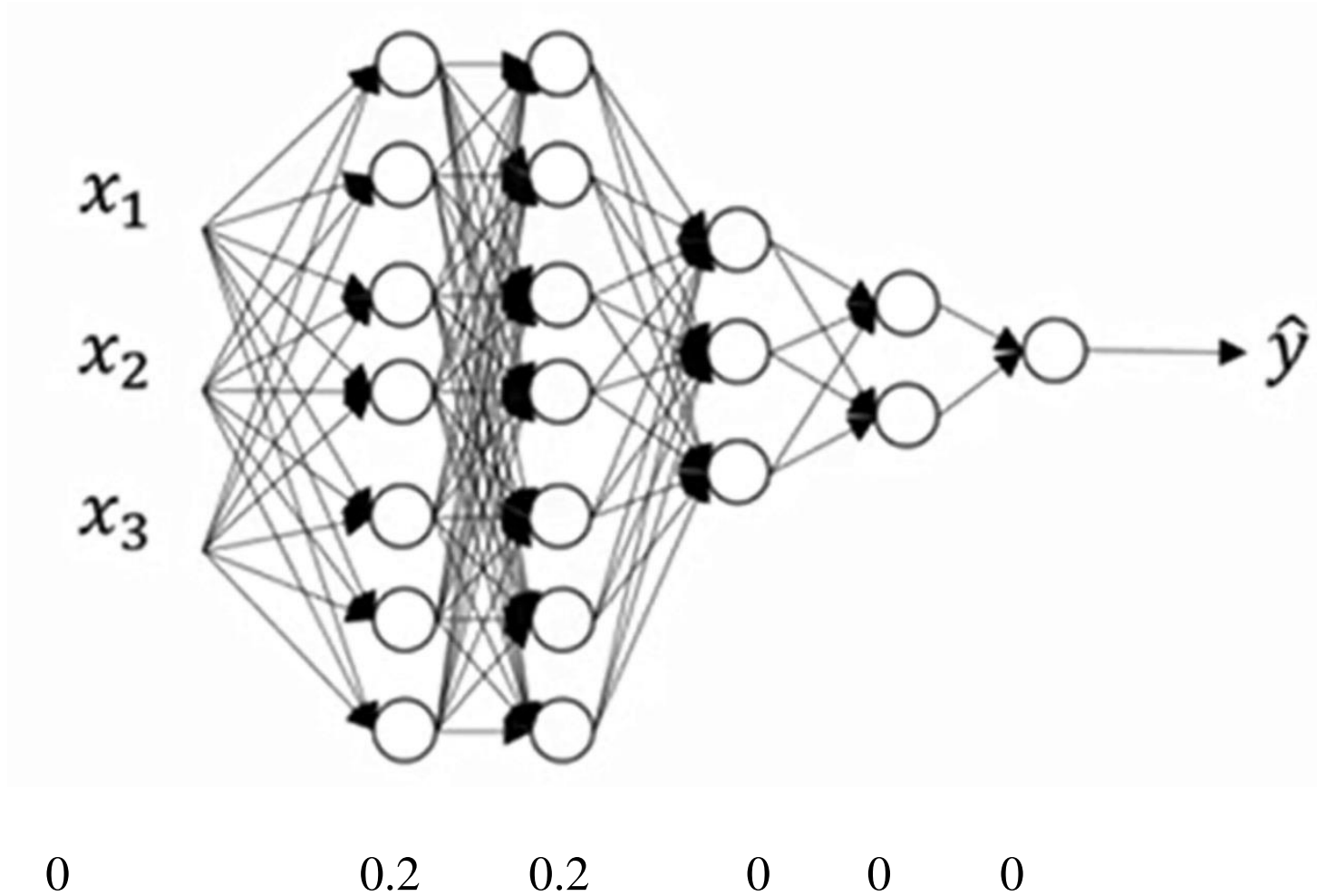
Drop out regularization: Prevents Overfitting

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



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.

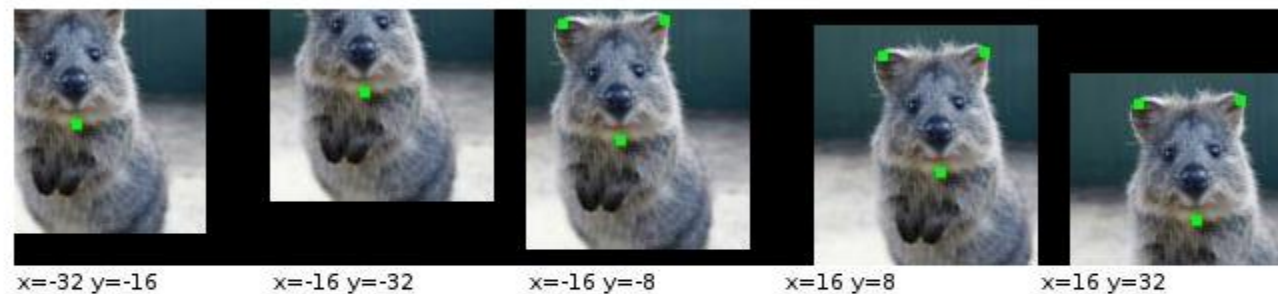
As getting additional data may be expensive and may not be possible

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

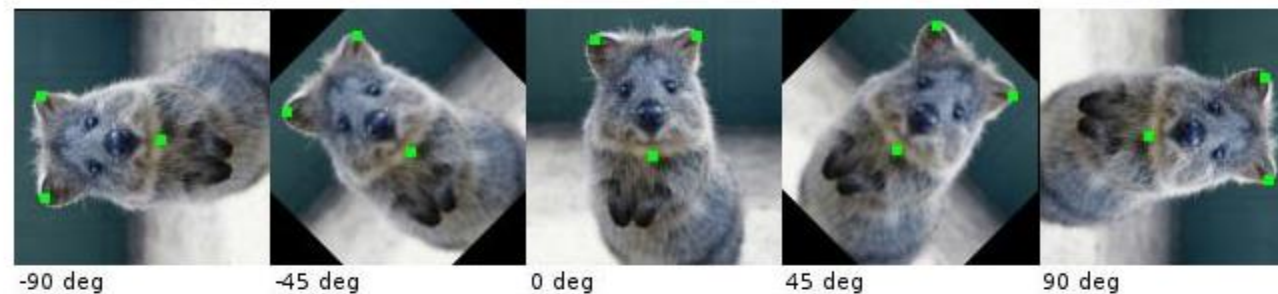
Randomly zooming in and zooming out can be another way

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

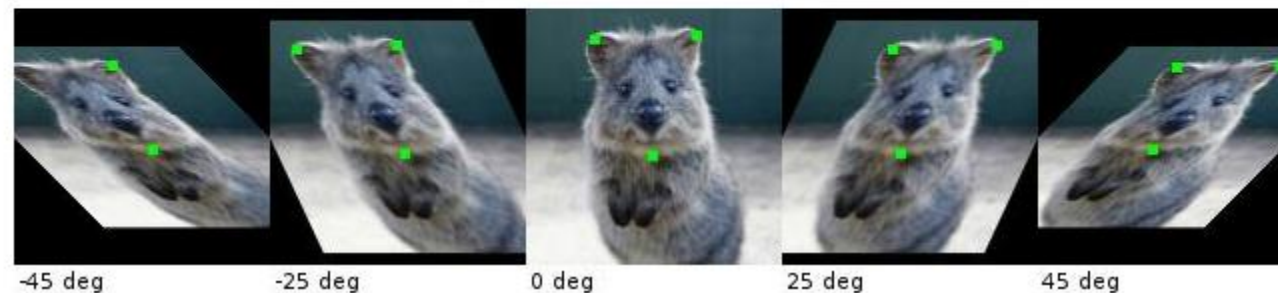
Affine: Translate



Affine: Rotate

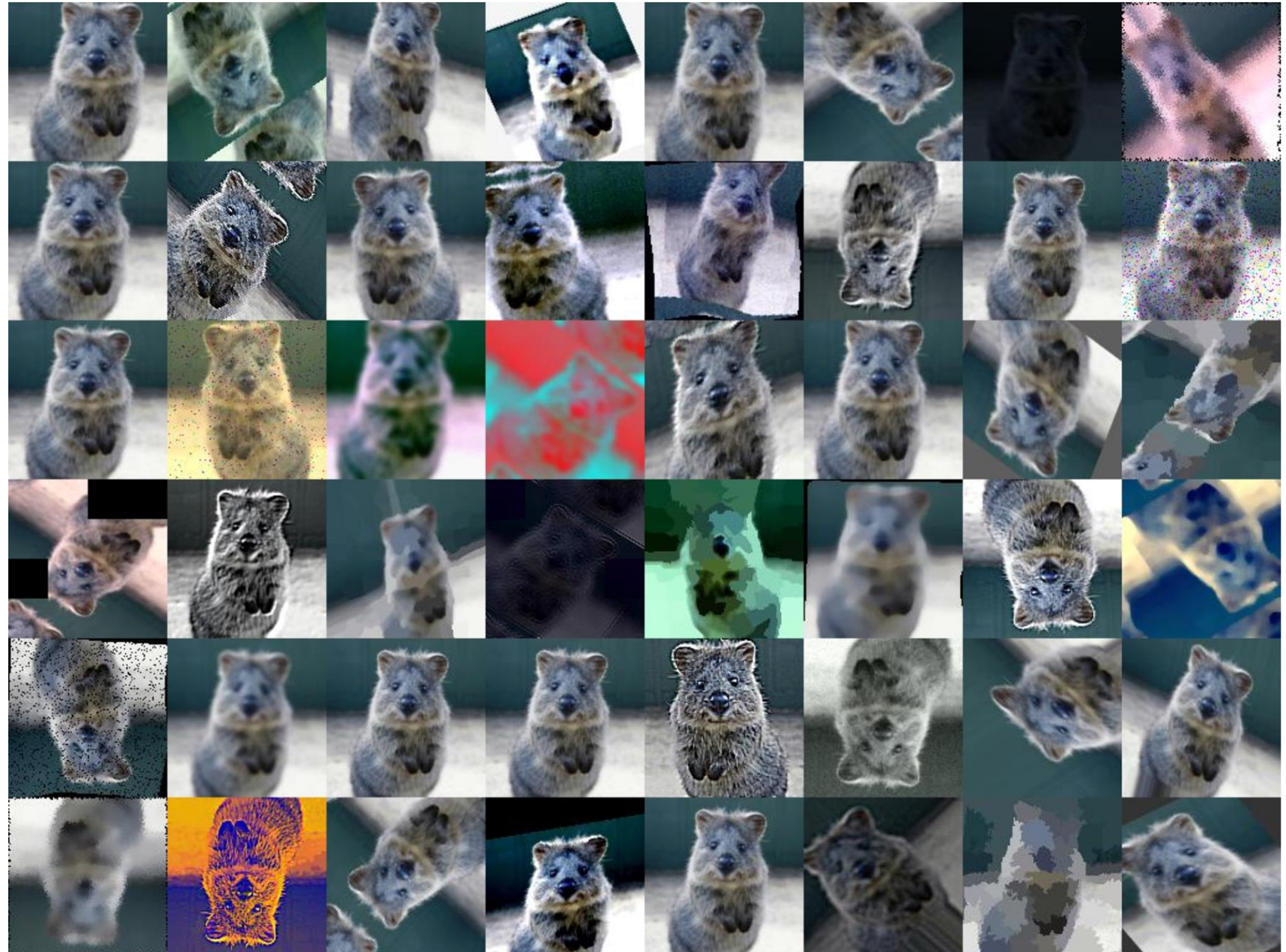


Affine: Shear

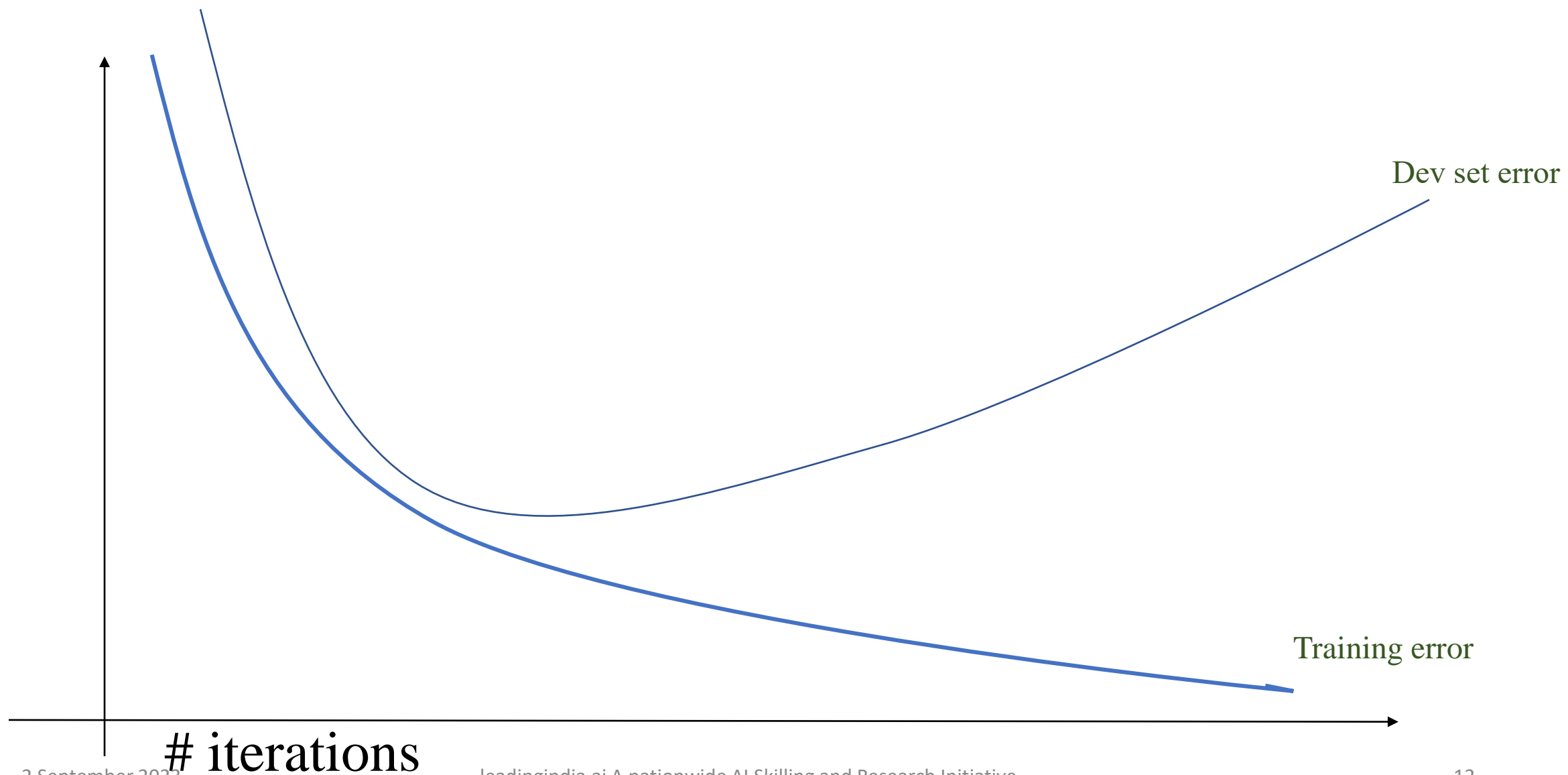


Data Augmentation

Noise Robustness



Early stopping



Early Stopping

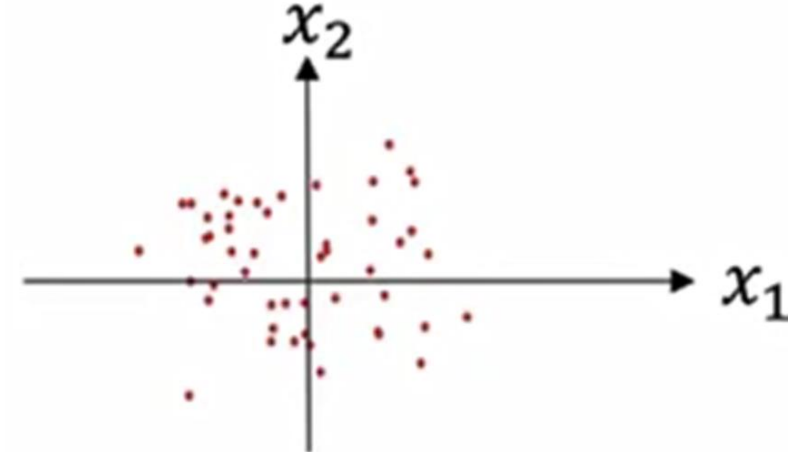
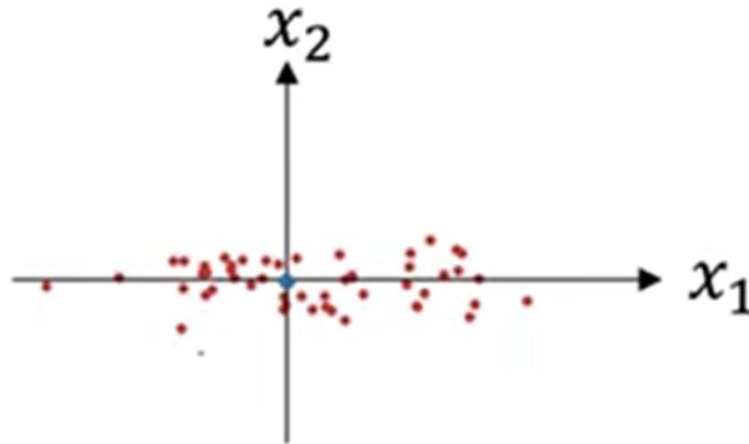
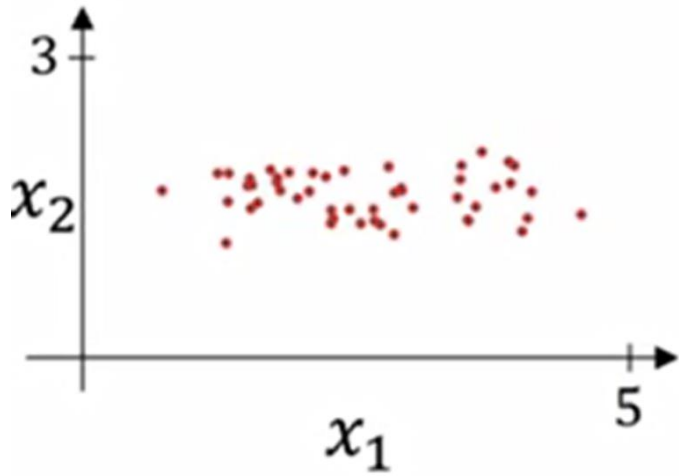
Sometime dev set error goes down and then it start going up. So you may decide to stop where the curve has started taking a different turn.

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



Speed up the training/ Why normalization

Use same normalizer in the test set also, exactly in the same way as training set

If the features are on different scale 1, 1000 and 0,1 weights will end up taking very different values

More steps may be needed to reach the optimal value and the learning can be slow.

Shape of the Normalized bowl will be more spherical and symmetrical making it easier to faster to optimize

Normalizing Training Sets

Subtract Mean

$$\mu = \frac{1}{m} \sum_{l=1}^m x^{(i)}$$
$$x = x - \mu$$

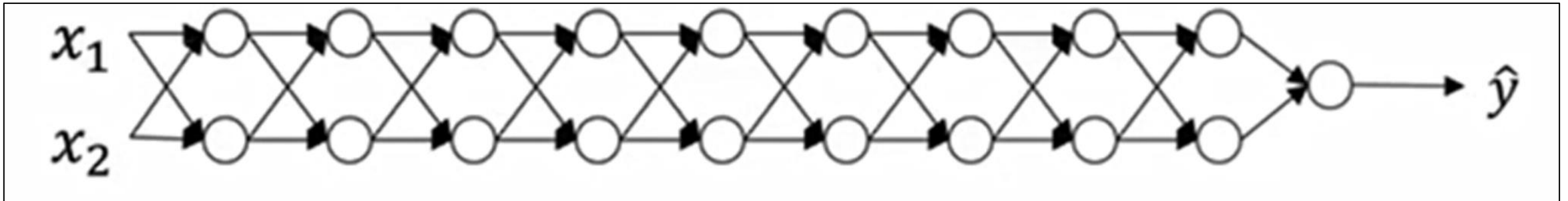
Normalize Variance

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

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

Exploding/vanishing gradients

Gradients/slope becoming too small or too large

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

If the value of features are large then weights need to be very small

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


Batch Norm

It is an extension of normalizing inputs and applies to every layer of the neural network


Given some intermediate values in Neural Network

- $\mu = \frac{1}{m} \sum z^{(i)}$
- $\sigma^2 = \frac{1}{m} \sum (z - \mu)^2$
- $z_{norm}^i = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$
- $\tilde{z}^i = \gamma z_{norm}^i + \beta$ where γ and β are learnable parameters
- If $\gamma = \sqrt{\sigma^2 + \epsilon}$ and $\beta = \mu$ then $\tilde{z}^i = z^{(i)}$


Why batch norm




Applying it on earlier layers helps in decoupling from the later layers.



That actually means that it provides a robustness to the changes in the covariance shift due to change in the input distribution.



So if there are frequent changes in the input examples than it will provide a cushion for the effect to taper off while going to later layers.



For test data we should prefer taking the exponentially weighted averages for μ and σ^2 of subsequent layers during training that will be better than the μ and σ^2 values of the training set itself.