# Hyperparameter tuning

**How to reduce the overfitting of NN and improve performance?**

1. Regularization: Regularization cannot be done in NN
    - due to L layered weight matrix $W = [W^1, W^2, W^3, \ldots\ldots, W^L]$

    Hence a hack is used called Forbenius Norm

$$Reg = \frac{\lambda}{2n}\sum_{k=1}^{k=L} ||W^k||_F^2$$

Where $n$ is the number of samples and $k$ is the current layer

we define $||W^k||_F^2$ as:

$$||W^k||_F^2 = \sum_{i=1}^{n^{k-1}}\sum_{j=1}^{n^k} (w_{ij}^k)^2$$

Where , $n^k$ is the number of neurons in the current layer $k$ and $n^{k-1}$ is the number of neurons in the previous layer $k - 1$

**How are the weights updated ?**

Ans: Loss is defined as

$$Loss(L) = \frac{1}{2n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \frac{\lambda}{2n}\sum_{k=1}^{L} ||W^k||_F^2$$

Hence Gradient becomes:

$$\frac{dL}{dW^k} = (From\ Backprop) + \frac{\lambda}{n}W^k$$

Hence weight Updation becomes:

$$w^k = w^k - \alpha(From\ Backprop) - \alpha\frac{\lambda}{n}W^k$$

$$w^k = (1 - \alpha\frac{\lambda}{n})w^k - \alpha(From\ Backprop)$$

Note: the extra $(1 - \alpha\frac{\lambda}{n})$ is known as weight decay

2. Dropout: Regularizes the NN by :
    - Dropping weights( Edges) of the NN
    - By creating a mask ($d^k$) through a random probability matrix ( $P(W^k)$ ) for the $k^{th}$ layer such that:

$$Mask\ (d^k)\ =\ 1\ if\ P(W^k)\ >\ dropout\ rate\ (r)$$
$$else\ Mask\ (d^k)\ =\ 0$$

- During test time, all the weights are upscaled by a factor of $p\ =\ 1 - r$

**What is the need for Upscaling of weights during test time?**

Ans: Weight updation does not take place for the ones that are dropped

- Making the weights not reach their optimal values
- Hence for optimal values, upscaling during test time is done

Note: During Test time, no Dropout takes place

3. Batch normalization: Standardizing the input is one of the important steps for reaching global minima
    - And since computing activation functions, weight multiplications, and biases, the input to hidden layers tends to have different distributions
    - These changed distributions get amplified as we go down the layers of NN
    - This is known as Internal Covariate Shift

Hence data standardization is performed as :

$$mean\ (\mu)\ =\ \frac{1}{m}\sum_{i=1}^{m} z_i$$

$$\text{Variance } (\sigma^2) = \frac{1}{m}\sum_{i=1}^{m} (z_i - \mu)^2$$

$$Z_{norm}\ =\ \frac{(z_i - \mu)}{\sqrt{\sigma^2 + \epsilon}}$$

where $m$ is the number of neuron in a layer and $\epsilon\ =\ e^{-10}$

**Is having normal distribution for all layers a good thing?**

Ans: No, since two layers have the exact same mean and variance, makes 2nd layer redundant, therefore :

$$\hat{Z}\ =\ \gamma\ \times Z_{norm}\ +\ \beta$$

Where $\gamma,\ \beta$ becomes two learnable parameters

4. Early Stopping: Sometimes the NN performance increases for a certain epoch and decreases on later training epochs
    - So in order to prevent the model from updating weights on the later training epochs
    - The weights of the best validation score model are stored using `ModelCheckpointCallback`

- After which the model runs for a certain threshold after which training stops
- This stopping of training is done through another callback using `EarlyStoppingCallback`

5. LearningRateDecay: Sometimes model gets stuck around the global minima,
    - due to a high learning rate

   And takes a lot of epochs to reach global minima
    - When the learning rate is quite small

   Hence to make the NN train faster with high accuracy,
    - An adaptive Learning rate is used such that
    - The learning rate is reduced gradually over epochs
    - So the NN first quickly reaches around the global minima
    - Then converges to global minima with a smaller learning rate
    - This is implemented using a callback `LearningRateScheduler`

# Practical Aspects

**What are all the hyperparameters for NN?**

1. Number of epochs
2. NN depth and complexity
3. Batch Normalization
4. Learning rate
5. Regularization
6. Dropout
7. Choosing optimizer
8. Collecting more data

Note: Collecting data should be the last resort, since
- Data in general is hard to get
- Collecting data is time-intensive
- Does not guarantee an improved performance of NN

**In what order should the major hyperparameters be tuned for NN?**

1. Learning Rate
2. β value of GD with Momentum
3. Number of Hidden units/ Neurons
4. Batch size
5. Number of layers of NN

6. Learning Rate Decay
7. $\beta_1$, $\beta_2$, $\epsilon$ of Adam

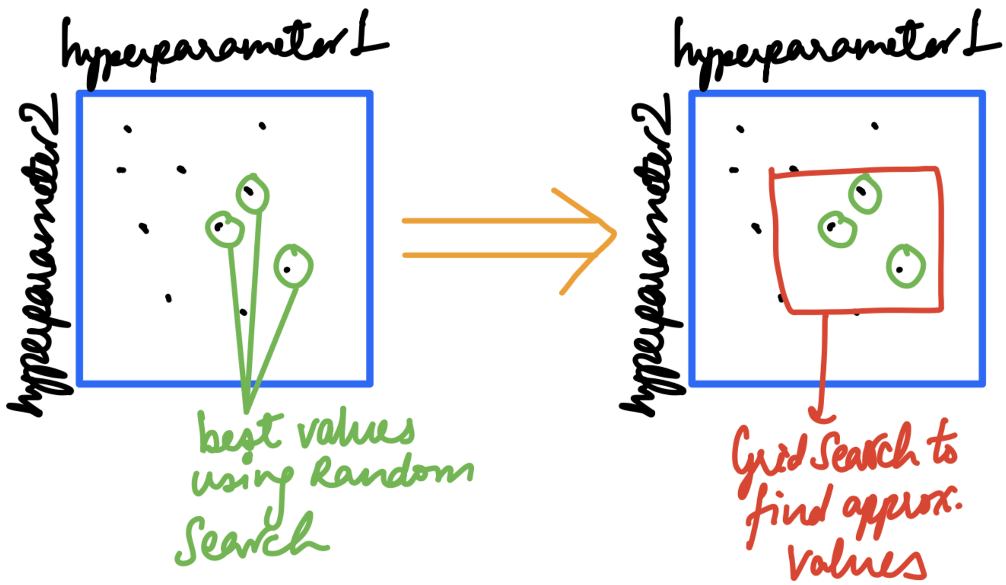
With such a variety of hyperparameters to experiment with,
- it becomes very important to know which hyperparameter to tune to enhance model performance
- This is known as Orthogonalization of NN

**What to tweak if NN has a bad training performance ?**
Ans: Clearly NN underfits hence:
- More epochs
- Deeper and Complex NN
- Different Optimizer
- More data

**What to tweak if NN has good training performance but bad validation performance ?**
Ans: Clearly NN underfits hence:
- Use simple NN
- Regularization
- Dropout
- Batch Normalization
- Diverse training samples

**What to tweak if NN has bad testing performance but good training and validation performance ?**
Ans: Though its not a good practice to tune NN for test data, yet some tweaks are:
- Changing loss function
- More Validation data

**How to find the correct value for the hyperparameters ?**
Ans: Perform Random Search and get some hyperparameter value ranges
- Followed by Grid Search to have more accurate results




Before Hyperparameter tuning, its very important to have an error analysis done of the model

- Hence its a good practice to have a human performance on the task
- Along with the maximum attainable performance known as Bayes Optimal error

**Why need Human performance ?**

Ans: Helps identifying whether model is having

- a low bias and high variance
- or a high variance and low bias

Note: The difference/gap between Human and Training error is called Avoidable Bias