Problems with Neural Network

- Vanishing / Exploding gradient
- Not Enough Data
- Slow training
- Overfitting

1) Vanishing / Exploding Gradient

- Weight initialize
- Activation function
- Batch Norm
- Gradient Clipping for Exploding Gradient only

2) Not enough Data

- Transfer learning
- Unsupervised pretraining

3) Slow Training

- Optimizers
- Learning rate scheduler
  - epoch change ↔ Learning Rate change

Overfitting

- l_1 and l_2 regularization
- Dropouts

Early Stopping → code

Parameter → (i) monitor → which value you want to monitor to check whether changing in loss or not.
Mostly "val_loss" use.
↳ Unseen data

(ii) min_delta ⇒ how much value consider as a change.

eg:- 0.00001 → if val_loss value less change than 0.00001 then we stop epochs after patience value.

(iii) patience ⇒ wait after checking the loss.

for eg: valloss change less than 0.00001. So, we can't stop the epoch. we wait and run 3 more epochs if val_loss still less than 0.00001 then we stop epochs. Otherwise continue the epochs.

patience value is 3

(iv) Verbose ⇒ when verbose = 0 (output don't show in written "epoch 161: early stopping".

if verbose = 1 then show early stopply message.

(v) Mode ⇒ One of {"auto", "min", "max"}.

→ "Min" mode: Training will stop when the quantity monitored has stoped decreasing.

→ "Max" mode: Training will stop when the quantity monitered has stopped ~~decreasing~~ Increasing.

→ "Auto" mode: The direction automatically inferred from the name of the monitered quantity.

mostly this mode use. Automatically figured out

(vi) Baseline ⇒ Baseline value for the monitored quantity. Training will stop if the model doesn't show improvement over the baseline. Generally, not give any baseline cz nobody knows actual loss or accuracy during training.

(vii) Restore_best_weights: whether to restore model weights from the epoch with the best value of the monitored quantity. If false, the model weights obtained at the last step of training are used.

Restore_best_weights: false → Training → Patience epoch value

↳ True → Training → where the Patience epoch start value

0.7783 → 0.77821 → 0.77822

True take this value. Patience 3 epoch. 2 epoch. False → take this value

CHEAT SHEET

Improving Neural Network Performance

1.) Vanishing Gradients
- Activation Functions
- Weight Initialization

2.) Overfitting
- Reduce complexity / increase Data
- Dropout Layers
- Regularization (L1 & L2)
- Early stopping

3.) Normalization
- Normalizing inputs
- Batch Normalization
- Normalizing Activations

4.) Gradient checking and Clipping

5.) Optimizers -
- Momentum
- Adagrad
- RMSprop
- Adam

6.) Learning rate scheduling

7.) Hyperparameter Tuning

- No. of hidden layers
- nodes / layer
- Batch size

Normalizing Input

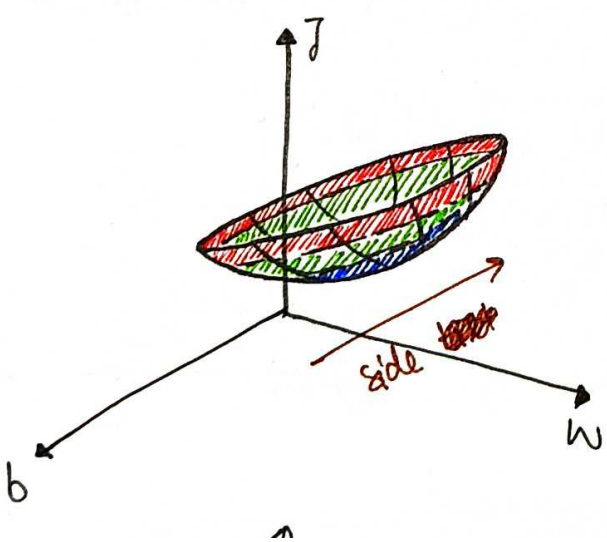
Feature Scaling

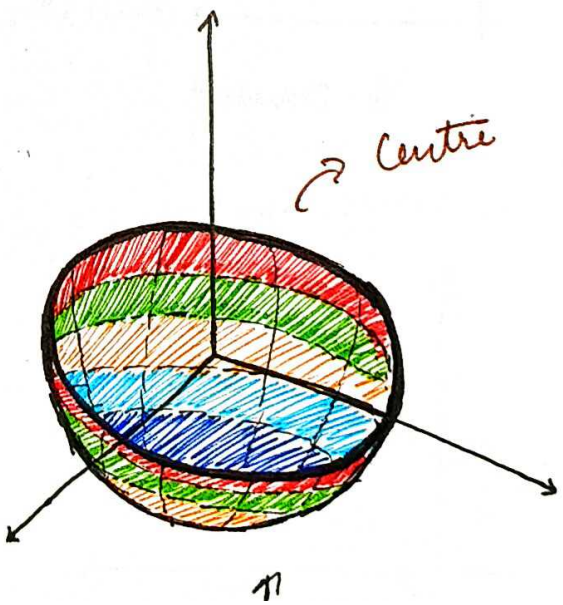
x_1 (age) → Weight (w_1)

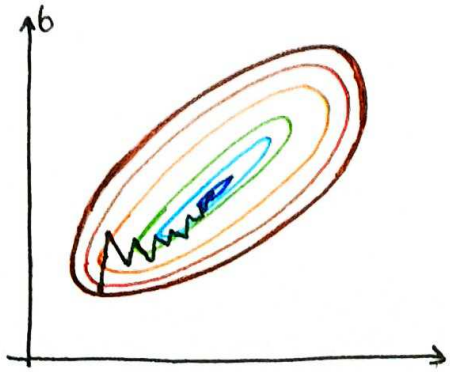
x_2 (salary) → (w_2)

$$w_2 = w_2 - \eta \frac{\partial L}{\partial w_2}$$

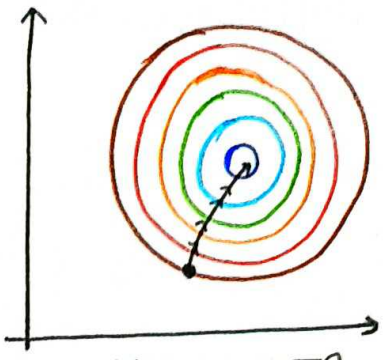
↳ When model update the weight. Model give more focus on w_2 than w_1 because salary in lakhs and easily see the changes and Age is not big value. So, model may can ignore. Model cannot predict accurately. Not train accurately







Unnormalized



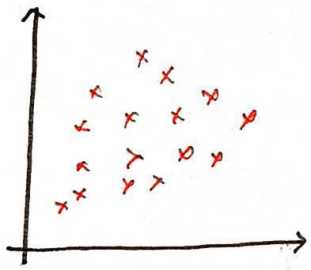
Normalized

Solution

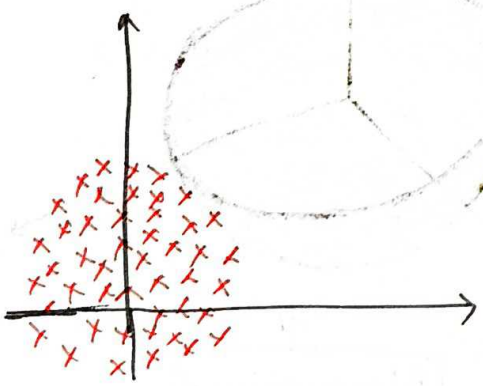
Standardization

$$\frac{x_i - \mu}{\sigma}$$

-1 to 1



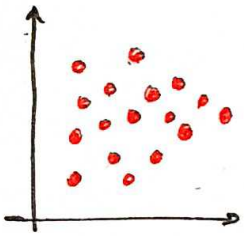
Standard



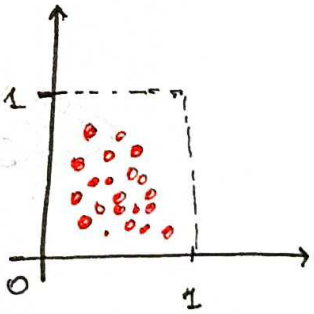
Normalization

$$\frac{x_i - x_{min}}{x_{max} - x_{min}}$$

0 to 1



Normalization



When we use Standardization and Normalization?

Standard → When we don't know min and max value of the column. like Cgp → 0 to 10 (Normal)

Salary → we don't know how much salary is min and max use Standardisation

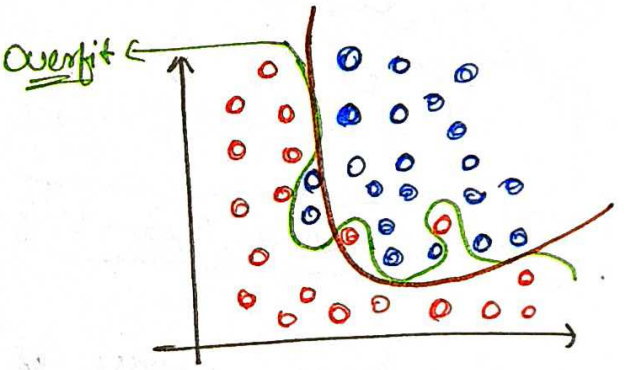
Normal → when we know min and max value of the colum → use Normal

cgpa = 0 to 10 → Normal

data → pdf → Normaly distributed or close to normaly distributed → use Standardizati.

Overfitting

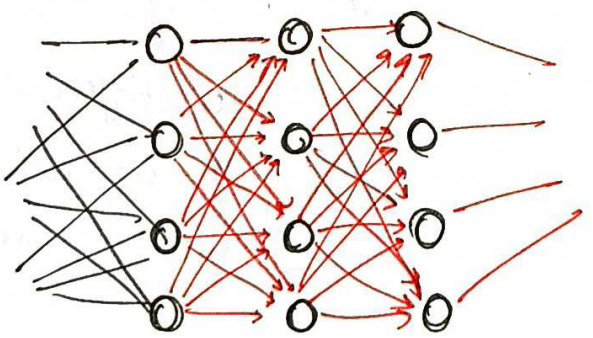
The problem of Overfitting



Overfit → Train data → good

↳ Test data → bad

ANN → overfitting → complex architecture



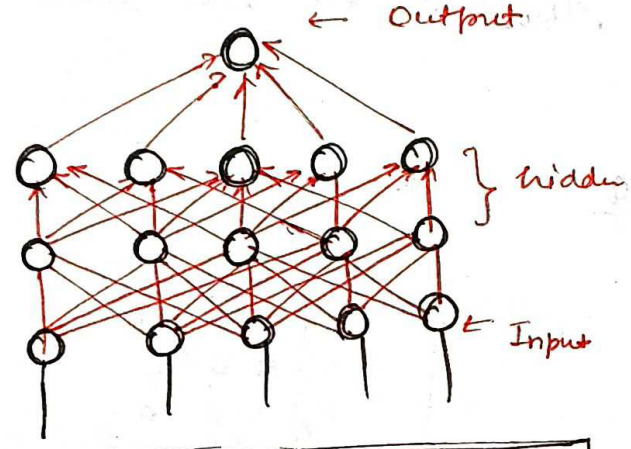
run on many epochs

Overfit many times

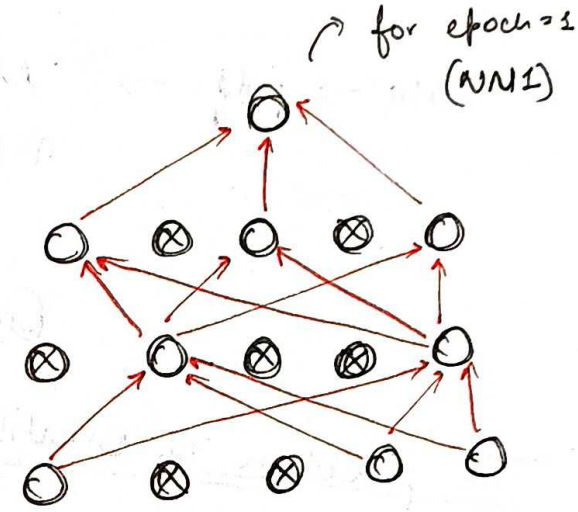
Possible Solution

1) Add more data
2) Reduce complexity
3) Early stopping
4) Regularization → L1, L2
5) Dropout

The concept of Dropouts

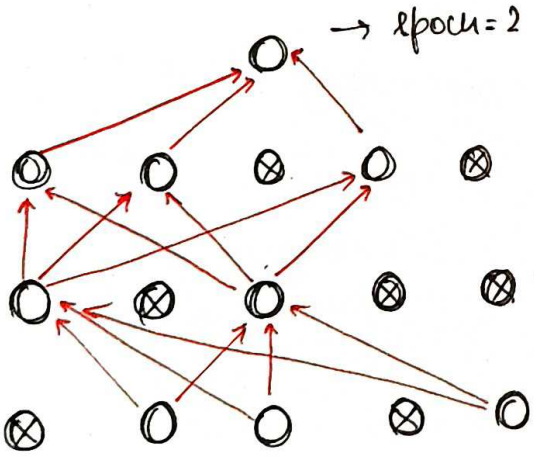


Standard Neural Network



After Applying Dropout

Drop random nodes of input and hidden layer in every epoch.



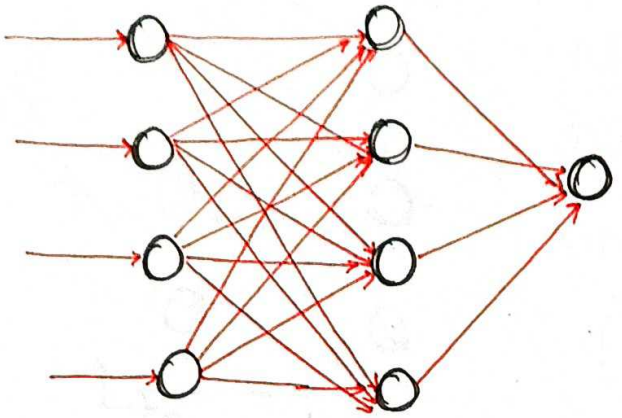
different NN2 from NN1

eg:→ 10 epochs

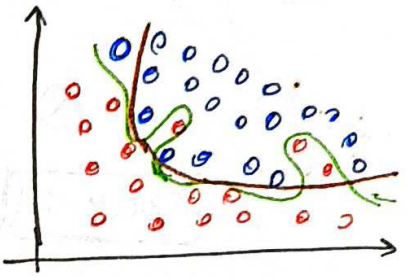
↓

10 different Neural Networks with same data

Why overfitting happen?



because node of neural network connected to each other and capture all small patterns.

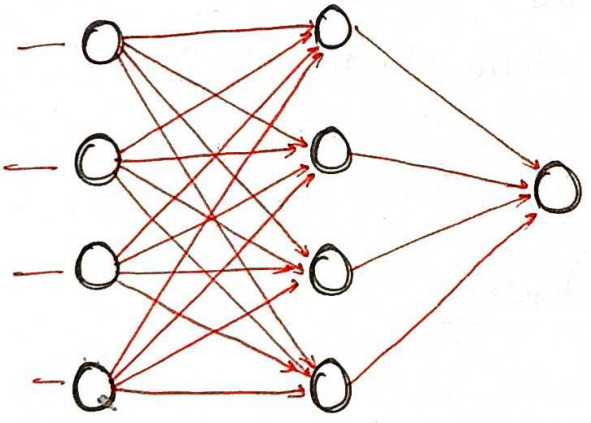


Solution

↳ Reduce the Nodes

↳ Changes in Nodes. So, Model can not focus on single thing or single pattern. Model should be focus on every thing and every pattern.

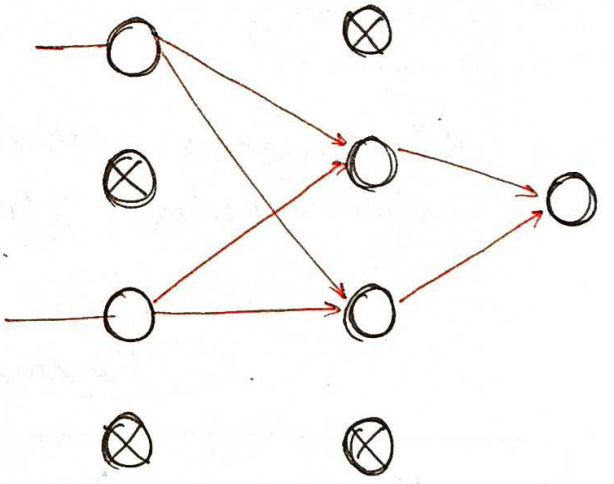
Why this works? → Reduce the Nodes



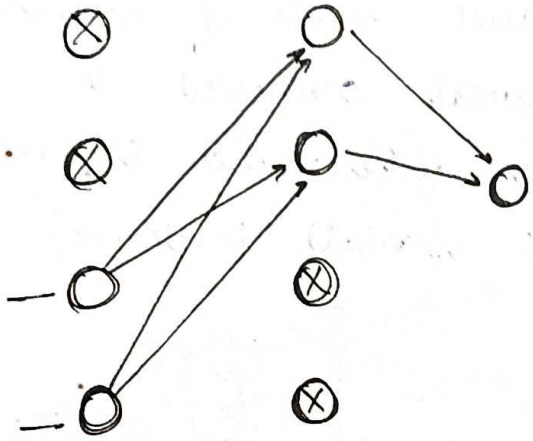
P = 0.5

P = 0.5

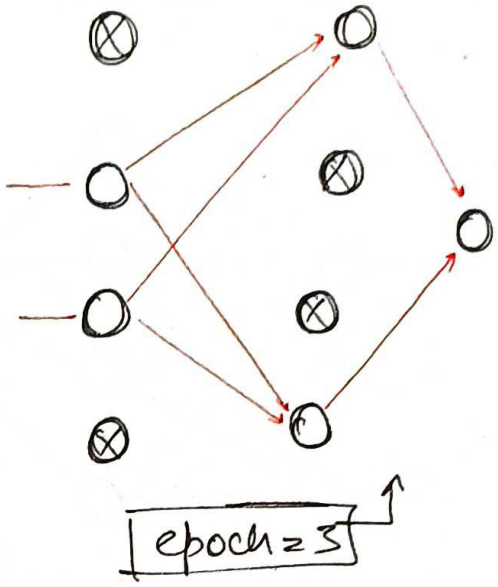
↳ Remove 50% of node in this layer



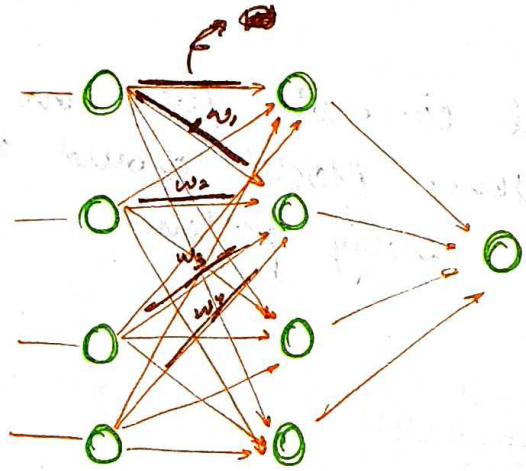
epochs = 2



epoch → 2



epoch = 3



→ Let assume, w_1 weight is more than other weights. So, in first epoch model focus on w_1 weight and first node of input layer.

In epoch, 1st node is canceled so, our model can focus on other node.

Using this method model can focus on every node and weight. Reduce Overfitting.

Random Forest Analogy

X_1 X_2 ------ X_{10} Y

50% of cols drop

Divide or drop types

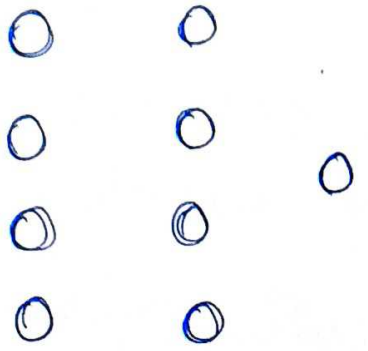
- Col
- Row

100 DT

Output

- 60 DT → 1
- 40 DT → 0

↳ bcz majority this main output



P_z 0.5

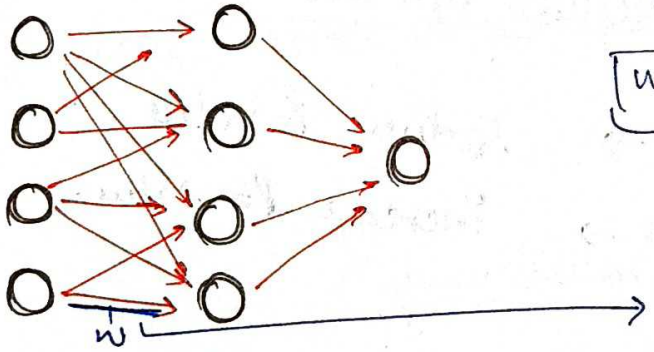
→ No. of combination of Neural Network is $2^{(8)}$ → No. of Nodes

→ $2^{(8)}$ → why 8 ? not 9 ? becaus output node never cancel.

epoch = 100

Training a united NN

How prediction works ?



P_z 0.25 P_z 0.25

100 epochs → Train model

Weight = W(1-P)

W x 0.75

why using this?

* When we write P_z 0.25 which means Probability of node not present in 100 epochs is 0.25.

OR

* ↑ 0.75 time ∧ hi tha probability ∧ Training ke time. (Node present, 100 epochs)

0.25 time drop tha training ke time.

↓

Next page Summary

Summarize

Training time → Nodes drop kar skte hai.

Testing time → all Nodes are available.

But we cannot assign whole node bcz every all nodes are not available. To jitne time available tha uski probability se multiply kar diya. That's why we multiply 0.75.
↳ Probability

P↓ → overfitting

P↑ → underfitting

> best value p is between 0.4 - 0.5

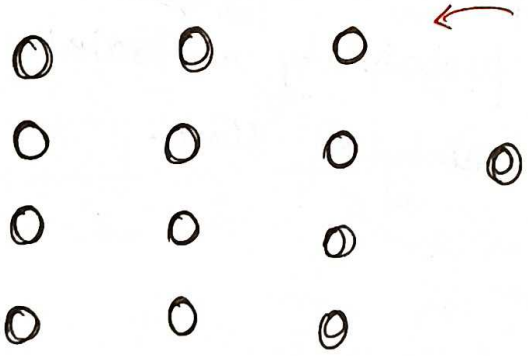
Practical Tips and Tricks

1) Overfitting in model → Reduce P-value

underfitting in model → Increase P= value.

2) last layer → dropout

↳ first try dropout on last layer then use every layer.



← First try dropout on last layer

3) CNN → 40-50% of P value get good result

ANN → 10-50 % of P value get good result

Drawbacks

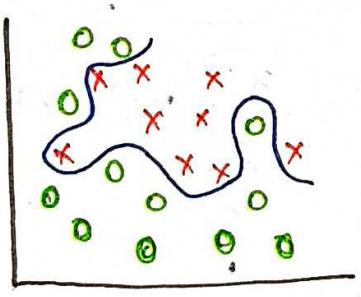
1) Convergence → delay / slow training
   ↳ reach at accurate weights and bias

2) Loss function changes → face difficulties during debugg

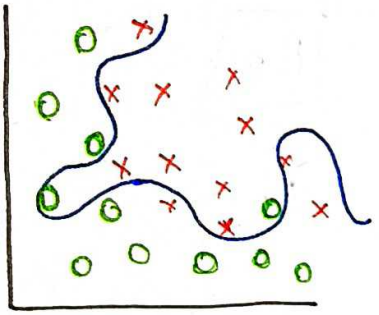
When you debugging your model you can face difficulties bcz many nodes are dropout and difficulties understand weight and bics.

Regularization

Overfitting



Train Dataset

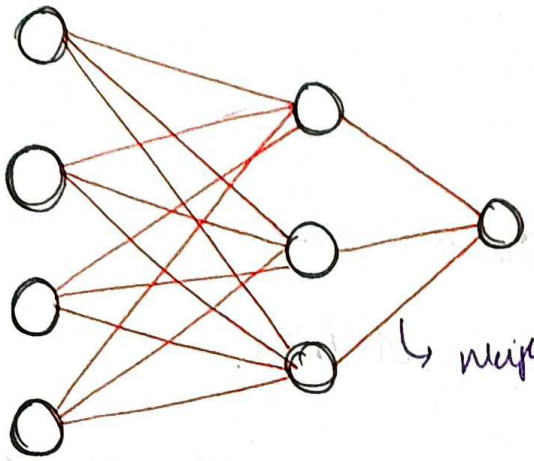


Test Dataset

Why neural Network Overfit?

Architectural

Because complex pattern cayglad small pattern



↳ weight ≃ 0

Regularization

ANN → Weight / bias

↳ min Lossfunction

$$C = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \text{Penalty Term}$$

L = mse

↳ binary

L2

$$C = L + \frac{\lambda}{2n}\sum_{i=1}^{k} \|w_i\|^2$$

L1

$$\frac{\lambda}{2n}\left[w_1^2 + w_2^2 + \dots + w_{10}^2\right]$$

let 10 weight

$w_1 \rightarrow w_{10}$

λ → hyperparameter

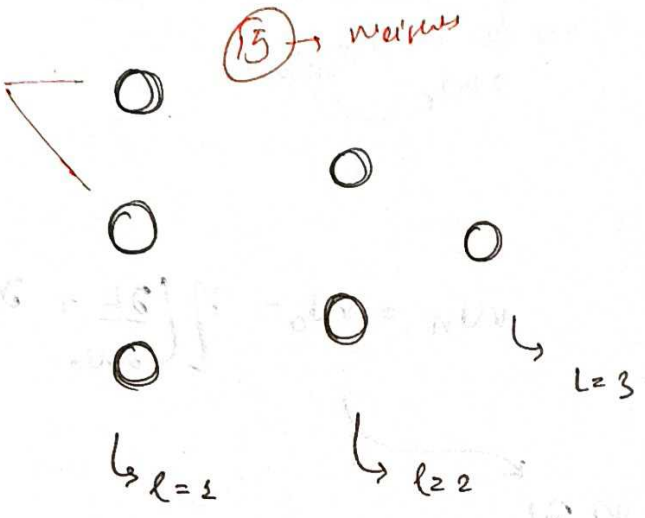
if λ → higher ↳ underfitting

lower ↳ still overfitting

When we apply regularization, value of weight going to 0. $w \simeq 0$

for L_1 → $C = L + \frac{\lambda}{2n} \sum_{i=1}^{k} ||W||$

Most of the book → $C = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{l=1}^{L} \sum_{i=1} \sum_{j=1} ||w_{ij}^{l}||^2$



entering layer by layer

Layer → L

row → i

column → j

$w_1^2 + w_2^2 + \dots$

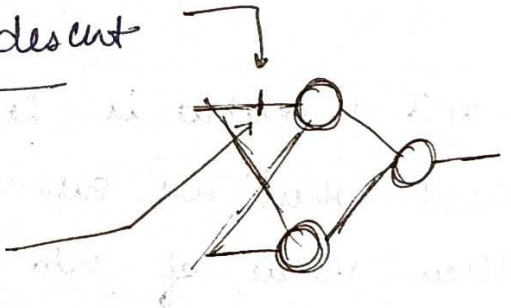
↳ only weights add not bias

Intuition behind Regularization

In Backpropagation → Gradient descent → use

$W_n = W_0 - \eta \frac{\partial L}{\partial W_0}$

change the particular weight

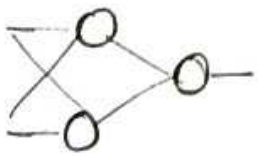


Loss function ↓

$L' = L + \frac{\lambda}{2} \sum ||W_i||^2$

↳ not dividing by n because we are solving only for one.

$$L' = L + \frac{\lambda}{2} \sum_i \|w_i\|^2$$



$$\frac{\partial L'}{\partial w_0} = \frac{\partial L}{\partial w_0} + \frac{\lambda}{2} \cdot 2 w_0$$

$w_1^2 + [w_2^1 + w_3^2 + \dots]$ → so will be 0.

differentiate w.r.t. output

$2w$

$$= \frac{\partial L}{\partial w_0} + \lambda w_0$$

$$w_n = w_0 - \eta \frac{\partial L}{\partial w_0} \longrightarrow w_n = w_0 - \eta \left(\frac{\partial L}{\partial w_0} + \lambda w_0 \right)$$

$$w_n = w_0 - \eta \lambda w_0 - \eta \frac{\partial L}{\partial w_0}$$

$$w_n = (1 - \eta\lambda) w_0 - \eta \frac{\partial L}{\partial w_0}$$ → almost same

→ difference

$$w_n = w_0 - \eta \frac{\partial L}{\partial w_0}$$

$1 - \eta\lambda$ → output is less than 1 and when we subtract with w_0 then value of w_0 will decrease.

$w_0 >> (1 - \eta\lambda) w_0$ → also called weight decay. $w_0 \downarrow\downarrow$

old w_0 ↳ new w_0

Jitne No. of epochs chalega utni bar w_0 ki value decrease hogi. Zero ke pass ja skta hai but completely zero nhi hota.

L1 is a Sparse Model
↳ where many node will be 0
And Make Simple Model or Neural Network.

But in L2 → Nods are near to zero but not completely zero. So, Our model not become very simple neural Network.

L2 is useful but try run both L1 and L2