# Unit – V

## (a) Challenges in Neural Network Optimization:-

→ Optimization tries to minimize the error or loss and trying to reduce them by adjusting weights & biases.

→ Challenges:-

a) Local Minima:-

- The grandfather of all optimization problems, local minima is a permanent challenge in the optimization of any deep learning alg.
- The local minima problem arises when the gradient encounter many local minimums that are like valleys, where the error is small, but it's not the lowest.

b) Saddle Points:-

Saddle Points are that regions where the gradient (rate of change) is zero, making it diff to move in right direction.

c) Vanishing Gradients:-

They occur during backpropagation where the gradients (changes in weights) become very small, preventing the network from learning effectively.

d) Overfitting:-

It happens when the model learns the training data too well, including noise & irrelevant details, making it perform poorly on new, unseen data.

c) Underfitting:-

It occurs when model is too simple or not trained enough and it fails to capture important patterns in data.

(d) Gradient-Descent Alg:-

→ It is an optimization alg used to min loss function by updating model parameters (weights ~~biases~~) in the direction of negative gradient.

→ It is a first order optimization alg.

→ working:-

1) Start with Initial Parameters:

· Initially, the parameters (weights) of model are set to random values.

2) Compute the Gradient:

· The gradient (rate of change ~~change in weights~~) of loss function is calculated w.r.t each parameter (weight).

· The gradient represents rate of change of loss function in direction each parameter.

3) Update the Parameters:-

· The parameters are updated in opposite direction of gradient to min loss function. Using formula

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

$\theta$ → model parameters (weights)

$\alpha$ → learning rate

$\nabla_\theta J(\theta)$ → gradient of loss function $J(\theta)$

4) Repeat process for several epochs until parameters converge to values that min loss function.

→ Disadv

- Can be slow for large datasets
- Can get stuck in local minima

② Stochastic Gradient Descent:-

→ SGD is a variant of basic GD alg. Unlike traditional one, which uses entire dataset to compute the gradient & update model parameters, SGD updates the parameters using only one data point at a time.

→ This makes the alg. faster but more noisy.

→ Working:-

1) Initialization:
start with random values for model parameters.

2) Select a single data point:
Instead of using whole dataset, SGD selects one data point randomly from the dataset.

3) Compute the Gradient:-
The gradient of loss function is computed with r. to the model parameters, using selected data point.

4) Update Parameters:-
The parameter are updated based on the gradient, using formula:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

$\nabla_\theta J(\theta)$ → gradient of loss function $J(\theta)$ w.r.t to parameters, based on selected data point.

5) Repeat process for each data point in the dataset. After one full pass through the data, this is called one epoch.
The parameters are updated in each epoch.

6) Iterate the process for multiple epochs until model parameters converge to values that min loss function.

→ Adv :-
- Faster Updates
- Escaping local minima

→ Disadv :-
- Noise can make it less stable
- Can take longer for convergence.

④ Mini - Batch Gradient Descent :-

→ It is an optimization alg that strikes a balance b/w Batch GD & SGD.

→ Instead of using the entire dataset or just a datapoint, MBGD updates the model parameters using a small subset of data, called mini-batch.

- Working :-

1) Divide the dataset into Mini-batches :-
- The entire dataset is divided into small batches.
- Each mini-batch contains 32 - 256 data points
- Size of mini-batch can act as hyperparameter. If chosen very less it can become noisy like SGD, if chosen more it can be slow like GD.

2) Initialize Parameters.

3) Compute the Gradient:-

For each mini-batch, the gradient of loss function is computed based of avergage of gradients of data points in that mini-batch.

4) Update Parameters:-

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

$\nabla_\theta J(\theta) \rightarrow$ gradient of $J(\theta)$ based on mini-batch.

5) Repeat for each mini-batch till an epoch.

6) Iterate for multiple epochs until convergence.

→ Adv:-

- Faster Update & req. less computation
- Memory efficient
- More stable

→ Disadv:-

- Choosing mini-batch size

(d) AdaGrad (Adaptive Gradient Alg):-

→ It is an optimization alg. that adapts the learning rate for each parameter individually based on its historical gradient.

→ It aims to improve learning rate by adjusting it for each parameter, allowing more frequent updates for parameters with smaller gradients and fewer updates ~~with~~ for parameters with larger gradients.

→ Working :-

1) Initialization of Parameters

2) Compute Gradients :-

For each parameter, AdaGrad computes gradient of the loss function wrt to that parameter (weights)

3) Accumulate Squared Gradients :-

- AdaGrad keeps track of the sum of squared gradients for each parameter. This helps in adjusting learning rate for each parameter.

$$G_{t,i} = G_{t-1,i} + g_{t,i}^2$$

where,

$G_{t,i}$ → accumulated squared gradient for paramet $i$ at '$t$'

$g_{t,i}$ → gradient of loss wrt '$i$' at '$t$'

4) Update Parameters :-

- The parameters are updated using

$$\theta_i = \theta_i - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}$$

$\theta_i$ → parameter

$\alpha$ → global learning rate

$\epsilon$ → small value added to prevent div by '0'.

5) Repeat for each data point or mini-batch (depending on variant of GD used) as the model parameters are updated after each iteration.

→ Disadv :-

- Learning rate will diminish over time
- Not ideal for non-convex functions.

- RMS Prop (Root Mean Square Propogation):-
~~(4) Adam (Adaptive Moment Estimation):-~~

- It is an ~~advanced~~ optimization alg. that modifies AdaGrad to address the issue of rapidly diminishing learning rates.
- It uses moving average of squared gradients to normalise the gradient, allowing for more stable & effective updates, especially in non-stationary setting (ex:- RNNs).
- Working:-
  1) Initialization
  2) Comput Gradients
  3) Calculate moving Average of Squared Gradients:-
     Instead of accumulating all squared gradients, RMS uses decaying average of squared alg.

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2$$

$v_t \rightarrow$ moving avg of squared gradients at 't'.

$g_t \rightarrow$ gradient of loss function

$\beta \rightarrow$ decay factor (typically close to 1) which controls how much weight is given to past gradients.

4) Update Parameters:-

$$\theta = \theta - \frac{\alpha}{\sqrt{v_t} + \epsilon} \cdot g_t$$

5) Repeat for each data point or mini-batch and the model parameters are updated iteratively.

→ Adv:-

- Avoids diminishing learning rates.
- Better performance in non-convex problems
- Faster convergence

→ Disadv:-

- Sensitive to hyperparameters ($\alpha$ & $\beta$)
- Choice of decay factor

⑥ Adam (Adaptive Moment Estimation):-

→ It is an ~~optim~~ advanced optimization alg. that combines the ideas of Momentum & RMSprop.

→ It computes adaptive learning rates ~~from~~ for each parameter, using both first moment (mean) & second moment (variance) of gradients.

→ Adam is one of the most widely used optimizers in DL due to its efficiency.

→ Working:-

1) Initialisation:-

- Adam starts by initializing the model parameters randomly
- Additionally, two moment estimates are initialized to zero:

$m_0 = 0$ (First moment, gradient mean)

$v_0 = 0$ (Second moment ~~est~~ estimate)

2) Compute Gradients :-

3) Update Moment Estimates :-

Adam computes 2 moment estimates.

- First moment estimate : The moving average of gradients ($m_t$)
- Second moment estimate : Moving avg of squared gradients ($v_t$)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$\beta_1$ & $\beta_2$ → hyperparameters that control decay rates for moment estimates

4) Bias Correction :

Since $m_t$ & $v_t$ are initialized to 0, their estimates are biased toward '0'.

To correct this bias,

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$\hat{m}_t$ & $\hat{v}_t$ → bias corrected estimates

5) Update parameters :-

$$\theta = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

6) Repeat for data point & minibatch and the model parameters are updated iteratively.

→ Disadv :-

Memory Usage, Hyperparameter Sensitivity, Risk of Overfitting.

## Large Scale Deep Learning:-

- It involves using DNNs to process & learn from vast amounts of data, which requires powerful hardware, like GPUs or specialized h/w to handle large scale computations.
- NNs are trained on massive datasets that could include millions or billions of examples.
- As the n/w processes more data, it improves its ability to make accurate predictions.
- The larger the dataset, the better model can generalize & perform complex task.
- Big Data - Large scale datasets that contain a lot of info like images, videos, text & sensor data.
- Using GPUs instead of traditional CPUs speeds up training.
- In large scale DL, the task is often distributed across multiple machines or even data centers to speed up training process.
- Appl:-

  Self Driving Cars, Recommendation Systems, Healthcare
- Challenges:-

  Data Quality

  Computational Power (large)

  Overfitting

(4) Computer Vision :-

-) It is a field in AI that enables machines to interpret & understand visual info from the world, like images or videos.

-) DL, particularly CNNs play crucial role in computer vision

-) CNNs are used to process image data. They work by applying filters to detect edges, textures, other features in the image at diff levels.

-) CNNs apply multiple layers like convolutional layers to extract features, pooling to extract features and in reducing computation cost by reducing spatial size of image.

-) Image classification

-) Object detection (classifying multiple objects in images)

-) Semantic segmentation aims to label each pixel in an image with its corresponding object class.

-) GANs are used to generate realistic images that resemble specific dataset.

-) Appl :-

Healthcare

Autonomous Driving

Surveillance & security

④ NLP:-

→ NLP in deep learning refers to process, understand & generate human language.

→ NLP enables computers to interact with & understand human lang. in a way that is meaningful & contextually relevant.

→ Deep L particular NNs have revolutionalized NLP tasks.

→ Text Classification (spam detection, topic categorization)

→ Machine Translation (sequence-to-sequence RNNs or transformers)

→ Named Entity Recognition (Names of people, org. locations)

→ Text Summarization

→ Sentiment Analysis

→ Question Answering

→ Speech Recognition & Synthesis

(Converting spoken lang to text) (Generating human-like speech from text)

→ Language Generation (Chatbots & Lang. Models)

→ RNNs, LSTMs, Transformers (GPT, BERT)

Generative Pre Trained Transformer