SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

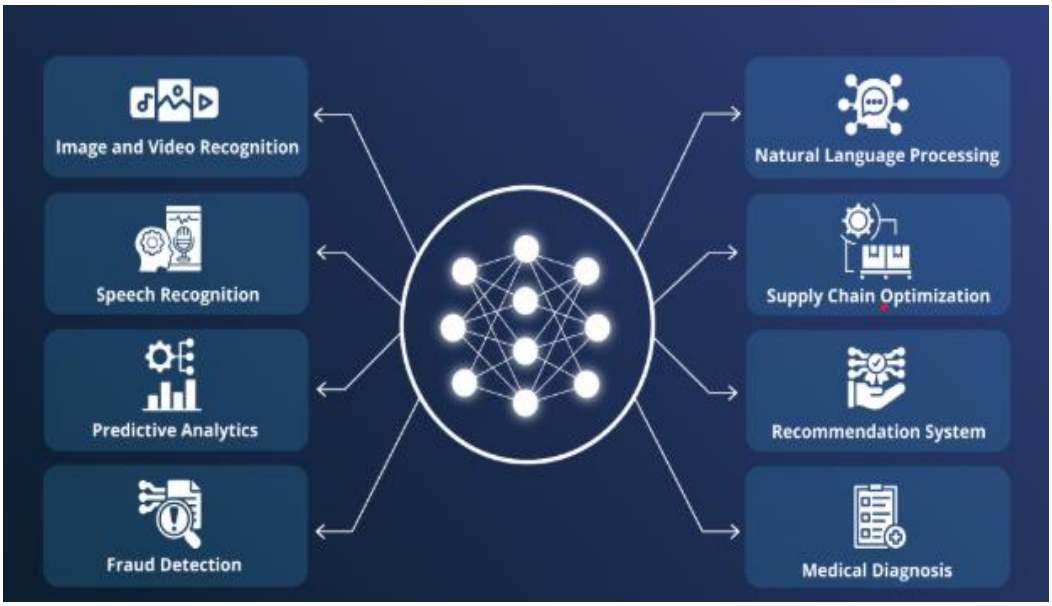
Subject Name: DEEP LEARNING

Year / Branch: II MCA – II SEMESTER

Regulation: R24

Prepared By: Mr. N P Gangadhar,

Assistant Professor



Deep learning is the key technology behind many recent breakthroughs in artificial intelligence

—Yann LeCun

II MCA – II SEMESTER			
COURSE CODE:	**24MCA221**	**CREDITS:**	**4**
COURSE TITLE:	**DEEP LEARNING**	**L-T-P:**	**4-0-0**

PREREQUISITES: *A Course on "Machine learning", Artificial Neural Networks and Knowledge on basic linear algebra may be helpful.*

COURSE EDUCATIONAL OBJECTIVES:

CEO 1: Toacquireknowledgeaboutdifferentmathematicaltoolsfor DeepLearning.
CEO 2: Tounderstandfundamentalsofartificialneural networks.
CEO 3: ToExplore various optimizers and Regularization Techniques
CEO 4: ToLearn about ConvolutionalNeural Networks.
CEO 5:Tocomprehend Object Detection, Autoencoder and GAN Architectures

UNIT-I: MATHEMATICAL TOOLS FOR DEEP LEARNING — **Lecture Hrs:12**

Linear Algebra :Matrix, Vector,Transpose,Tensor, operations on elements, Systems of Linear equations,Rank, Norm,expressing a Matrix, Determinant, Trace, Eigen values and Eigen Vectors, Singular Value Decomposition(SVD). **Statistics** – Probability, Random Variable, Binomial Distribution, Poisson Distribution, Normal Distribution, Sampling, Central limit Theorem. **Calculus**- Derivatives, rules forderivatives,Partial derivatives.

UNIT-II: FUNDAMENTALSOFARTIFICIALNEURALNETWORK(ANN) — **Lecture Hrs:12**

Understanding the Biological Neuron, Exploring the Artificial Neuron, Early Implementation of ANN – RmcCulloch-Pits mosel of Neuron, Rosenblatt's Perceptron,Types of Activation Functions-linear function, non-linear function, softmax function. Architectures of neural network- Single layer feed forward network, Multi-layer feed forward ANN, Recurrent Neural Network, Convolutional Networks. Learning Process in ANN – Weight of Interconnection between neurons, Gradient Descent and Backpropagation.

UNIT-III: TRAININGDEEPNEURALNETWORK — **Lecture Hrs:12**

Initializing Weights- He/ Kaiming initialization, Xavier initialization. Batch, Mini-batch and stochastic gradient descent. Regularization-L1/ L2 regularization, Early stopping, Dropout regularization, data augmentation. Normalization of inputs-Batch Normalization, Batch Normas regularizer.

UNIT-IV: CONVOLUTIONAL NETWORKS — **Lecture Hrs:10**

Building blocks of CNN, Building a Convolution Neural Network, Popular CNN Architectures-LeNet-5, AlexNet, ZFNET, VGG-16, GoogleNet and ResNet Object Detection- one stage deection techniques – YOLO, SSD,Two stage object detection techniques – R-CNN, fast R-CNN, faster R-CNN, Mask R-CNN, Applications of Object Detection

UNIT-V: SEQUENCE-BASEDMODELS ANDOTHERDLARCHITECTURES — **Lecture Hrs:12**

Recurrent Neural Network – Data Preparation for RNN Vanishing Gradient problem and RNN, Applications of RNN, Types of RNN, Limitations of RNN, Longshort-termmemory (LSTM), Gated RNNs, Bidirectional RNNs. Other DL Architectures- Autoencoder, Architecture and its applications, GAN, GAN Architecture and its applications,

TEXTBOOKS:

1. *AmitKumarDas,SaptarsiGoswami,PabitraMitra,AmlanChakrabarti,"DeepLearning", Pearson Paperback,First Edition,2021.*

2. *IanGoodfellow,YoshuaBengio,AaronCourville, "DeepLearning",MITPress,2016.*

REFERENCE BOOKS:

1. *JoshPattersonandAdamGibson,"Deeplearning:Apractitioner'sapproach",O'ReillyMedia,First Edition,2017.*
2. *NikhilBuduma, "Fundamentals of Deep Learning, Designing next-generationmachineintelligence algorithms",O'Reilly,ShroffPublishers,2019.*

COURSE OUTCOMES: *On successful completion of this course, students will be able to:*		POs related to COs
CO1	Understand the mathematical background required for Deep learning.	**PO1,PO2**
CO2	Analyze the fundamental concepts of Artificial neural networks.	**PO1,PO2,PO3**
CO3	Understand and apply the deep neural networks	**PO1,PO2,PO3,PO4**
CO4	Explore the Purpose of Convolution Neural Network, Popular Architectures of CNN	**PO1,PO2,PO3,PO4,PO8**
CO5	To learn about sequence-based models like RNN, LSTM and Gated RNN and other DL architectures and use for real-world problems	**PO1,PO2,PO3,PO4,PO8**

CO-PO MAPPING(DETAILED; HIGH:3; MEDIUM:2; LOW:1)

Course	POs / COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8
C401: Deep Learning	C401.1	3	3	-	-	-	-	-	-
	C401.2	3	3	2	-	-	-	-	-
	C401.3	3	3	2	3	-	-	-	-
	C401.4	3	3	3	2	-	-	-	3
	C401.5	3	3	3	3	-	-	-	3
	C401	3	3	2.5	2.67				3

UNIT III -TRAINING DEEP NEURAL NETWORK

Unit Overview

This unit covers initializing weights using Xavier and He initialization, different forms of gradient descent such as batch, mini-batch, and stochastic gradient descent, and common regularization techniques including L1, L2, early stopping, dropout, and data augmentation. It also explains normalization of inputs, especially batch normalization, and the idea of batch normalization acting as a regularizer during training.

Learning Outcomes

After completing this unit, students will be able to:

1. Explain the purpose of weight initialization in neural networks.
2. Distinguish between He/Kaiming initialization and Xavier initialization.
3. Compare batch gradient descent, mini-batch gradient descent, and stochastic gradient descent.
4. Explain the need for regularization in deep learning.
5. Describe L1, L2, early stopping, dropout, and data augmentation.
6. Explain normalization of inputs and the role of batch normalization.
7. Describe how batch normalization can also provide a regularization effect.

Importance of Studying this Unit

This unit is important because training a neural network is not only about choosing an architecture. Good performance also depends on proper initialization, efficient optimization, prevention of overfitting, and stable normalization. These techniques are fundamental in practical deep learning systems and are used in most modern neural-network training pipelines.

Key Concepts

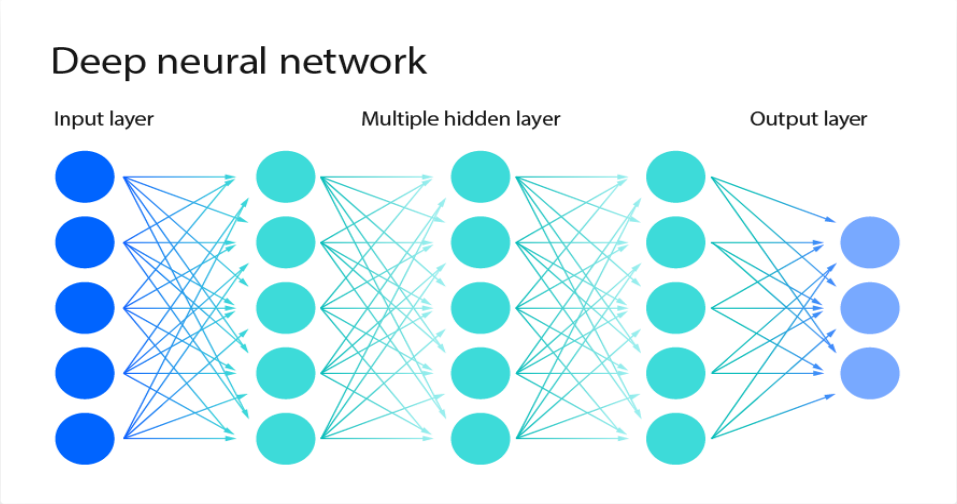
Weight initialization, Xavier initialization, He/Kaiming initialization, gradient descent, batch gradient descent, mini-batch gradient descent, stochastic gradient descent, regularization, L1 regularization, L2 regularization, early stopping, dropout, data augmentation, normalization, batch normalization, batch norm as regularizer.

Deep Neural Network (DNN)

- A Deep Neural Network (DNN) in deep learning is an Artificial Neural Network (ANN) with multiple hidden layers between its input and output, allowing it to learn complex,

hierarchical patterns from data, essentially enabling automatic feature extraction for tasks like image recognition and natural language processing.

- The "depth" comes from these many layers, enabling greater abstraction and performance than traditional shallow networks, and forming the core of modern deep learning applications.



Step-by-Step Explanation

Input Layer

- Receives raw input data (features).
- Example: pixels of an image, numerical values, text embeddings.
- No computation is done here—data is only passed forward.

Hidden Layers

- A Deep Neural Network has **multiple hidden layers** (2 or more).
- Each neuron performs:
- $z = (w1x1 + w2x2 + ... + b)$
- $a = activation(z)$
- Early layers learn **simple features**.
- Deeper layers learn **complex patterns**.

Weights and Bias

- **Weights (w)** control the importance of inputs.
- **Bias (b)** allows flexibility in shifting activation.
- These parameters are learned during training.

Activation Function

- Introduces non-linearity.
- Common choices:
  - ReLU (hidden layers)
  - Sigmoid / Softmax (output layer)

Output Layer

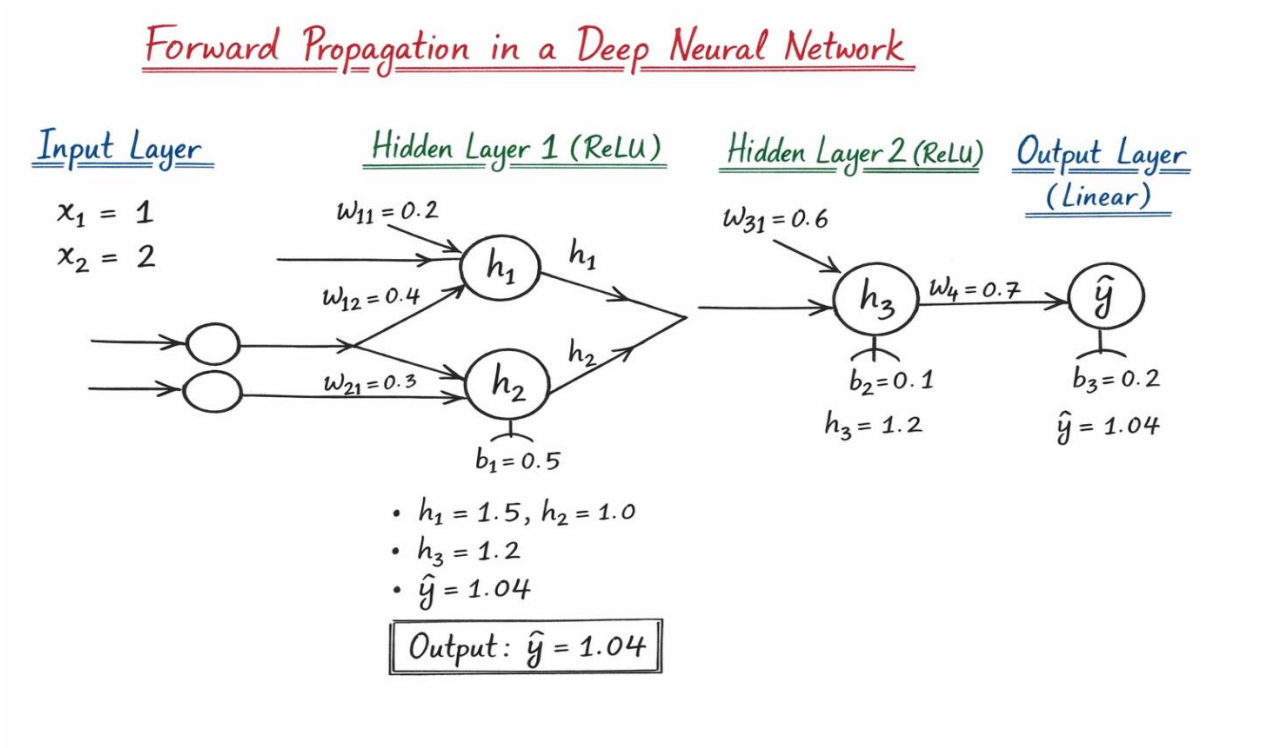
- Produces the final prediction.
- Examples:
  - Binary classification → Sigmoid
  - Multi-class classification → Softmax
  - Regression → Linear activation

Forward Propagation

- Data flows from **input → hidden layers → output**.
- Each layer transforms the data progressively.

Backpropagation (Training Phase)

- Error is calculated at the output.
- Error is propagated backward.
- Weights are updated using **gradient descent**.



Key Characteristics:

- **Multiple Layers:** Unlike basic ANNs, DNNs have many hidden layers (sometimes hundreds) that progressively process data, extracting increasingly complex features.
- **Automatic Feature Extraction:** DNNs learn relevant features directly from raw data (like pixels in an image) without manual engineering, a major advantage.
- **Non-linear Transformations:** Each layer applies non-linear functions, allowing the network to model intricate, non-linear relationships in data.
- **Backpropagation:** They learn by adjusting internal connections (weights) based on errors, using algorithms like gradient descent to minimize mistakes.
- **Foundation of Deep Learning:** DNNs are the fundamental architecture powering most deep learning models, driving AI advancements in various fields.

How it Works:

1. **Input Layer:** Receives raw data (e.g., image pixels, text).
2. **Hidden Layers:** Data flows through these layers, with each layer performing complex transformations, building representations from simple features (edges) to complex ones (objects).

3. **Output Layer:** Produces the final result (e.g., classifying an image as a cat or dog).

Applications:

- Image and Speech Recognition
- Natural Language Processing (NLP)
- Predictive Analytics
- Self-driving Cars

Initializing Weights

- Weight initialization is the process of assigning initial values to weights before training a neural network. Proper initialization helps avoid vanishing and exploding gradients.

Why Weight initialization

To Break Symmetry

If all weights are initialized to the **same value (e.g., zero)**:

- All neurons in a layer learn the **same features**
- Network becomes useless
- **Random initialization** ensures each neuron learns **different features**.

To Avoid Vanishing Gradients

- Very small initial weights → gradients shrink as they move backward
- Early layers stop learning

To Avoid Exploding Gradients

- Very large initial weights → gradients grow uncontrollably
- Training becomes unstable

To Speed Up Convergence

- Poor initialization → slow or no learning
- Proper initialization → faster training and fewer epochs

To Maintain Stable Activations

- Ensures outputs of each layer have similar variance
- Prevents saturation of activation functions

To Improve Model Accuracy

- Helps the model reach a **better optimum**
- Reduces chances of getting stuck in poor solutions

He / Kaiming Initialization

- He initialization (also called Kaiming initialization) was developed later to handle the ReLU family of activations.
- Because ReLU zeroes out negative values, it effectively halves the variance of the input signal something the original Xavier scheme doesn't account for.
- **Formula:** Let n_{in}= number of input neurons

Normal Distribution: Weights are drawn from a normal distribution:

$$w \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

Uniform Distribution

$$w \sim U\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right)$$

Xavier (Glorot) Initialization

- Xavier initialization (named after Xavier Glorot and Yoshua Bengio) was one of the first mathematically grounded schemes designed to maintain stable variances across layers.
- It was developed assuming the network uses symmetric activation functions like sigmoid or tanh.
- **Formula**

  n_{in}=number of input neurons
- n_{out}= number of output neurons

Normal Xavier

Weights are from a normal distribution with mean 0 and variance:

$$w \sim N(0, \frac{2}{n_{in}+n_{out}})$$

Uniform Xavier

Weights are drawn from a uniform distribution:

$$w \sim U\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right)$$

Gradient based Learning

- Gradient-based learning is a type of machine learning in which the **optimization algorithm** uses gradients to **update the model parameters during training.**
- This approach is commonly used in **deep learning and neural networks** because it allows the model to learn **complex representations of the input data**.
- It works by **iteratively adjusting the model's weights and biases** to minimize the cost function (error).
  At each step, the parameters are updated in the direction of the negative gradient of the cost function until the error becomes **very small or zero.**

Working of gradient descent:

Step 1 – Define the Goal:

- The goal of **Gradient Descent** is to **minimize the cost function** (also called the loss function), which measures the difference between:
- **Actual output (y)** (the ground truth label).
- **Predicted output (ŷ)** (what the model predicts).

Step 2 – Calculate the Cost Function:

- The **Cost Function (J)** is computed as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Loss\,(y_i, \hat{y}_i)$$

- where:
- m= total number of training examples.
- θ= model parameters (e.g., weights).

The **Loss function** measures error for a single data point.

Step 3 – Compute the Gradient (Direction):

- Calculate the **partial derivative** of the cost function with respect to each parameter θ:

$$\frac{\partial J(\theta)}{\partial \theta}$$

- This tells us the direction in which the cost function increases the most.

Step 4 – Take a Step in the Opposite Direction (Steepest Descent):

- To **reduce the cost function**, we move in the **negative direction of the gradient**.
- The **update rule** is:

$$\theta := \theta - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta}$$

- where:
- α= learning rate (a small positive number).
- θ= current parameter values.

Step 5 – Learning Rate Consideration:

- If α(learning rate) is:
- **Too high** → Might overshoot the minimum → Risk divergence.
- **Too small** → Takes a lot of time → More computational effort.
- Proper choice of αhelps in fast yet stable convergence.

Step 6 – Iterate Until Convergence:

- Repeat steps 3 and 4 iteratively.
- The model keeps adjusting θ in each iteration.
- The process continues until:
- The **change in cost function becomes negligible**.
- Or **maximum number of iterations is reached**.
- Or the **cost function approaches zero**.

Step 7 – Termination:

- Once the cost is minimized and the parameters stabilize, the model stops updating.
- At this point, the model is considered **trained**.

Types of gradient descent

There are three types of gradient descent learning algorithms

1. Batch gradient descent,
2. Stochastic gradient descent
3. Mini-batch gradient descent.

Batch Gradient Descent (Vanilla Gradient Descent)

- ❖ In this method, the **error is calculated for every sample in the dataset**, but the model parameters are **not updated until the entire dataset has been processed**.
- ❖ One full pass through the dataset is called an **epoch**.
- ❖ This gives a **stable error gradient** and usually leads to **stable convergence**.

Advantages

1. Fewer updates → more **efficient** than updating after each sample.
2. Updates are **stable** and smoother, which helps in convergence.

3. Easy to use with **parallel processing** (faster calculations on modern hardware).

Disadvantages

1. May **stop early at a not-so-good solution** (suboptimal).
2. Needs to **store the whole dataset in memory**.
3. **Slow training** for large datasets (updates happen only once per epoch).
4. Collecting all errors before updating adds extra complexity.
5. Needs **more time and computing power**.

Stochastic Gradient Descent (SGD)

- In SGD, the model updates **after every single training sample**, not after the whole dataset.
- This makes updates **frequent** and sometimes **faster** than batch gradient descent.
- But, because updates happen so often, the error path is **noisy** (it jumps up and down instead of moving smoothly).

Advantages

1. Frequent updates show **immediate progress** in learning.
2. Very **easy to understand and implement**.
3. Frequent updates let the model **learn quickly in the beginning**.
4. Noise in updates helps avoid getting stuck in **local minima**.
5. **Faster** and needs **less memory** (doesn't require whole dataset at once).
6. Works well for **large datasets**.

Disadvantages

1. Too many updates can be **computationally heavy** for large datasets.
2. Frequent updates create **noisy gradients** (error jumps instead of going down smoothly).
3. The noise can make it **hard for the model to settle at the true minimum**.

Mini-Batch Gradient Descent

- It's a mix of **Batch Gradient Descent** and **Stochastic Gradient Descent (SGD)**.
- The training dataset is split into **small groups (mini-batches)**, usually between **50–256 samples**.
- The model updates its parameters after each mini-batch, not after the whole dataset (like Batch) and not after every single sample (like SGD).
- This makes it the **most popular method in deep learning**.

Advantages

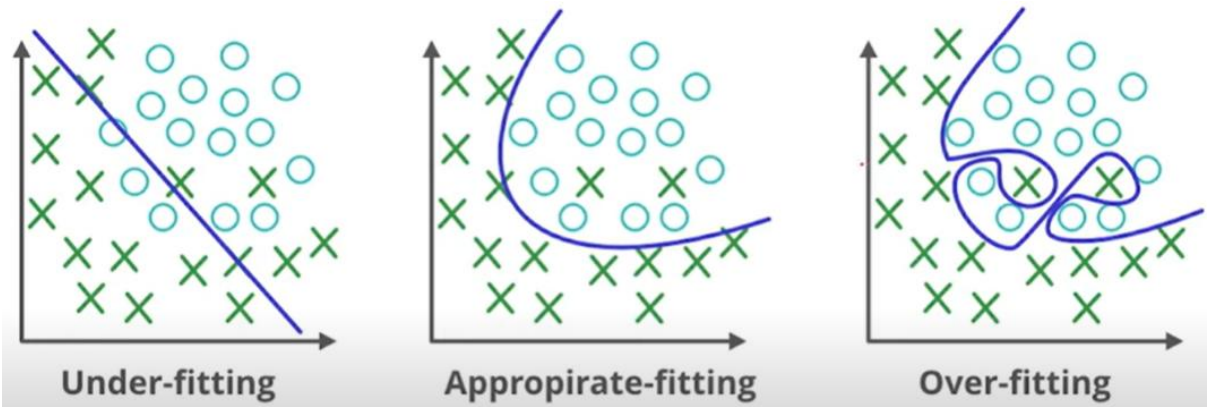
1. Updates happen more often than Batch, so the model **learns faster** and avoids getting stuck in bad solutions (**local minima**).
2. More **computationally efficient** than SGD (since updates are done in batches, not sample by sample).
3. Doesn't need the **whole dataset in memory at once**, which is good for large datasets.

Disadvantages

1. Requires setting an extra parameter: **mini-batch size**.
2. Like Batch, it still needs to **accumulate errors** for a batch before updating.
3. Can lead to **more complex behavior** compared to the other methods.

Regularization

- ❖ Regularization is a technique used to reduce errors by fitting the function appropriately on the given training set and avoiding overfitting.
- ❖ Overfitting and underfitting are common challenges in neural networks.
- ❖ Overfitting occurs when a model performs very well on training data but fails to generalize to new or unseen data, while underfitting happens when a model performs poorly on both training and validation datasets.



The main benefits of regularization

- **Reducing overfitting:** By constraining the model's complexity, regularization helps prevent the model from memorizing noise or irrelevant patterns in the training data.
- **Improving generalization:** Regularized models tend to perform better on new, unseen data because they focus on capturing the underlying patterns rather than fitting the training data perfectly.
- **Enhancing model stability:** Regularization makes models less sensitive to small fluctuations in the training data, leading to more stable and reliable predictions.
- **Enabling feature selection:** Some regularization techniques, such as L1 regularization, can automatically identify and discard irrelevant features, resulting in more interpretable models

Types of Regularizations

The commonly used techniques are:

1. Lasso Regularization – L1 regularization
2. Ridge Regularization – L2 regularization

Lasso Regularization/ L1 regularization

- A regression model which uses the L1 regularization technique is called LASSO (Least Absolute Shrinkage and Selection Operator) regression

- L1 Regularization is a technique used to reduce overfitting in neural networks by adding a penalty to the loss function during training. This penalty is the sum of the absolute values of the weights in the model.
- In each training step, the model tries to minimize the total loss.
- Because large weights increase the loss (due to the penalty), the optimizer shrinks the weights.
- Some weights become exactly 0 — meaning those neurons or input features are ignored.
- This makes the model simpler and more efficient.
- Let's say the original loss function (like Mean Squared Error or Cross-Entropy) is:

  Loss = L(y_true, y_pred)
- With **L1 Regularization**, the new loss becomes:

  Loss = L(y_true, y_pred) + λ * Σ |w|

Where:

- L(y_true, y_pred) = original loss (error between actual and predicted)
- λ (lambda) = regularization strength (a small positive number, like 0.01)
- Σ |w| = sum of absolute values of all weights in the network

EX:

◆ 3. $\sum |w|$

This is the **sum of the absolute values of the weights**.

- If your model has 3 weights:
  - $w_1 = 2.0$
  - $w_2 = -1.5$
  - $w_3 = 0.0$

Then:

$$\sum |w| = |2.0| + |-1.5| + |0.0| = 2.0 + 1.5 + 0.0 = 3.5$$

Let's say:

- Original loss = 0.25
- λ = 0.01
- Sum of weights (|w|) = 3.5

Then:

$$\text{Total Loss} = 0.25 + 0.01 \cdot 3.5 = 0.25 + 0.035 = 0.285$$

Ridge Regression

- A regression model that uses the L2 regularization technique is called Ridge regression.
- L2 Regularization is a technique used to prevent overfitting by discouraging the model from having large weight values.
- It adds a penalty to the loss function based on the square of the weights.
- where
    - 'm' number of feature
    - 'n' number of examples
    - Y_i actual target value
    - $Y_i\hat{}$ predicted target value
    - 'w' weight

$$\text{Loss} = L(y_{\text{true}}, y_{\text{pred}}) + \lambda \cdot \sum w^2$$

Where:

Term	Meaning
$L(y_{\text{true}}, y_{\text{pred}})$	Original loss (like MSE or Cross-Entropy)
λ	Regularization strength (a small value like 0.01)
$\sum w^2$	Sum of the **squared weights** in the model

Let's say:

- Weights: $w = [2.0, -1.5, 0.0]$
- Then: $\sum w^2 = 2.0^2 + (-1.5)^2 + 0.0^2 = 4.0 + 2.25 + 0 = 6.25$
- Lambda (λ) = 0.01
- Original Loss (e.g., MSE) = 0.25

Then:

$$\text{Total Loss} = 0.25 + 0.01 \cdot 6.25 = 0.25 + 0.0625 = 0.3125$$

Early Stopping

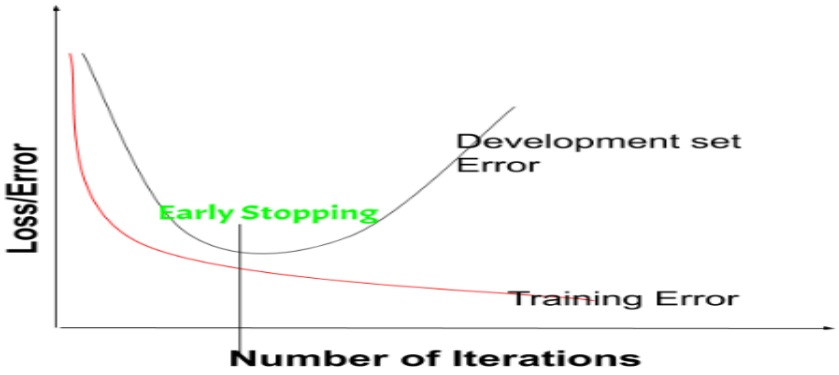
❖ Early stopping is a smart trick to stop training at the right time, before overfitting happens

❖ A problem with training neural networks is in the choice of the number of training epochs to use.

❖ Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model.

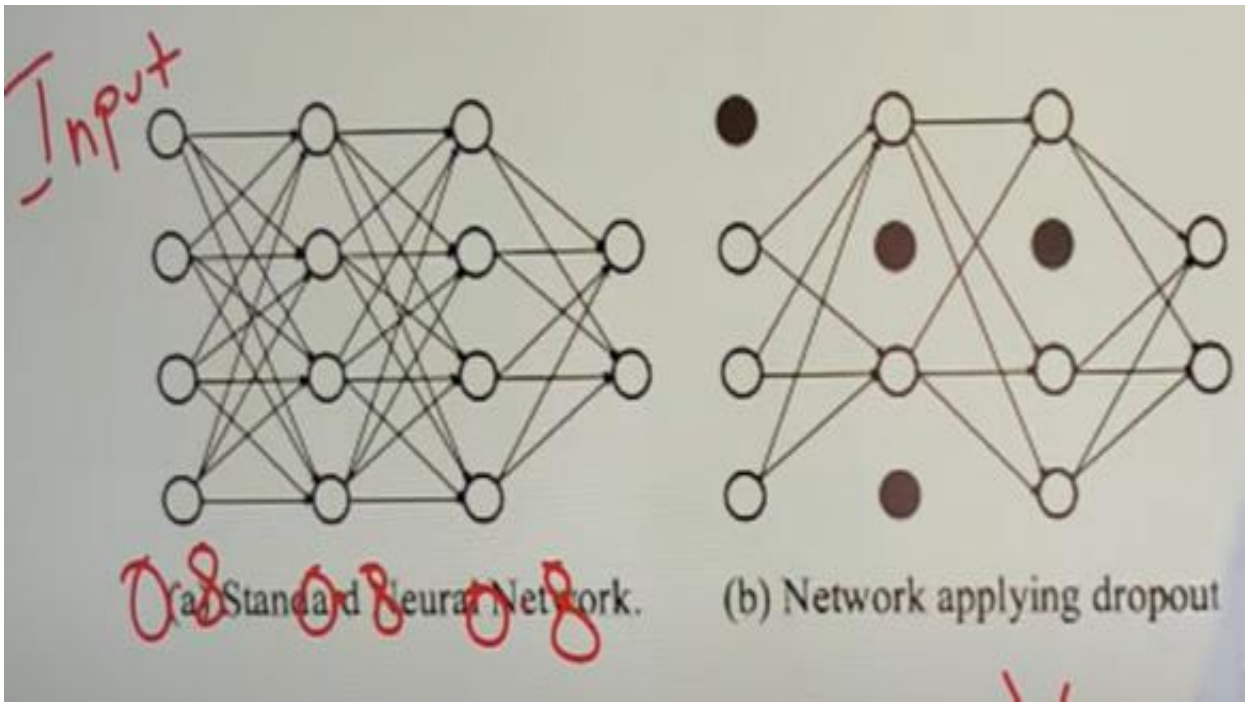
How It Works (Step-by-Step):

- Train your model with many epochs (say 100 or 200).
- After each epoch, check how the model performs on the validation dataset.
- Keep track of validation error (or loss).
- If the validation error starts increasing, it means the model is starting to overfit.
- Stop training at the point where validation error is lowest.



Drop Out Regularization

- Dropout regularization is a widely used technique in deep learning to prevent overfitting in neural networks.
- It works by randomly "dropping out," or setting to zero, a fraction of the units (i.e., neurons) in the neural network during training.
- This means that at each training iteration, certain neurons are temporarily removed from the network with a probability defined by the dropout rate.



- For n neurons , we get 2^n different Thinned Networks.
- During training, all 2^n different Thinned Networks are trained for each batch.

- While testing the original network is tested

Here's how dropout regularization typically works:

- **During Training**: In each training iteration, a fraction of neurons in the hidden layers are randomly selected to be "dropped out" or temporarily removed from the network. The dropout rate is a hyperparameter that determines the probability of any particular neuron being dropped out. For example, a dropout rate of 0.5 means that each neuron has a 50% chance of being dropped out.
- **Forward Pass**: During the forward pass, the network behaves as usual, but with the selected neurons set to zero. This means that the activations and outputs of these neurons are ignored for that particular training iteration.
- **Backward Pass**: During the backward pass (backpropagation), only the active neurons (i.e., those that were not dropped out) contribute to the gradient computation. This means that the network's parameters are updated based only on the active neurons.
- **Testing Phase**: During the testing or inference phase, dropout is typically turned off, and all neurons are used. However, the weights of the connections are usually scaled by the dropout rate to account for the fact that more units are active during testing than during training

Data Augmentation

- Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data.
- It includes making minor changes to the dataset or using deep learning to generate new data points.

Augmented vs. Synthetic data

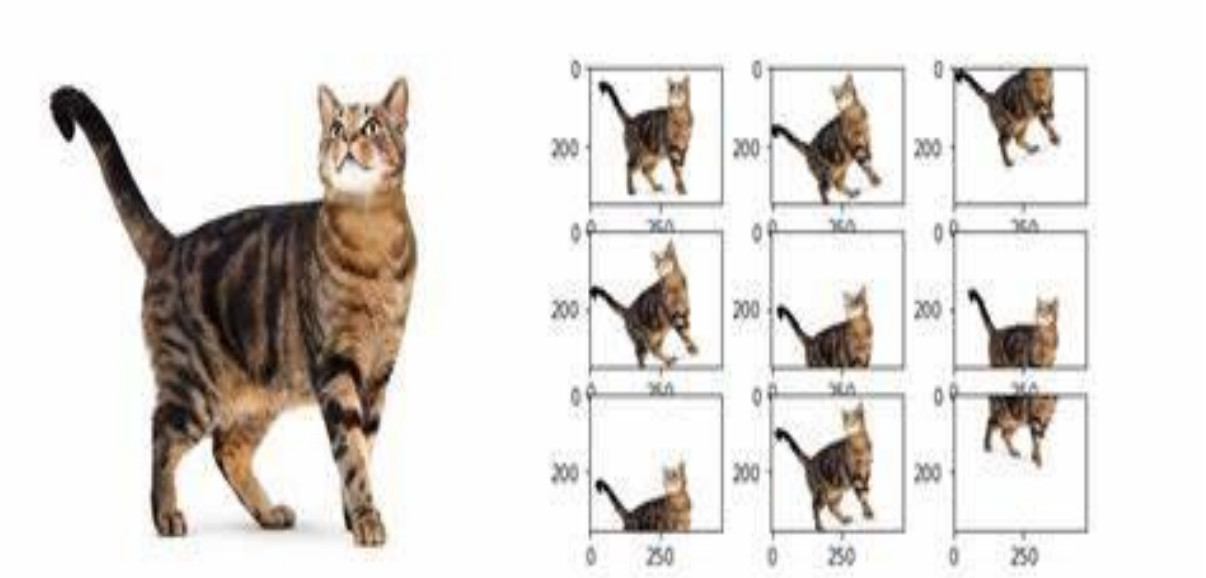
- **Augmented data** is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.
- **Synthetic data** is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.

When Should You Use Data Augmentation?

1. To prevent models from overfitting.
2. The initial training set is too small.
3. To improve the model accuracy.

Image Augmentation

1. **Geometric transformations**: randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
2. **Color space transformations**: randomly change RGB color channels, contrast, and brightness.
3. **Kernel filters**: randomly change the sharpness or blurring of the image.
4. **Random erasing**: delete some part of the initial image.
5. **Mixing images**: blending and mixing multiple images.



Normalization

- Normalization in machine learning and Deep Learning is a data preprocessing technique used to scale feature values to a common range or scale so that no single feature dominates due to large numeric differences.
- It makes sure that all features contribute fairly when training a model.
- **Normalization resizes feature values** to a standard range, typically **0 to 1** or **−1 to 1**.
- It **adjusts values so they're on a similar scale** without changing the underlying patterns.

Why Normalize Data?

Prevents Feature Domination: Features with large ranges won't overpower others. **Speeds Up Training**: Algorithms like gradient descent converge faster when data features are scaled similarly.

Improves Model Accuracy: Especially for models sensitive to feature magnitude differences.

Normalization of inputs

- Normalization of inputs means scaling and transforming the raw input data before feeding it into a deep learning model so that all features (variables) are on a similar scale or distribution.
- It's a key data preprocessing step used especially in neural networks and many other algorithms.

- When your dataset has multiple features (e.g., age, income, distance, etc.), those features might have **very different value ranges** — some large, some small.
- If you train a model directly on such data:
- The model might **learn slowly** or inefficiently
- Certain features may **dominate training** simply because of their scale
- Optimization (like gradient descent) can become **unstable**
- Activation functions (e.g., sigmoid, tanh) can **saturate** and slow down learning

Common Normalization Techniques

Min-Max Normalization

Scales values to a fixed range, typically 0 to 1:

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Z-Score Normalization (Standardization)

Centers the data to have mean 0 and unit variance (std = 1):

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma}$$

Batch Normalization

- Batch Normalization is a technique used inside neural networks to normalize the inputs (activations) of each layer during training so that they have a consistent distribution — typically zero mean and unit variance within each mini-batch.
- As neural networks train, the **distribution of activations changes** as weights are updated.
- This problem originally called internal covariate shift can slow training and make optimization more difficult.
- Batch Norm helps keep each layer's inputs stable across training steps, which:
- Makes training **faster** and more stable
- Allows use of **higher learning rates**
- Helps avoid issues like **vanishing/exploding gradients**

- Acts like a mild **regularizer** and improves generalization

Batch Normalization Works

For each mini-batch during training:

Compute Mean and Variance

Calculate the mean (average) and variance of the activations for each feature over the batch.

Normalize Activations

Each activation is normalized:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Here:

- μ = batch mean
- σ^2 = batch variance
- ε = small number to stabilize computation

Scale and Shift

- After normalization, two **learnable parameters** — **γ (gamma)** and **β (beta)** — are applied:

$$y = \gamma\hat{x} + \beta$$

- This lets the network **adjust the normalized output** to better fit what it needs to learn

Batch Normas regularizer

- Batch Normalization (BatchNorm) was originally introduced to stabilize and speed up training in deep neural networks by normalizing each layer's activations using statistics (mean and variance) computed from the current mini-batch.
- Beyond that main purpose, **BatchNorm also has a regularization effect** meaning it helps reduce overfitting and improve generalization, similarly to classic regularizers like dropout or weight decay.

Unit Highlights

- Proper **weight initialization** improves training stability and speed.
- **He/Kaiming initialization** is commonly used with ReLU-type activations.

- **Xavier initialization** is also known as Glorot initialization.
- **Mini-batch gradient descent** is the most common practical training approach.
- **L1 and L2 regularization** help reduce overfitting.
- **Dropout, early stopping, and data augmentation** are important regularization methods.
- **Batch normalization** stabilizes training and can also provide a regularization effect.

TEXT BOOKS:

1. *Amit Kumar Das, Saptarsi Goswami, Pabitra Mitra, Amlan Chakrabarti, "Deep Learning", Pearson Paperback, First Edition,2021.*

REFERENCE BOOKS:

1. *Josh Patterson and Adam Gibson, "Deep learning: Apractitioner's approach ",O' Reilly Media, First Edition,2017.*
2. *Nikhil Buduma, "Fundamentals of Deep Learning, Designing next generation machine intelligence algorithms",O'Reilly,ShroffPublishers,2019.*

S.No	Questions
UNIT III -TRAINING DEEP NEURAL NETWORK	
PART-A (Two Marks Questions)	
1.	he weight initialization.
2.	t is Xavier initialization?
3.	is He initialization used for ReLU?
4.	he Batch Gradient Descent.
5.	t is Mini-batch Gradient Descent?
6.	erentiate SGD and Batch GD.
7.	he L1 regularization.
8.	does L2 reduce overfitting?
9.	t is Dropout?
10.	t is Early stopping?
11.	he Data Augmentation.
12.	is Batch Normalization required?
PART-B (Ten Marks Questions)	
1.	ain weight initialization techniques (Zero, Xavier, He).
2.	pare Batch, Mini-batch and Stochastic Gradient Descent.
3.	ain L1 and L2 regularization with mathematical formulation.
4.	ribe Dropout regularization and its working.
5.	ain Early stopping and Data Augmentation techniques.
6.	ain Batch Normalization with steps and advantages.
7.	uss challenges in training deep neural networks.
8.	pare regularization techniques used in deep learning.