**DL :**

**UNIT 1: Introduction to Machine Learning and Deep Learning**

1. **What is Machine Learning and Deep Learning?**
   * **Machine Learning (ML):** Machine learning refers to algorithms that allow computers to learn from data and improve over time without being explicitly programmed. It involves supervised learning, unsupervised learning, and reinforcement learning.
   * **Deep Learning (DL):** A subfield of ML that involves neural networks with many layers (also known as deep neural networks). Deep learning excels in processing large amounts of data and is especially useful for tasks like image recognition, speech recognition, and natural language processing (NLP).
2. **Supervised and Unsupervised Learning:**
   * **Supervised Learning:** Involves training a model using labeled data. The algorithm learns to map inputs to the correct outputs, e.g., classification and regression tasks.
   * **Unsupervised Learning:** The model learns patterns from data without any labels. It looks for structures like clusters or associations, e.g., clustering and dimensionality reduction.
3. **Bias-Variance Tradeoff:**
   * The bias-variance tradeoff involves balancing **bias** (error due to overly simplistic models) and **variance** (error due to overly complex models).
   * A **high-bias** model underfits, while a **high-variance** model overfits.
4. **Hyperparameters:**
   * These are parameters set before training the model, such as learning rate, batch size, and the number of layers. Proper selection can significantly affect model performance.
5. **Underfitting and Overfitting:**
   * **Underfitting:** Occurs when the model is too simple and cannot capture the underlying patterns in the data.
   * **Overfitting:** Occurs when the model is too complex and fits the noise in the data instead of the true pattern.
6. **Regularization:**
   * Regularization techniques (like L2 regularization, dropout) are used to prevent overfitting by adding a penalty for large model weights or randomly dropping units during training to promote generalization.
7. **Limitations of Machine Learning:**
   * Machine learning can struggle with **data quality**, **interpretability**, and **computation cost**. It also requires large amounts of data to perform well.
8. **History of Deep Learning:**
   * Deep learning dates back to the 1950s with the **perceptron**, but it gained momentum in the 2000s with improvements in neural network architectures and the availability of more data and computational power.
9. **Advantages and Challenges of Deep Learning:**
   * **Advantages:** Deep learning can handle unstructured data like images, audio, and text. It also has the ability to learn hierarchical features.
   * **Challenges:** It requires large datasets, high computational resources, and can be difficult to interpret.
10. **Learning Representations from Data:**
    * Deep learning models learn hierarchical representations from raw data (e.g., pixel values in images or words in text) by learning through layers of transformations.
11. **Understanding How Deep Learning Works (in three figures):**
    * **Figure 1:** Input data passed through layers of a neural network, transforming into increasingly abstract representations.
    * **Figure 2:** Training process involving forward propagation (calculating output) and backward propagation (updating weights).
    * **Figure 3:** After sufficient training, the model can make predictions based on the learned representations.
12. **Common Architectural Principles of Deep Networks:**
    * **Feedforward architecture**: Information moves in one direction, from input to output.
    * **Recurrent networks**: Information cycles through loops, ideal for sequential data like time series.
13. **Architecture Design:**
    * **Depth:** How many layers the network has.
    * **Width:** Number of neurons per layer.
    * **Activation functions:** Used to introduce non-linearity.
    * **Regularization:** Techniques like dropout or L2 regularization to prevent overfitting.
14. **Applications of Deep Learning:**
    * **Computer vision:** Object recognition, face detection.
    * **Speech recognition:** Voice assistants like Siri or Alexa.
    * **Natural language processing (NLP):** Sentiment analysis, translation.
15. **Popular Industry Tools:**
    * **TensorFlow, Keras, PyTorch, Caffe, Shogun:** These are frameworks and libraries used for building, training, and deploying deep learning models. TensorFlow and PyTorch are the most popular for research and industry applications.

**UNIT 2: Introduction to Neural Networks**

1. **The Biological Neuron:**
   * The **biological neuron** is the inspiration for artificial neurons. It consists of dendrites (inputs), a cell body (sums inputs), and an axon (output).
2. **The Perceptron:**
   * The **perceptron** is a simple linear classifier, one of the earliest neural network models. It uses weighted sums of inputs, applies a threshold function, and outputs 0 or 1.
3. **Multilayer Feed-Forward Networks:**
   * These networks consist of multiple layers of neurons, where each layer's output is passed to the next. The feed-forward nature means there is no feedback loop.
4. **Training Neural Networks:**
   * **Backpropagation** is the method for training neural networks, where the network's error is propagated backward to adjust weights and minimize loss.
   * **Forward propagation** is the process of passing inputs through the network to get an output.
5. **Activation Functions:**
   * **Linear:** A simple activation function that outputs a weighted sum of inputs.
   * **Sigmoid:** S-shaped curve, outputs values between 0 and 1, often used for binary classification.
   * **Tanh (Hyperbolic Tangent):** Outputs values between -1 and 1, often used in hidden layers.
   * **Hard Tanh:** A variation of Tanh with hard boundaries.
   * **Softmax:** Converts logits to probabilities, used in multi-class classification.
   * **Rectified Linear Unit (ReLU):** Outputs zero for negative values, and the value itself for positive ones. Common in hidden layers for deep networks.
6. **Loss Functions:**
   * **Regression loss:** Mean squared error (MSE) is typically used.
   * **Classification loss:** Cross-entropy loss is used for classification tasks.
   * **Reconstruction loss:** Used for models like autoencoders.
7. **Hyperparameters:**
   * **Learning Rate:** Controls the size of weight updates.
   * **Momentum:** Helps speed up training and avoid local minima.
   * **Regularization:** Reduces overfitting by penalizing large weights.
   * **Sparsity:** Encourages the model to use fewer features.
8. **Deep Feedforward Networks – Example of XOR:**
   * The XOR problem is a classic example used to demonstrate the power of multi-layer neural networks, which can learn non-linear decision boundaries.
9. **Cost Functions, Error Backpropagation:**
   * **Cost function:** Measures the difference between predicted and true values.
   * **Backpropagation:** Calculates gradients and updates weights to minimize the cost function.
10. **Gradient-Based Learning:**
    * **Gradient Descent** is used to minimize the loss function. It updates weights by following the negative gradient of the loss with respect to the weights.
11. **Vanishing and Exploding Gradients:**
    * This refers to the problem where gradients become too small (vanishing) or too large (exploding), making training unstable.
12. **Deep Learning with PyTorch and Jupyter/Colab:**
    * **PyTorch** is a deep learning framework that allows dynamic computation graphs. **Jupyter notebooks** and **Google Colab** are often used for prototyping and experimentation.

**UNIT 3: Convolutional Neural Networks (CNNs)**

1. **CNN Architecture Overview:**
   * CNNs are designed for grid-like data, like images. They involve layers that perform convolutions, pooling, and fully connected layers.
2. **Basic Structure of a Convolutional Network:**
   * **Padding:** Adds extra pixels around the border of the image to control the size of the output.
   * **Strides:** Determines how much the filter moves when scanning the image.
   * **ReLU Layer:** Introduces non-linearity to the model.
   * **Pooling:** Reduces dimensionality by taking the max or average value from a patch of data.
   * **Fully Connected Layers:** After convolutions and pooling, the network typically ends with fully connected layers that output the final prediction.
3. **Training a Convolutional Network:**
   * Training involves adjusting the weights of the convolutional filters using backpropagation to minimize the loss function.

**UNIT 4: Recurrent Neural Networks (RNNs)**

1. **Recurrent Neural Networks (RNNs):**
   * RNNs are designed for sequential data. They have loops that allow information to be passed from one time step to the next, maintaining context.
2. **Bidirectional RNNs:**
   * These process the input sequence in both forward and reverse directions, capturing dependencies from both ends of the sequence.
3. **Encoder-Decoder Sequence-to-Sequence Architectures:**
   * Used for tasks like translation, where one network encodes the input sequence, and another decodes it to the output sequence.
4. **The Challenge of Long-Term Dependencies:**
   * RNNs struggle to remember information from earlier in long sequences. This issue is solved by architectures like **LSTM** and **GRU**.
5. **Long Short-Term Memory (LSTM):**
   * LSTM is an advanced RNN variant designed to remember long-term dependencies by using gates to control the flow of information.
6. **Optimization for Long-Term Dependencies:**
   * Techniques like **Gradient Clipping** and better weight initialization help in dealing with long-term dependencies.

**UNIT 5: Deep Generative Models**

1. **Boltzmann Machines:**
   * A type of stochastic neural network that can model the distribution of binary data. It’s the basis for the **Deep Belief Network**.
2. **Deep Belief Networks (DBNs):**
   * Composed of multiple layers of Boltzmann Machines, DBNs learn hierarchical features of data.
3. **Generative Adversarial Networks (GANs):**
   * **GANs** consist of two networks: the **Generator**, which creates fake data, and the **Discriminator**, which tries to distinguish between real and fake data. Both networks are trained simultaneously to improve each other.
4. **Applications of GANs:**
   * Image generation, style transfer, video prediction, etc.

**UNIT 6: Deep Reinforcement Learning**

1. **Markov Decision Process (MDP):**
   * A mathematical framework for modeling decision-making, consisting of states, actions, rewards, and a transition model.
2. **Challenges of Reinforcement Learning:**
   * **Exploration vs Exploitation:** Bal

**HPC :**

**UNIT 1: Introduction to Parallel Computing**

This unit introduces parallel computing, discussing its necessity, platforms, and models.

1. **Motivating Parallelism:**
   * Parallel computing involves breaking down tasks into smaller subtasks that can run simultaneously on multiple processors or cores, improving computational efficiency.
   * Parallelism is essential for high-performance computing applications, especially as the demand for faster computation grows (e.g., simulations, real-time data processing, AI).
2. **Modern Processor:**
   * **Stored-program computer architecture:** This is a fundamental design in modern computing where the program (software) and data are stored in the same memory, allowing the CPU to fetch and execute instructions from memory.
   * **General-purpose Cache-based Microprocessor Architecture:** Modern processors use caches (small, fast memory) to reduce the time it takes to access frequently used data, enhancing overall CPU performance. This cache-based architecture allows the CPU to execute instructions faster by reducing access time to memory.
3. **Parallel Programming Platforms:**
   * **Implicit Parallelism:** This refers to systems that automatically handle parallel execution. The hardware or software framework (like a GPU or multi-core CPU) takes care of parallel execution without the programmer needing to explicitly define it.
   * **Dichotomy of Parallel Computing Platforms:** Parallel platforms can generally be categorized into:
     + **Shared-memory systems** (all processors have access to a common memory) and
     + **Distributed-memory systems** (each processor has its own memory and communicates through a network).
   * **Physical Organization of Parallel Platforms:** The physical arrangement of processors and memory in a parallel system affects performance. For instance, multi-core processors have several cores on a single chip, while distributed systems may have processors spread across different machines.
   * **Communication Costs in Parallel Machines:** In parallel systems, data needs to be communicated between processors, which incurs costs in terms of time (latency) and bandwidth. Optimizing communication is crucial for performance.
4. **Levels of Parallelism:**
   * **SIMD (Single Instruction, Multiple Data):** In SIMD, a single instruction is executed on multiple data elements in parallel. For example, vector processors or GPUs use SIMD to perform operations like addition on arrays of numbers simultaneously.
   * **MIMD (Multiple Instruction, Multiple Data):** MIMD allows different processors to execute different instructions on different data. This model is more flexible and is commonly used in large parallel systems like supercomputers.
   * **SIMT (Single Instruction, Multiple Threads):** A form of SIMD where multiple threads execute the same instruction on different pieces of data. This is often used in GPU architectures, where each thread in a warp performs the same operation on different data.
   * **SPMD (Single Program, Multiple Data):** In SPMD, all processors execute the same program but work on different parts of the data. This is common in distributed memory systems.
   * **Data Flow Models:** These models focus on executing computations when data is available, rather than strictly following a set order of operations.
   * **Demand-driven Computation:** In contrast to traditional execution models, demand-driven computation processes tasks based on data requirements rather than a strict execution schedule.
5. **Architectures:**
   * **N-wide superscalar architectures:** These allow multiple instructions to be executed per cycle by having multiple execution units (e.g., integer, floating-point) on the processor.
   * **Multi-core:** A multi-core processor contains multiple processing units (cores) on the same chip, enabling concurrent execution of tasks.
   * **Multi-threaded:** A multi-threaded processor can handle multiple threads (smaller units of a program) concurrently, improving performance for certain types of tasks (e.g., web servers, real-time processing).

**UNIT 2: Principles of Parallel Algorithm Design**

This unit discusses how to design algorithms that can effectively utilize parallel computing.

1. **Preliminaries:**
   * This covers the basic concepts of parallel algorithm design: identifying tasks, dividing the workload, synchronizing tasks, and ensuring minimal communication overhead.
2. **Decomposition Techniques:**
   * **Task Decomposition:** Splitting a problem into independent tasks that can be executed simultaneously.
   * **Data Decomposition:** Dividing data into smaller chunks and assigning each chunk to a different processor for processing.
   * **Recursive Decomposition:** Problems are broken down recursively until the subproblems are small enough to be solved concurrently.
3. **Characteristics of Tasks and Interactions:**
   * **Task Dependency:** Some tasks depend on the results of others, requiring synchronization mechanisms like barriers or locks.
   * **Interaction Overheads:** Communication between tasks can introduce overheads. Minimizing the number of interactions between tasks improves performance.
4. **Mapping Techniques for Load Balancing:**
   * Efficiently distributing tasks across available processors ensures that no processor is idle while others are overloaded.
   * Techniques like **static mapping** (pre-defined task assignments) and **dynamic mapping** (tasks assigned during execution based on load) are common approaches.
5. **Parallel Algorithm Models:**
   * **Data Parallelism:** Focuses on performing the same operation on different data elements in parallel.
   * **Task Parallelism:** Involves executing different tasks simultaneously, typically used in divide-and-conquer algorithms.
   * **Work Pool and Master-Slave Model:** In the work pool model, tasks are assigned from a central pool to workers. In the master-slave model, one master processor controls the execution of tasks on slave processors.
6. **Complexities:**
   * **Sequential vs Parallel Complexity:** The complexity of an algorithm can change when parallelized. For example, parallel algorithms may reduce the time complexity (e.g., from O(n2)O(n^2)O(n2) to O(n)O(n)O(n)), but may introduce communication overhead.
   * **Anomalies in Parallel Algorithms:** Sometimes parallel algorithms do not perform as expected due to issues like load imbalances, excessive synchronization, or inefficient memory access patterns.

**UNIT 3: Basic Communication**

In parallel systems, efficient communication between processors is critical for achieving good performance.

1. **Communication Operations:**
   * **One-to-All Broadcast:** A single processor sends data to all other processors in a system. Used for distributing shared information.
   * **All-to-One Reduction:** Multiple processors send data to a single processor, which aggregates or reduces the data (e.g., summing the values).
   * **All-to-All Broadcast and Reduction:** Data is exchanged among all processors in a system. This is useful in problems where each processor needs information from every other processor.
   * **All-Reduce and Prefix-Sum Operations:** These operations combine data across processors and redistribute it. **All-Reduce** is often used in parallel applications like machine learning for aggregating gradients.
2. **MPI Communication:**
   * **Scatter, Gather, Broadcast:** MPI provides efficient methods for distributing data among processes (scatter), collecting data from processes (gather), and distributing the same data to all processes (broadcast).
   * **Blocking and Non-blocking MPI:** In **blocking operations**, a process waits for the operation to complete before proceeding. In **non-blocking operations**, processes can continue working while waiting for the communication to finish.
   * **All-to-All Personalized Communication:** Each processor sends a unique piece of data to every other processor.
   * **Circular Shift:** Data is shifted cyclically between processors in a circular manner (used in problems like matrix rotation).

**UNIT 4: Sources of Overhead in Parallel Programs**

Overhead in parallel programs is a major factor affecting performance, and understanding it is essential.

1. **Performance Measures and Analysis:**
   * **Amdahl’s Law:** It predicts the theoretical speedup of a program when only part of it is parallelized. The formula is:

S=1(1−P)+PNS = \frac{1}{(1 - P) + \frac{P}{N}}S=(1−P)+NP​1​

where SSS is the speedup, PPP is the parallel portion of the program, and NNN is the number of processors.

* + **Gustafson’s Law:** Focuses on the scalability of parallel programs. It suggests that as the problem size increases, the performance of a parallel system increases more than predicted by Amdahl’s Law.
  + **Speedup Factor and Efficiency:** Speedup measures how much faster the parallel program runs compared to the sequential one. Efficiency measures how well the processors are utilized.

1. **Scalability and Asymptotic Analysis:**
   * **Granularity:** Refers to the size of tasks that are executed in parallel. Fine-grained tasks may lead to higher communication overhead, while coarse-grained tasks may underutilize processors.
   * **Scalability:** The ability of a system to handle larger problem sizes or more processors without significant performance degradation.
   * **Asymptotic Analysis:** It evaluates the performance of an algorithm in terms of its growth as the input size increases, crucial for understanding parallel program performance.
2. **Matrix Computation:**
   * **Matrix-Vector and Matrix-Matrix Multiplication:** Parallel algorithms are commonly used in matrix operations, as they can be decomposed into smaller tasks that can run concurrently on different processors or threads.

**UNIT 5: Introduction to GPU**

This unit introduces GPU architecture and CUDA programming, which are critical for parallel computing tasks like AI and deep learning.

1. **GPU Architecture Overview:**
   * GPUs consist of many smaller processing units (cores) that can execute simple operations on large sets of data simultaneously. They are ideal for tasks requiring high throughput, like graphics rendering, simulations, and machine learning.
2. **CUDA Programming Model:**
   * **CUDA C Programming Model:** CUDA allows the user to write programs that can run on the GPU. It uses a **kernel** (a function executed on the GPU) and threads (units of execution). CUDA supports features like memory management, synchronization, and error handling.
   * **Memory Model:** CUDA provides different types of memory such as global, shared, and local memory. Efficient memory usage is crucial for high-performance computing.

**UNIT 6: Scope of Parallel Computing**

This unit highlights practical applications of parallel computing, especially in AI and machine learning.

1. **Parallel Search Algorithms:**
   * **DFS and BFS:** Depth-first and breadth-first search algorithms are widely used in graph theory. They can be parallelized by dividing the search space among processors.
2. **Parallel Sorting:**
   * Sorting algorithms like **Bubble Sort** and **Merge Sort** can be parallelized. Parallel sorting helps in processing large datasets faster.
3. **Distributed Computing and Frameworks:**
   * **Kubernetes:** A platform for automating containerized applications, often used for managing large-scale distributed computing environments.
   * **GPU Applications in AI/ML:** GPUs are essential for training deep learning models due to their ability to handle large-scale matrix operations in parallel.