DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
JORHAT ENGINEERING COLLEGE

PROJECT REPORT

SUBMITTED IN COMPLETION FOR ACADEMIC INTERNSHIP BY

BHIKRANT BORAH, CHINMOY SHARMA, AKASH SARMAH

INTERNSHIP CARRIED OUT AT:

PROJECT SUPERVISED BY:

MOUSOOMI BORA,ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE ASSAM KAZIRANGA UNIVERSITY

KAZIRANGA UNIVERSITY

KNOWLEDGE & BEYOND

INTERNSHIP COMPLETION CERTIFICATE

This is to certify that

Akash Sarmah (Roll. 220710007009), a student of Jorhat Engineering College successfully completed the summer internship at Department of Computer Science and Engineering from 15th June, 2024 to 15th July, 2024.

During the internship period, Akash has worked on the project entitled **Performance optimization of LSTM using Meta heuristic algorithm**. All his work has been the highest standards and he has demonstrated excellent skills in Deep learning area. Akash has been always dedicated to the work allotted and has adhered to the deadline. The work and contributions made during the internship have been commendable and of significant value to the department.

We wish him all the success in future endeavours.

Mousoomi Bora

Mousoomi Bora

Assistant Professor,

Department of Computer Science & Engineering

The Assam Kaziranga University, Koraikhowa, NH 37, Assam

KAZIRANGA UNIVERSITY

KNOWLEDGE & BEYOND

INTERNSHIP COMPLETION CERTIFICATE

This is to certify that

Chinmoy Sharma (Roll. 220710007021), a student of Jorhat Engineering College successfully completed the summer internship at Department of Computer Science and Engineering from 15th June, 2024 to 15th July, 2024.

During the internship period, Chinmoy has worked on the project entitled **Performance optimization of LSTM using Meta heuristic algorithm**. All his work has been the highest standards and he has demonstrated excellent skills in Deep learning area. Chinmoy has been always dedicated to the work allotted and has adhered to the deadline. The work and contributions made during the internship have been commendable and of significant value to the department.

We wish him all the success in future endeavours.

Mousoomi Bora

Mousoomi Bora

Assistant Professor,

Department of Computer Science & Engineering

The Assam Kaziranga University, Koraikhowa, NH 37, Assam

KAZIRANGA UNIVERSITY

KNOWLEDGE & BEYOND

INTERNSHIP COMPLETION CERTIFICATE

This is to certify that

Bhikrant Borah (Roll. 220710007018), a student of Jorhat Engineering College successfully completed the summer internship at Department of Computer Science and Engineering from 15th June, 2024 to 15th July, 2024.

During the internship period, Bhikrant has worked on the project entitled **Performance optimization of LSTM using Meta heuristic algorithm**. All his work has been the highest standards and he has demonstrated excellent skills in Deep learning area. Bhikrant has been always dedicated to the work allotted and has adhered to the deadline. The work and contributions made during the internship have been commendable and of significant value to the department.

We wish him all the success in future endeavours.

Mousoomi Bora

Mousoomi Bora

Assistant Professor,

Department of Computer Science & Engineering

The Assam Kaziranga University, Koraikhowa, NH 37, Assam

ABSTRACT

In recent years, Machine Learning (ML) and Deep Learning (DL) have transformed industries through data-driven insights and automation. Within this landscape, Long Short-Term Memory (LSTM) networks, a deep learning technique, have shown particular promise for educational applications due to their ability to analyze sequential data. This project utilizes LSTM networks to predict cumulative CGPA and campus placement outcomes, providing educators with actionable insights into student performance trajectories. Additionally, meta-heuristic optimization, specifically Genetic Algorithms, is employed to fine-tune the LSTM model's hyperparameters, enhancing predictive accuracy. This approach underscores the potential of deep learning and advanced optimization to drive impactful educational forecasting and support academic success.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION 08

1.1.1. A TASTE OF MACHINE LEARNING.
1.1.2. RELATION TO DATA MINING.
1.1.3. RELATION TO OPTIMIZATION.
1.1.4. DEEP LEARNING.
1.1.5. METAHEURISTIC ALGORITHM

CHAPTER 2: TECHNOLOGY LEARNT 10

2.1. INTRODUCTION TO MACHINE LEARNING
2.1.1. HISTORY OF AI.
2.1.2. WHAT IS AI?
2.1.3. WHAT IS ML?
2.1.4. RELATIONSHIP BETWEEN AI AND ML.
2.1.5. MACHINE LEARNING APPROACH.

2.2. SUPERVISED LEARNING
2.2.1. DEFINE SUPERVISED LEARNING
2.2.2. SUPERVISED LEARNING FLOW

2.3. UNSUPERVISED LEARNING
2.2.1. DEFINE UNSUPERVISED LEARNING
2.2.2. TYPES OF UNSUPERVISED LEARNING

2.4. REINFORCEMENT LEARNING
2.4.1. DEFINE REINFORCEMENT LEARNING
2.4.2. FEATURES OF REINFORCEMENT LEARNING

2.5. BASIC FUNDAMENTALS OF MACHINE LEARNING
2.5.1. DATA PREPROCESSING
2.5.2. ALGORITHMS
2.5.3. TRAINING AND TESTING
2.5.4. PERFORMANCE METRICS
2.5.5. OVERFITTING AND UNDERFITTING

2.6. WHAT IS DL?
2.6.1. KEY CHARACTERISTICS

2.7. FUNDAMENTALS OF DEEP LEARNING
2.7.1. NEURAL NETWORKS
2.7.2. LAYERS
2.7.3. WEIGHT AND BIAS
2.7.4. ACTIVATION FUNCTIONS
2.7.5. LOSS FUNCTIONS
2.7.6. TRAINING PROCESS
2.7.6.1. BACKWARD PROPAGATION
2.7.6.1.1. FORWARD PROPAGATION
2.7.6.1.2. GRADIENT DESCENT
2.7.6.1.3. LEARNING RATE
2.7.6.1.4. OPTIMIZERS

2.7.7. OVERFITTING AND REGULARIZATION
2.7.7.1. FIXING OVERFITTING
2.7.7.1.1. EARLY STOPPING
2.7.7.1.2. REGULARIZATION(L2 REGULARIZATION AND DROPOUTS)
2.7.7.1.3. BATCH NORMALIZATION

2.8. INTRODUCTION TO LSTM NETWORKS
2.8.1. ARCHITECTURE OF LSTM MODEL
2.8.2. TRAINING LSTM NETWORKS
2.8.3. EVALUATION METRICS FOR LSTM MODELS
2.8.4. VARIATIONS AND ENHANCEMENTS OF LSTM
2.8.5. SUMMARY OF OUR MODEL
2.9. INTRODUCTION TO METAHEURISTIC ALGORITHM
2.9.1. WHAT IS METAHEURISTIC ALGORITHM?
2.9.2. WHY USE METAHEURISTIC ALGORITHMS?
2.10. GENETIC ALGORITHMS (GA)
2.10.1. WHAT IS GA AND IT'S KEY CHARACTERISTICS
2.10.2. KEY CONCEPTS OF GENETIC ALGORITHMS
2.10.3. GENETIC ALGORITHM STEPS (WITH REFERENCE THROUGH OUR CODE)
2.10.4. HYPER PARAMETER TUNING VS GENETIC ALGORITHM

CHAPTER 3: LIBRARIES 35
3.1. NUMPY (NUMERICAL PYTHON)
3.2. TENSORFLOW
3.3. KERAS
3.4. PANDAS
3.5. MATPLOTLIB
3.6. SCIKIT-LEARN
3.7. DEAP (DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON)

CHAPTER 4: DATASET 38
4.1. CSV FILE
4.2. ALL THE COLUMNS IN OUR DATASET

CHAPTER 5: METHODOLOGY 40
5.1. DATA PREPROCESSING
5.2. DATA RESHAPING
5.3. FUNCTION FOR LSTM MODELS OF PLACEMENT AND CGPA
5.4. FITNESS FUNCTION FOR PLACEMENT AND CGPA
5.5. GA COMPONENTS AND CUSTOM MUTATION FUNCTION
5.6. GENETIC ALGORITHM FOR CGPA MODEL AND PLACEMENT MODEL
5.7. CGPA MODEL AND PLACEMENT MODEL WITH THE BEST HYPERPARAMETERS
5.8. PLOTTING ROC CURVE AND PERFORMANCE METRICS FOR PLACEMENT MODEL
5.9. TESTING OF THE MODELS

CHAPTER 6: RESULTS AND DISCUSSION 47
6.1. CGPA PREDICTION
6.2. PLACEMENT PREDICTION
6.3. GENERAL OBSERVATIONS

CHAPTER 7: TESTING 50
7.1. TEST CASES

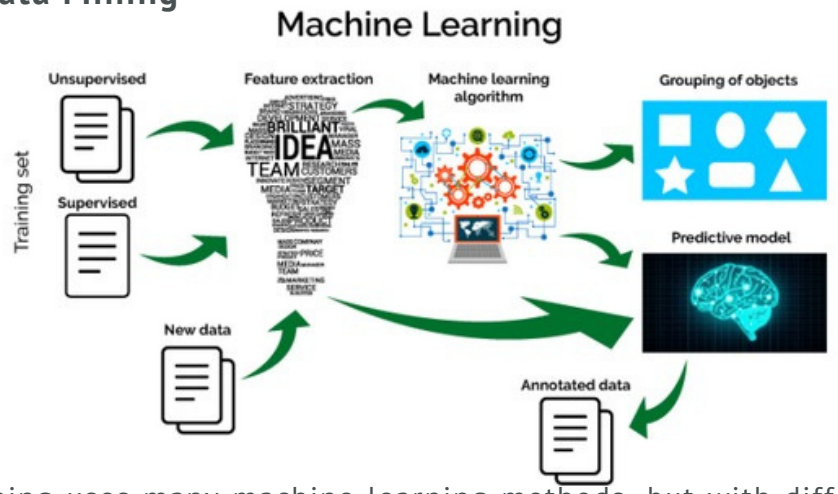
CHAPTER 8: CONCLUSION AND FUTURE DIRECTIONS 51

INTRODUCTION

A Taste of Machine Learning:

- Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959. Over the past
- two decades Machine Learning has become one of the mainstays of information technology. With the ever-increasing amounts of data becoming available there
- is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress.

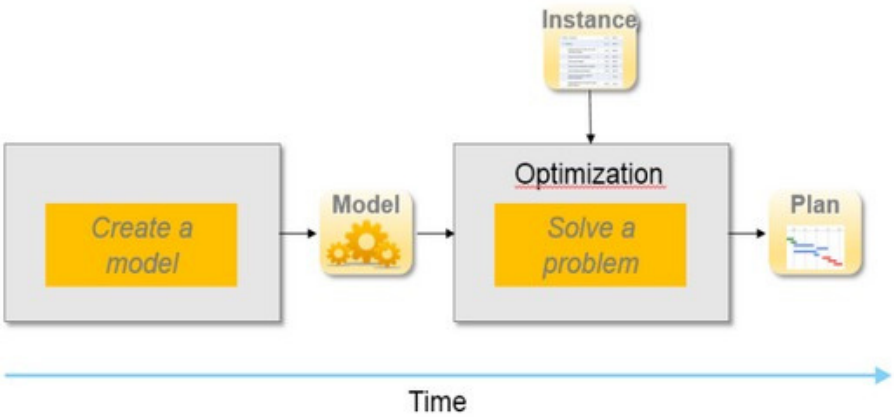
Relation to Data Mining



- Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy.

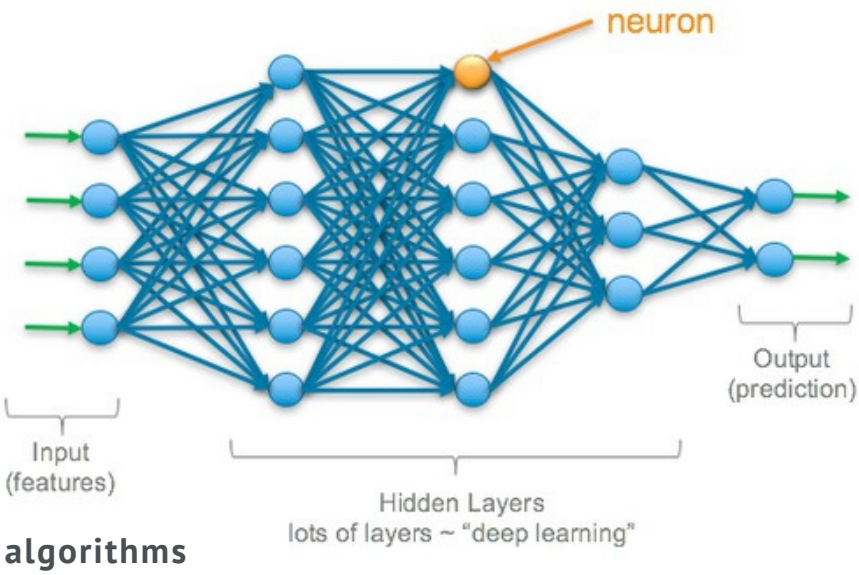
Relation to Optimization

- Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples.
- Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances.



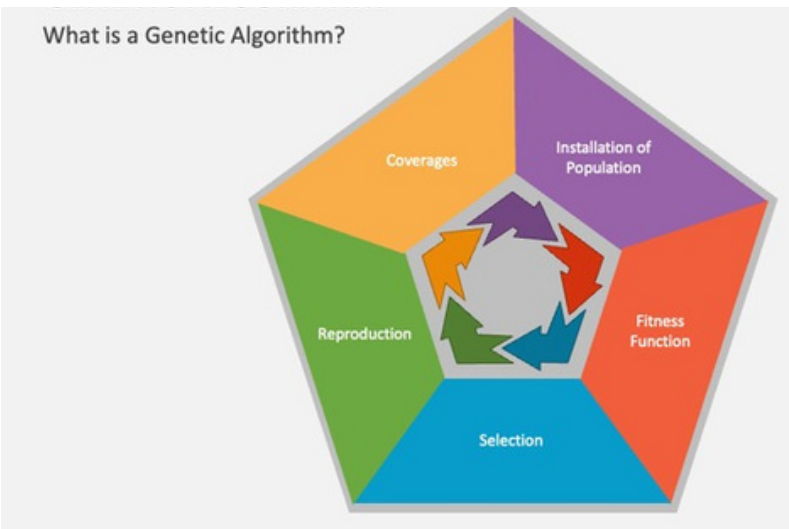
Deep Learning

- Deep learning models learn hierarchical representations of data, making them powerful for tasks that require understanding intricate patterns and features. Their performance improves significantly with more data and computational power. The 1980s saw a shift towards machine learning, particularly with the
- introduction of backpropagation for neural networks. The field experienced a resurgence in the 2010s due to deep learning breakthroughs, exemplified by AlexNet's success in image recognition and AlphaGo's victory over a Go champion.



Metaheuristic algorithms

- Metaheuristic algorithms are high-level optimization methods that emerged in the late 20th century, inspired by natural and biological processes. They were developed as flexible, problem-independent approaches to tackle complex, often NP-hard optimization problems that traditional algorithms struggled with due to computational limits. Early examples include genetic algorithms (inspired by Darwinian evolution, developed by John Holland in the 1970s) and simulated annealing (inspired by metallurgical annealing, introduced in the 1980s). These algorithms are "heuristic" in nature—focused on exploration and exploitation of the solution space—using iterative, probabilistic techniques to move toward optimal or near-optimal solutions. Today, metaheuristics like ant colony optimization, and tabu search are widely applied in fields ranging from machine learning to logistics, where finding precise solutions is challenging due to the vastness or complexity of the problem space.



TECHNOLOGY LEARNT

INTRODUCTION TO MACHINE LEARNING

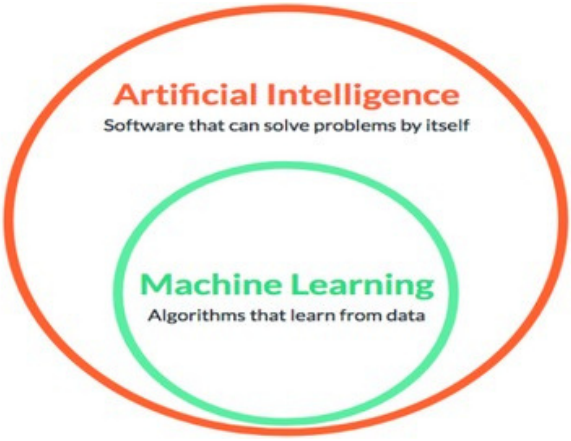
History of AI : The history of Artificial Intelligence (AI) began in the 1950s with the Dartmouth Conference, where the term "AI" was coined. Early developments included symbolic reasoning and natural language processing. The 1980s saw a shift towards machine learning, particularly with the introduction of backpropagation for neural networks. The field experienced a resurgence in the 2010s due to deep learning breakthroughs, exemplified by AlexNet's success in image recognition and AlphaGo's victory over a Go champion. Today, AI continues to evolve rapidly, influencing various sectors while raising important ethical considerations.

What is AI? Artificial Intelligence (AI) is referred to as simulation of human intelligence processes by computer systems. These processes include learning (the acquisition of information and rules for using it), reasoning (using rules to reach approximate or definite conclusions), and self-correction. AI systems are designed to perform tasks that typically require human intelligence, such as understanding natural language, recognizing patterns, solving problems, and making decisions.AI can automate repetitive tasks,AI systems can adapt to new inputs and learn from experiences.

What is ML? Machine Learning (ML) is a transformative field within artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform specific tasks without explicit programming. Instead of being instructed on how to perform a task, ML systems learn from data, identify patterns, and make decisions based on their experiences.

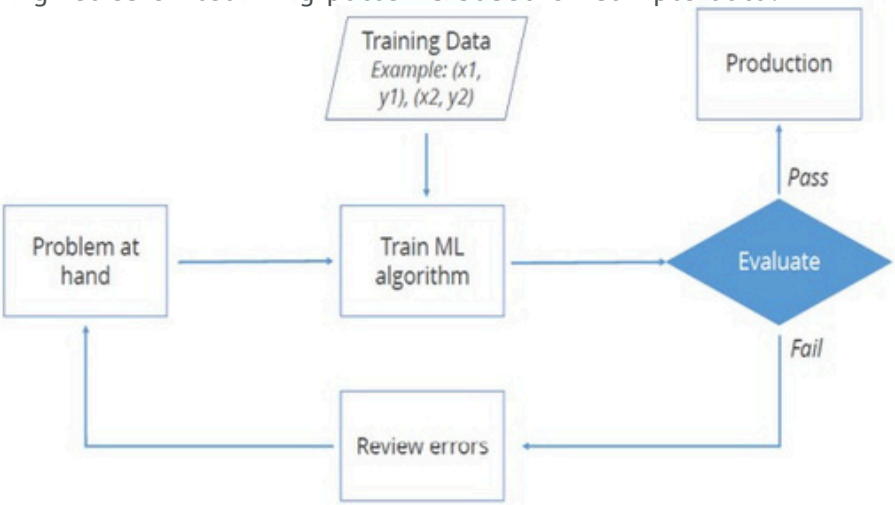
Relationship between AI and ML

Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it.



Machine Learning Approach

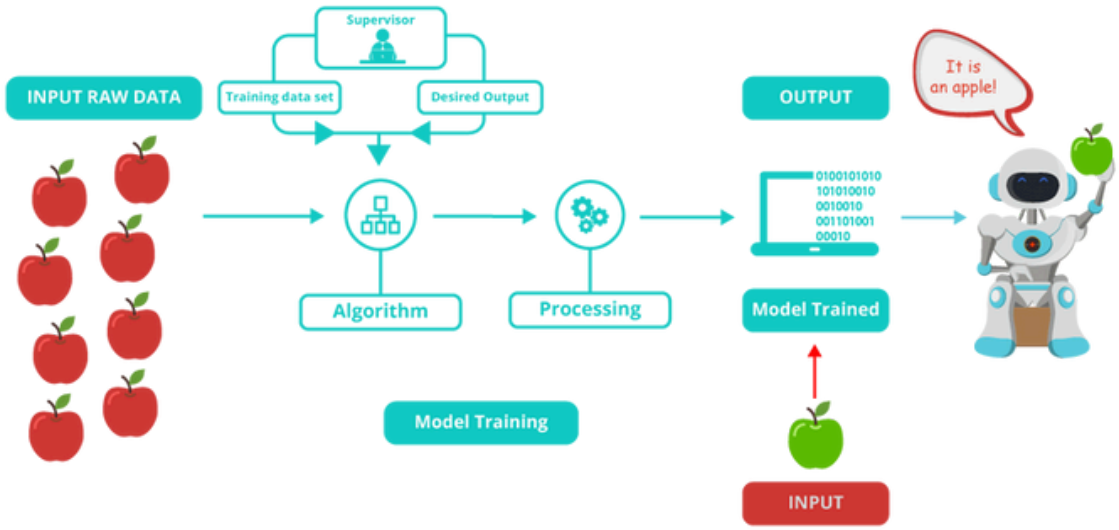
Machine Learning relies on learning patterns based on sample data.



Supervised Learning

Define Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.



In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

❖ Supervised Learning Flow

√ Data Preparation

- Clean data
- Label data (x, y)
- Feature Engineering
- Reserve 80% of data for Training (Train_X) and 20% for Evaluation (Train_E)

✓ **Training Step**

- Design algorithmic logic
- Train the model with Train X
- Derive the relationship between x and y, that is, y = f(x)

✓ **Evaluation or Test Step**

- Evaluate or test with Train E
- If accuracy score is high, you have the final learned algorithm y = f(x)
- If accuracy score is low, go back to training step

✓ **Production Deployment**

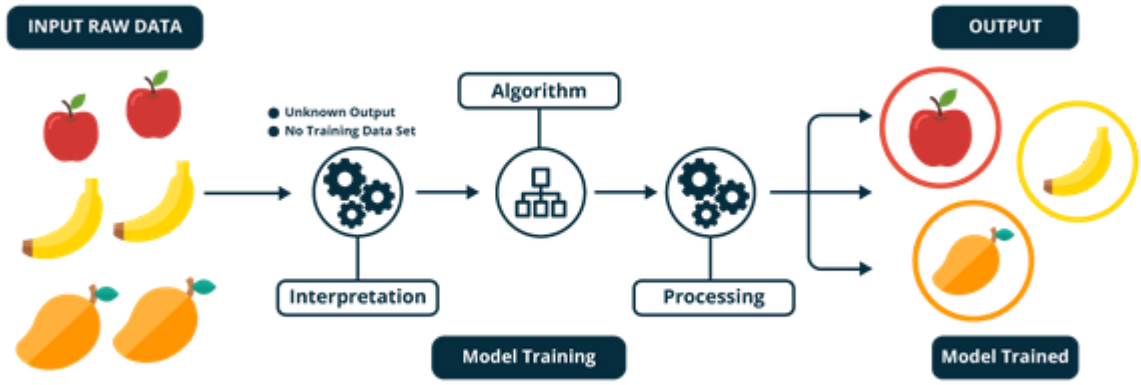
- Use the learned algorithm y = f(x) to predict production data.

The algorithm can be improved by more training data, capacity, or algo redesign.

Unsupervised Learning

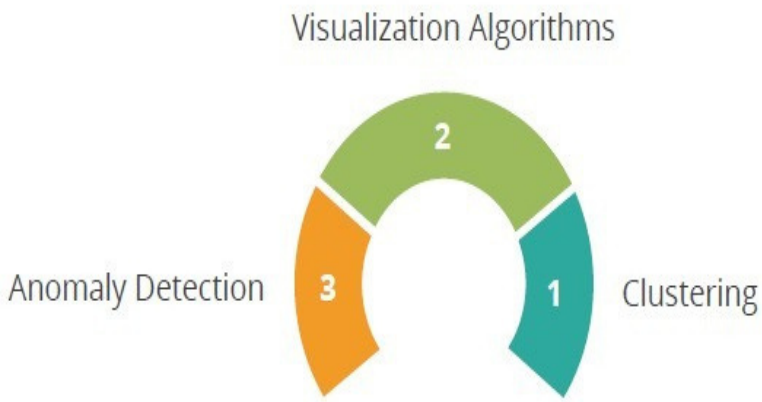
Define Unsupervised Learning

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.



Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Types of Unsupervised Learning

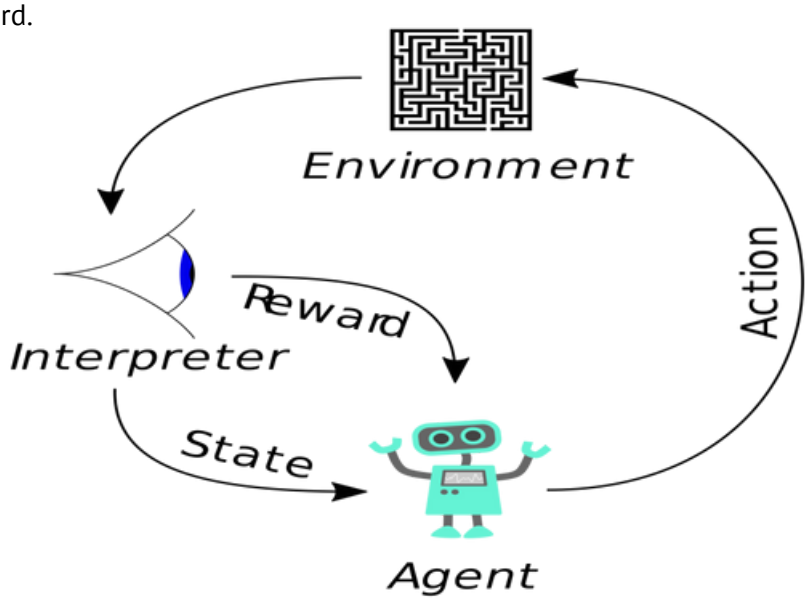


Clustering

The most common unsupervised learning method is cluster analysis. It is used to find data clusters so that each cluster has the most closely matched data.

Reinforcement Learning

Define Reinforcement Learning Reinforcement Learning is a type of Machine Learning that allows the learning system to
observe the environment and learn the ideal behavior based on trying to maximize some notion of cumulative reward.



It differs from supervised learning in that labelled input/output pairs need not be presented, and sub-optimal actions need not be explicitly corrected. Instead the focus is finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge)

- Features of Reinforcement Learning
  - The learning system (agent) observes the environment, selects and takes certain actions, and gets rewards in return (or penalties in certain cases). The agent learns
  - the strategy or policy (choice of actions) that maximizes its rewards over time.

In a manufacturing unit, a robot uses deep reinforcement learning to identify a device from one box and put it in a container.
The robot learns this by means of a rewards-based learning system, which incentivizes it for the right action.

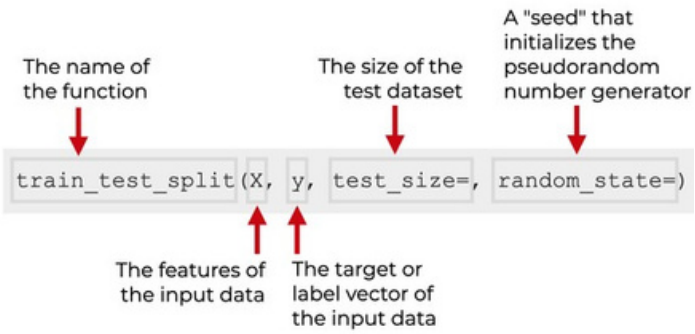
Basic Fundamentals of Machine Learning

The basic fundamentals of machine learning:

1.Data Preprocessing:

Data Preparation: The foundation of any ML model is data. This can be structured (like tables in databases) or unstructured (like images, text, or audio).Machine Learning depends largely on test data. Data preparation involves data selection, filtering, transformation, etc.

Train-Test Split: used to divide a dataset into two subsets(train and test data)



Types of data:

a.**Labeled Data**: Data that has been annotated with labels or categories. Each example in the dataset has a corresponding output. Used for supervised learning.

b.**Unlabeled Data**: Data that does not have any labels or annotations. It consists only of input features without corresponding outputs. Used for unsupervised learning and tasks like clustering or dimensionality reduction. c.**Training Data**: A subset of the dataset used to train the machine learning model. It contains both features and labels (in supervised learning) and is used to adjust the model's parameters.

d.**Validation Data**:A subset of data used to tune the model's hyperparameters and evaluate its performance during training. It helps prevent overfitting by providing a checkpoint for model evaluation before testing.

e.**Test Data**:A separate subset of data used to evaluate the final performance of a trained model. It is not used during the training or validation phases and provides an unbiased estimate of the model's generalization ability.

Features: These are individual measurable properties or characteristics used as input variables for the model. Selecting the right features is crucial for model performance.

The transformation stage in the data preparation process includes an important step known as **Feature Engineering**

Feature Engineering techniques:

- **Numerical data:**
  - **Feature Scaling**

✓ Feature scaling is an important step in the data transformation stage of data preparation process.

✓ Feature Scaling is a method used in Machine Learning for standardization of independent variables of data features

Techniques of Feature Scaling:

- **Standardization**

Standardization is a popular feature scaling method, which gives data the property of a standard normal distribution (also known as Gaussian distribution).

All features are standardized on the normal distribution (a mathematical model).

The **mean** of each feature is centered at **0** and the feature column has a **standard deviation** of **1**

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

The ML library scikit-learn implements a class for standardization called StandardScaler, as demonstrated here:

```
>>> from sklearn.preprocessing import StandardScaler
>>> stdsc = StandardScaler()
>>> X_train_std = stdsc.fit_transform(X_train)
>>> X_test_std = stdsc.transform(X_test)
```

- **Normalization**

In most cases, normalization refers to rescaling of data features between 0 and 1, which is a special case of Min-Max scaling.

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}}$$

In the given equation, subtract the min value for each feature from each feature instance and divide by the spread between max and min.
In effect, it measures the relative percentage of distance of each instance from the min value for that feature.

The ML library scikit-learn has a MinMaxScaler class for normalization.

It implements normalization as explained on the previous slide.

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> mms = MinMaxScaler()
>>> X_train_norm = mms.fit_transform(X_train)
>>> X_test_norm = mms.transform(X_test)
```

- **Categorical data:**
  - **Label Encoding:**
    - Converts each category into a unique integer.
    - Suitable for ordinal categorical data (where the order matters).

```
# 1. Label Encoding
label_encoder = LabelEncoder()
df['Color_LabelEncoded'] = label_encoder.fit_transform(df['Color'])
```

  - **One-Hot Encoding:**
    - Creates binary columns for each category, where only one column is set to 1 (hot) for a given instance, and the others are set to 0.
    - Suitable for nominal categorical data (where order does not matter).

```
# 2. One-Hot Encoding
df_one_hot = pd.get_dummies(df['Color'], prefix='Color')
df = pd.concat([df, df_one_hot], axis=1)
```

  - **Ordinal Encoding:**
    - Similar to label encoding but specifically for ordinal data, where categories have a meaningful order.

```
# 6. Ordinal Encoding (specify order)
ordinal_encoder = OrdinalEncoder(categories=[['Blue', 'Green', 'Red']])  # Define the order
df['Color_OrdinalEncoded'] = ordinal_encoder.fit_transform(df[['Color']])
```

- **Binary Encoding:**
  - Converts categories into binary code and creates new columns based on the binary representation.
  - Useful for high cardinality categorical features.

```
# 5. Binary Encoding
binary_encoder = ce.BinaryEncoder(cols=['Color'])
df_binary = binary_encoder.fit_transform(df['Color'])
df = pd.concat([df, df_binary], axis=1)
```

- **Target Encoding:**
  - Replaces categories with the mean of the target variable for each category.
  - Helps in capturing the relationship between the category and the target but can lead to overfitting.

```
# 4. Target Encoding
target_mean = df.groupby('Color')['Target'].mean()
df['Color_TargetEncoded'] = df['Color'].map(target_mean)
```

2.Algorithms:

Machine learning algorithms are the mathematical models that process the input data to learn patterns. Common types include:

1. **Linear Regression:** Used for predicting continuous values.
2. **Logistic Regression:** Used for binary classification tasks.
3. **Decision Trees:** A flowchart-like structure used for classification and regression.
4. **Support Vector Machines (SVM):** Used for classification tasks by finding the hyperplane that best separates data points of different classes.
5. **Neural Networks:** Inspired by the human brain, these are used for complex tasks such as image and speech recognition.

REGRESSION :

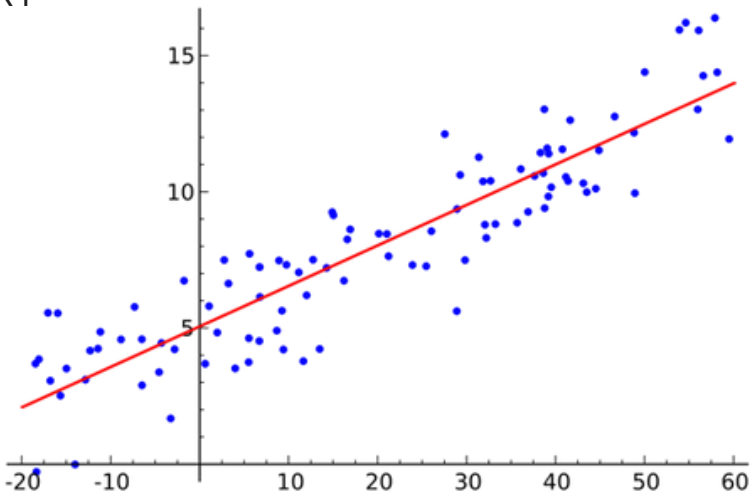
In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships among variables.It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors')

- **Linear Regression(linear model):** It is a linear approach for modeling the relationship between a scalar dependent variable y and an independent variable x.
  - The equation is written as:

$$y = wx + b$$

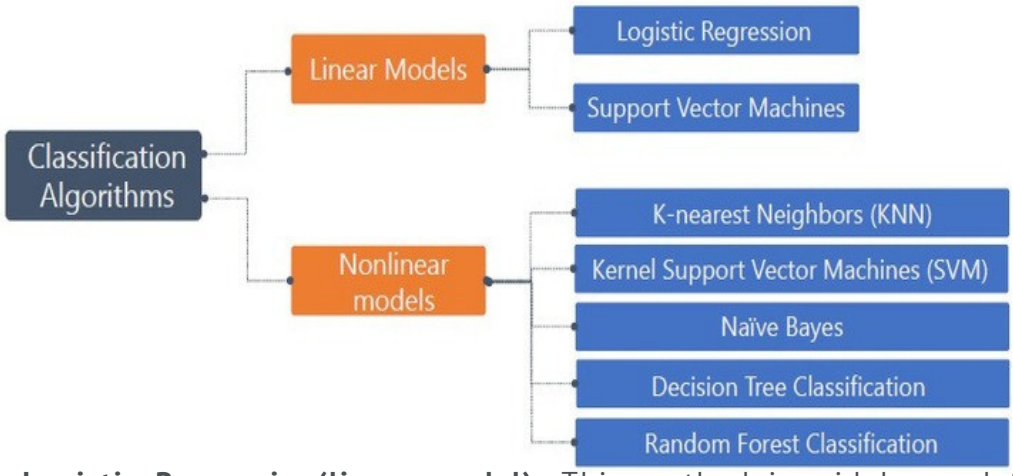
where x, y, w are vectors of real numbers and w is a vector of weight parameters.

where b is the bias or the value of output for zero input



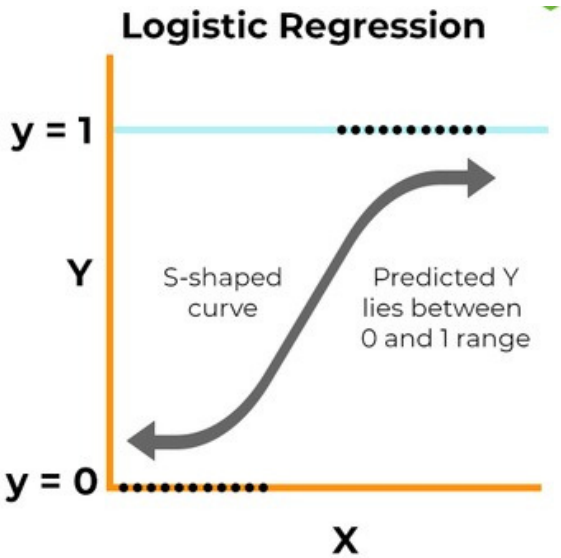
CLASSIFICATION:

- It specifies the class to which data elements belong to.
- It predicts a class for an input variable.
- It is best used when the output has finite and discreet values.



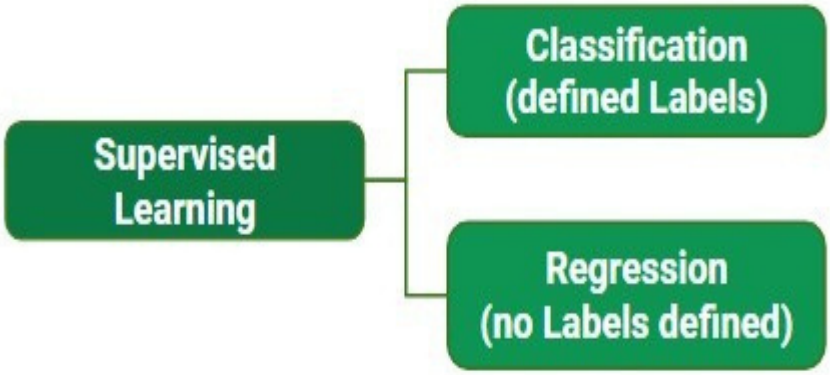
- **Logistic Regression(linear model):** This method is widely used for binary classification problems(2 output classes). It can also be extended to multi-class classification problems(>2 output classes).

A binary dependent variable can have only two values, like 0 or 1, win or lose, pass or fail, healthy or sick, etc.



The probability in the logistic regression is often represented by the Sigmoid function (also called the logistic function or the S-curve)

$$S(t) = \frac{1}{1 + e^{-t}}$$



3.Training and Testing:

- **Training:** The process of teaching the model using a dataset. The model learns by adjusting its parameters to minimize the difference between predicted and actual outcomes (loss).

```
# Initialize model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

- **Testing:** After training, the model is evaluated on a separate dataset to assess its performance and generalization ability. This helps to identify overfitting or underfitting.

```
# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

  - MSE: A lower MSE indicates that the model's predictions are closer to the actual values.
  - R^2 Score: A score closer to 1 indicates a better fit; it tells how well the model explains the variance in the data.

4.Performance Metrics:

To evaluate the effectiveness of a machine learning model, various metrics are used, such as:

1. **Accuracy:** The percentage of correct predictions.
2. **Precision and Recall:** Metrics that provide insights into the model's performance, especially in classification tasks.
3. **F1 Score:** The harmonic mean of precision and recall, useful for imbalanced datasets.
4. **Mean Squared Error (MSE):U**sed for regression tasks to measure the average squared difference between predicted and actual values. A lower MSE indicates that the model's predictions are closer to the actual values.

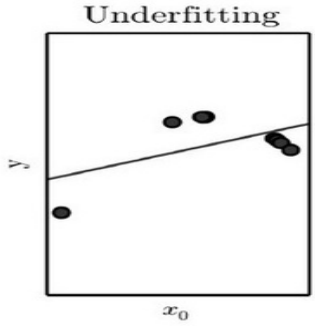
5.Overfitting and Underfitting:

1. **Overfitting:** Occurs when a model learns the training data too well, capturing noise and outliers, leading to poor performance on unseen data.
2. **Underfitting:** Happens when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and testing datasets.

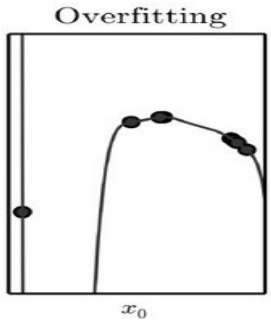
Loss Trends for Underfitting, Overfitting, and Good Fit

To distinguish these cases, let's look at the loss behavior during training and validation:

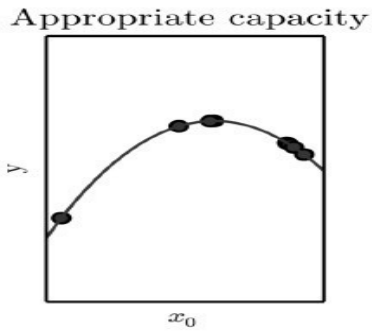
- **Underfitting:**
  - Training Loss: High Validation Loss: High
  - (similar to training loss) Explanation: The model
  - cannot fit the training data well, indicating it's too simplistic or lacks the necessary complexity.



- **Overfitting:**
  - Training Loss: Very low
  - Validation Loss: High
  - Explanation: The model has fit the training data too well (including noise), which causes it to perform poorly on new data. This discrepancy is a sign of overfitting.



- **Good Fit:**
  - Training Loss: Moderate to low
  - Validation Loss: Similar to training loss (slightly higher is okay) Explanation: The model
  - performs well on both the training and validation data, indicating it has achieved a balance between bias and variance.



What is DL?

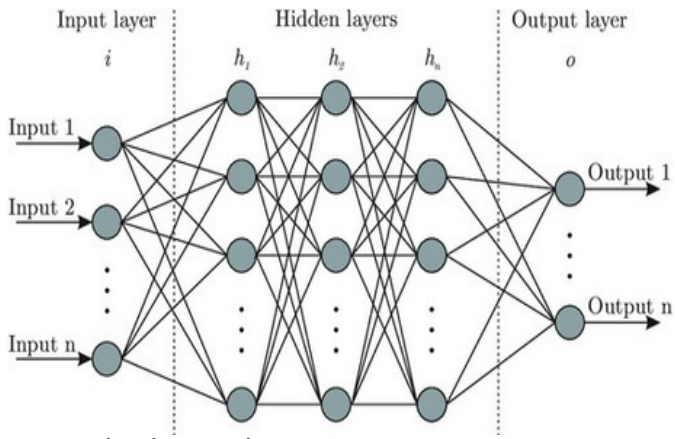
Deep Learning is a specialized subset of machine learning that uses artificial neural networks with multiple layers (deep neural networks) to model complex patterns in large datasets. It mimics the way the human brain processes information.

Key Characteristics:

- **Layered Structure:** Deep learning models consist of an input layer, multiple hidden layers, and an output layer, allowing them to learn hierarchical representations of data.
- **Automatic Feature Extraction:** Unlike traditional ML, deep learning automatically extracts features from raw data, reducing the need for manual feature engineering.
- **Data Requirements:** Deep learning typically requires large amounts of data and significant computational power to train effectively.

Fundamentals of Deep Learning

- **Neural Networks**: Deep learning is based on artificial neural networks, which are inspired by the structure and function of the human brain. A neural network consists of layers of interconnected nodes (neurons) that process data. **Layers**:
- Neural networks typically consist of an input layer, one or more hidden layers, and an output layer. The "depth" of a network refers to the number of hidden layers, hence the term "deep learning."



  - Input Layer: Accepts the input data.
  - Hidden Layers: Perform intermediate processing of inputs.
  - Output Layer: Produces the final result or prediction.
- **Weight and Bias:** Weight parameter represents the strength of the connection between units i.e directly proportional to the strength of the associated input neuron in deciding output. Bias can be considered as the line of intercept in a linear equation
- **Activation Functions**: These functions determine the output of a neuron based on ~~its input.Activation fu~~nctions are essential components of deep learning models because they introduce non-linearity, allowing models to capture complex patterns.
  - **Linear** :

Formula: $f(x) = x$
Range: $(-\infty, \infty)$
Uses: Output layer in regression tasks (e.g., CGPA prediction).

Pros: Maintains original data scale, No vanishing gradient issue (derivative = 1), Directly interpretable outputs.
Cons: No non-linearity; can't capture complex patterns.
Unbounded outputs (may produce extreme values).
Not suitable for hidden layers in complex tasks.

- **Sigmoid**

Formula: $\sigma(x) = \frac{1}{1+e^{-x}}$
Range: (0, 1)
Uses: Commonly used in binary classification problems, especially in the output layer.
Pros: Smooth gradient, maps values between 0 and 1.
Cons: Can lead to vanishing gradients; outputs are not zero-centered.

- **Tanh (Hyperbolic Tangent)**

Formula: $\tanh(x) = \frac{2}{1+e^{-2x}} - 1$
Range: (-1, 1)
Uses: Frequently used in hidden layers of recurrent neural networks (RNNs).
Pros: Zero-centered outputs, helps with vanishing gradient issues compared to sigmoid.
Cons: Still susceptible to vanishing gradients.

- **ReLU (Rectified Linear Unit)**

Formula: $f(x) = \max(0, x)$
Range: [0, ∞)
Uses: Widely used in dense layers for non linearity,helping capture complex patterns.
Pros: Computationally efficient, helps mitigate the vanishing gradient problem.
Cons: "Dying ReLU" problem where neurons can get stuck at zero and stop learning.

- **Leaky ReLU**

Formula: $f(x) = x$ if $x > 0$; $f(x) = \alpha x$ if $x \leq 0$ (commonly $\alpha = 0.01$)
Range: (-∞, ∞)
Uses: Alternative to ReLU to prevent dying ReLU issues.
Pros: Allows a small gradient when $x \leq 0$, preventing dead neurons.
Cons: Additional hyperparameter α to tune.

In our model,we used:

- ReLU, Tanh, or Sigmoid are randomly selected for hidden layers in the LSTM.
- Linear is used in the output layer for continuous value prediction (CGPA).
- Sigmoid is used in the output layer for binary classification (placement prediction).

Loss Functions : Loss functions measure how well a model's predictions match the true values, guiding the model's learning process by providing a feedback signal. Lower loss indicates better model performance. Different loss functions are chosen based on the type of problem: regression, classification, etc. **Loss Functions Used in the Model** Mean Squared Error (MSE):

- Used for: The CGPA prediction model (a regression task). Description: Calculates
- the average of the squared differences between predicted and true values, penalizing larger errors more heavily. MSE is commonly used in regression tasks as it's sensitive to large errors, which helps the model learn to make more accurate predictions.

```
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mse'])
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of samples.

Binary Cross-Entropy:

- Used for: The Placement prediction model (a binary classification task).
- Description: Measures the difference between the predicted probability distributions and the actual binary labels. The output is a lower loss when predicted probabilities are closer to the actual labels (0 or 1), making it suitable for binary classification tasks where the target is either 0 or 1.

```
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

Here, y_i is the actual class (0 or 1), $\hat{y}_i$ is the predicted probability for class 1, and n is the number of samples.

Some other commonly used loss functions:

- Mean Absolute Error (MAE):
  - Use case: Regression.
  - Description: Computes the average absolute difference between predicted and true values. Unlike MSE, it's less sensitive to large outliers.
- Categorical Cross-Entropy:
  - Use case: Multi-class classification. Description: Measures the difference between
  - predicted probabilities and actual classes in multi-class tasks, penalizing incorrect predictions based on their probability.

- Huber Loss:
  - Use case: Regression with robustness to outliers.
  - Description: Combines MSE and MAE, applying MSE for small errors and MAE for large errors, making it robust to outliers.

- **Training Process**: Training a neural network involves optimizing the model parameters (weights and biases) so that it accurately maps inputs to the desired outputs. This process is achieved through an iterative approach of forward propagation, loss calculation, backward propagation (backpropagation), and parameter updates via optimization algorithms like gradient descent. Each of these components contributes uniquely to the learning process, enabling the model to make accurate predictions.
  - **Forward Propagation:** The training process begins with forward propagation, where input data is passed through the layers of the neural network. Each layer applies a series of transformations, typically a weighted sum followed by an activation function. For example, in a simple neural network layer, the transformation for an input x is:

    $$z = w \cdot x + b$$

    where:

    - w is the weight associated with the input,
    - b is the bias term,
    - z is the linear transformation output.

    The result zzz is then passed through an activation function, such as ReLU, sigmoid, or tanh, to introduce non-linearity into the model. The output at each layer serves as the input to the next layer, and this process continues until the output layer provides a prediction.

    After obtaining the prediction from forward propagation, the next step is to calculate the loss or error, which measures how far the predicted output is from the actual output. For regression tasks, Mean Squared Error (MSE) is commonly used, while Binary Cross-Entropy (BCE) is used for binary classification tasks. Lowering this error over time is the goal of training, and it provides a basis for improving the model's performance.

  - **Gradient Descent:**To minimize the loss, the model's parameters (weights and biases) are adjusted in a direction that reduces the error. Gradient descent is the optimization algorithm that facilitates this process by iteratively updating parameters to reach a minimum loss.
    - **Mathematical Intuition for Gradient Descent**
      - Gradient descent relies on the derivative (gradient) of the loss function with respect to each parameter, indicating the direction and rate of change in the loss concerning each parameter. Mathematically, for a parameter θ (weight or bias), the update rule in gradient descent is:

        $$\theta := \theta - \eta \cdot \frac{\partial \mathcal{L}}{\partial \theta}$$

        where:

        - η is the **learning rate**, which controls the step size in each update,
        - $\frac{\partial \mathcal{L}}{\partial \theta}$ is the gradient of the loss function $\mathcal{L}$ with respect to θ.

- The gradient $\partial L/\partial \theta$ points in the direction of the steepest increase in loss, so moving in the opposite direction (subtracting the gradient) helps reduce the loss.This process repeats until the gradient is close to zero, meaning we have reached a minimum of the loss function. However, finding a global minimum is challenging, and the model may reach a local minimum instead.

- The **learning rate** controls the step size in gradient descent. A larger learning rate leads to faster updates but risks overshooting the minimum, while a smaller learning rate ensures more gradual convergence but can be slower and potentially get stuck in local minima. Careful tuning of the learning rate is essential for efficient training. Optimizers

While gradient descent is a core optimization technique, there are several advanced optimizers designed to improve training efficiency and stability:

- SGD (Stochastic Gradient Descent): Uses one or a few samples for each update, adding variability to the updates. This can help the model escape local minima but may result in noisy updates.
- Momentum: Builds on SGD by accelerating updates in the direction of the persistent gradient, helping to smooth the trajectory and speed up convergence in certain directions.
- Adam (Adaptive Moment Estimation): Combines aspects of Momentum and RMSProp (Root Mean Square Propagation) optimizers. Adam maintains a per-parameter learning rate, adjusting it based on estimates of the first and second moments of gradients. It's widely used due to its stability and adaptability across various problems.
- RMSProp: Adjusts the learning rate for each parameter based on recent gradient magnitudes, favoring parameters that consistently point in similar directions. This is especially useful for non-convex optimization problems.

Choosing the right optimizer can enhance model training, as it affects the speed of convergence and helps avoid getting stuck in local minima.

This entire procedure of computing the gradient of the loss function with respect to each parameter with the help of gradient descent is referred to as Backpropagation,calculating gradients from the output layer backward to the input layer.

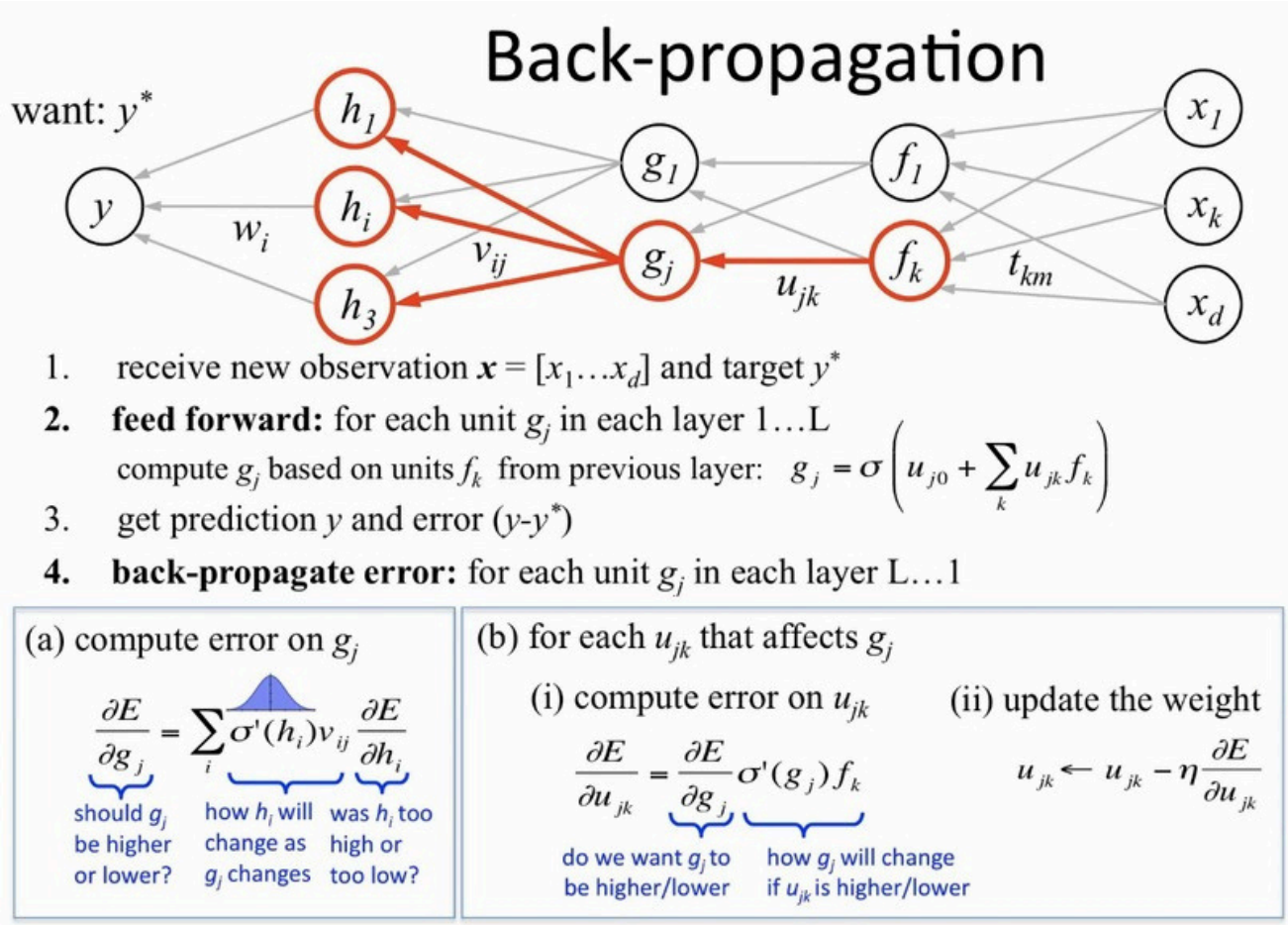
Backpropagation is an essential algorithm in neural networks that adjusts weights to minimize the error (or "loss") between predicted and actual outputs. It leverages the chain rule from calculus to propagate error gradients backward through the network, optimizing each weight in the process.

- **Steps in Backpropagation**
  - **Forward Pass:** Compute the activations of each neuron in the network, eventually yielding the predicted output y for a given input.
  - **Calculate the Loss:** Use the predicted output y and the true label y to calculate the loss L.

$$L = \frac{1}{2}(\hat{y} - y)^2$$

- **Backward Pass (Gradient Calculation):**
  - Start from the output layer and move backwards.
  - Compute the gradient of L with respect to the weights and activations at each layer.
- For each layer, we update weights w by moving them in the opposite direction of the gradient to reduce the loss:

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



Fixing Overfitting : Overfitting occurs when a model learns noise in the training data instead of the underlying patterns.To fix overfitting issue we used:

Early Stopping

- Purpose: Early stopping is a technique to prevent overfitting by monitoring the model's performance on validation data and stopping the training process once the performance stops improving. How It Works:
-
    - During training, the model's validation loss (or accuracy) is tracked. If the
    - validation loss stops improving for a specified number of epochs (controlled by the patience parameter), training halts, and the weights from the best epoch are restored.
- Benefits:
    - Early stopping prevents excessive training, which can lead to overfitting.
    - It saves computational resources by avoiding unnecessary epochs.
    - It ensures that the model retains weights from the epoch with the best validation performance, improving generalization.
- Fine-Tuning Tips:
    - Choosing an appropriate patience level is important. A low patience value might cause premature stopping, while a high patience could result in overfitting.
    - It's useful to monitor *val_loss* for regression tasks or *val_accuracy* for classification tasks, depending on the target metric.

Regularization

Regularization adds penalties to the model's objective function to discourage complex models, which are prone to overfitting.

- Types of Regularization:
    - L2 Regularization:
        - L2 regularization (also known as weight decay) adds a penalty proportional to the squared magnitude of weights.
        - It encourages the model to keep weights small, thus reducing overfitting.
        - In neural networks, L2 regularization can be applied to the dense or LSTM layers, where weights are critical.
        - We applied L2 in both your models (kernel_regularizer=l2(l2_penalty)) to constrain weight magnitudes.
    - Dropout:
        - Dropout is a regularization technique where random nodes (neurons) in the network are ignored (dropped out) during each training step.
        - This forces the model to learn multiple, independent representations, improving robustness and reducing overfitting.
        - Dropout is typically applied between dense layers or after LSTM layers.
        - For LSTM networks, a dropout rate around 0.2-0.5 is common. However, very high dropout rates can lead to underfitting.
- Benefits:
    - Regularization techniques control the complexity of the model by limiting the capacity of weights and forcing neurons to rely on fewer features.
    - This prevents overfitting on training data and improves generalization to new data.

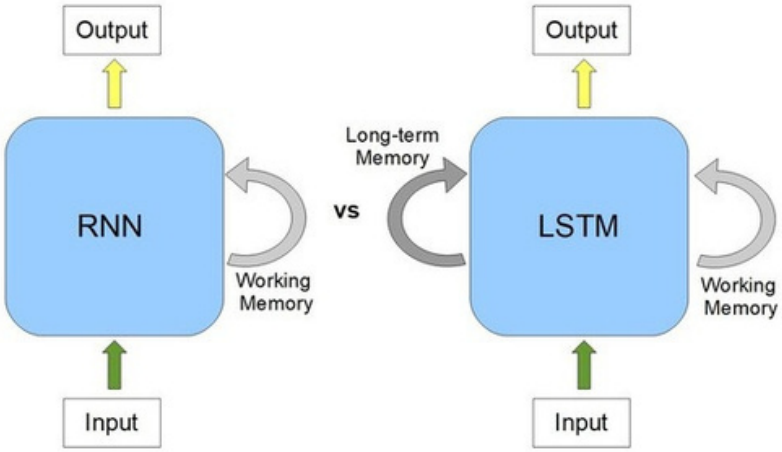
Batch Normalization

- Purpose: Batch normalization stabilizes and accelerates training by normalizing the inputs of each layer within each mini-batch. It helps to:
  - Reduce internal covariate shift (i.e., changes in the distribution of layer inputs as training progresses).
  - Allow for higher learning rates, reducing training time.
  - Improve generalization by adding slight regularization effects.
- How It Works:
  - During training, it computes the mean and variance for each mini-batch and normalizes the data within the batch.
  - Each feature in the batch is scaled to have mean 0 and variance 1, followed by scaling and shifting with learned parameters.
- Benefits:
  - Batch normalization can act as a form of regularization by reducing overfitting.
  - It reduces the need for dropout or reduces dropout rates, as it stabilizes learning.
  - It helps manage the problem of vanishing and exploding gradients, which can hinder training deep networks.

Introduction to LSTM Networks

A Recurrent Neural Network (RNN) is a type of neural network designed to handle sequential data, making it well-suited for tasks like time series analysis, natural language processing, and speech recognition. Unlike traditional feedforward networks, RNNs have a feedback loop, allowing them to retain information from previous inputs by maintaining a "memory" of past states. This characteristic enables RNNs to understand the context within a sequence, as each output depends not only on the current input but also on the previous computations.
However, RNNs face limitations when dealing with long-term dependencies in sequences. As information from earlier steps moves through the network, it often gets "forgotten" due to the vanishing gradient problem. This issue limits the network's ability to capture patterns over extended sequences, leading to performance degradation in tasks where long-term dependencies are crucial.
What is the Vanishing Gradient Problem?
In a deep neural network, backpropagation is used to calculate gradients for each layer, which are then used to update the weights. In networks with many layers, the gradients tend to exponentially decrease (or sometimes increase) as they move backward from the output layer to the input layer. When gradients become very small, they "vanish," effectively preventing the earlier layers from learning and slowing down or even stalling the training process.

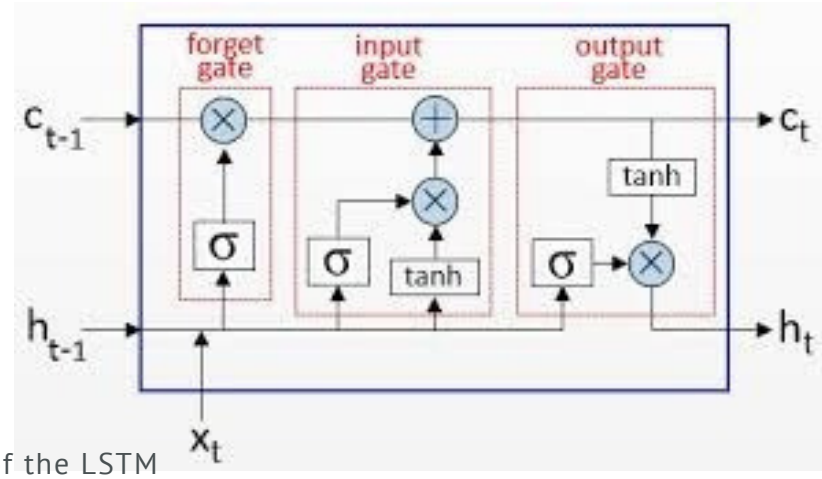
LSTMs were introduced to address this limitation by using a more complex architecture. LSTMs are a type of Recurrent Neural Network (RNN) designed to handle sequential data and to handle long term dependencies overcoming vanishing gradient problem
Applications of LSTMs: LSTMs are widely used in time-series prediction, natural language processing, speech recognition, and anomaly detection, as they effectively capture patterns across sequences.



Architecture of LSTM Model

LSTM networks have a unique cell structure compared to standard RNNs, allowing them to store and update information over long sequences. Each LSTM cell contains:

- **Cell State (Ct)**: The cell state carries information across the entire sequence. It acts like a memory, allowing the LSTM to remember or forget information over time.
- **Hidden State (ht)**: This is the output of the cell for the current time step, which goes on to the next cell in the sequence.
- **Gates**: Each LSTM cell contains three gates to control the flow of information:
  a. **Forget Gate**: Decides which information should be discarded from the cell state.
  b. **Input Gate**: Determines which values to update in the cell state with new information.
  c. **Output Gate**: Controls the information that moves from the cell state to the hidden state (i.e., output of the LSTM cell).



Xt: Input of the LSTM

Key Properties and Advantages of LSTMs

- Long-term Dependency Capture: By controlling the flow of information through the gates, LSTMs can retain important past information across many time steps.
- Mitigating the Vanishing Gradient Problem: Through its gating mechanisms, the LSTM helps gradients to propagate effectively, even over long sequences.
- Adaptive Memory Mechanism: The cell state enables selective memory retention, allowing LSTMs to focus on relevant parts of a sequence.

Training LSTM Networks

- LSTMs require the input data to be shaped as 3D tensors: (samples, time steps, features). Time steps represent the length of each sequence, and features represent the number of input features. Loss Functions:
-
  - For regression tasks (e.g., predicting CGPA), the Mean Squared Error (MSE) loss function is commonly used.
  - For classification tasks (e.g., predicting placement), Binary Crossentropy or Categorical Crossentropy can be used.
- Optimizers: Optimizers like Adam, RMSprop, and SGD are common choices for training LSTMs. Adam is often preferred because of its adaptive learning rate capabilities.

Evaluation Metrics for LSTM Models

- For Regression (CGPA Prediction):
  - Mean Squared Error (MSE): Measures the average squared difference between predicted and actual values, used as a primary metric in continuous predictions.
  - Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual values, often used for interpretability.
- For Classification (Placement Prediction):
  - Accuracy: Measures the ratio of correct predictions to total predictions.
  - Precision, Recall, F1-Score: Important for imbalanced datasets where accuracy alone may not be enough.
  - ROC-AUC: Measures the model's ability to distinguish between classes. An AUC closer to 1 indicates a better model.

Variations and Enhancements of LSTMs

  - Bidirectional LSTM (BiLSTM): This variant processes the sequence in both forward and backward directions, capturing contextual information from both the past and future, which is used in our project for placement model

Summary of our Model

  - **Model 1: final_cgpa_model (named "sequential_141"):**
    - This model has a single LSTM layer, followed by a dropout layer and a dense output layer. It has 485,405 parameters in total, with a significant portion used by the optimizer.

Model: "sequential_141"

Layer (type)	Output Shape	Param #
lstm_141 (LSTM)	(None, 200)	161,600
dropout_141 (Dropout)	(None, 200)	0
dense_141 (Dense)	(None, 1)	201

Total params: 485,405 (1.85 MB)
Trainable params: 161,801 (632.04 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 323,604 (1.23 MB)

- **Model 2: final_placement_model (named "sequential_144"):**
  - This model uses a bidirectional LSTM layer to capture dependencies in both directions, followed by dropout, batch normalization, and a dense output layer. It has fewer parameters (187,751) compared to the first model, making it potentially lighter and faster.

Model: "sequential_144"

Layer (type)	Output Shape	Param #
bidirectional_84 (Bidirectional)	(None, 174)	61,944
dropout_144 (Dropout)	(None, 174)	0
batch_normalization_84 (BatchNormalization)	(None, 174)	696
dense_144 (Dense)	(None, 1)	175

Total params: 187,751 (733.41 KB)
Trainable params: 62,467 (244.01 KB)
Non-trainable params: 348 (1.36 KB)
Optimizer params: 124,936 (488.04 KB)

Introduction to Metaheuristic Algorithm

What is Metaheuristic Algorithm?

A metaheuristic algorithm is an optimization technique designed to find approximate solutions to complex problems where traditional methods (like calculus-based optimization or exhaustive search) are impractical. Metaheuristic algorithms are inspired by natural processes, such as evolution, swarming, or physical annealing, and work by balancing two key aspects: exploration (searching new areas) and exploitation (refining promising areas). Examples include genetic algorithms, particle swarm optimization, simulated annealing, and ant colony optimization.

Why Use Metaheuristic Algorithms?

- **Efficient in Complex Spaces:** Metaheuristics work well in big, complicated search spaces, even when functions aren't smooth or simple For instance, in tuning an LSTM neural network, there are many parameters like learning rate, batch size, and number of layers. Using a genetic algorithm (GA), we can explore a vast search space by evolving solutions across generations, saving time compared to a grid search that tries every combination. **Escaping Local Optima:** They avoid getting stuck in suboptimal solutions by using techniques like mutation (in genetic algorithms) or controlled randomness (in simulated annealing).

- **Adaptable and Flexible:** Metaheuristics are highly customizable and can be adjusted for different types of problems, like fine-tuning parameters in machine learning.
- **Good Exploration and Refinement**: They balance broad searching (to explore new options) with focused improvement (to refine the best options), which is ideal for finding strong solutions efficiently.
- **Parallel-Friendly:** Many metaheuristics allow multiple solutions to be tested at once such as maximizing both accuracy and speed of a model, speeding up the search and making them scalable for larger problems.

Genetic Algorithms (GA) : A metaheuristic algorithm

- The genetic algorithm works based on the principle of natural selection and biological evolution process. It is a random optimization method that changes the solution of problems in data individuals of a gene sting structure in genetic space by a certain coding scheme, converts objective function into fitness value and evaluates the advantages and disadvantages of individuals and as the basis for the genetic operation

- **Key Characteristics:**
  - **Population-Based:** GA maintains a population of solutions that evolve through selection, crossover (recombination), and mutation.
  - **Fitness Function:** Each solution is evaluated using a fitness function, which determines how well it solves the problem at hand.
  - **Convergence:** Over successive generations, GA aims to improve the population's overall fitness, converging towards optimal solutions.

- **Key Concepts of Genetic Algorithms**
  - **Population**: A collection of candidate solutions (individuals). Each individual represents a potential solution, each defined by unique set of parameters to the problem we are trying to solve. **Chromosome**: The representation of an individual. In our case, it's a list of hyperparameters for a machine learning model. **Fitness Function**: A function that evaluates how good an individual is at solving the problem. The fitness score indicates the quality of the solution. **Selection**: The process of choosing individuals based on their fitness scores to be parents for the next generation. **Crossover (Recombination)**: A genetic operator that combines two parent solutions to produce offspring. **Mutation**: A genetic operator that introduces random changes to an individual to maintain diversity in the population. **Iteration**: The process of repeating the evaluation, selection, crossover, and mutation steps for a specified number of generations or until a stopping criterion is met.
  -

Genetic Algorithm Steps
(with reference through our code)

- **Step 1: Create initial population of individuals.**

```
toolbox = base.Toolbox()
toolbox.register("attr_units", random.randint, 50, 200)
toolbox.register("attr_activation", random.choice, ['relu', 'tanh', 'sigmoid'])
toolbox.register("attr_dropout", random.uniform, 0.0, 0.5)
toolbox.register("attr_optimizer", random.choice, ['adam', 'rmsprop', 'sgd', 'adadelta']
toolbox.register("attr_learning_rate", random.uniform, 1e-4, 1e-2)

toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_units, toolbox.attr_activation, toolbox.attr_dropout,
                  toolbox.attr_optimizer, toolbox.attr_learning_rate), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

  - In the GA, the population consists of a set of candidate solutions (individuals), each defined by a unique set of parameters
  - Each individual represents a combination of:
    - Number of LSTM units (between 50 and 200)
    - Activation function (e.g., 'relu', 'tanh', 'sigmoid')
    - Dropout rate (between 0 and 0.5)
    - Optimizer (e.g., 'adam', 'rmsprop', etc.)
    - Learning rate (between 10^{-4} and 10^{-2})
  - This setup defines the "genome"(a specific hyperparameter) of each individual in the population.

- **Step 2: Evaluate fitness of individuals**.
  - The fitness function is critical as it guides the GA by scoring each individual's performance. For our models, the fitness is evaluated by training a neural network with the individual's hyperparameters and then assessing its performance on the test set.

```
def evaluate_cgpa_model(individual):
    # Extract hyperparameters
    units = individual[0]
    activation = individual[1]
    dropout = individual[2]
    optimizer = individual[3]
    learning_rate = individual[4]

    # Create model with specific hyperparameters
    model = create_cgpa_model(units, activation, dropout, optimizer, learning_rate)
    model.fit(X_train_cgpa_reshaped, y_train_cgpa, epochs=10, verbose=0)
    # Evaluate on test set
    loss, mse = model.evaluate(X_test_cgpa_reshaped, y_test_cgpa, verbose=0)
    return mse,
```

    - Fitness Evaluation :
      - For CGPA model: Compute MSE(x_i), where lower MSE is better.
      - For Placement model: Compute Accuracy(x_i), where higher accuracy is better

- **Step 3: Select the individuals; selection method is used to probabilistically select individuals from a population for later breeding.**
  - In GA, selection methods like roulette wheel, tournament, or rank-based are used to choose individuals for reproduction. In our code, a tournament selection method is used to select individuals based on their fitness scores.

```
toolbox.register("select", tools.selTournament, tournsize=3)
```

  - The selection process, where tournsize=3 means that, for each selection, three individuals are randomly chosen, and the fittest of these is selected. This promotes better-performing individuals while still allowing diversity.

- **Step 4: Apply genetic operators such as Crossover and mutation.**
  - Genetic operators like crossover and mutation generate new individuals (children) by mixing and slightly altering the "genes" (hyperparameters) of selected individuals.
    - Crossover:
      - We use cxTwoPoint, a two-point crossover function, to combine genes of two parents and create offspring.

```
toolbox.register("mate", tools.cxTwoPoint)
```

      - This function selects two crossover points in the parent genomes and exchanges the segments between these points to produce two children.
    - Mutation:(mutation operator should allow diversity)
      - In GA, mutation introduces random changes to ensure diversity. In our code, the custom_mutate function modifies each gene with a probability of 20% (Pm), maintaining genetic diversity.

```
def custom_mutate(individual):
    individual[0] = random.randint(50, 200) if random.random() < 0.2 else individual[0]
    individual[1] = random.choice(['relu', 'tanh', 'sigmoid']) if random.random() < 0.2 else individual[1]
    individual[2] = max(0.0, min(1.0, individual[2] + random.gauss(0, 0.1))) if random.random() < 0.2 else individual[2]
    individual[3] = random.choice(['adam', 'rmsprop', 'sgd', 'adadelta']) if random.random() < 0.2 else individual[3]
    individual[4] = max(1e-4, min(1e-2, individual[4] + random.gauss(0, 0.001))) if random.random() < 0.2 else individual[4]
    return individual,

toolbox.register("mutate", custom_mutate)
```

      - Here, each parameter has a 20% chance of changing, aligning with the concept of setting Pm around 10-20% for diversity.

- **Step 5: Finished all generations in the genetic algorithm if stopping criteria are agreed otherwise it evaluates fitness of individuals again**
  - The GA iterates over generations until a certain number is reached (ngen=10 in our code). After each generation, the fitness of the population is re-evaluated, and the best-performing individuals are selected for further crossover and mutation.

```
population = toolbox.population(n=10)
ngen = 10
cxpb = 0.5
mutpb = 0.2

result_cgpa, logbook = algorithms.eaSimple(population, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, verbose=True)
```

- The eaSimple function runs the GA loop with:
  - cxpb=0.5 as the probability of applying crossover
  - mutpb=0.2 as the mutation probability ngen=10
  - as the number of generations
- Each cycle produces a new population, evolving towards better solutions.

Optimization Goal

Through each generation, GA mathematically optimizes the fitness function by:

$$\min_{\mathbf{x}^i} \mathrm{MSE}(\mathbf{x}^i) \quad \text{or} \quad \max_{\mathbf{x}^i} \mathrm{Accuracy}(\mathbf{x}^i)$$

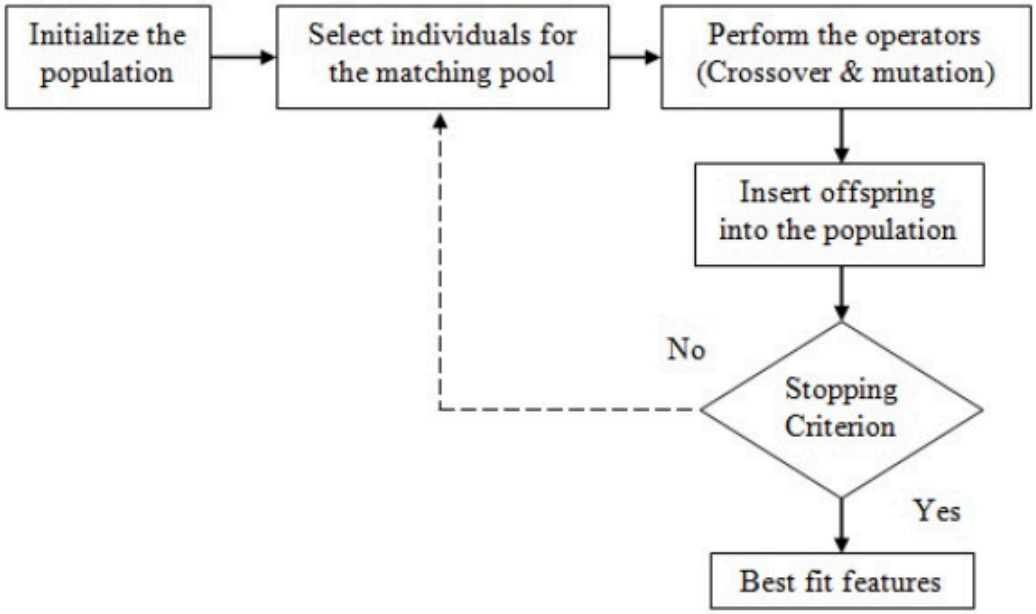


Fig. 3: Phases of GA for Feature Selection

LIBRARIES

1. NumPy (Numerical Python)

•Purpose: Essential for numerical computations, handling arrays and matrices.
•Key Features: Provides powerful n-dimensional arrays and mathematical functions for working with them.
•Use Case: Commonly used for data manipulation, matrix operations, and implementing custom algorithms.

Example:

```
import numpy as np
# Create an array
arr = np.array([1, 2, 3, 4])
# Matrix operations
arr_squared = arr ** 2
```

2. TensorFlow

•Purpose: Open-source framework for building machine learning models, especially deep learning.
•Key Features: Provides tools for defining, training, and deploying models. Supports CPU/GPU computation.
•Use Case: Building neural networks, regression, classification, and other ML tasks. TensorFlow can be used with Keras (high-level API).

Example:

```
import tensorflow as tf

# Create a constant tensor
tensor = tf.constant([1, 2, 3, 4])

# Perform operations
result = tensor + 10
```

3. Keras

•Purpose: High-level neural networks API, running on top of TensorFlow.
•Key Features: Easy-to-use, modular, and intuitive API for defining and training deep learning models.
•Use Case: Quick model prototyping, defining architectures (like CNNs, RNNs), and training models with minimal code.

Example:

```
ifrom tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)), # Flattening the input image
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
]) # Compile and train the model
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

4. Pandas

•Purpose: Library for data manipulation and analysis, especially with structured data (like CSV files).
•Key Features: DataFrames, data manipulation, handling missing data, and simple data analysis.
•Use Case: Importing datasets, cleaning data, and preparing it for machine learning models.

Example:

```
import pandas as pd

# Load a CSV file
df = pd.read_csv('data.csv')

# Data operations
df['column'] = df['column'].fillna(0) # Handle missing data
```

5. Matplotlib

•Purpose: Plotting and visualizing data.
•Key Features: Create line charts, histograms, scatter plots, and more.
•Use Case: Visualizing model performance (e.g., accuracy, loss) or exploring data.

Example:

```
import matplotlib.pyplot as plt

# Plot a simple graph
plt.plot([1, 2, 3], [1, 4, 9])
plt.title('Example Plot')
plt.show()
```

6. Scikit-Learn

•Purpose: Machine learning library with tools for classification, regression, clustering, and more.
•Key Features: Implements common algorithms like decision trees, SVM, KNN, and provides tools for model evaluation.
•Use Case: Building traditional machine learning models (non-deep learning), model selection, and evaluation.

Example

(Training a classifier):

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Train a model
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
```

7. DEAP(Distributed Evolutionary Algorithms in Python)

- Purpose: designed to facilitate the development and implementation of evolutionary algorithms for solving optimization and machine learning problems.
- Key Features:
  (i)Customizable evolutionary operators.
  (ii)Supports multi-objective optimization.
  (iii)Parallelization for large-scale problems.
- Use Case: optimize complex problems like function maximization, genetic programming, machine learning hyperparameter tuning, and evolutionary robotics.

Example: *Maximizing f(x)=x^2*

```
import random
from deap import base, creator, tools

# Setup individual and fitness type
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# Initialize individual and evaluate function
toolbox = base.Toolbox()
toolbox.register("individual",tools.initIterate,creator.Individual,lambda: [random.uniform(-5,
5)])
toolbox.register("evaluate", lambda ind: (ind[0]**2,))

# Create population and run algorithm population
=                                toolbox.population(n=10)
tools.selBest(population, 1)[0] # Get best solution
```

DATASET

CSV

A CSV (Comma-Separated Values) file is a simple, text-based format used to store tabular data, where each line represents a data record, and each field is separated by a comma. In machine learning, CSV files are widely used to organize and transfer datasets, making them easy to read into data processing libraries like Pandas, which can then be used to train models.

```
data = pd.read_csv(io.BytesIO(uploaded['KU_STUDENT_DATA_ON_CAMPUS_PLACEMENT.csv']))
print(data.head())
```

```
  Branch               Cepo  Program  End term exam SGPA - 1st semester   \
0    CSE  Currently enrolled  B.Tech                                6.50
1    CSE  Currently enrolled  B.Tech                                7.70
2    CSE          Passed out  B.Tech                                6.96
3    CSE          Passed out  B.Tech                                6.20
4    CSE  Currently enrolled  B.Tech                                6.20

   End term exam SGPA - 2nd semester  End term exam SGPA - 3rd semester  \
0                               7.20                               6.10
1                               8.57                               7.70
2                               7.14                               6.71
3                               6.30                               7.88
4                               6.30                               6.30

   End term exam SGPA - 4th semester  End term exam SGPA - 5th semester  \
0                               8.20                               6.80
1                               9.09                               7.62
2                               7.88                               7.05
3                               8.03                               8.68
4                               6.30                               7.01

   End term exam SGPA - 6th semester  End term exam SGPA - 7th semester  ...  \
0                               6.60                               6.30  ...
1                               7.84                               6.50  ...
2                               8.14                               6.33  ...
3                               9.14                               8.61  ...
4                               7.04                               6.30  ...

   EDU_LN  SCHL_RCV  URB_RUR INT_CONN How many hrs you study after school?  \
0     Yes        No    Urban     Good                                   1
1      No        No    Urban     Good                                   3
2      No        No    Rural     Good                                   3
3      No        No    Urban     Good                                   2
4     Yes        No    Urban     Good                                   2

   How many value added program you have entered? (coursera/ AWS/IBM etc)  \
0                                                  2
1                                                  7
2                                                  2
3                                                  2
4                                                  2

   SPOR_PSN  COC_PART COC_PART_ROLE Cam_plc
0       Yes       yes   Team leader     Yes
1       Yes       yes   Team leader      No
2        No       yes     Volunteer      No
3        No       yes     Volunteer      No
4        No       yes     Volunteer     Yes
```

All the columns in our dataset:
The dataset contains 369 entries(369 students) with 31 columns, and it includes various attributes related to Engineering students' academic and extracurricular profiles.An overview of the dataset:

- Academic Performance:
  - Department of Engineering
  - Semester-wise SGPA (End term exam SGPA - 1st semester through 8th semester).
  - Final CGPA after the 8th semester.
  - Grades in Class X and Class XII.
- Personal and Educational Background:
  - Branch, program, and enrollment status (e.g., "Currently enrolled", "Passed out").
  - Gender and family type.
- Extracurricular and Additional Activities:
  - Participation in sports, clubs, and other co-curricular roles.
  - Number of value-added programs completed (like Coursera or AWS certifications).
  - Number of internships.
- Placement Indicator:
  - The "Cam_plc" column indicates if the student was placed on campus.

This dataset provides a rich combination of continuous (e.g., SGPA values), categorical (e.g., Branch, Program, Gender), and binary data, suitable for a model that predicts CGPA and placement outcomes using LSTM and genetic algorithms. The time-series nature of semester SGPA data aligns well with LSTM, while other variables can aid in refining predictions.

METHODOLOGY

1. IMPORTING NECESSARY LIBRARIES

The libraries used are pandas,sklearn,numpy,tensorflow,random and pickle.

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import random
import tensorflow as tf
import pickle
import io

# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)
tf.random.set_seed(42)
```

Screenshot 1

2. Data Preprocessing

- Lowercasing: Converts all string data to lowercase for consistency.
- Categorical Features: Categorical features are one-hot encoded using OneHotEncoder with the drop='first' parameter to prevent multicollinearity.
- Numerical Features: Numerical data is standardized using StandardScaler to ensure the features have a mean of 0 and a standard deviation of 1.

```
data = data.apply(lambda x: x.str.lower() if x.dtype == "object" else x)

categorical_features = data.select_dtypes(include=['object']).columns.tolist()
encoder = OneHotEncoder(sparse_output=False, drop='first')
encoded_categorical_data = encoder.fit_transform(data[categorical_features])

numerical_features = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
scaler = StandardScaler()
scaled_numerical_data = scaler.fit_transform(data[numerical_features])

encoded_categorical_df = pd.DataFrame(encoded_categorical_data, columns=encoder.get_feature_names_out(categorical_features))
scaled_numerical_df = pd.DataFrame(scaled_numerical_data, columns=numerical_features)
processed_data = pd.concat([encoded_categorical_df, scaled_numerical_df], axis=1)
```

Screenshot 2

3. Extracting the target variable

Target Variables: The target variables are CGPA and placement outcomes.

```
X_cgpa = processed_data.drop(['CGPA after 8th semester'], axis=1)
y_cgpa = processed_data['CGPA after 8th semester']

X_placement = processed_data.drop(['Cam_plc_yes'], axis=1)
y_placement = processed_data['Cam_plc_yes']
```

Screenshot 3

4. TRAIN_TEST_SPLIT & DATA RESHAPING FOR LSTM

Train-Test Split: The dataset is split into training and testing sets using an 80-20 ratio. Additionally, the data is reshaped to be compatible with LSTM input.

```
X_train_cgpa, X_test_cgpa, y_train_cgpa, y_test_cgpa
= train_test_split(X_cgpa, y_cgpa, test_size=0.2, random_state=42)

X_train_placement, X_test_placement, y_train_placement, y_test_placement
= train_test_split(X_placement, y_placement, test_size=0.2, random_state=42)

# Reshape data for LSTM
X_train_cgpa_reshaped
= np.reshape(X_train_cgpa.values, (X_train_cgpa.shape[0], X_train_cgpa.shape[1], 1))

X_test_cgpa_reshaped
= np.reshape(X_test_cgpa.values, (X_test_cgpa.shape[0], X_test_cgpa.shape[1], 1))

X_train_placement_reshaped
= np.reshape(X_train_placement.values, (X_train_placement.shape[0], X_train_placement.shape[1], 1))

X_test_placement_reshaped
= np.reshape(X_test_placement.values, (X_test_placement.shape[0], X_test_placement.shape[1], 1))
```

Screenshot : 4

5. Designing function for LSTM models of CGPA

CGPA Prediction Model : A Sequential LSTM model is built with hyperparameters tuned using a genetic algorithm. The model consists of LSTM layers, dropout for regularization, and dense output layers. The optimizer and learning rate are also selected based on the genetic algorithm.

```
def create_cgpa_model(units, activation, dropout, optimizer, learning_rate, dropout_rate=0.2, l2_penalty=0.001):
    model = Sequential()
    model.add(LSTM(units=units, activation=activation, input_shape=(X_train_cgpa_reshaped.shape[1], 1)))
    model.add(Dropout(dropout))
    model.add(Dense(1, activation='linear', kernel_regularizer=l2(l2_penalty)))

    if optimizer == 'adam':
        opt = Adam(learning_rate=learning_rate)
    elif optimizer == 'rmsprop':
        opt = RMSprop(learning_rate=learning_rate)
    elif optimizer == 'sgd':
        opt = SGD(learning_rate=learning_rate)
    elif optimizer == 'adadelta':
        opt = Adadelta(learning_rate=learning_rate)

    model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mse'])
    return model
```

Screenshot 5

6. Designing function for LSTM models of placement

Placement Prediction Model : Similarly, a Bidirectional LSTM model with batch normalization is used for predicting placement. The model is trained using binary cross-entropy loss and various optimizers (Adam, RMSprop, SGD, adadelta).

```
def create_placement_model(units, activation, dropout, optimizer, learning_rate, dropout_rate=0.5, l2_penalty=0.01):
  # changed l2, dropout_rate :0.2
    model = Sequential()
    model.add(Bidirectional(LSTM(units=units, activation=activation, input_shape
                                =(X_train_placement_reshaped.shape[1], 1))))
    model.add(Dropout(dropout))
    model.add(BatchNormalization())
    model.add(Dense(1, activation='sigmoid', kernel_regularizer=l2(l2_penalty)))

    if optimizer == 'adam':
        opt = Adam(learning_rate=learning_rate)
    elif optimizer == 'rmsprop':
        opt = RMSprop(learning_rate=learning_rate)
    elif optimizer == 'sgd':
        opt = SGD(learning_rate=learning_rate)
    elif optimizer == 'adadelta':
        opt = Adadelta(learning_rate=learning_rate)

    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

Screenshot : 6

7. Defining the fitness function for Placement and CGPA

```
def evaluate_cgpa_model(individual):
    units = individual[0]
    activation = individual[1]
    dropout = individual[2]
    optimizer = individual[3]
    learning_rate = individual[4]

    model = create_cgpa_model(units, activation, dropout, optimizer, learning_rate)
    model.fit(X_train_cgpa_reshaped, y_train_cgpa, epochs=10, verbose=0)
    loss, mse = model.evaluate(X_test_cgpa_reshaped, y_test_cgpa, verbose=0)
    return mse,

def evaluate_placement_model(individual):
    units = individual[0]
    activation = individual[1]
    dropout = individual[2]
    optimizer = individual[3]
    learning_rate = individual[4]

    model = create_placement_model(units, activation, dropout, optimizer, learning_rate)
    model.fit(X_train_placement_reshaped, y_train_placement, epochs=10, verbose=0)
    loss, acc = model.evaluate(X_test_placement_reshaped, y_test_placement, verbose=0)
    return 1 - acc,
```

Screenshot 7

Evaluation: Both models are trained with early stopping based on validation performance. The CGPA model evaluates mean squared error (MSE), while the placement model evaluates accuracy.

8. Register GA Components

```
toolbox = base.Toolbox()
toolbox.register("attr_units", random.randint, 50, 200)
toolbox.register("attr_activation", random.choice, ['relu', 'tanh', 'sigmoid'])
toolbox.register("attr_dropout", random.uniform, 0.0, 0.5)
toolbox.register("attr_optimizer", random.choice, ['adam', 'rmsprop', 'sgd', 'adadelta'])
toolbox.register("attr_learning_rate", random.uniform, 1e-4, 1e-2)

toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_units, toolbox.attr_activation, toolbox.attr_dropout,
                  toolbox.attr_optimizer, toolbox.attr_learning_rate), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxTwoPoint)
```

Screenshot : 8

9. Custom mutation function to ensure dropout rate remains within valid range

```
# Custom mutation function to ensure dropout rate remains within valid range
def custom_mutate(individual):
    individual[0]
    = random.randint(50, 200) if random.random() < 0.2 else individual[0]
    individual[1]
    = random.choice(['relu', 'tanh', 'sigmoid']) if random.random() < 0.2 else individual[1]
    individual[2]
    = max(0.0, min(1.0, individual[2] + random.gauss(0, 0.1))) if random.random() < 0.2 else individual[2]
    individual[3]
    = random.choice(['adam', 'rmsprop', 'sgd', 'adadelta']) if random.random() < 0.2 else individual[3]
    individual[4]
    = max(1e-4, min(1e-2, individual[4] + random.gauss(0, 0.001))) if random.random() < 0.2 else individual[4]
    return individual,

toolbox.register("mutate", custom_mutate)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate_cgpa_model)
```

Screenshot 9

10. Run the Genetic Algorithm for CGPA Model

10 individuals are initialized, and genetic operations like crossover and mutation are applied to evolve the population over 10 generations.

```
population = toolbox.population(n=10)
ngen = 10
cxpb = 0.5
mutpb = 0.2

result_cgpa, logbook = algorithms.eaSimple(population, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, verbose=True)

best_individual_cgpa = tools.selBest(population, 1)[0]
print(f'Best CGPA Model Hyperparameters: {best_individual_cgpa}')

# Save best hyperparameters for CGPA model
with open('best_hyperparameters_cgpa.pkl', 'wb') as f:
    pickle.dump(best_individual_cgpa, f)
```

Screenshot : 10

11. Run the Genetic Algorithm for Placement Model

```
toolbox.unregister("evaluate")
toolbox.register("evaluate", evaluate_placement_model)

population = toolbox.population(n=10)
result_placement, logbook = algorithms.eaSimple(population, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, verbose=True)

best_individual_placement = tools.selBest(population, 1)[0]
print(f'Best Placement Model Hyperparameters: {best_individual_placement}')

# Save best hyperparameters for Placement model
with open('best_hyperparameters_placement.pkl', 'wb') as f:
    pickle.dump(best_individual_placement, f)
```

Screenshot 11

12. Train the CGPA model with the best hyperparameters

```
best_units_cgpa = best_individual_cgpa[0]
best_activation_cgpa = best_individual_cgpa[1]
best_dropout_cgpa = best_individual_cgpa[2]
best_optimizer_cgpa = best_individual_cgpa[3]
best_learning_rate_cgpa = best_individual_cgpa[4]

final_cgpa_model
= create_cgpa_model(best_units_cgpa, best_activation_cgpa, best_dropout_cgpa,
                    best_optimizer_cgpa, best_learning_rate_cgpa)
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
                               restore_best_weights=True)

history_cgpa = final_cgpa_model.fit(
    X_train_cgpa_reshaped, y_train_cgpa,
    epochs=100,
    validation_data=(X_test_cgpa_reshaped, y_test_cgpa),
    callbacks=[early_stopping]
)

loss_cgpa, mse_cgpa = final_cgpa_model.evaluate(X_test_cgpa_reshaped, y_test_cgpa)
print('Final MSE for CGPA:', mse_cgpa)
```

Screenshot 12

13. Train the Placement model with the best hyperparameters

```
best_units_placement = best_individual_placement[0]
best_activation_placement = best_individual_placement[1]
best_dropout_placement = best_individual_placement[2]
best_optimizer_placement = best_individual_placement[3]
best_learning_rate_placement = best_individual_placement[4]

final_placement_model = create_placement_model(best_units_placement,
                                               best_activation_placement, best_dropout_placement,
                                               best_optimizer_placement, best_learning_rate_placement)
early_stopping = EarlyStopping(monitor='val_accuracy', patience=20, restore_best_weights=True)

history_placement = final_placement_model.fit(
    X_train_placement_reshaped, y_train_placement,
    epochs=100,
    validation_data=(X_test_placement_reshaped, y_test_placement),
    callbacks=[early_stopping]
)
loss_placement, acc_placement = final_placement_model.evaluate(X_test_placement_reshaped, y_test_placement)
print('Final Accuracy for Placement:', acc_placement)
```

Screenshot 13

14. Plotting ROC Curve for Placement Model

For the placement model, ROC-AUC measures the trade-off between true positive and false positive rates, providing an overall score of classification quality.

```
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Make predictions on the test set
y_pred_prob_placement = final_placement_model.predict(X_test_placement_reshaped).ravel()

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test_placement, y_pred_prob_placement)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Placement Model')
plt.legend(loc='lower right')
plt.show()

# Find optimal threshold using Youden's J statistic
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]

print(f'Optimal Threshold for Placement Prediction: {optimal_threshold:.4f}')
```

Screenshot 14

15. Plot training loss vs validation loss for CGPA model

```
plt.figure(figsize=(12, 6))
plt.plot(history_cgpa.history['loss'], label='Training Loss')
plt.plot(history_cgpa.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss vs Validation Loss for CGPA Model')
plt.legend()
plt.show()
```

Screenshot 15

16. Plot training loss vs validation loss for Placement model

```
plt.figure(figsize=(12, 6))
plt.plot(history_placement.history['loss'], label='Training Loss')
plt.plot(history_placement.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss vs Validation Loss for Placement Model')
plt.legend()
plt.show()
```

Screenshot 16

17. Function for prediction of CGPA and Placement

```
def predict_student_cgpa(student_index):
    student_data_a = X_cgpa.iloc[student_index]
    cgpa_input = np.array(student_data_a).reshape((1, len(student_data_a), 1))
    predicted_cgpa = final_cgpa_model.predict(cgpa_input)
    print('Predicted CGPA:', predicted_cgpa[0][0])

    # Inverse transform to get the predicted CGPA on the original scale
    y_cgpa = data['CGPA after 8th semester']
    y_train_cgpa = np.array(y_cgpa.values)
    scaler = StandardScaler()
    y_train_cgpa_scaled = scaler.fit_transform(y_train_cgpa.reshape(-1, 1))
    predicted_cgpa_original = scaler.inverse_transform(predicted_cgpa)
    print('Predicted CGPA (original scale):', predicted_cgpa_original[0][0])

def predict_student_placement(student_index, threshold=optimal_threshold):
    student_data_b = X_placement.iloc[student_index]
    placement_input = np.array(student_data_b).reshape((1, len(student_data_b), 1))
    predicted_placement_prob = final_placement_model.predict(placement_input)
    predicted_placement = int(predicted_placement_prob >= threshold)
    print('Predicted Placement:', 'Yes' if predicted_placement == 1 else 'No')
```

Screenshot 17

RESULTS AND DISCUSSION

- **1. CGPA Prediction:**
  - The genetic algorithm optimized model achieved a final mean squared error (MSE) of 0.0126 on the test set, reflecting its ability to predict CGPA with minimal error. The best LSTM configuration included 200 unit , tanh activation, 0.4047 dropout,
  - and Adam optimizer with a learning rate of 0.00761.

- **2. Placement Prediction:**
  - The placement model showed accuracy of 0.9324 , with the ROC-AUC providing a solid indication of classifier performance. The optimal threshold for placement
  - prediction was determined using Youden's J statistic, improving the model's practical application in determining whether a student will be placed.

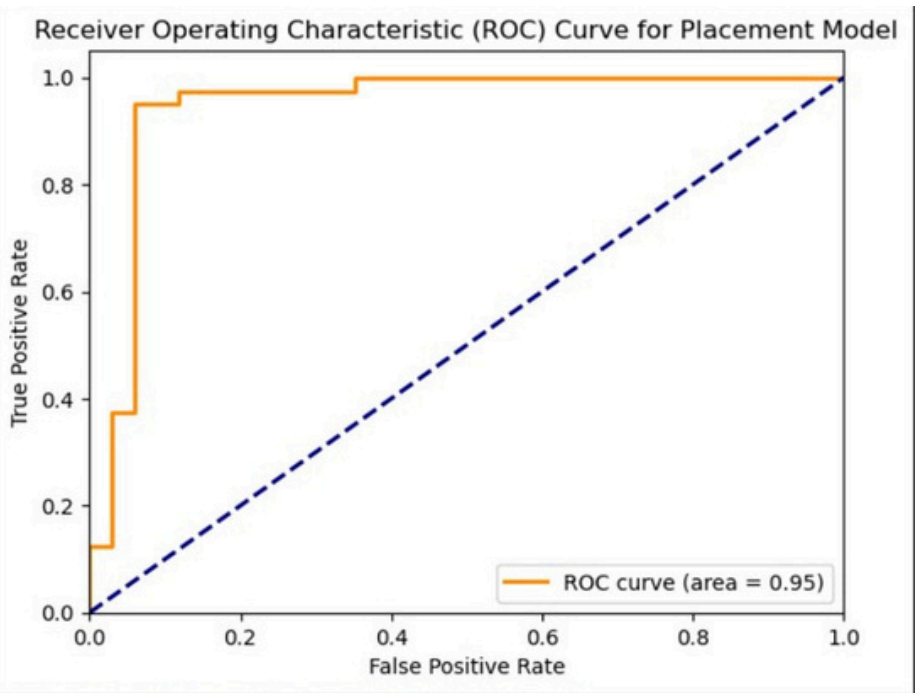
- **3. General Observations:**
  - Both models benefit from proper hyperparameter tuning via genetic algorithms, showcasing improved performance.
  - Overfitting is controlled using dropout, L2 regularization, and early stopping, ensuring the models generalize well to unseen data.

RESULT: `Final MSE for CGPA: 0.012644040398299694`

Screenshot 18

RESULT: `Final Accuracy for Placement: 0.9324324131011963`

Screenshot 19

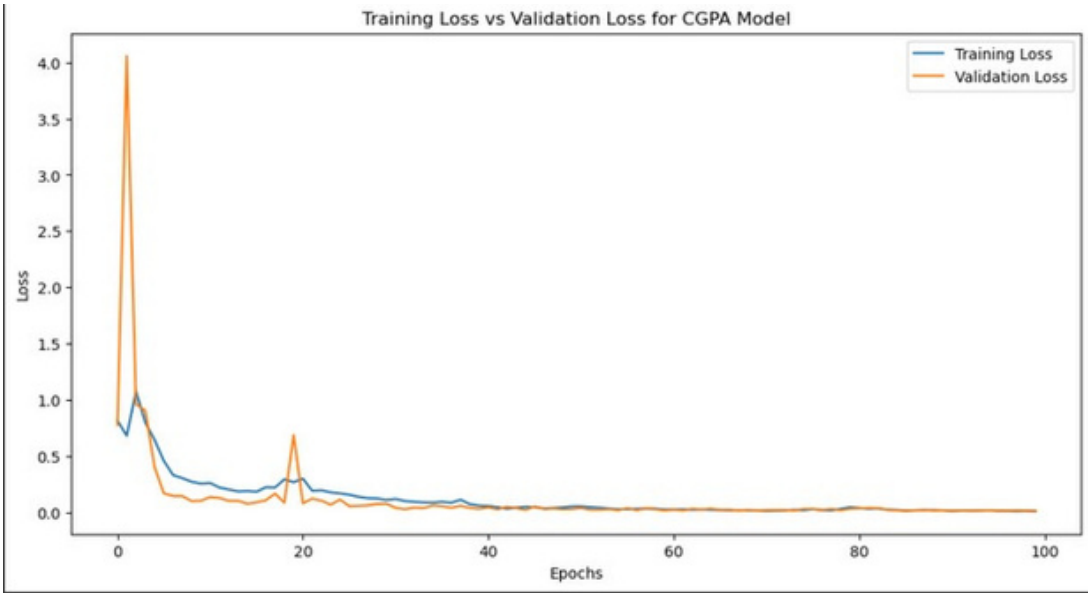


Screenshot 20

The placement model's ROC curve demonstrates strong performance, as indicated by an Area Under the Curve (AUC) of 0.95. This high AUC score suggests that the model is highly capable of distinguishing between the positive (placement) and negative classes with great accuracy.

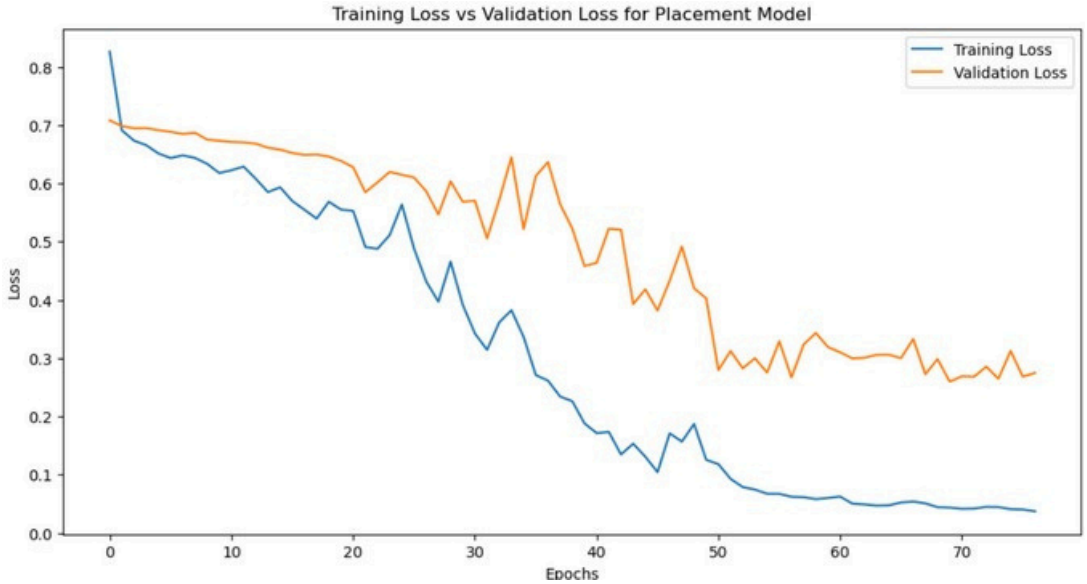
An AUC of 0.95 means that the model has a 95% chance of correctly ranking a randomly chosen positive instance higher than a randomly chosen negative one. The curve closely follows the top-left corner, which is ideal for ROC curves, indicating a low false positive rate along with a high true positive rate. Additionally, the optimal threshold of 0.6065 helps to fine-tune the decision boundary for making accurate placement predictions

Thus,the placement model performs very well in terms of classification ability, making it a reliable predictor for the placement outcome.



Screenshot 21

Based on the training and validation loss curve for the CGPA prediction model, The CGPA prediction model exhibits effective learning and strong generalization. Initially, both training and validation losses decrease rapidly, reflecting the model's quick adaptation to the data. After around 20 epochs, losses stabilize at low levels and remain close to each other, indicating that the model captures underlying patterns without overfitting or underfitting. Although there is a slight fluctuation in validation loss around the 20th epoch, it quickly returns to a low value, aligning with training loss. By around 30 epochs, the training and validation losses converge, showing the model is well-tuned, with minimal risk of overfitting. This stable alignment of loss values suggests the model reliably generalizes to unseen data, accurately mapping the relationship between features and CGPA



Screenshot 22

The training and validation loss curve for the placement model demonstrates a steady decline in training loss, indicating that the model is learning effectively on the training data. The validation loss also decreases overall, though with some fluctuations, suggesting that the model generalizes reasonably well. The consistent gap between training and validation loss may point to a slight overfitting, but the model shows solid performance across epochs.

TESTING

TEST:

```
student=data.iloc[0]
print("original CGPA:",student["CGPA after 8th semester"])
print("original placement:",student["Cam_plc"])

predict_student_cgpa(0)
predict_student_placement(0)
```

```
original CGPA: 6.3
original placement: yes
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Predicted CGPA: -0.70290637
Predicted CGPA (original scale): 6.3516536
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Predicted Placement: Yes
```

Screenshot 23

TEST:

```
student=data.iloc[20]
print("original CGPA:",student["CGPA after 8th semester"])
print("original placement:",student["Cam_plc"])

predict_student_cgpa(20)
predict_student_placement(20)
```

```
original CGPA: 8.2
original placement: yes
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
Predicted CGPA: 1.4116232
Predicted CGPA (original scale): 8.293216
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
Predicted Placement: Yes
```

Screenshot 24

TEST:

```
student=data.iloc[100]
print("original CGPA:",student["CGPA after 8th semester"])
print("original placement:",student["Cam_plc"])

predict_student_cgpa(100)
predict_student_placement(100)
```

```
original CGPA: 7.66
original placement: no
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
Predicted CGPA: 0.8765051
Predicted CGPA (original scale): 7.8018694
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
Predicted Placement: No
```

Screenshot 25

The test cases demonstrate that our predictive models for CGPA and placement outcomes perform effectively and achieve reliable results. The LSTM-based model for CGPA prediction consistently produces accurate estimates that closely match the actual CGPA values, while the Bidirectional LSTM model for placement prediction correctly identifies placement outcomes.

CONCLUSION

The predictive models developed for CGPA and placement outcomes demonstrate strong accuracy and reliability, proving effective across a wide range of student profiles. Our LSTM model accurately predicts CGPA, aligning closely with actual academic performance, while the Bidirectional LSTM model reliably identifies placement outcomes. This performance consistency indicates the robustness of our model design, including the preprocessing steps that helped the models recognize meaningful patterns within the data.

The success of these models highlights their practical utility for educational institutions, enabling more precise forecasts of student success and placement potential. By leveraging these insights, institutions can enhance support strategies, tailor academic interventions, and optimize career planning efforts to improve overall student outcomes.

In conclusion, our findings confirm that these models can serve as valuable tools in educational contexts, supporting data-driven decision-making to foster student success and employability.

FUTURE DIRECTIONS

To further enhance our predictive models for CGPA and placement outcomes, several future directions can be explored:

- Collecting and incorporating data from multiple institutions would allow the models to generalize better across diverse educational settings, enhancing their robustness and applicability. Integrating these models with a
- recommendation system could provide personalized academic and career guidance to students Developing a real-time prediction framework would
- allow continuous monitoring of student progress, providing institutions with timely insights for proactive intervention and support.