**TRIBHUVAN UNIVERSITY**

LUMBINI ICT CAMPUS

Gaindakot-4, Nawalpur



**Lab report on:**

**Artificial Intelligence**

**Submitted by**: **Submitted to:**

Name: Bipin Paudel Er. Bibek Chalise

Faculty: BCA Semester: VII

**Table of Contents**

[**Implementation of PROLOG** 3](#_Toc159872647)

[**Implementation of Breadth-first search algorithm.** 7](#_Toc159872648)

[**Implementation of Depth-first search algorithm.** 10](#_Toc159872649)

[**Implemention of Machine Learning** 12](#_Toc159872650)

[**Implemtation of Neutral Networks** 16](#_Toc159872651)

# **Implementation of PROLOG**

**Introduction**

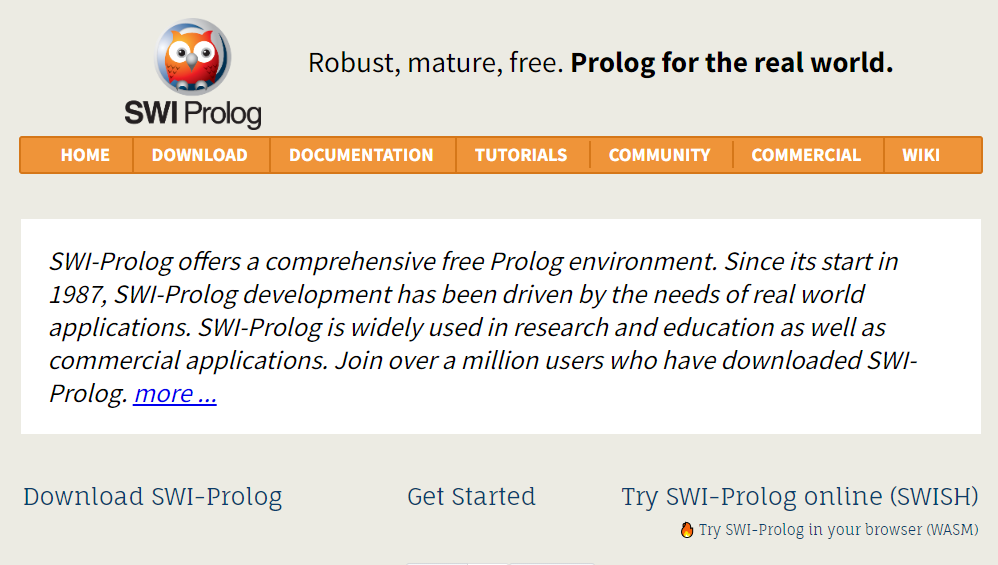
Prolog (Programming in Logic) is a general-purpose programming language that is based on the principles of logic programming. It is widely used for tasks such as natural language processing, automated reasoning, and symbolic computation.

Prolog has a declarative programming style, which means that you specify what you want the program to do, rather than how to do it. The Prolog interpreter then uses a set of logical rules and facts to deduce how to achieve the desired result. This approach is different from most imperative programming languages, which require you to specify how the program should execute each step-in detail.

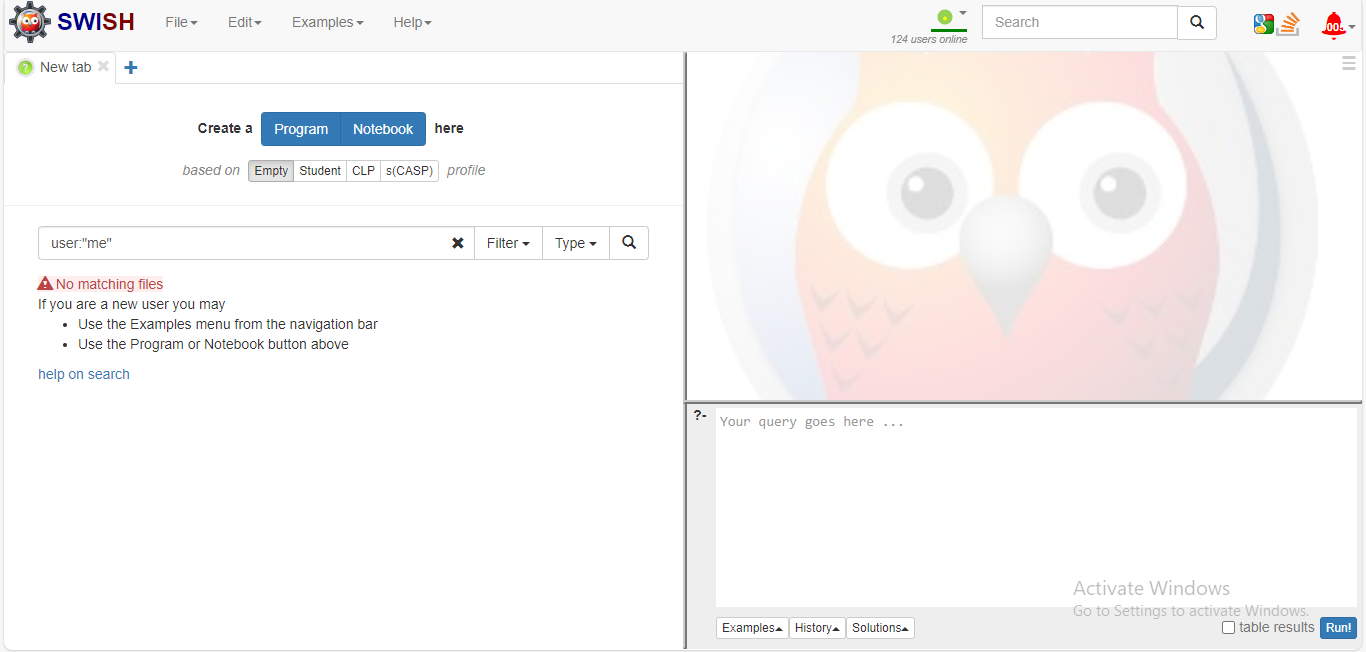
Prolog is particularly well-suited for tasks that involve searching for solutions to complex problems or queries, such as solving puzzles or answering questions based on a set of facts. It is also often used in artificial intelligence and expert systems applications.

Prolog has a syntax that is based on logical statements and operators, and it includes built-in support for data types such as atoms, numbers, and lists. It also includes a range of built-in predicates and control structures that can be used to manipulate and reason about data.

**Prolog Code Editor:**

1. Go to Offical Website of Prolog <https://www.swi-prolog.org/>

1. Click to the Try swi-prolog online. Which will redirect to prolog web editor page.



1. Click on create a program and it will show textarea to write prolog code of facts and rules. The query code should be written in bottom right text editor.

**Demonstration of Fact Table**

**Facts**

A fact is a statement that represents a piece of information or a relationship between different pieces of information. Facts are written in the form of logical statements and are stored in a Prolog program or database.

For example, a fact might be "John is a father," which represents the relationship between the person "John" and the role "father”. In Prolog syntax, this fact would be written as: father(john).

**Rules**

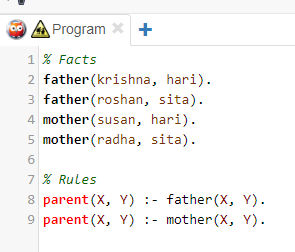
A rule is a logical statement that defines a relationship or pattern between different facts or variables. Rules are written in the form of a head and a body, separated by a colon. The head is a logical statement that represents the conclusion or result of the rule, and the body is a list of logical statements that represent the conditions or premises that must be satisfied in order for the rule to be applied.

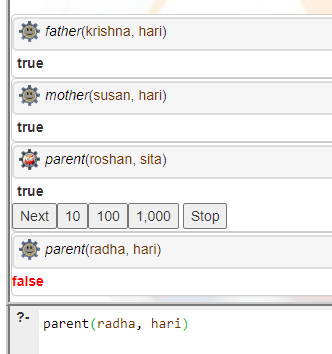
For example, a rule might be "if X is a parent and Y is a child of X, then X is a mother or father of Y." In Prolog syntax, this rule would be written as:

parent\_of(X, Y) :- mother(X, Y).

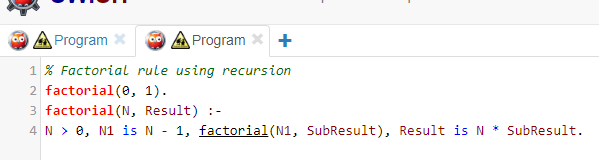
parent\_of(X, Y) :- father(X, Y).

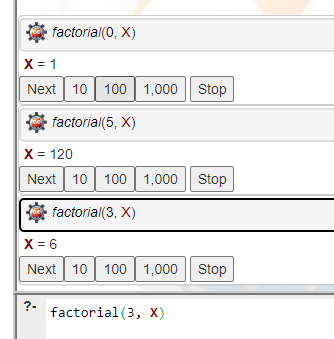
Facts and rules are used together in Prolog programs to represent and reason about knowledge. Prolog's built-in search and unification algorithms can be used to query and manipulate the facts and rules in a program to find solutions to problems or answer questions.

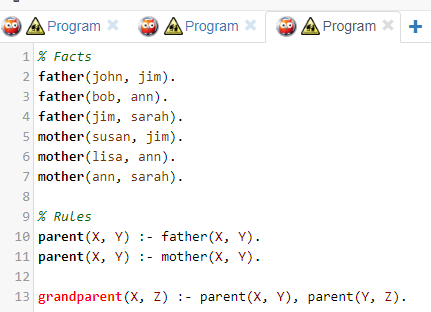
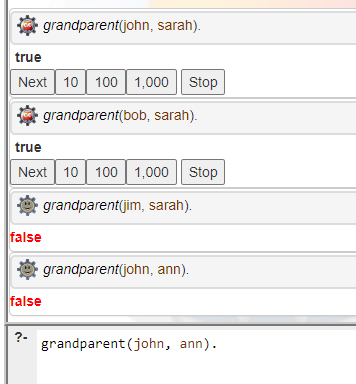
**Example 1 (Family Relationship) :**



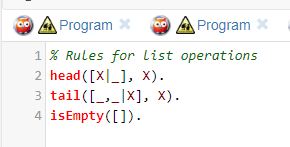
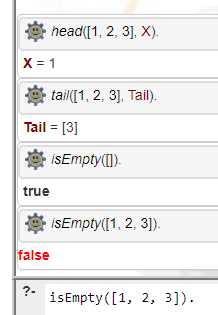
**Example 2 (Factorial using recursion):**





**Example 3 (Family Relationships) :**

**Example 4 (List Operations) :**

# **Implementation of Breadth-first search algorithm.**

**Theory:**

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

**Code:**

*#include <stdio.h>*

*#include <stdlib.h>*

*#define SIZE 40*

*struct queue {*

*int items[SIZE];*

*int front;*

*int rear;*

*};*

*struct queue\* createQueue();*

*void enqueue(struct queue\* q, int);*

*int dequeue(struct queue\* q);*

*void display(struct queue\* q);*

*int isEmpty(struct queue\* q);*

*void printQueue(struct queue\* q);*

*struct node {*

*int vertex;*

*struct node\* next;*

*};*

*struct node\* createNode(int);*

*struct Graph {*

*int numVertices;*

*struct node\*\* adjLists;*

*int\* visited;*

*};*

*// BFS algorithm*

*void bfs(struct Graph\* graph, int*

*startVertex) {*

*struct queue\* q = createQueue();*

*graph->visited[startVertex] = 1;*

*enqueue(q, startVertex);*

*while (!isEmpty(q)) {*

*printQueue(q);*

*int currentVertex = dequeue(q);*

*printf("Visited %d\n", currentVertex);*

*struct node\* temp = graph->adjLists[currentVertex];*

*while (temp) {*

*int adjVertex = temp->vertex;*

*if (graph->visited[adjVertex] == 0) {*

*graph->visited[adjVertex] = 1;*

*enqueue(q, adjVertex);*

*}*

*temp = temp->next;*

*}*

*}*

*}*

*// Creating a node*

*struct node\* createNode(int v) {*

*struct node\* newNode =*

*malloc(sizeof(struct node));*

*newNode->vertex = v;*

*newNode->next = NULL;*

*return newNode;*

*}*

*// Creating a graph*

*struct Graph\* createGraph(int vertices) {*

*struct Graph\* graph = malloc(sizeof(struct*

*Graph));*

*graph->numVertices = vertices;*

*graph->adjLists = malloc(vertices \**

*sizeof(struct node\*));*

*graph->visited = malloc(vertices \**

*sizeof(int));*

*int i;*

*for (i = 0; i < vertices; i++) {*

*graph->adjLists[i] = NULL;*

*graph->visited[i] = 0;*

*}*

*return graph;*

*}*

*// Add edge*

*void addEdge(struct Graph\* graph, int src,*

*int dest) {*

*// Add edge from src to dest*

*struct node\* newNode = createNode(dest);*

*newNode->next = graph->adjLists[src];*

*graph->adjLists[src] = newNode;*

*// Add edge from dest to src*

*newNode = createNode(src);*

*newNode->next = graph->adjLists[dest];*

*graph->adjLists[dest] = newNode;*

*}*

*// Create a queue*

*struct queue\* createQueue() {*

*struct queue\* q = malloc(sizeof(struct*

*queue));*

*q->front = -1;*

*q->rear = -1;*

*return q;*

*}*

*// Check if the queue is empty*

*int isEmpty(struct queue\* q) {*

*if (q->rear == -1)*

*return 1;*

*else*

*return 0;*

*}*

*// Adding elements into queue*

*void enqueue(struct queue\* q, int value) {*

*if (q->rear == SIZE - 1)*

*printf("\nQueue is Full!!");*

*else {*

*if (q->front == -1)*

*q->front = 0;*

*q->rear++;*

*q->items[q->rear] = value;*

*}*

*}*

*// Removing elements from queue*

*int dequeue(struct queue\* q) {*

*int item;*

*if (isEmpty(q)) {*

*printf("Queue is empty");*

*item = -1;*

*} else {*

*item = q->items[q->front];*

*q->front++;*

*if (q->front > q->rear) {*

*printf("Resetting queue ");*

*q->front = q->rear = -1;*

*}*

*}*

*return item;*

*}*

*// Print the queue*

*void printQueue(struct queue\* q) {*

*int i = q->front;*

*if (isEmpty(q)) {*

*printf("Queue is empty");*

*} else {*

*printf("\nQueue contains \n");*

*for (i = q->front; i < q->rear + 1; i++) {*

*printf("%d ", q->items[i]);*

*}*

*}*

*}*

*int main() {*

*struct Graph\* graph = createGraph(6);*

*addEdge(graph, 0, 1);*

*addEdge(graph, 0, 2);*

*addEdge(graph, 1, 2);*

*addEdge(graph, 1, 4);*

*addEdge(graph, 1, 3);*

*addEdge(graph, 2, 4);*

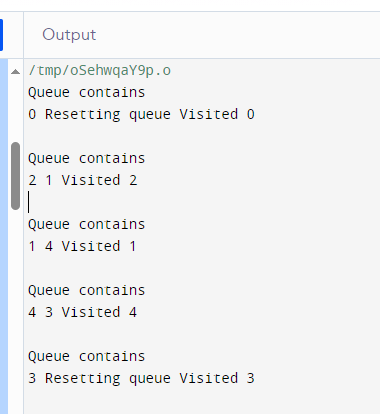
*addEdge(graph, 3, 4);*

*bfs(graph, 0);*

*return 0;*

*}*

**Output:**

****

# **Implementation of Depth-first search algorithm.**

**Theory:**

DFS (Depth-first search) is a technique used for traversing trees or graphs. Here backtracking is used for traversal. In this traversal first, the deepest node is visited and then backtracks to its parent node if no sibling of that node exists

**Code:**

*#include <stdio.h>*

*#include <stdlib.h>*

*struct node {*

*int vertex;*

*struct node\* next;*

*};*

*struct Graph {*

*int numVertices;*

*int\* visited;*

*struct node\*\* adjLists; // Corrected syntax here*

*};*

*void DFS(struct Graph\* graph, int vertex);*

*struct node\* createNode(int v);*

*struct Graph\* createGraph(int vertices);*

*void addEdge(struct Graph\* graph, int src, int dest);*

*void printGraph(struct Graph\* graph);*

*// DFS algorithm*

*void DFS(struct Graph\* graph, int vertex) {*

*struct node\* adjList = graph->adjLists[vertex];*

*struct node\* temp = adjList;*

*graph->visited[vertex] = 1;*

*printf("Visited %d \n", vertex);*

*while (temp != NULL) {*

*int connectedVertex = temp->vertex;*

*if (graph->visited[connectedVertex] == 0) {*

*DFS(graph, connectedVertex);*

*}*

*temp = temp->next;*

*}*

*}*

*// Create a node*

*struct node\* createNode(int v) {*

*struct node\* newNode = malloc(sizeof(struct node));*

*newNode->vertex = v;*

*newNode->next = NULL;*

*return newNode;*

*}*

*// Create graph*

*struct Graph\* createGraph(int vertices) {*

*struct Graph\* graph = malloc(sizeof(struct Graph));*

*graph->numVertices = vertices;*

*graph->adjLists = malloc(vertices \* sizeof(struct node\*));*

*graph->visited = malloc(vertices \* sizeof(int));*

*int i;*

*for (i = 0; i < vertices; i++) {*

*graph->adjLists[i] = NULL;*

*graph->visited[i] = 0;*

*}*

*return graph;*

*}*

*// Add edge*

*void addEdge(struct Graph\* graph, int src, int dest) {*

*// Add edge from src to dest*

*struct node\* newNode = createNode(dest);*

*newNode->next = graph->adjLists[src];*

*graph->adjLists[src] = newNode;*

*// Add edge from dest to src*

*newNode = createNode(src);*

*newNode->next = graph->adjLists[dest];*

*graph->adjLists[dest] = newNode;*

*}*

*// Print the graph*

*void printGraph(struct Graph\* graph) {*

*int v;*

*for (v = 0; v < graph->numVertices; v++) {*

*struct node\* temp = graph->adjLists[v];*

*printf("\n Adjacency list of vertex %d\n ", v);*

*while (temp) {*

*printf("%d -> ", temp->vertex);*

*temp = temp->next;*

*}*

*printf("\n");*

*}*

*}*

*int main() {*

*struct Graph\* graph = createGraph(4);*

*addEdge(graph, 0, 1);*

*addEdge(graph, 0, 2);*

*addEdge(graph, 1, 2);*

*addEdge(graph, 2, 3);*

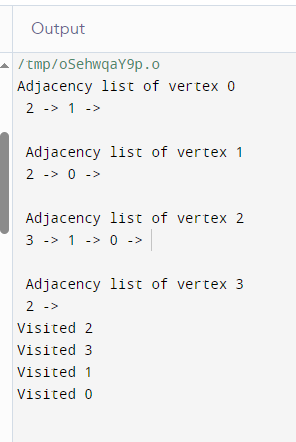
*printGraph(graph);*

*DFS(graph, 2);*

*return 0;*

*}*

**Output**

****

# **Implemention of Machine Learning**

**Machine Learning**

Machine learning is a subset of artificial intelligence that gives systems the ability to learn and optimize processes without having to be consistently programmed. Simply put, machine learning uses data, statistics and trial and error to “learn” a specific task without ever having to be specifically coded for the task.

1. **Supervised Learning**: Learns from labeled data to make predictions.
2. **Unsupervised Learning:** Finds patterns in unlabeled data.
3. **Reinforcement Learning:** Learns by interacting with an environment and receiving rewards or penalties.

Machine learning is used in various fields to make computers improve their performance on tasks by learning from data.

**Supervised Learning:**

Supervised learning is a category of machine learning that uses labeled datasets to train algorithms to predict outcomes and recognize patterns.

**Types of supervised learning**

Supervised learning in machine learning is generally divided into two categories: classification and regression.

**Classification**

Classification algorithms are used to group data by predicting a categorical label or output variable based on the input data. Classification is used when output variables are categorical, meaning there are two or more classes.

**Program 1: Classification Algorithim using Random Forest Classifier**

*# Import necessary libraries*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn.metrics import accuracy\_score, classification\_report*

*from sklearn.datasets import make\_classification*

*# Generate dummy data*

*X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=10, n\_clusters\_per\_class=2, random\_state=42)*

*# Split the data into training and testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Create a RandomForestClassifier*

*classifier = RandomForestClassifier(random\_state=42)*

*# Train the classifier on the training data*

*classifier.fit(X\_train, y\_train)*

*# Make predictions on the test data*

*y\_pred = classifier.predict(X\_test)*

*# Evaluate the classifier*

*accuracy = accuracy\_score(y\_test, y\_pred)*

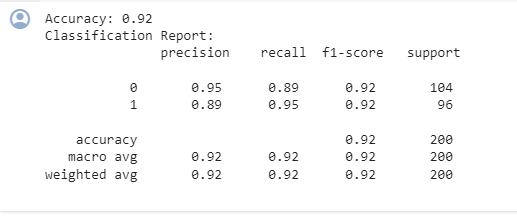
*classification\_rep = classification\_report(y\_test, y\_pred)*

*# Print the results*

*print("Accuracy:", accuracy)*

*print("Classification Report:\n", classification\_rep)*

Output:



Program 2: New Data Input For Classification

*# Assuming 'new\_data' is your new input data for classification*

*new\_data = X\_test[:5] # Example: Use the first 5 samples from the test set*

*# Make predictions using the trained classifier*

*classification\_predictions = classifier.predict(new\_data)*

*# Print the classification predictions*

*print("Classification Predictions:", classification\_predictions)*

Output:



**Regression**

Regression algorithms are used to predict a real or continuous value, where the algorithm detects a relationship between two or more variables.

A common example of a regression task might be predicting a salary based on work experience. For instance, a supervised learning algorithm would be fed inputs related to work experience (e.g., length of time, the industry or field, location, etc.) and the corresponding assigned salary amount. After the model is trained, it could be used to predict the average salary based on work experience.

**Program 3: Regression Algorithm using Random Forest Regressor**

*# Import necessary libraries*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.ensemble import RandomForestRegressor*

*from sklearn.metrics import mean\_squared\_error*

*from sklearn.datasets import make\_regression*

*# Generate dummy data for regression*

*X, y = make\_regression(n\_samples=1000, n\_features=20, noise=0.1, random\_state=42)*

*# Split the data into training and testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Create a RandomForestRegressor*

*regressor = RandomForestRegressor(random\_state=42)*

*# Train the regressor on the training data*

*regressor.fit(X\_train, y\_train)*

*# Make predictions on the test data*

*y\_pred = regressor.predict(X\_test)*

*# Evaluate the regressor*

*mse = mean\_squared\_error(y\_test, y\_pred)*

*# Print the results*

*print("Mean Squared Error:", mse)*

**Output:**

****

**Program 4: New Data Input for Regression**

*# Assuming 'new\_data' is your new input data for regression*

*new\_data = X\_test[:5] # Example: Use the first 5 samples from the test set*

*# Make predictions using the trained regressor*

*regression\_predictions = regressor.predict(new\_data)*

*# Print the regression predictions*

*print("Regression Predictions:", regression\_predictions)*

**Output:**

****

# **Implemtation of Neutral Networks**

**Neutrol Network**

A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain. Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex.

**Artifical Neural Network**

An Artificial Neural Network (ANN) is a computational model inspired by the human brain’s neural structure. It consists of interconnected nodes (neurons) organized into layers. Information flows through these nodes, and the network adjusts the connection strengths (weights) during training to learn from data, enabling it to recognize patterns, make predictions, and solve various tasks in machine learning and artificial intelligence.

**Program 1: Simple ANN Program**

*# Import necessary libraries*

*import numpy as np*

*import tensorflow as tf*

*from tensorflow import keras*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.datasets import make\_classification*

*# Generate dummy data for classification*

*X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=10, n\_clusters\_per\_class=2, random\_state=42)*

*# Split the data into training and testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Standardize the data*

*scaler = StandardScaler()*

*X\_train = scaler.fit\_transform(X\_train)*

*X\_test = scaler.transform(X\_test)*

*# Build the ANN model*

*model = keras.Sequential([*

*keras.layers.Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),*

*keras.layers.Dense(1, activation='sigmoid')*

*])*

*# Compile the model*

*model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])*

*# Train the model*

*model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)*

*# Evaluate the model on the test set*

*loss, accuracy = model.evaluate(X\_test, y\_test)*

*print("Test Accuracy:", accuracy)*

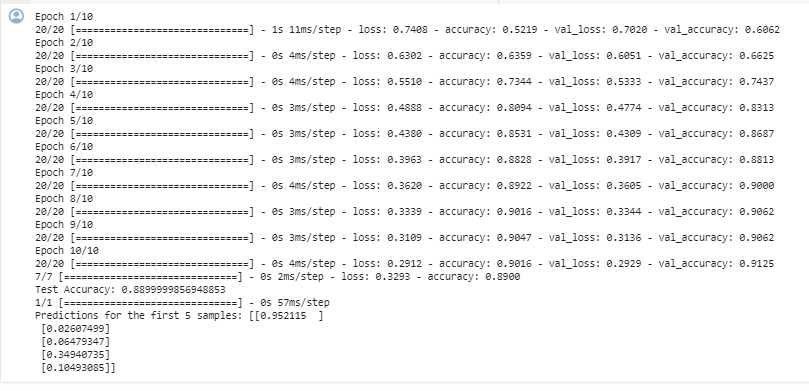
*# Make predictions on new data*

*new\_data = X\_test[:5]*

*predictions = model.predict(new\_data)*

*print("Predictions for the first 5 samples:", predictions)*

**Output:**



Program 2: Program of ANN with Hidden Layers

*# Import necessary libraries*

*import numpy as np*

*import tensorflow as tf*

*from tensorflow import keras*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.datasets import make\_classification*

*# Generate dummy data for classification*

*X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=10, n\_clusters\_per\_class=2, random\_state=42)*

*# Split the data into training and testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Standardize the data*

*scaler = StandardScaler()*

*X\_train = scaler.fit\_transform(X\_train)*

*X\_test = scaler.transform(X\_test)*

*# Build a more advanced ANN model*

*model = keras.Sequential([*

*keras.layers.Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)),*

*keras.layers.Dropout(0.5), # Dropout layer for regularization*

*keras.layers.Dense(64, activation='relu'),*

*keras.layers.BatchNormalization(), # Batch normalization for better convergence*

*keras.layers.Dense(32, activation='relu'),*

*keras.layers.Dropout(0.3), # Another dropout layer*

*keras.layers.Dense(1, activation='sigmoid')*

*])*

*# Compile the model*

*model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])*

*# Train the model*

*model.fit(X\_train, y\_train, epochs=20, batch\_size=64, validation\_split=0.2)*

*# Evaluate the model on the test set*

*loss, accuracy = model.evaluate(X\_test, y\_test)*

*print("Test Accuracy:", accuracy)*

*# Make predictions on new data*

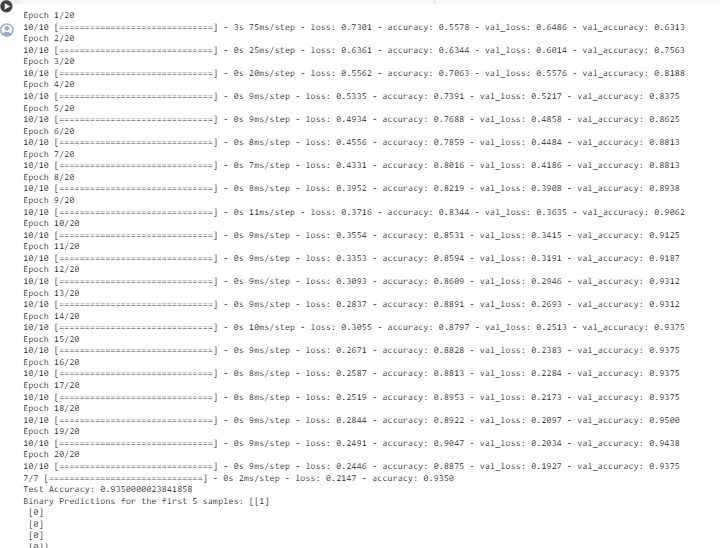
*new\_data = X\_test[:5]*

*# Apply a threshold to convert probabilities to binary predictions*

*binary\_predictions = (predictions > 0.5).astype(int)*

*print("Binary Predictions for the first 5 samples:", binary\_predictions)*

**Output:**

****