Name : Shreya Singh	Class/Roll No. :55/D16AD	Grade :

Title of Experiment :

Design and implement a fully connected deep neural network with at least 2 hidden layers for a classification application. Use appropriate Learning Algorithm, output function and loss function.

Objective of Experiment :

The objective of the Iris flower dataset classification problem is to develop a deep learning model that can accurately classify iris flowers into one of three species based on their four features: sepal length, sepal width, petal length, and petal width. The model should learn to differentiate between the different species (setosa, versicolor, and virginica) by training on a labeled dataset and achieve high accuracy on unseen data during testing

Outcome of Experiment :

The experiment yielded a trained deep neural network model that achieved a certain accuracy on the Iris flower test dataset. This accuracy reflects the model's capability to effectively classify iris flowers into their respective species based on their feature attributes.

Problem Statement :

Design and implement a fully connected deep neural network with at least 2 hidden layers for a iris flower dataset classification problem. Use appropriate Learning Algorithm, output function and loss function.

Description / Theory :

Fully Connected Deep Neural Network: A fully connected deep neural network, also known as a multi-layer perceptron (MLP), consists of multiple layers of interconnected nodes or neurons. Each neuron in a layer is connected to every neuron in the subsequent layer. The layers typically include an input layer, one or more hidden layers, and an output layer. These layers collectively allow the network to learn complex relationships between input data and desired output labels through a process known as forward and backward propagation.

Activation Functions: Activation functions introduce non-linearity to the neural network, enabling it to learn and approximate non-linear relationships within the data. In this implementation, the Rectified Linear Unit (ReLU) activation function is used for the hidden layers. It replaces negative values with zero and allows positive values to pass through unchanged, helping the network capture complex patterns effectively.

Output Layer Activation Function: The softmax activation function is used in the output layer for multi-class classification problems. It converts the raw output scores of the network into probability distributions across different classes. This function ensures that the sum of probabilities for all classes is equal to 1, making it suitable for selecting the most likely class prediction.

Loss Function: The categorical cross-entropy loss function is utilized in this classification problem. It measures the dissimilarity between predicted probability distributions and the true one-hot encoded class labels. Minimizing this loss during training aligns the model's predicted probabilities with the actual class label.

Program :

```
import numpy as np
import tensorflow as tf
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.utils import to_categorical



# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target


# Convert labels to one-hot encoded format
y_onehot = to_categorical(y, num_classes=3)


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.2, random_state=42)


# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Build the neural network model
model = Sequential([
Dense(64, activation='relu', input_shape=(4,)),
Dense(32, activation='relu'),
Dense(3, activation='softmax')
])


# Compile the model
model.compile(optimizer=SGD(learning_rate=0.01),
loss=CategoricalCrossentropy(),
metrics=['accuracy'])



# Train the model
model.fit(X_train_scaled, y_train, epochs=100, batch_size=16, validation_split=0.1)
```

```
Epoch 75/100
7/7 [==============================] - 0s 6ms/step - loss: 0.2811 - accuracy: 0.8889 - val_loss: 0.4319 - val_accuracy: 0.9167
Epoch 76/100
7/7 [==============================] - 0s 6ms/step - loss: 0.2787 - accuracy: 0.8981 - val_loss: 0.4302 - val_accuracy: 0.9167
Epoch 77/100
7/7 [==============================] - 0s 6ms/step - loss: 0.2765 - accuracy: 0.9167 - val_loss: 0.4276 - val_accuracy: 0.9167
Epoch 78/100
7/7 [==============================] - 0s 7ms/step - loss: 0.2745 - accuracy: 0.9074 - val_loss: 0.4253 - val_accuracy: 0.9167
Epoch 79/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2723 - accuracy: 0.9074 - val_loss: 0.4236 - val_accuracy: 0.9167
Epoch 80/100
7/7 [==============================] - 0s 7ms/step - loss: 0.2699 - accuracy: 0.9167 - val_loss: 0.4215 - val_accuracy: 0.9167
Epoch 81/100
7/7 [==============================] - 0s 7ms/step - loss: 0.2682 - accuracy: 0.9167 - val_loss: 0.4192 - val_accuracy: 0.9167
Epoch 82/100
7/7 [==============================] - 0s 7ms/step - loss: 0.2656 - accuracy: 0.9167 - val_loss: 0.4172 - val_accuracy: 0.9167
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

```
1/1 [==============================] - 0s 87ms/step - loss: 0.1872 - accuracy: 0.9667
Test loss: 0.1871630996465683
Test accuracy: 0.9666666388511658
```

Results and Discussions :

The trained deep neural network achieved an accuracy of around 95% on the test dataset, demonstrating its effectiveness in classifying iris flowers into their respective species. The accuracy suggests that the network successfully learned the intricate relationships between the input features and the target classes. Although the model's performance is satisfactory, further exploration could involve hyperparameter tuning to optimize aspects such as learning rate, batch size, and number of neurons in hidden layers. This could potentially lead to even higher accuracy and improved generalization. Inclusion of techniques like dropout and regularization might further enhance the model's robustness against overfitting.