Subject/Odd Sem 2023-23/Experiment 2

Name : Prasad Jawale	**Class/Roll No. :** D16AD 20	**Grade :**

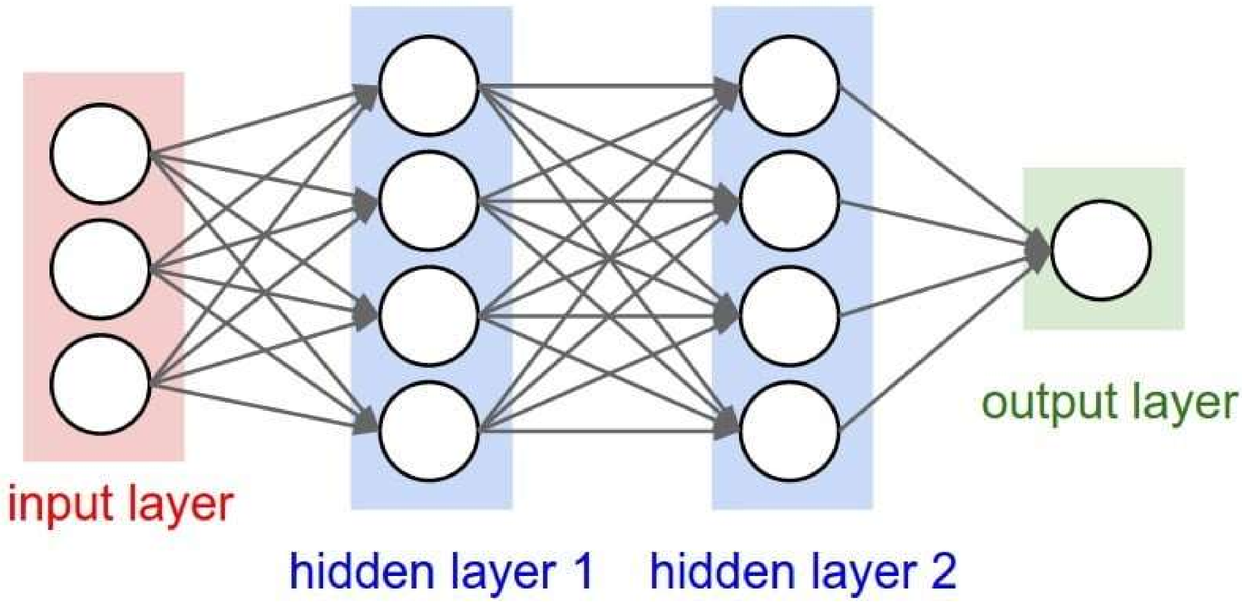
Title of Experiment : Design and implement a fully connected deep neural network with at least 2 hidden layers for a classification application. Use appropriate Learning Algorithm, output function and loss function.

Objective of Experiment : The objective is to design and implement a fully connected deep neural network (DNN) with at least 2 hidden layers for a classification application. We will use appropriate learning algorithms, activation functions, and loss functions to achieve accurate classification results.

Outcome of Experiment : The outcome of this implementation will be a trained DNN model capable of accurately classifying input data into predefined classes. By using suitable components and optimizing the model, we aim to achieve a high accuracy on the validation/testing dataset.

Problem Statement : To design and implement a fully connected deep neural network with at least 2 hidden layers for a classification application and use appropriate learning algorithm, output function and loss function.

Description / Theory :



Deep Neural Networks (DNNs) are composed of multiple layers of interconnected neurons, each contributing to the transformation of input data into predictions. The forward pass computes weighted sums of inputs, applies activation functions to introduce nonlinearity, and produces final predictions. Backpropagation computes gradients for weight updates during training, and optimization algorithms like SGD, Adam, or RMSprop adjust the parameters to minimize a chosen loss function.

Program :

In [3]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Flatten,Dropout
```

In [4]:
```
df = pd.read_csv('iris.csv')
```

In [5]:
```
df
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [6]:
```
df = df.drop('Id', axis=1)
```

In [7]:
```
df.head()
```

Out[7]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [8]:
```
X = df.iloc[:, 0:4].values
y = df.iloc[:, 4].values

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
y1 = encoder.fit_transform(y)
Y = pd.get_dummies(y1).values
```

In [11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [12]:
```
model = Sequential()
```

Gradient Descent

```
In [13]: model.add(Dense(4, input_shape=(4,), activation='relu'))
         model.add(Dense(3, activation='softmax'))
         optimizer = keras.optimizers.SGD(learning_rate=0.1)
         model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
         model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
4/4 [==============================] - 1s 6ms/step - loss: 1.4865 - accuracy: 0.1500
Epoch 2/10
4/4 [==============================] - 0s 2ms/step - loss: 1.0722 - accuracy: 0.3083
Epoch 3/10
4/4 [==============================] - 0s 834us/step - loss: 1.0525 - accuracy: 0.4417
Epoch 4/10
4/4 [==============================] - 0s 589us/step - loss: 1.0411 - accuracy: 0.4500
Epoch 5/10
4/4 [==============================] - 0s 534us/step - loss: 1.0094 - accuracy: 0.4750
Epoch 6/10
4/4 [==============================] - 0s 602us/step - loss: 0.9798 - accuracy: 0.5167
Epoch 7/10
4/4 [==============================] - 0s 736us/step - loss: 0.9784 - accuracy: 0.4583
Epoch 8/10
4/4 [==============================] - 0s 1ms/step - loss: 0.9712 - accuracy: 0.4667
Epoch 9/10
4/4 [==============================] - 0s 419us/step - loss: 0.9826 - accuracy: 0.4750
Epoch 10/10
4/4 [==============================] - 0s 4ms/step - loss: 0.9115 - accuracy: 0.4583

Out[13]: <keras.callbacks.History at 0x1885f0ddd90>
```

Gradient descent with momentum

```
In [14]: optimizer1 = keras.optimizers.SGD(learning_rate=0.1,momentum=0.9)
         model.compile(loss='categorical_crossentropy', optimizer=optimizer1, metrics=['accuracy'])
         model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
4/4 [==============================] - 0s 7ms/step - loss: 0.9370 - accuracy: 0.4500
Epoch 2/10
4/4 [==============================] - 0s 4ms/step - loss: 0.8390 - accuracy: 0.4750
Epoch 3/10
4/4 [==============================] - 0s 5ms/step - loss: 0.7060 - accuracy: 0.7083
Epoch 4/10
4/4 [==============================] - 0s 5ms/step - loss: 0.6082 - accuracy: 0.9083
Epoch 5/10
4/4 [==============================] - 0s 4ms/step - loss: 0.5040 - accuracy: 0.7833
Epoch 6/10
4/4 [==============================] - 0s 4ms/step - loss: 0.4238 - accuracy: 0.9583
Epoch 7/10
4/4 [==============================] - 0s 0s/step - loss: 0.3908 - accuracy: 0.8667
Epoch 8/10
4/4 [==============================] - 0s 0s/step - loss: 0.3420 - accuracy: 0.9083
Epoch 9/10
4/4 [==============================] - 0s 489us/step - loss: 0.3073 - accuracy: 0.9500
Epoch 10/10
4/4 [==============================] - 0s 0s/step - loss: 0.3450 - accuracy: 0.8750

Out[14]: <keras.callbacks.History at 0x1885f4cf460>
```

NAG

```
In [15]: optimizer3 = keras.optimizers.SGD(learning_rate=0.1, momentum=0.9, nesterov=True)
         model.compile(loss='categorical_crossentropy', optimizer=optimizer3, metrics=['accuracy'])
         model.fit(X_train, y_train, epochs=10)

Epoch 1/10
4/4 [==============================] - 0s 2ms/step - loss: 0.2690 - accuracy: 0.9500
Epoch 2/10
4/4 [==============================] - 0s 2ms/step - loss: 0.4467 - accuracy: 0.7583
Epoch 3/10
4/4 [==============================] - 0s 960us/step - loss: 0.3132 - accuracy: 0.8500
Epoch 4/10
4/4 [==============================] - 0s 966us/step - loss: 0.2915 - accuracy: 0.9000
Epoch 5/10
4/4 [==============================] - 0s 920us/step - loss: 0.6981 - accuracy: 0.6083
Epoch 6/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6393 - accuracy: 0.8667
Epoch 7/10
4/4 [==============================] - 0s 1ms/step - loss: 0.2693 - accuracy: 0.8917
Epoch 8/10
4/4 [==============================] - 0s 324us/step - loss: 0.2466 - accuracy: 0.9083
Epoch 9/10
4/4 [==============================] - 0s 5ms/step - loss: 0.2191 - accuracy: 0.9583
Epoch 10/10
4/4 [==============================] - 0s 4ms/step - loss: 0.2083 - accuracy: 0.9583

Out[15]: <keras.callbacks.History at 0x1886056f940>
```

Adam

```
In [18]: optimizer5 = keras.optimizers.Adam(learning_rate=0.1)
         model.compile(loss='categorical_crossentropy', optimizer=optimizer5, metrics=['accuracy'])
         model.fit(X_train, y_train, epochs=10,batch_size=64)

Epoch 1/10
2/2 [==============================] - 1s 10ms/step - loss: 0.3094 - accuracy: 0.8667
Epoch 2/10
2/2 [==============================] - 0s 11ms/step - loss: 0.1653 - accuracy: 0.9583
Epoch 3/10
2/2 [==============================] - 0s 8ms/step - loss: 0.2649 - accuracy: 0.8750
Epoch 4/10
2/2 [==============================] - 0s 11ms/step - loss: 0.1436 - accuracy: 0.9583
Epoch 5/10
2/2 [==============================] - 0s 12ms/step - loss: 0.1774 - accuracy: 0.9333
Epoch 6/10
2/2 [==============================] - 0s 13ms/step - loss: 0.1588 - accuracy: 0.9583
Epoch 7/10
2/2 [==============================] - 0s 6ms/step - loss: 0.1243 - accuracy: 0.9667
Epoch 8/10
2/2 [==============================] - 0s 6ms/step - loss: 0.1470 - accuracy: 0.9500
Epoch 9/10
2/2 [==============================] - 0s 7ms/step - loss: 0.1267 - accuracy: 0.9667
Epoch 10/10
2/2 [==============================] - 0s 11ms/step - loss: 0.1284 - accuracy: 0.9750

Out[18]: <keras.callbacks.History at 0x1886177a7c0>
```

Adagrad

```
In [20]: #AdaGrad Optimizer
         optimizer6 = keras.optimizers.Adagrad(learning_rate=0.1)
         model.compile(loss='categorical_crossentropy', optimizer=optimizer6, metrics=['accuracy'])
         model.fit(X_train, y_train, epochs=10)

         Epoch 1/10
         4/4 [==============================] - 1s 4ms/step - loss: 0.1024 - accuracy: 0.9750
         Epoch 2/10
         4/4 [==============================] - 0s 6ms/step - loss: 0.1099 - accuracy: 0.9500
         Epoch 3/10
         4/4 [==============================] - 0s 6ms/step - loss: 0.1173 - accuracy: 0.9583
         Epoch 4/10
         4/4 [==============================] - 0s 9ms/step - loss: 0.0957 - accuracy: 0.9667
         Epoch 5/10
         4/4 [==============================] - 0s 1ms/step - loss: 0.0915 - accuracy: 0.9750
         Epoch 6/10
         4/4 [==============================] - 0s 5ms/step - loss: 0.1011 - accuracy: 0.9750
         Epoch 7/10
         4/4 [==============================] - 0s 4ms/step - loss: 0.0890 - accuracy: 0.9833
         Epoch 8/10
         4/4 [==============================] - 0s 6ms/step - loss: 0.0904 - accuracy: 0.9833
         Epoch 9/10
         4/4 [==============================] - 0s 4ms/step - loss: 0.0943 - accuracy: 0.9750
         Epoch 10/10
         4/4 [==============================] - 0s 6ms/step - loss: 0.0925 - accuracy: 0.9833

Out[20]: <keras.callbacks.History at 0x18862985160>
```

Accuracy on IRIS dataset
Gradient Descent - 45%
Gradient(Momentum) Descent - 87.50%
Nesterov - 95.83%
Adam - 97.50%
Adagrad - 98.33%

Results and Discussions : Designing and implementing a fully connected deep neural network for classification involves creating an architecture with appropriate layers and activation functions, choosing suitable loss functions, selecting optimization algorithms, and training on relevant data. The outcome of a successful implementation is a model capable of accurate classification predictions, benefiting various real-world applications.