In [141…

```
import numpy as np

class ANN:
    def __init__(self,input_size,hidden_layers,hidden_neurons,output_size,learning_rate):
        self.weights=[]
        self.bias=[]
        self.hidden_layers=hidden_layers
        self.learning_rate=learning_rate;

        for i in range(hidden_layers+1):
            if i==0:
                self.weights.append(np.random.randn(hidden_neurons,input_size))
                self.bias.append(np.full((hidden_neurons,1),1))
            elif i==hidden_layers:
                self.weights.append(np.random.randn(output_size,hidden_neurons))
                self.bias.append(np.full((output_size,1),1))
            else:
                self.weights.append(np.random.randn(hidden_neurons,hidden_neurons))
                self.bias.append(np.full((hidden_neurons,1),1))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def first_order_sigmoid(self, x):
        return self.sigmoid(x) * (1 - self.sigmoid(x))

    def forward(self,x):
        activations=[]
        activations.append(x)
        for i in range(self.hidden_layers+1):
            x=np.dot(self.weights[i],activations[i])+self.bias[i]
            activations.append(self.sigmoid(x))
        return activations

    def backward(self,activations,di,m):
        delta=(activations[-1]-di.T) * self.first_order_sigmoid(np.dot(self.weights[-1],activations[-2])+self.bias[-1])
        for i in range(self.hidden_layers,-1,-1):
            if i==self.hidden_layers:
                prev=np.array(self.weights[i])
                self.weights[i]=self.weights[i]-(self.learning_rate/m) * np.dot(delta,activations[i].T)
                self.bias[i]=self.bias[i]-(self.learning_rate/m) * np.sum(delta,axis=1,keepdims=True)
            else:
                delta=np.dot(prev.T, delta) * self.first_order_sigmoid(np.dot(self.weights[i], activations[i])+self.bias[i])
                prev=np.array(self.weights[i])
                self.weights[i]=self.weights[i]-(self.learning_rate/m) * np.dot(delta,activations[i].T)
                self.bias[i]=self.bias[i]-(self.learning_rate/m) * np.sum(delta,axis=1,keepdims=True)

    def train(self,x,y,epochs):
        for i in range(epochs):
            activations=self.forward(x)
            m=x.shape[1]
            self.backward(activations,y,m)
            if(i%1000==0):
                print("Error at %d epoch : "%(i),np.sum(activations[-1]-y.T))

    def predict(self,x):
        predictions=[]
        for input in x:
            prediction = self.forward(np.array(input))
            predictions.append(prediction[-1])
        return predictions
```

In [142…

```
input_size=4
hidden_layers=2
neurons_in_hidden_layer=5
output_size=3
learning_rate=0.1

model=ANN(input_size,hidden_layers,neurons_in_hidden_layer,output_size,learning_rate)
```

In [155…

```
import pandas as pd
import seaborn as sns
# df=pd.read_csv('Iris.csv')
df=sns.load_dataset('iris')

# x = df.drop(['Id','Species'], axis=1)
x = df.drop(['species'], axis=1)
y=df['species']
```

In [ ]:

```
df
```

In [145…

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(y)
```

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,encoded_labels,test_size=0.2,random_state=62)

from tensorflow.keras.utils import to_categorical

labels = np.array(y_train)
y_train = to_categorical(labels)
y_train=np.array(y_train)
```

In [146…

```python
# x_train=x_train.values
# x_train=np.array(x_train)
# x_train=x_train.T
# x_test=x_test.values

x_train = x_train.to_numpy().T
x_test = x_test.to_numpy()
```

In [ ]:

```python
print('x_train \n',x_train)
print('x_test\n',x_test)
print('y_train\n',y_train)
```

In [ ]:

```python
epochs=100000
model.train(x_train,y_train,epochs)
```

In [149…

```python
# hh=model.forward([[6.7],[3.0],[5.2],[2.3]])
# print(hh[-1])
```

In [150…

```python
test_sample=[]
for i in x_test:
    test_sample.append([[x] for x in i.tolist()])
```

In [151…

```python
# y_pred=model.predict(test_sample)
# y_pred = np.hstack([np.argmax(arr, axis=0) for arr in y_pred]).flatten()

y_pred = np.argmax(np.hstack(model.predict(test_sample)), axis=0)
```

In [152…

```python
print(y_test)
```

```
[2 2 0 0 2 1 2 0 1 1 2 2 0 1 1 1 0 1 0 0 0 0 2 1 2 0 2 0 2 0]
```

In [153…

```python
print(y_pred)
```

```
[2 2 0 0 2 1 2 0 1 1 2 2 0 1 1 1 0 1 0 0 0 0 2 1 2 0 2 0 2 0]
```

In [154…

```python
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,confusion_matrix
accuracy=accuracy_score(y_test,y_pred)
precision=precision_score(y_test,y_pred,average='weighted')
recall=recall_score(y_test,y_pred,average='weighted')
f1score=f1_score(y_test,y_pred,average='weighted')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1score)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

In [ ]: