Practical 3: Performing multi-class classification using feed forward neural network

```
import pandas as pd
import numpy as np
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

```
df = pd.read_csv('Flower.csv',header = None)
```

```
X = df.iloc[:,:-1]
X.shape
```

```
(150, 4)
```

```
y = df.iloc[:,-1]
y.shape
```

```
(150,)
```

```
y
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: 4, Length: 150, dtype: object
```

```
from sklearn.preprocessing import LabelEncoder
```

```
lb = LabelEncoder
y = lb.fit_transform(y) #Converting categorical data into 0,1,2
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
x1train,x1test,y1train,y1test = train_test_split(X,y,test_size = 0.25,random_state = 1)
```

```
import keras
```

```
import tensorflow as tf
```

```
encoded_Y = tf.keras.utils.to_categorical(y)
encoded_Y #converting scalar y into vector
```

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
```

```
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
```

```
model = Sequential()
model.add(Dense(8,activation = 'relu',input_dim = 4))
model.add(Dense(6,activation = 'relu'))
model.add(Dense(3,activation = 'softmax'))
```

```
C:\Users\CompLab14\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass ar
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ['accuracy'])
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest = train_test_split(X,encoded_Y,test_size = 0.25,random_state = 1)
```

```
model.fit(xtrain,ytrain,epochs=100,batch_size=10)
```

```
Epoch 1/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3389 - loss: 0.9473
Epoch 2/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4273 - loss: 0.8779
Epoch 3/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 966us/step - accuracy: 0.6671 - loss: 0.8232
Epoch 4/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - accuracy: 0.6977 - loss: 0.7681
Epoch 5/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7264 - loss: 0.7457
Epoch 6/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - accuracy: 0.6706 - loss: 0.7247
Epoch 7/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 893us/step - accuracy: 0.6111 - loss: 0.6968
Epoch 8/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6621 - loss: 0.6598
Epoch 9/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6561 - loss: 0.6528
Epoch 10/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 969us/step - accuracy: 0.6488 - loss: 0.6352
```

```
Epoch 11/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7044 - loss: 0.6143
Epoch 12/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7111 - loss: 0.5792
Epoch 13/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7559 - loss: 0.5440
Epoch 14/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 997us/step - accuracy: 0.6989 - loss: 0.5472
Epoch 15/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - accuracy: 0.7006 - loss: 0.5286
Epoch 16/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7234 - loss: 0.5012
Epoch 17/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - accuracy: 0.7055 - loss: 0.5183
Epoch 18/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6830 - loss: 0.5140
Epoch 19/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 897us/step - accuracy: 0.7310 - loss: 0.4906
Epoch 20/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 912us/step - accuracy: 0.7287 - loss: 0.4699
Epoch 21/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 960us/step - accuracy: 0.7030 - loss: 0.4770
Epoch 22/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7059 - loss: 0.4620
Epoch 23/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6786 - loss: 0.4896
Epoch 24/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - accuracy: 0.6653 - loss: 0.4891
Epoch 25/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - accuracy: 0.7052 - loss: 0.4819
Epoch 26/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - accuracy: 0.6659 - loss: 0.4927
Epoch 27/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7395 - loss: 0.4358
Epoch 28/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 988us/step - accuracy: 0.6709 - loss: 0.4726
Epoch 29/100
```

```
predictions = model.predict(xtest)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

```
predictions
```

```
array([[9.9822778e-01, 1.7721991e-03, 5.7439370e-10],
       [9.2833795e-02, 8.9889520e-01, 8.2709659e-03],
       [2.2854209e-03, 9.5068127e-01, 4.7033284e-02],
       [9.9756634e-01, 2.4336851e-03, 2.4035063e-09],
       [3.2728358e-06, 4.1140428e-01, 5.8859241e-01],
       [1.1537509e-03, 9.1618478e-01, 8.2661517e-02],
       [1.6658030e-05, 3.9283022e-01, 6.0715318e-01],
       [9.9561393e-01, 4.3861121e-03, 2.6967074e-08],
       [9.9343866e-01, 6.5612136e-03, 1.2087978e-07],
       [7.0561146e-07, 1.1576290e-01, 8.8423634e-01],
       [1.4024805e-03, 8.3573496e-01, 1.6286254e-01],
       [9.9723333e-01, 2.7666548e-03, 6.2956631e-09],
       [1.3396799e-07, 5.6694675e-02, 9.4330525e-01],
       [2.3186104e-03, 9.5050609e-01, 4.7175340e-02],
       [6.4111216e-04, 8.0678463e-01, 1.9257420e-01],
       [9.9315804e-01, 6.8418384e-03, 9.1706838e-08],
       [3.0553727e-03, 9.0964019e-01, 8.7304428e-02],
       [7.7213609e-04, 7.9616636e-01, 2.0306145e-01],
       [9.9607348e-01, 3.9265258e-03, 2.1847477e-08],
       [9.9502605e-01, 4.9738945e-03, 2.9321061e-08],
       [4.4713347e-04, 6.6700059e-01, 3.3255231e-01],
       [6.8860740e-04, 7.5809771e-01, 2.4121371e-01],
       [8.8047760e-05, 6.2884116e-01, 3.7107077e-01],
       [9.9594611e-01, 4.0538604e-03, 1.5174345e-08],
       [9.4609942e-07, 1.5944219e-01, 8.4055686e-01],
       [3.2317629e-03, 9.1766161e-01, 7.9106644e-02],
       [9.9818999e-01, 1.8100301e-03, 7.0082590e-10],
       [9.9678850e-01, 3.2115483e-03, 7.4964346e-09],
       [3.8303496e-04, 7.5457454e-01, 2.4504249e-01],
       [1.5616072e-06, 9.9250227e-02, 9.0074813e-01],
       [8.2061195e-04, 8.5689002e-01, 1.4228934e-01],
       [3.4734114e-08, 3.8548436e-02, 9.6145153e-01],
       [3.6518166e-03, 9.0394241e-01, 9.2405826e-02],
       [1.3352759e-07, 3.8359787e-02, 9.6164012e-01],
       [9.5403072e-07, 6.9368303e-02, 9.3063068e-01],
       [9.9688220e-01, 3.1178750e-03, 6.7796408e-09],
       [6.1250926e-04, 6.6299617e-01, 3.3639127e-01],
       [9.9593627e-01, 4.0637506e-03, 1.7458364e-08]], dtype=float32)
```

```
for i in range(1, len(predictions),3):
    print(predictions[i],ytest[i])
```

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

```
for i in range(1, len(predictions),3):
    print(np.argmax(predictions[i]),y1test[i])
```

```
1 1
2 2
0 0
1 1
1 1
1 1
0 0
1 1
1 1
1 1
2 2
2 2
0 0
```

Start coding or generate with AI.