Implement ann for binary classification and regression use different activation and optimisers find the model perfomance , use dropout layers check the performance of model

Binary Classification Problem using ANN

In [ ]:
```
#for local paths
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [ ]:
```
#import libraries
import numpy as np
import pandas as pd
```

In [ ]:
```
#importing dataset
path = ''
data = pd.read_csv(path+'.csv')
```

In [ ]:
```
#fetch dimensions of pandas and numpy
data.shape
```

In [ ]:
```
#displays first five rows of data
data.head()
```

In [ ]:
```
#provides a concise summary of a Data
data.info()
```

In [ ]:
```
#returns a Series with True and False
data.duplicated().sum()
```

In [ ]:
```
#removes specified row or column
data.drop(columns=[''],inplace=True)
```

In [ ]:
```
data.head(3)
```

In [ ]:
```
# convert categorical variables into dummy
df = pd.get_dummies(data,columns=[''],drop_first=True)
df.head(3)
```

In [ ]:
```
x = df.drop(columns=[''])
y = df['']
```

In [ ]:
```
#splitting data
from sklearn.model_selection import train_test_split

train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.2,random_state=108)
```

In [ ]:
```
train_x.shape,test_x.shape,train_y.shape,test_y.shape
```

In [ ]:
```
#removes the mean and scales each feature to unit variance
from sklearn.preprocessing import StandardScaler

scal = StandardScaler()

train_x_scal = scal.fit_transform(train_x)
test_x_scal = scal.fit_transform(test_x)
```

In [ ]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [ ]:
```
#create models layer-by-layer
model = Sequential()

#input layer
model.add(Dense(3,activation='sigmoid',input_dim=11))
#output layer
model.add(Dense(1,activation='sigmoid'))
```

In [ ]:
```
model.summary()
```

Model compilation

In [ ]:
```
#binary_crossentropy
model.compile(loss='binary_crossentropy',optimizer='Adam')
```

In [ ]:
```
#create histograms
hist = model.fit(train_x_scal, train_y,epochs=10)
```

In [ ]:
```
hist.history
```

In [ ]:
```
#creating static, animated, and interactive visualizations
import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])
plt.show()
```

In [ ]:
```
#weights for layer 0
model.layers[0].get_weights()
```

In [ ]:
```
#weights for layer 1
model.layers[1].get_weights()
```

In [ ]:
```
#predictions
model.predict(test_x_scal)
```

In [ ]:
```
#output in classes now
y_pred = np.where(model.predict(test_x_scal)>0.5,1,0)
y_pred
```

In [ ]:
```
from sklearn.metrics import accuracy_score

accuracy_score(test_y,y_pred)
```

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

#input layer
model.add(Dense(11,activation='relu',input_dim=11))
model.add(Dense(11,activation='relu'))
#output layer
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

In [ ]:
```
#binary_crossentropy
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])

hist_ = model.fit(train_x_scal, train_y,epochs=100,validation_split=0.2)
```

In [ ]:
```
plt.plot(hist_.history['loss'])
plt.plot(hist_.history['val_loss'])
plt.show()
```

In [ ]:
```
plt.plot(hist_.history['accuracy'])
plt.plot(hist_.history['val_accuracy'])
plt.show()
```

In [ ]:
```
#output in classes now
y_pred = np.where(model.predict(test_x_scal)>0.5,1,0)
```

In [ ]:
```
accuracy_score(test_y,y_pred)
```

Regression Problem using ANN

In [ ]:
```
#importing dataset
path = ''
data = pd.read_csv(path+'.csv')
data.shape
```

In [ ]:
```
##displays first five rows of data
data.head()
```

```
In [ ]: #returns the number of missing values in the dataset
        data.isnull().sum()
```

```
In [ ]: #provides a concise summary of a Data
        data.info()
```

```
In [ ]: #removes specified row or column
        data.drop(columns=[''],inplace=True)
```

```
In [ ]: #fetch dimensions of pandas and numpy
        data.shape
```

```
In [ ]: #select all rows but first column will be excluded
        x = data.iloc[:,0:-1]
        y = data.iloc[:,-1]
```

```
In [ ]: #splitting data
        from sklearn.model_selection import train_test_split

        train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.2,random_state=108)

        train_x.shape,train_y.shape,test_x.shape,test_y.shape
```

```
In [ ]: #scales the data to a fixed range, typically between 0 and 1
        from sklearn.preprocessing import MinMaxScaler

        scale = MinMaxScaler()

        train_x_scale = scale.fit_transform(train_x) #scaled np array
        test_x_scale = scale.fit_transform(test_x)   #scaled np array
```

```
In [ ]: ##create models layer-by-layer
        model = Sequential()

        model.add(Dense(7,activation='relu',input_dim=7))
        model.add(Dense(1,activation='linear'))
```

```
In [ ]: model.summary()
```

model compilation

```
In [ ]: model.compile(loss='mean_squared_error',optimizer='Adam')

        hist = model.fit(train_x_scale,train_y,epochs=10,validation_split=0.2)
```

```
In [ ]: #prediction
        y_pred = model.predict(test_x_scale)
```

```
In [ ]: #for regression
        from sklearn.metrics import r2_score

        r2_score(test_y,y_pred)
```

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

        model.add(Dense(7,activation='relu',input_dim=7))
        model.add(Dense(7,activation='relu'))
        model.add(Dense(1,activation='linear'))
```

```
In [ ]: model.summary()
```

```
In [ ]: model.compile(loss='mean_squared_error',optimizer='Adam')

        hist = model.fit(train_x_scale,train_y,epochs=100,validation_split=0.2)
```

```
In [ ]: #prediction
        y_pred = model.predict(test_x_scale)

        r2_score(test_y,y_pred)
```

In [ ]:
```
#plotting training and validation loss

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.show()
```