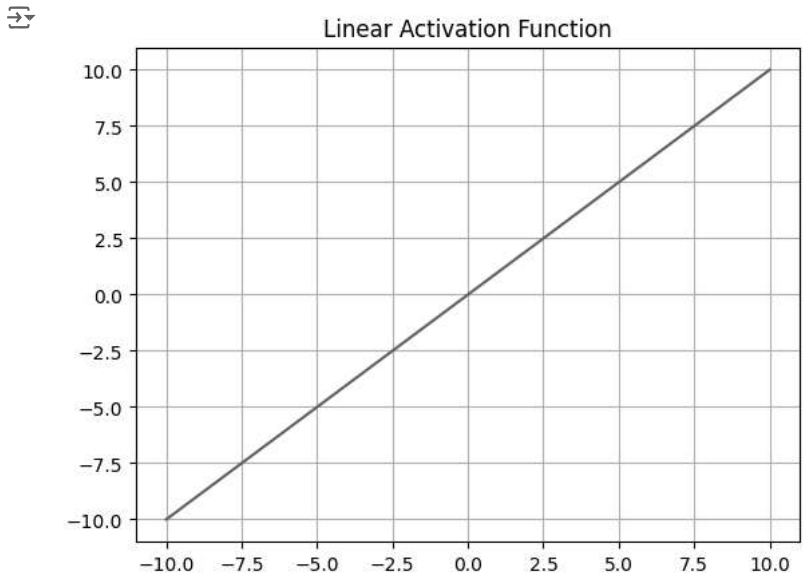
Name: Aparna Iyer

PRN: 22070126017

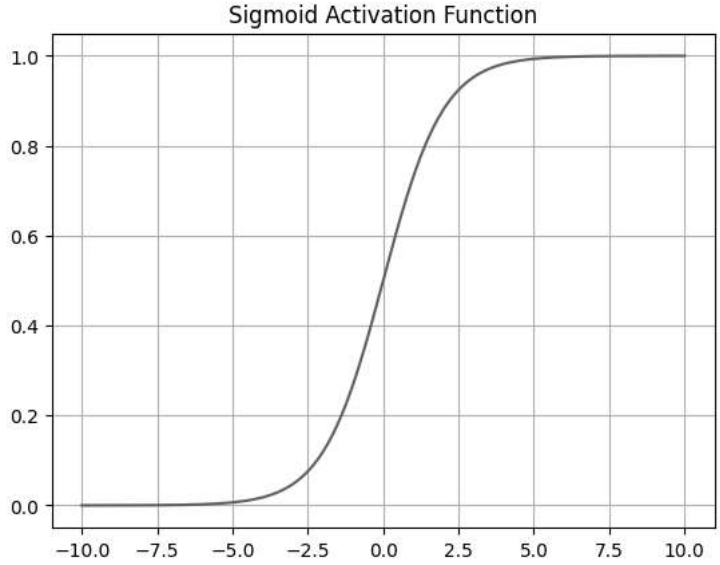
Batch: AI-ML A1

1. Implement the Python Program for different activation functions.

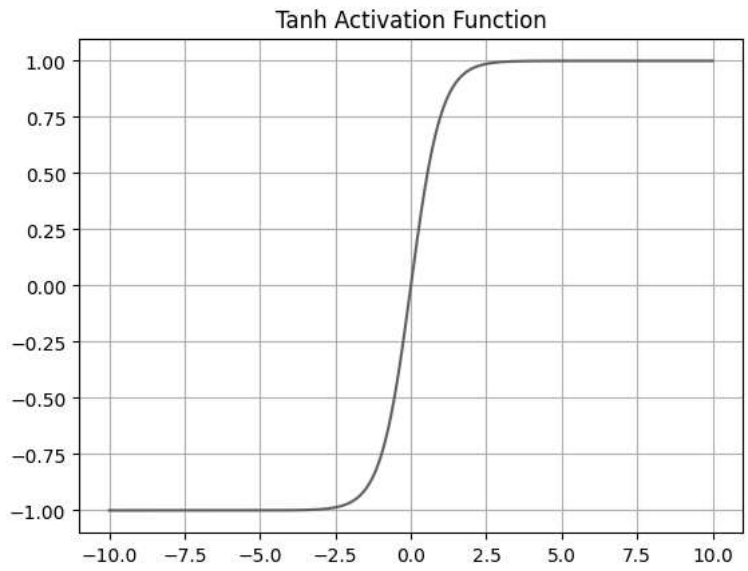
```
#Activation Functions

#Neural Network Basics



#Linear Activation Function
import numpy as np
import matplotlib.pyplot as plt

def linear(x):
  return x

#Plotting the linea function

x=np.linspace(-10,10,400)
y=linear(x)
plt.plot(x,y)
plt.title("Linear Activation Function")
plt.grid()
plt.show()
```



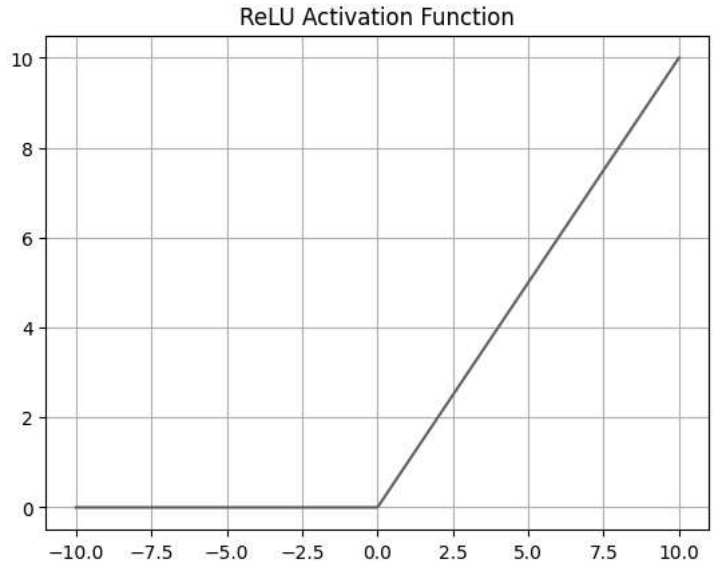

```
#Sigmoid Function: Maps any real-valued number to 0 or 1 (Non-Linear)

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

x=np.linspace(-10,10,400)
y=sigmoid(x)
plt.plot(x,y)
plt.title("Sigmoid Activation Function")
plt.grid()
plt.show()
```



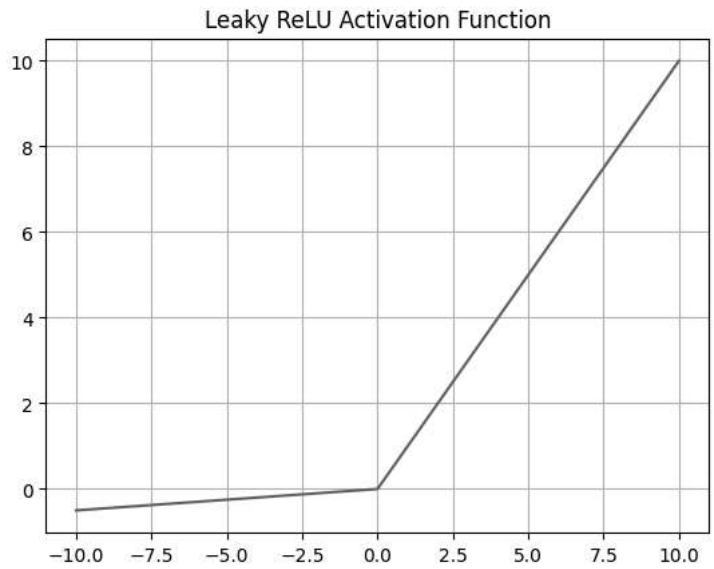

```
#Tanh Function: Maps any real-valued number to the range (-1,1). (Hyperbolic Tangent)

def tanh(x):
  return np.tanh(x)

x=np.linspace(-10,10,400)
y=tanh(x)
plt.plot(x,y)
plt.title("Tanh Activation Function")
plt.grid()
plt.show()
```



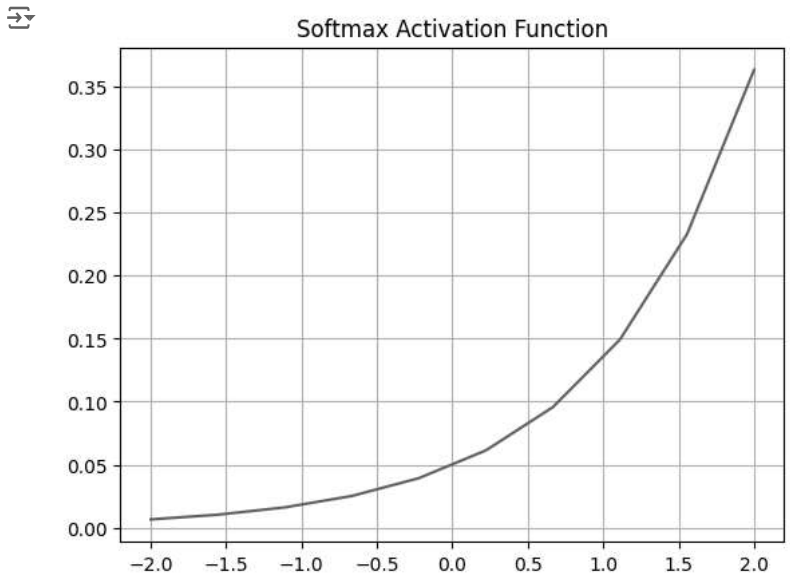

```
#ReLU : Maximum of (0,x)

def relu(x):
  return np.maximum(0,x)

x=np.linspace(-10,10,400)
y=relu(x)
plt.plot(x,y)
plt.title("ReLU Activation Function")
plt.grid()
plt.show()
```




```
#Leaky ReLU: An attempt to fix the 'dying ReLU' problem by allowing small, non-negative

def leaky_relu(x,alpha=0.05):
  return np.where(x>0,x,alpha*x)

x=np.linspace(-10,10,400)
y=leaky_relu(x)
plt.plot(x,y)
plt.title("Leaky ReLU Activation Function")
plt.grid()
plt.show()
```



```
#Softmax Function : Used in Output Layer.
#No. of neurons in output layer = No. of classes

#y=exp(X)/sum(exp(x))

def softmax(x):
  exp_x = np.exp(x-np.max(x)) #Shift for numerical stability
  return exp_x/exp_x.sum(axis=0)

#Plotting the softmax function

x=np.linspace(-2,2,10)
y=softmax(x)
plt.plot(x,y)
plt.title("Softmax Activation Function")
plt.grid()
plt.show()
```



2. Implement the Python Program for building a simple NN from scratch (without Tensorflow and Keras).

```
import numpy as np

class NeuralNetwork:
  def __init__(self,input_size,hidden_size,output_size):
    self.input_size=input_size
    self.hidden_size=hidden_size
    self.output_size=output_size

    #Initialize weights  (Random Weights)

    self.weights_input_hidden = np.random.randn(self.input_size,self.hidden_size)
    self.weights_hidden_output = np.random.randn(self.hidden_size,self.output_size)

    #Initialize biases

    self.bias_hidden = np.zeros((1,self.hidden_size))
    self.bias_output = np.zeros((1,self.output_size))

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

  def sigmoid_derivative(self,x):
    return x*(1-x)

  def feedforward(self,X):
    #Input to hidden
    self.hidden_activation = np.dot(X,self.weights_input_hidden) + self.bias_hidden
    self.hidden_output = self.sigmoid(self.hidden_activation)

    #Hidden to output
    self.output_activation = np.dot(self.hidden_output,self.weights_hidden_output) + self.bias_output
    self.predicted_output = self.sigmoid(self.output_activation)

    return self.predicted_output

  def backward(self, X, y, learning_rate):
    # Compute the output layer error
    output_error = y - self.predicted_output
    output_delta = output_error * self.sigmoid_derivative(self.predicted_output)

    # Compute the hidden layer error
    hidden_error = np.dot(output_delta,self.weights_hidden_output.T)
    hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_output)

  # Update weights and biases
    self.weights_hidden_output += np.dot(self.hidden_output.T, output_delta) * learning_rate
    self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate
    self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
    self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * learning_rate


  def train(self,X,y,learning_rate,epochs):
    for epoch in range(epochs):
      output=self.feedforward(X) #Forward Pass
      self.backward(X,y,learning_rate)
      if epoch % 2000 == 0:
        loss=np.mean(np.square(y-output))
        print("Epoch %d: Loss %.4f" %(epoch,loss))


X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[1],[1],[0],[1]])

nn = NeuralNetwork(input_size = 2, hidden_size = 2, output_size = 1)
nn.train(X,y,learning_rate=0.1,epochs=10000)

#Test the trained model

output = nn.feedforward(X)
print("Predictions after training:")
print(output)
```

```
Epoch 0: Loss 0.2713
Epoch 2000: Loss 0.0050
Epoch 4000: Loss 0.0017
```

```
Epoch 6000: Loss 0.0009
Epoch 8000: Loss 0.0006
Predictions after training:
[[0.97649261]
 [0.99736814]
 [0.03193829]
 [0.98093458]]
```

```
X.shape
```

```
(4, 2)
```

```
y.shape
```

```
(4, 1)
```

3. Implement the Python Program for any dataset from Kaggle from scratch (without Tensorflow and Keras).

```
#Testing the above model on a Kaggle Dataset

import pandas as pd
#EDA on the Dataset

#Titanic Test Set

titanic_df = pd.read_csv("/content/train.csv")



#First 5 rows of DataFrame
titanic_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2833

Next steps: Generate code with titanic_df View recommended plots

```
#Preprocessing: Handling Missing Values, Scaling the Data and Label Encoding

import sklearn as sk
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Handle missing values
age_imputer = SimpleImputer(strategy='median')
fare_imputer = SimpleImputer(strategy='median')
embarked_imputer = SimpleImputer(strategy='most_frequent')

titanic_df['Age'] = age_imputer.fit_transform(titanic_df[['Age']])
titanic_df['Fare'] = fare_imputer.fit_transform(titanic_df[['Fare']])
titanic_df['Cabin'] = titanic_df['Cabin'].fillna('Unknown')

# Encode categorical variables
label_encoder = LabelEncoder()
titanic_df['Sex'] = label_encoder.fit_transform(titanic_df['Sex'])
titanic_df['Embarked'] = label_encoder.fit_transform(titanic_df['Embarked'])
titanic_df['Cabin'] = label_encoder.fit_transform(titanic_df['Cabin'])
```

```
# Drop columns that are not features
drop_columns = ['PassengerId', 'Name', 'Ticket','Survived']

X1 = np.array(titanic_df.drop(columns=drop_columns))
y1 = np.array(titanic_df['Survived'])
print("The shape of X1 is: ",X1.shape) #The number of rows and columns of X1
```

```
The shape of X1 is:  (891, 8)
```

```
print("The shape of y1 is: ",y1.shape)
```

```
The shape of y1 is:  (891,)
```

```
y1_reshaped=y1.reshape(891,1) #We are reshaping y1 to ensure that we can apply the Neural Network as X1 is a 2D Array while y1 is a 1D Arra


#Training the Neural Network on the Predictor and Target Sets
nn = NeuralNetwork(input_size = 8, hidden_size = 2, output_size = 1)
nn.train(X1,y1_reshaped,learning_rate=0.1,epochs=10000)
```

```
<ipython-input-10-1d7ec6417839>:20: RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-x))
Epoch 0: Loss 0.2500
Epoch 2000: Loss 0.2565
Epoch 4000: Loss 0.2904
Epoch 6000: Loss 0.3731
Epoch 8000: Loss 0.2395
```

```
#Test the trained model

output1 = nn.feedforward(X1)
print("Predictions after training:")
print(output1)
```

```
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]
 [0.00688312]]
<ipython-input-10-1d7ec6417839>:20: RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-x))
```

```
#Another dataset: Kaggle Tennis Matches Dataset

df_matches=pd.read_csv("/content/TennisMatchesKaggle.csv",low_memory=False)


df_matches.head()
```

tourney_date	match_num	winner_id	winner_seed	winner_entry	...	l_1st
31-12-2018	300	105453	2	NaN	...	3
31-12-2018	299	106421	4	NaN	...	3
31-12-2018	298	105453	2	NaN	...	1
31-12-2018	297	104542	NaN	PR	...	3
31-12-2018	296	106421	4	NaN	...	4

```
#EDA on the Dataset
import seaborn as sns

# Check data types of each column
print(df_matches.dtypes)
```

```
tourney_id             object
tourney_name           object
surface                object
draw_size               int64
tourney_level          object
tourney_date           object
match_num               int64
winner_id               int64
winner_seed            object
winner_entry           object
winner_name            object
winner_hand            object
winner_ht             float64
winner_ioc             object
winner_age            float64
loser_id                int64
loser_seed             object
loser_entry            object
loser_name             object
loser_hand             object
loser_ht              float64
loser_ioc              object
loser_age             float64
score                  object
best_of               float64
round                  object
minutes               float64
w_ace                 float64
w_df                  float64
w_svpt                float64
w_1stIn               float64
w_1stWon              float64
w_2ndWon              float64
w_SvGms               float64
w_bpSaved             float64
w_bpFaced             float64
l_ace                 float64
l_df                  float64
```

```
l_svpt                float64
l_1stIn               float64
l_1stWon              float64
l_2ndWon              float64
l_SvGms               float64
l_bpSaved             float64
l_bpFaced             float64
winner_rank           float64
winner_rank_points    float64
loser_rank            float64
loser_rank_points     float64
league                 object
dtype: object
```

```
from sklearn.preprocessing import LabelEncoder

#Label-Encoding Categorical Columns

# Identify categorical columns
categorical_cols = df_matches.select_dtypes(include=['object']).columns

# Initialize the LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in categorical_cols:
    df_matches[col] = le.fit_transform(df_matches[col])



#Null Value Identification and Imputation

nan_counts = df_matches.isna().sum()

print(nan_counts)
```

```
tourney_id                0
tourney_name              0
surface                   0
draw_size                 0
tourney_level             0
tourney_date              0
match_num                 0
winner_id                 0
winner_seed               0
winner_entry              0
winner_name               0
winner_hand               0
winner_ht            174546
winner_ioc                0
winner_age            18856
loser_id                  0
loser_seed                0
loser_entry               0
loser_name                0
loser_hand                0
loser_ht             197794
loser_ioc                 0
loser_age             39602
score                     0
best_of                   0
round                     0
minutes              275448
w_ace                251147
w_df                 251004
w_svpt               251145
w_1stIn              251145
w_1stWon             251145
w_2ndWon             251145
w_SvGms              271126
w_bpSaved            251150
w_bpFaced            251150
l_ace                251152
l_df                 251184
l_svpt               251146
l_1stIn              251145
l_1stWon             251145
l_2ndWon             251145
l_SvGms              271126
l_bpSaved            251147
l_bpFaced            251147
winner_rank          139459
```

```
winner_rank_points    176163
loser_rank            146155
loser_rank_points     180401
league                     0
dtype: int64
```

```
# Convert the Series, nan_counts to a dictionary
nan_counts_dict = nan_counts.to_dict()
```

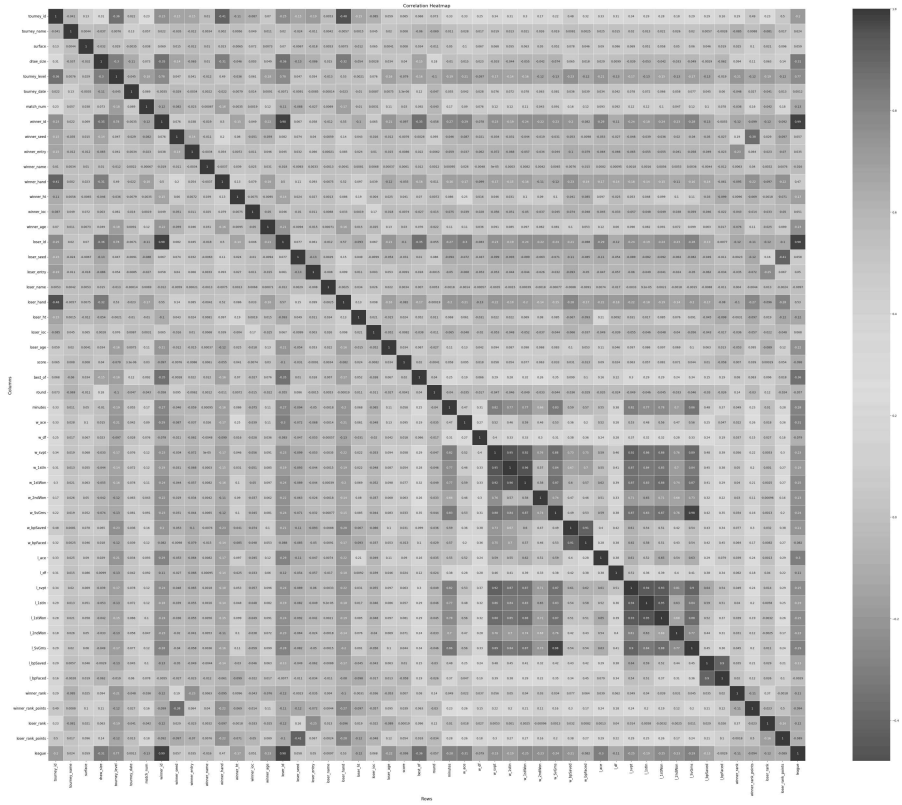
```
for key, value in nan_counts_dict.items():
  if nan_counts_dict[key]!=0: #Number of Null Values in the column is not equal to 0
    #Perform Mode Imputation
    df_matches[key] = df_matches[key].fillna(df_matches[key].mode()[0])
```

```
df_matches.head() #First 5 rows of the DataFrame
```

	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_num
0	18526	273	3	22	0	6664	300
1	18526	273	3	22	0	6664	299
2	18526	273	3	22	0	6664	298
3	18526	273	3	22	0	6664	297
4	18526	273	3	22	0	6664	296

5 rows × 50 columns

```
#Correlation Matrix

# Compute the correlation matrix
corr_matrix = df_matches.corr()

# Create a heatmap
plt.figure(figsize=(60, 48))
heatmap = sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.xlabel('Rows', fontsize=15)
plt.ylabel('Columns', fontsize=15)

# Increase the tick labels font size
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add title
plt.title('Correlation Heatmap', fontsize=16)

# Display the heatmap
plt.show()
```




```
#Selecting features for independent variable X, and dependent variable Y

y2=np.array(df_matches['winner_rank'])
X2=np.array(df_matches.drop(columns=['winner_rank','tourney_date','tourney_level','league','winner_id','surface','draw_size','winner_name',
```

```
print("The shape of X2 is: ",X2.shape) #The number of rows and columns of X2
```

```
The shape of X2 is:  (373436, 31)
```

```
print("The shape of y2 is: ",y2.shape) #The number of rows and columns of y2
```

```
The shape of y2 is:  (373436,)
```

```
y2_reshaped=y2.reshape(373436,1) #We are reshaping y2 to ensure that both are 2D arrays.
```