*****DESIGN TRAIN AND TEST MULTI LAYER PERCEPTRON FOR TABULAR DATA AND VERIFY VARIOUS ACTIVATION FUNCTIONS AND OPTIMIZERS TENSORFLOW***************

```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/diabetes-dataset/diabetes.csv
```

```
data=pd.read_csv("/kaggle/input/diabetes-dataset/diabetes.csv")
data
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6

1              1       85             66             29        0  26.6

2              8      183             64              0        0  23.3
```

```
3              1     89             66             23        94  28.1

4              0    137             40             35       168  43.1

..           ...    ...            ...            ...       ...   ...

763           10    101             76             48       180  32.9

764            2    122             70             27         0  36.8

765            5    121             72             23       112  26.2

766            1    126             60              0         0  30.1

767            1     93             70             31         0  30.4


     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

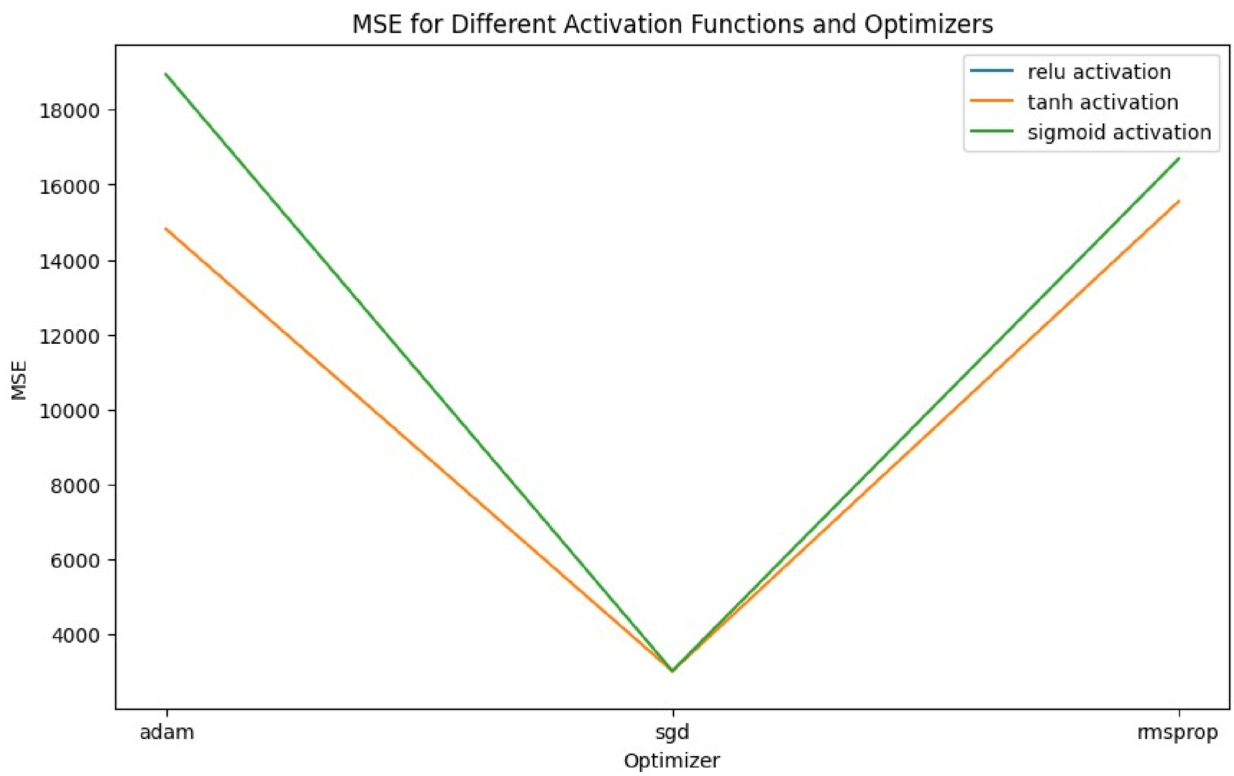
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load and prepare the data
data = load_diabetes()
X = data.data
y = data.target

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the model
def create_model(activation='relu', optimizer='adam'):
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],),
activation=activation))
    model.add(Dense(32, activation=activation))
    model.add(Dense(1))  # No activation for regression output

    # Compile the model
    model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['mse'])
    return model

# Define a list of activation functions and optimizers to test
activation_functions = ['relu', 'tanh', 'sigmoid']
optimizers = ['adam', 'sgd', 'rmsprop']

results = {}

# Train and test the model with different configurations
for activation in activation_functions:
    for optimizer in optimizers:
        model = create_model(activation=activation,
optimizer=optimizer)
        model.fit(X_train, y_train, epochs=100, batch_size=32,
verbose=0)

        # Evaluate the model
        loss, mse = model.evaluate(X_test, y_test, verbose=0)
        results[(activation, optimizer)] = mse
        print(f"Activation: {activation}, Optimizer: {optimizer}, MSE:
{mse:.4f}")

# Plotting
plt.figure(figsize=(10, 6))
for activation in activation_functions:
    plt.plot([optimizer for (act, optimizer) in results.keys() if act
== activation],
             [results[(activation, optimizer)] for optimizer in
optimizers],
             label=f'{activation} activation')

plt.title('MSE for Different Activation Functions and Optimizers')
plt.xlabel('Optimizer')
plt.ylabel('MSE')
```

```python
plt.legend()
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Activation: relu, Optimizer: adam, MSE: 2850.3616
Activation: relu, Optimizer: sgd, MSE: nan
Activation: relu, Optimizer: rmsprop, MSE: 2824.1575
Activation: tanh, Optimizer: adam, MSE: 14815.9014
Activation: tanh, Optimizer: sgd, MSE: 3014.3652
Activation: tanh, Optimizer: rmsprop, MSE: 15555.7920
Activation: sigmoid, Optimizer: adam, MSE: 18943.2266
Activation: sigmoid, Optimizer: sgd, MSE: 3018.5166
Activation: sigmoid, Optimizer: rmsprop, MSE: 16693.6094
```




```python
from tensorflow.keras.optimizers import SGD

# Use a smaller learning rate
optimizer = SGD(learning_rate=0.0001)

# Recreate and compile the model with the adjusted optimizer
```

```python
model = create_model(activation='relu', optimizer=optimizer)

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# Evaluate the model
loss, mse = model.evaluate(X_test, y_test, verbose=0)
print(f"Activation: relu, Optimizer: sgd, MSE: {mse:.4f}")
```

```
Activation: relu, Optimizer: sgd, MSE: 2831.2041
```

To eliminate NaN (Not a Number) values in the MSE when using the ReLU activation function with the SGD optimizer, the following changes can be made:

Reduce Learning Rate: A high learning rate can cause issues with convergence, especially with the ReLU activation function, leading to NaN values. Lowering the learning rate for the SGD optimizer can help stabilize training.

Initialize Weights Properly: Proper weight initialization (like He initialization) can also help prevent issues with ReLU activation.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adagrad
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load and prepare the data
data = load_diabetes()
X = data.data
y = data.target

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the model
def create_model(activation='relu', optimizer='adam'):
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],),
activation=activation))
    #model.add(BatchNormalization())  # Add Batch Normalization
```

```
    model.add(Dense(32, activation=activation))
    #model.add(BatchNormalization())  # Add Batch Normalization
    model.add(Dense(1))  # No activation for regression output

    # Compile the model with gradient clipping
    optimizer = tf.keras.optimizers.get(optimizer)
    if isinstance(optimizer, SGD):
        optimizer.learning_rate = 0.0001  # Reduced learning rate

    model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['mse'])
    return model

# Define a list of activation functions and optimizers to test
activation_functions = ['relu', 'tanh', 'sigmoid']
optimizers = ['adam', 'sgd', 'rmsprop', 'Adagrad']

results = {}

# Train and test the model with different configurations
for activation in activation_functions:
    for optimizer in optimizers:
        model = create_model(activation=activation,
optimizer=optimizer)
        model.fit(X_train, y_train, epochs=100, batch_size=32,
verbose=0)

        # Evaluate the model
        loss, mse = model.evaluate(X_test, y_test, verbose=0)
        results[(activation, optimizer)] = mse
        print(f"Activation: {activation}, Optimizer: {optimizer}, MSE:
{mse:.4f}")

# Plotting
plt.figure(figsize=(10, 6))
for activation in activation_functions:
    plt.plot([optimizer for (act, optimizer) in results.keys() if act
== activation],
             [results[(activation, optimizer)] for optimizer in
optimizers],
             label=f'{activation} activation')

plt.title('MSE for Different Activation Functions and Optimizers')
plt.xlabel('Optimizer')
plt.ylabel('MSE')
# Option 1: Manually set the y-axis limits
#plt.ylim(0, 30000)  # Set lower and upper bounds for the y-axis

# Option 2: Use a logarithmic scale for the y-axis
#plt.yscale('log')
```

```
plt.legend()
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Activation: relu, Optimizer: adam, MSE: 2956.9028
Activation: relu, Optimizer: sgd, MSE: 3147.7812
Activation: relu, Optimizer: rmsprop, MSE: 2907.7595
Activation: relu, Optimizer: Adagrad, MSE: 23059.0273
Activation: tanh, Optimizer: adam, MSE: 14791.1758
Activation: tanh, Optimizer: sgd, MSE: 2860.1355
Activation: tanh, Optimizer: rmsprop, MSE: 15218.6572
Activation: tanh, Optimizer: Adagrad, MSE: 25984.4414
Activation: sigmoid, Optimizer: adam, MSE: 20165.0508
Activation: sigmoid, Optimizer: sgd, MSE: 2957.6797
Activation: sigmoid, Optimizer: rmsprop, MSE: 16949.9629
Activation: sigmoid, Optimizer: Adagrad, MSE: 25664.1484
```