Write a Python program to demonstrate the use of the sigmoid, tanh, and ReLU activation functions:

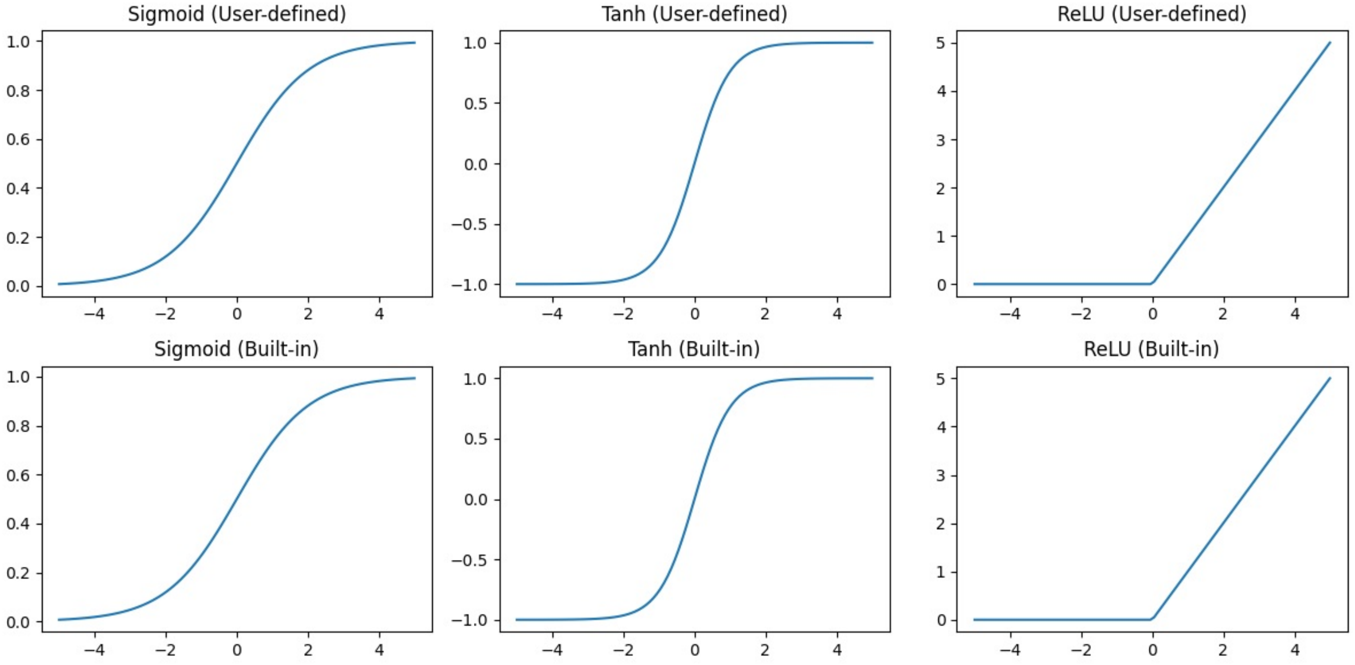
a. using user-defined functions.

b. using built-in functions

Illustrate the behavior of these activation functions by plotting graphs

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt

# User-defined activation functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

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

# Input values
x = np.linspace(-5, 5, 100)

# Using user-defined activation functions
sigmoid_values = sigmoid(x)
tanh_values = tanh(x)
relu_values = relu(x)

# Using built-in activation functions
sigmoid_builtin = 1 / (1 + np.exp(-x))
tanh_builtin = np.tanh(x)
relu_builtin = np.maximum(0, x)

# Plotting the graphs
plt.figure(figsize=(12, 6))

plt.subplot(2, 3, 1)
plt.plot(x, sigmoid_values, label='Sigmoid')
plt.title('Sigmoid (User-defined)')

plt.subplot(2, 3, 2)
plt.plot(x, tanh_values, label='Tanh')
plt.title('Tanh (User-defined)')

plt.subplot(2, 3, 3)
plt.plot(x, relu_values, label='ReLU')
plt.title('ReLU (User-defined)')

plt.subplot(2, 3, 4)
plt.plot(x, sigmoid_builtin, label='Sigmoid')
plt.title('Sigmoid (Built-in)')

plt.subplot(2, 3, 5)
plt.plot(x, tanh_builtin, label='Tanh')
plt.title('Tanh (Built-in)')

plt.subplot(2, 3, 6)
plt.plot(x, relu_builtin, label='ReLU')
plt.title('ReLU (Built-in)')

plt.tight_layout()
plt.show()
```



In [ ]:

```
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Demonstrate the use of SGD and ADAM Optimizers in training the neural Networks. 2

In [3]:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam

np.random.seed(0)
X = np.random.rand(1000, 10)
y = np.random.randint(0, 2, 1000)

model = Sequential()
model.add(Dense(64, input_dim=10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])

model.fit(X, y, epochs=10, batch_size=32)

model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])

model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4859 - loss: 0.6952
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5082 - loss: 0.6915
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4860 - loss: 0.6947
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4718 - loss: 0.6957
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5205 - loss: 0.6927
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5195 - loss: 0.6926
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5042 - loss: 0.6921
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5086 - loss: 0.6921
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5077 - loss: 0.6926
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5025 - loss: 0.6935
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.5252 - loss: 0.6918
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5268 - loss: 0.6925
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5520 - loss: 0.6895
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5243 - loss: 0.6894
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5161 - loss: 0.6896
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5438 - loss: 0.6858
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5696 - loss: 0.6841
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5638 - loss: 0.6822
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5678 - loss: 0.6846
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 903us/step - accuracy: 0.5734 - loss: 0.6821
```

Out[3]:

```
<keras.src.callbacks.history.History at 0x15bcb88f1d0>
```

In [ ]:

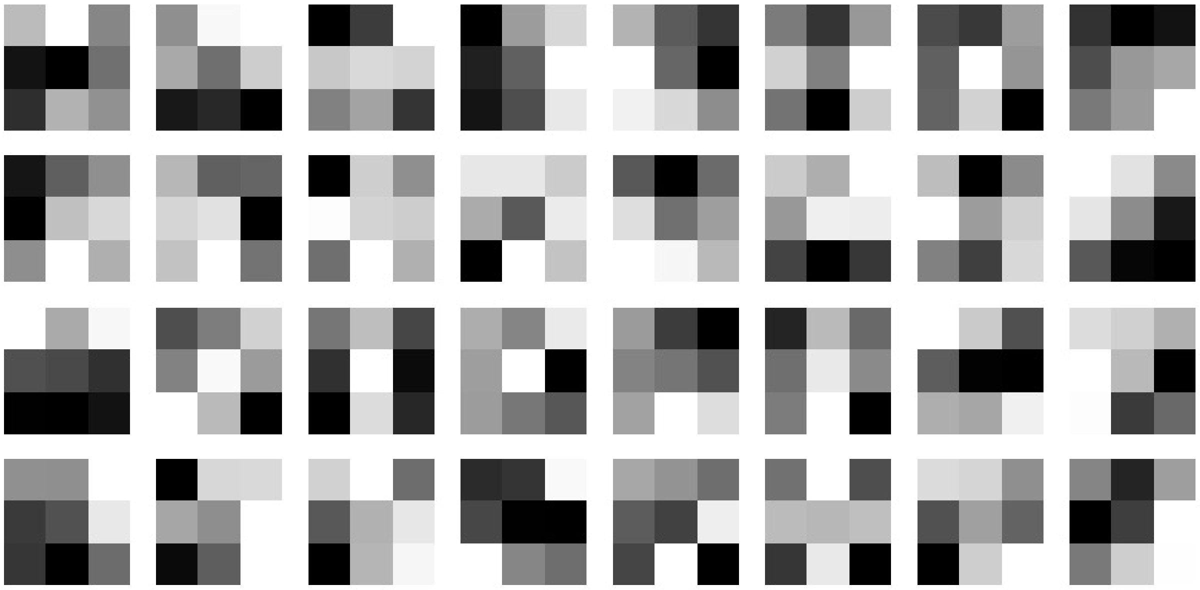
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Build a simple CNN architecture for image classification. Train the CNN on a dataset like MNIST or CIFAR-10. Visualize the learned filters and feature maps. 3

In [1]:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))

filters, biases = model.layers[0].get_weights()

n_filters = filters.shape[3]
plt.figure(figsize=(10, 10))
for i in range(n_filters):
    f = filters[:, :, 0, i]
    plt.subplot(8, 8, i+1)
    plt.imshow(f, cmap='gray')
    plt.axis('off')

plt.show()
```

```
C:\Users\Sanga\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_con
v.py:99: 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__(
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.9006 - loss: 0.3274 - val_accuracy: 0.9867 - val_loss:
0.0394
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.9854 - loss: 0.0480 - val_accuracy: 0.9812 - val_loss:
0.0617
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9887 - loss: 0.0355 - val_accuracy: 0.9899 - val_loss:
0.0335
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9924 - loss: 0.0229 - val_accuracy: 0.9918 - val_loss:
0.0245
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9947 - loss: 0.0159 - val_accuracy: 0.9902 - val_loss:
0.0305
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Implement a basic
RNN for sequence prediction.

In [1]:

```
import tensorflow as tf
import numpy as np

np.random.seed(0)
sequence_length = 10
X = np.random.random((1000, sequence_length, 1))
y = np.sin(X).reshape(-1)

model = tf.keras.models.Sequential([
    tf.keras.layers.SimpleRNN(64, input_shape=(sequence_length, 1), return_sequences=False),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X, y, epochs=10, batch_size=32, validation_split=0.2)

new_sequence = np.random.random((1, sequence_length, 1))

predicted_value = model.predict(new_sequence)

print("Predicted value for the next step:", predicted_value)
```

```
Epoch 1/10
C:\Users\Sanga\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: 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__(**kwargs)
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1502 - val_loss: 0.0701
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0704 - val_loss: 0.0655
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0701 - val_loss: 0.0621
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0634 - val_loss: 0.0616
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0622 - val_loss: 0.0607
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0641 - val_loss: 0.0597
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0649 - val_loss: 0.0617
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0641 - val_loss: 0.0607
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0620 - val_loss: 0.0639
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0673 - val_loss: 0.0615
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
Predicted value for the next step: [[0.36005083]]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Implement an LSTM and compare its performance on a sequential task.

In [1]:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

num_words = 10000
maxlen = 500
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=num_words)

train_data = sequence.pad_sequences(train_data, maxlen=maxlen)
test_data = sequence.pad_sequences(test_data, maxlen=maxlen)

model = Sequential()
model.add(Embedding(num_words, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data, train_labels,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
Epoch 1/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 29s 174ms/step - accuracy: 0.5985 - loss: 0.6602 - val_accuracy: 0.8026 - val_loss:
0.4273
Epoch 2/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 27s 170ms/step - accuracy: 0.8233 - loss: 0.4050 - val_accuracy: 0.8558 - val_loss:
0.3452
Epoch 3/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 27s 170ms/step - accuracy: 0.8618 - loss: 0.3357 - val_accuracy: 0.8090 - val_loss:
0.4406
Epoch 4/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 27s 172ms/step - accuracy: 0.8811 - loss: 0.2924 - val_accuracy: 0.8304 - val_loss:
0.4823
Epoch 5/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 28s 180ms/step - accuracy: 0.8943 - loss: 0.2755 - val_accuracy: 0.8670 - val_loss:
0.3088
Epoch 6/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 28s 178ms/step - accuracy: 0.9070 - loss: 0.2417 - val_accuracy: 0.8120 - val_loss:
0.5008
Epoch 7/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 27s 169ms/step - accuracy: 0.9003 - loss: 0.2537 - val_accuracy: 0.8786 - val_loss:
0.2888
Epoch 8/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 26s 167ms/step - accuracy: 0.9211 - loss: 0.2105 - val_accuracy: 0.8694 - val_loss:
0.3076
Epoch 9/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 27s 170ms/step - accuracy: 0.9188 - loss: 0.2132 - val_accuracy: 0.8846 - val_loss:
0.2939
Epoch 10/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 28s 175ms/step - accuracy: 0.9364 - loss: 0.1730 - val_accuracy: 0.8752 - val_loss:
0.3026
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js