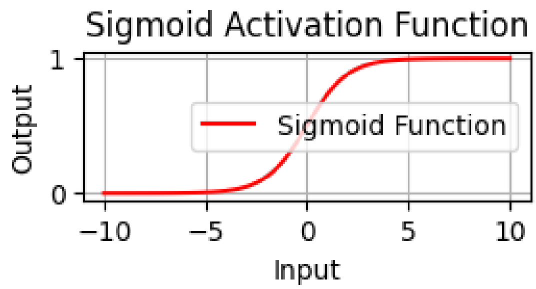
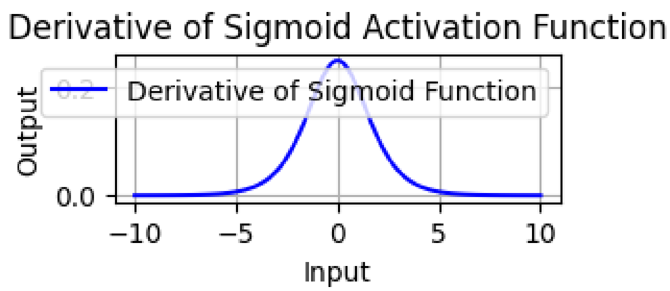
Activation Functions

June 5, 2025

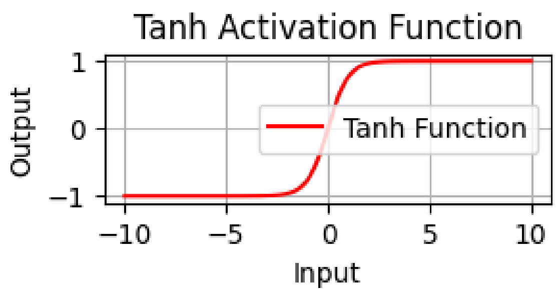
[19]:
```python
#SIGMOID
import numpy as np
import matplotlib.pyplot as plt

# Define the sigmoid function
def sigmoid_function(x):
    return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid function
def sig_der(x):
    return sigmoid_function(x) * (1 - sigmoid_function(x))

# Generate random inputs
X = np.linspace(-10, 10, 100)
y = sigmoid_function(X)
z = sig_der(X)

# Plot the sigmoid function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='Sigmoid Function', color='red')
plt.title('Sigmoid Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot the derivative of the sigmoid function
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Sigmoid Function', color='blue')
plt.title('Derivative of Sigmoid Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```

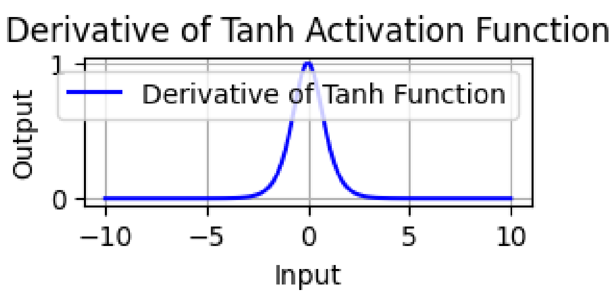




[20]:
```
#TAN
import numpy as np
import matplotlib.pyplot as plt

# Tanh function and its derivative
def tanh(x):
    return np.tanh(x)

def tanh_derivative(x):
    return 1 - np.tanh(x) ** 2

# Generate input data
X = np.linspace(-10, 10, 100)
y = tanh(X)
z = tanh_derivative(X)

# Plot tanh function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='Tanh Function', color='red')
plt.title('Tanh Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
```

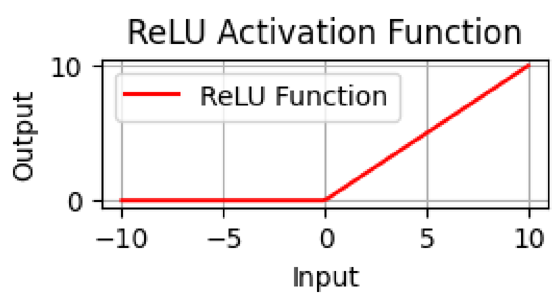
```
plt.legend()
plt.show()

# Plot derivative of tanh
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Tanh Function', color='blue')
plt.title('Derivative of Tanh Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```

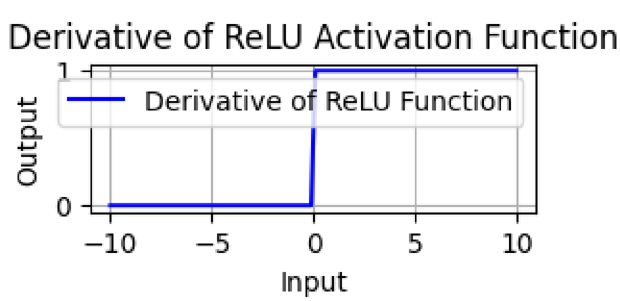




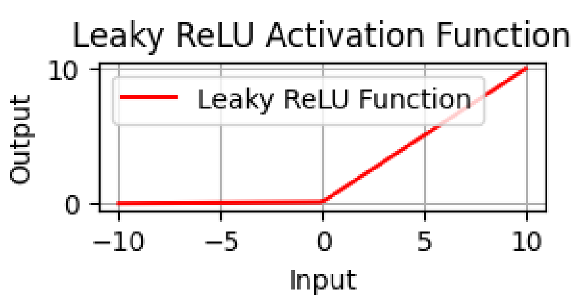
[21]:
```
#RELU
import numpy as np
import matplotlib.pyplot as plt

# ReLU function and its derivative
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x > 0, 1, 0)
```

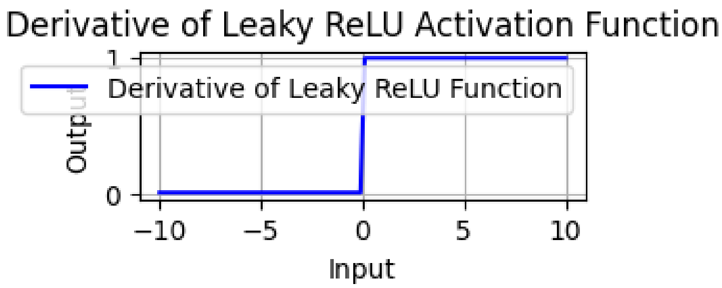
```python
# Generate input data
X = np.linspace(-10, 10, 100)
y = relu(X)
z = relu_derivative(X)

# Plot ReLU function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='ReLU Function', color='red')
plt.title('ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of ReLU
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of ReLU Function', color='blue')
plt.title('Derivative of ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```





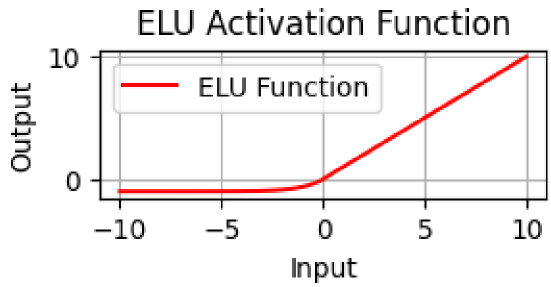
[22]:
```
#LEAKY RELU
import numpy as np
import matplotlib.pyplot as plt

# Leaky ReLU function and its derivative
def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)

def leaky_relu_derivative(x, alpha=0.01):
    return np.where(x > 0, 1, alpha)

# Generate input data
X = np.linspace(-10, 10, 100)
y = leaky_relu(X)
z = leaky_relu_derivative(X)

# Plot Leaky ReLU function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='Leaky ReLU Function', color='red')
plt.title('Leaky ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of Leaky ReLU
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Leaky ReLU Function', color='blue')
plt.title('Derivative of Leaky ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```

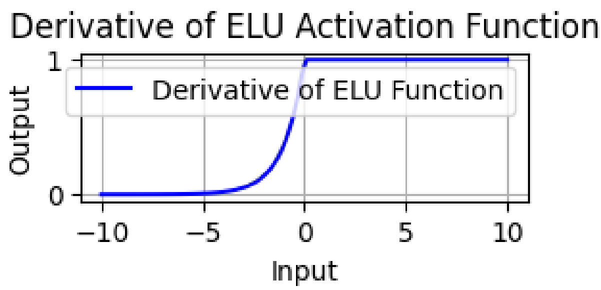




[23]:
```python
#ELU
import numpy as np
import matplotlib.pyplot as plt

# ELU function and its derivative
def elu(x, alpha=1.0):
    return np.where(x > 0, x, alpha * (np.exp(x) - 1))

def elu_derivative(x, alpha=1.0):
    return np.where(x > 0, 1, alpha * np.exp(x))

# Generate input data
X = np.linspace(-10, 10, 100)
y = elu(X)
z = elu_derivative(X)

# Plot ELU function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='ELU Function', color='red')
plt.title('ELU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of ELU
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of ELU Function', color='blue')
plt.title('Derivative of ELU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
```

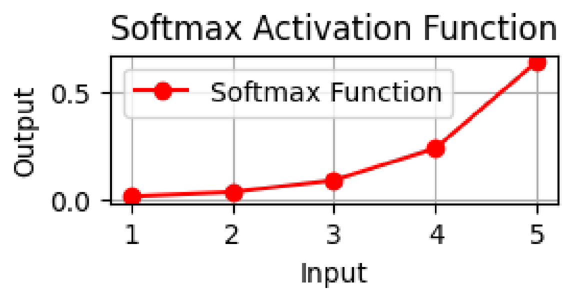
```
plt.legend()
plt.show()
```

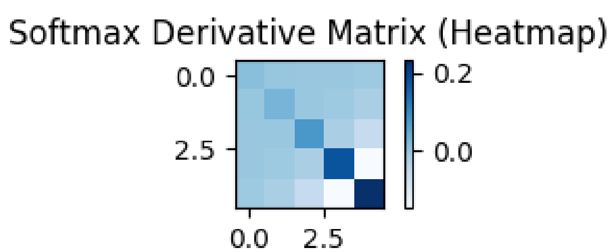




[24]:
```
#SOFTMAX
import numpy as np
import matplotlib.pyplot as plt

# Softmax function and its derivative
def softmax(x):
    exp_x = np.exp(x - np.max(x))  # Subtract max for numerical stability
    return exp_x / np.sum(exp_x)

def softmax_derivative(x):
    s = softmax(x).reshape(-1, 1)
    return np.diagflat(s) - np.dot(s, s.T)

# Generate input data
X = np.array([1, 2, 3, 4, 5])
y = softmax(X)
z = softmax_derivative(X)

# Plot softmax function
plt.figure(figsize=(3, 1))
```

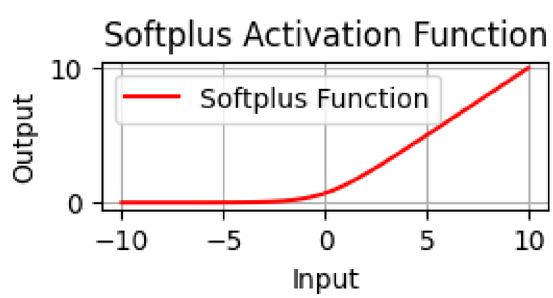
```
plt.plot(X, y, label='Softmax Function', color='red', marker='o')
plt.title('Softmax Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Softmax derivative visualization (heatmap for better visualization)
plt.figure(figsize=(3, 1))
plt.imshow(z, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Softmax Derivative Matrix (Heatmap)')
plt.show()
```

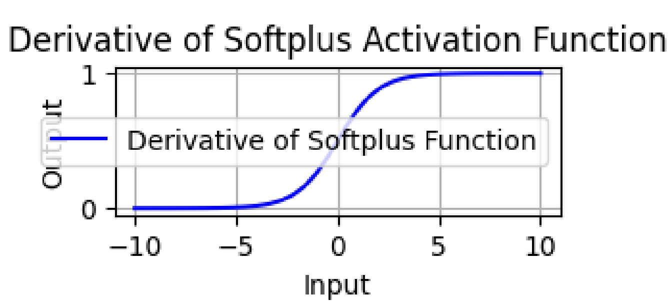




[25]:
```
#SOFTMAX PLUS
import numpy as np
import matplotlib.pyplot as plt

# Softplus function and its derivative
def softplus(x):
    return np.log(1 + np.exp(x))

def softplus_derivative(x):
```

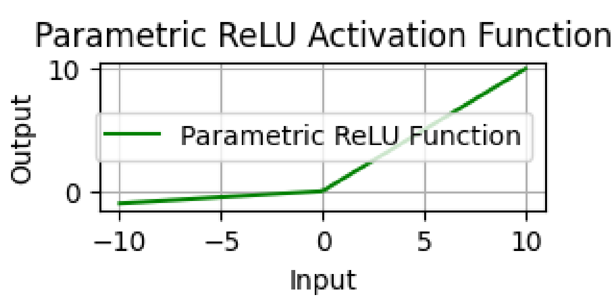
```
    return 1 / (1 + np.exp(-x))

# Generate input data
X = np.linspace(-10, 10, 100)
y = softplus(X)
z = softplus_derivative(X)

# Plot Softplus function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='Softplus Function', color='red')
plt.title('Softplus Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of Softplus
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Softplus Function', color='blue')
plt.title('Derivative of Softplus Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```

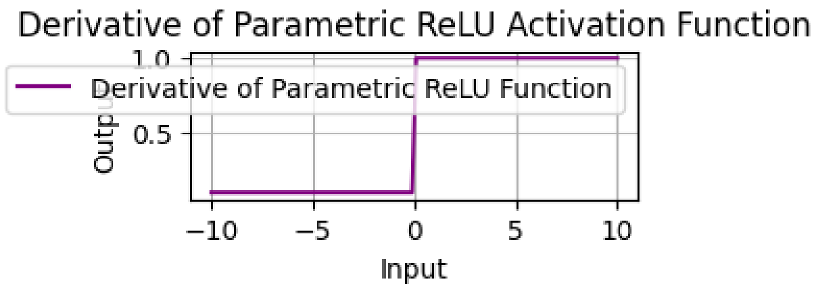




[27]:

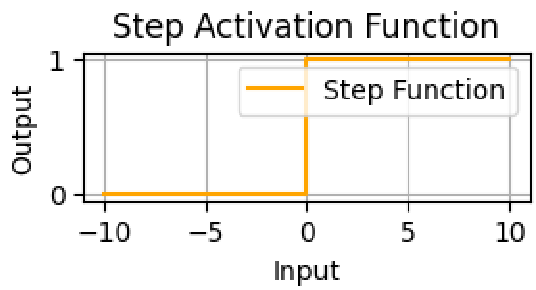
```python
# PARAMETRIC RELU
import numpy as np
import matplotlib.pyplot as plt

# Parametric ReLU function and its derivative
def prelu(x, alpha=0.1):
    return np.where(x >= 0, x, alpha * x)

def prelu_derivative(x, alpha=0.1):
    return np.where(x >= 0, 1, alpha)

# Generate input data
X = np.linspace(-10, 10, 100)
y = prelu(X)
z = prelu_derivative(X)

# Plot Parametric ReLU function
plt.figure(figsize=(3, 1))
plt.plot(X, y, label='Parametric ReLU Function', color='green')
plt.title('Parametric ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of Parametric ReLU
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Parametric ReLU Function', color='purple')
plt.title('Derivative of Parametric ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```

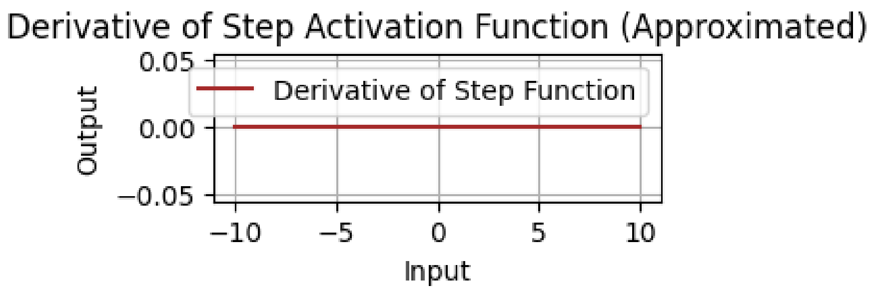




[29]:
```
#STEP FUNCTION
import numpy as np
import matplotlib.pyplot as plt

# Step function and its derivative
def step_function(x):
    return np.where(x >= 0, 1, 0)

def step_function_derivative(x):
    # The derivative of the step function is undefined at 0 and 0 elsewhere,
    # but for practical purposes, it can be approximated as 0 everywhere.
    return np.zeros_like(x)

# Generate input data
X = np.linspace(-10, 10, 100)
y = step_function(X)
z = step_function_derivative(X)

# Plot Step function
plt.figure(figsize=(3, 1))
plt.step(X, y, label='Step Function', color='orange', where='mid')
plt.title('Step Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()

# Plot derivative of Step function
plt.figure(figsize=(3, 1))
plt.plot(X, z, label='Derivative of Step Function', color='brown')
plt.title('Derivative of Step Activation Function (Approximated)')
plt.xlabel('Input')
```

```
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```





[ ]:

```

```