Activation Functions Explained

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

x = np.linspace(-5, 5, 400)

y_sigmoid = sigmoid(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y_sigmoid, label="Sigmoid")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Sigmoid Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
def tanh(x):
    return np.tanh(x)

x = np.linspace(-5, 5, 400)

y_tanh = tanh(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y_tanh, label="Tanh")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Tanh Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
def relu(x):
    return np.maximum(0, x)

x = np.linspace(-5, 5, 400)

y_relu = relu(x)
```

```
plt.figure(figsize=(6, 4))
plt.plot(x, y_relu, label="ReLU")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("ReLU Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
def identity(x):
    return x

x = np.linspace(-5, 5, 400)

y_identity = identity(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y_identity, label="Identity")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Identity Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

x = np.linspace(-5, 5, 400)
y_softmax = softmax(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y_softmax, label="Softmax")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Softmax Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
 .....expalin this code line by line
```

This code demonstrates the use of various activation functions commonly used in machine learning and neural networks. Let's break down the code line by line:

1. Importing Libraries

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

- `import numpy as np`: This imports the `numpy` library, a powerful library for numerical computing in Python. It is used to handle arrays and perform mathematical operations.
- `import matplotlib.pyplot as plt`: This imports the `matplotlib` library for creating static, animated, and interactive visualizations in Python. Specifically, `pyplot` is used for plotting graphs.

2. Sigmoid Activation Function

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

- This defines a function `sigmoid(x)`, which computes the Sigmoid activation function.
- The formula for Sigmoid is: $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$, where e is the base of the natural logarithm.
- `np.exp(-x)` computes the exponential function e^{-x}, and the result is used to compute the Sigmoid function.

```python
x = np.linspace(-5, 5, 400)
```

- `np.linspace(-5, 5, 400)` generates 400 equally spaced numbers between -5 and 5. This creates a smooth range of inputs for the Sigmoid function.

```python
y_sigmoid = sigmoid(x)
```

- This calculates the output of the Sigmoid function for each value in `x`.

```python
plt.figure(figsize=(6, 4))
plt.plot(x, y_sigmoid, label="Sigmoid")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Sigmoid Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

- This block of code generates the plot for the Sigmoid function:
  - `plt.figure(figsize=(6, 4))`: Sets the size of the figure to 6 inches by 4 inches.

- `plt.plot(x, y_sigmoid, label="Sigmoid")`: Plots the Sigmoid function, with `x` as the input and `y_sigmoid` as the output.
- `plt.axhline(0, color='black', linewidth=0.5)`: Draws a horizontal line at y=0.
- `plt.axvline(0, color='black', linewidth=0.5)`: Draws a vertical line at x=0.
- `plt.legend()`: Displays the legend for the plot.
- `plt.title()`, `plt.xlabel()`, `plt.ylabel()`: Sets the title and axis labels.
- `plt.grid(True)`: Displays grid lines for better readability.
- `plt.show()`: Displays the plot.

3. Tanh Activation Function

```
python

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

- Defines a function `tanh(x)` that computes the Tanh activation function. The Tanh function is defined as $\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, and `np.tanh(x)` is a built-in NumPy function that computes this for you.

```
python

x = np.linspace(-5, 5, 400)
y_tanh = tanh(x)
```

- Generates the input range `x` and calculates the output of the Tanh function.

```
python

plt.figure(figsize=(6, 4))
plt.plot(x, y_tanh, label="Tanh")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Tanh Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

- Similar to the previous plotting code, it generates and displays the plot for the Tanh function.

4. ReLU Activation Function

```
python

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

- Defines the ReLU (Rectified Linear Unit) activation function. It returns the maximum of 0 and `x` for each element in `x`, effectively replacing negative values with 0 and keeping positive values

unchanged.

```
python

x = np.linspace(-5, 5, 400)
y_relu = relu(x)
```

- Generates the input range `x` and calculates the output of the ReLU function.

```
python

plt.figure(figsize=(6, 4))
plt.plot(x, y_relu, label="ReLU")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("ReLU Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

- Generates and displays the plot for the ReLU function.

5. Identity Activation Function

```
python

def identity(x):
    return x
```

- Defines the identity function. The identity function simply returns the input `x` unchanged.

```
python

x = np.linspace(-5, 5, 400)
y_identity = identity(x)
```

- Generates the input range `x` and calculates the output of the identity function.

```
python

plt.figure(figsize=(6, 4))
plt.plot(x, y_identity, label="Identity")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Identity Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

- Generates and displays the plot for the identity function.

6. Softmax Activation Function

```
python

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()
```

- Defines the Softmax activation function. Softmax is used to convert a vector of values into a probability distribution. The formula is:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

  - First, `np.exp(x - np.max(x))` calculates the exponentials of the input values, while subtracting `np.max(x)` helps prevent overflow issues with large values.
  - Then, `e_x / e_x.sum()` normalizes the exponential values to sum up to 1.

```
python

x = np.linspace(-5, 5, 400)
y_softmax = softmax(x)
```

- Generates the input range `x` and calculates the output of the Softmax function.

```
python

plt.figure(figsize=(6, 4))
plt.plot(x, y_softmax, label="Softmax")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.title("Softmax Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

- Generates and displays the plot for the Softmax function.

Summary:

This code defines and visualizes five different activation functions commonly used in machine learning:

1. **Sigmoid**
2. **Tanh**
3. **ReLU**
4. **Identity**
5. **Softmax**

For each activation function, the code generates an input range (`x`), computes the output using the respective activation function, and then plots the graph using `matplotlib`.