Assignment on Activation Function

MTech in Applied AI
Deep Learning Techniques

by

Sanjeev Kumar Pandey[*]
Faculty: Dr Anamika Gupta

Submitted April 12, 2025

[*] Student ID: 31050, Enrolment No:MT23AAI001, Email: pandey.sanjeev@yahoo.com

Table of Contents

1 Assignment on Activation Function

Activation functions play a critical role in neural networks by introducing non-linearity. Take the sigmoid function for example.

Explore:

1. Its response to changes in weights (ω) and bias (b).
2. Visualize a multi-layer 2D activation tower.

2 Effect of change on weight and biases

2.1 Python code - effect of change on weight and biases

The code depicts the response to changes in weights (ω) and bias (b).

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

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

x = np.linspace(-10, 10, 500)

# Weight variations
plt.figure(figsize=(12, 6))
for w in [0.5, 1, 2, 5]:
    plt.plot(x, sigmoid(w*x), label=f'w={w}, b=0')
plt.title('Sigmoid Response to Weight Changes (b=0)')
plt.xlabel('x')
plt.ylabel('  (  x  + b)')
plt.legend()
plt.grid(True)
plt.savefig('response_to_weight_changes.png')
plt.close()

# Bias variations
plt.figure(figsize=(12, 6))
for b in [-2, 0, 2, 5]:
    plt.plot(x, sigmoid(x + b), label=f'  =1, b={b}')
plt.title('Sigmoid Response to Bias Changes (  =1)')
plt.xlabel('x')
plt.ylabel('  (  x  + b)')
plt.legend()
plt.grid(True)
plt.savefig('response_to_bias_changes.png')
plt.close()
```

Listing 1: Code

2.2 Output - Response to change on weight

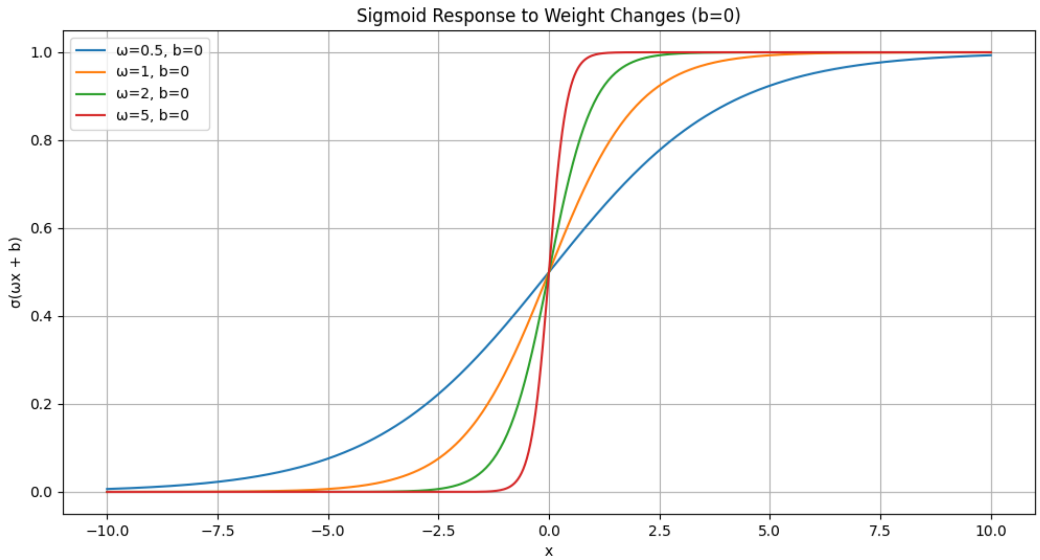

Figure 1: Loss vs Weights and Biases Plot

2.3 Output - Response to change on bias

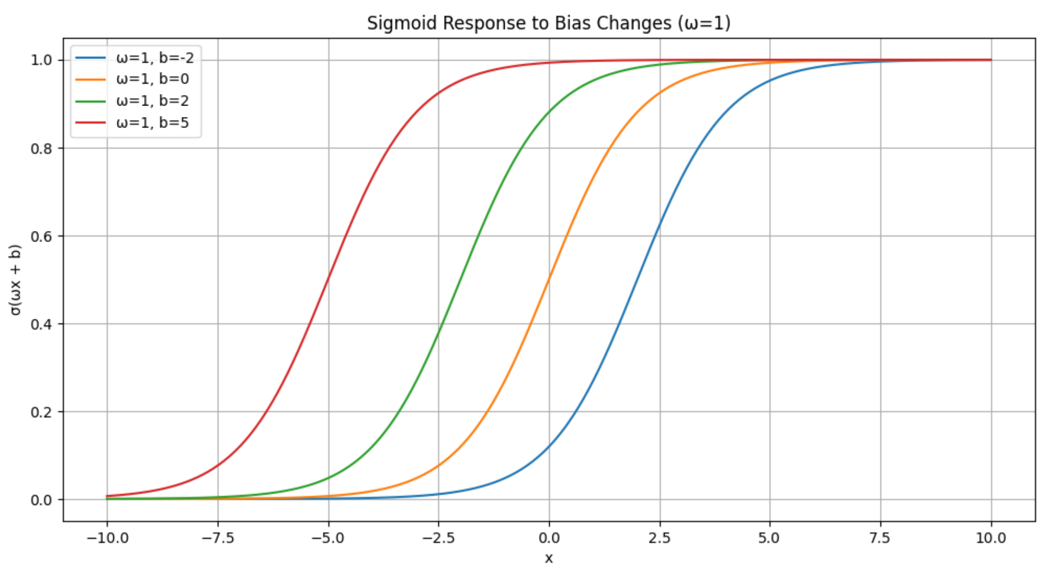

Figure 2: Loss vs Weights and Biases Plot

2.4 Analysis - Response to change on weight and biases

The provided program visualizes how changes in **weight** (ω) and **bias** (b) affect the sigmoid activation function, which is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = \omega x + b \tag{2.1}$$

Effect of Changing the Weight (ω)

When $b = 0$, the sigmoid function becomes:

$$\sigma(\omega x) = \frac{1}{1 + e^{-\omega x}} \tag{2.2}$$

- Increasing ω makes the sigmoid **steeper**, i.e., it transitions more quickly from 0 to 1.
- Decreasing ω **flattens** the sigmoid, making the transition more gradual.

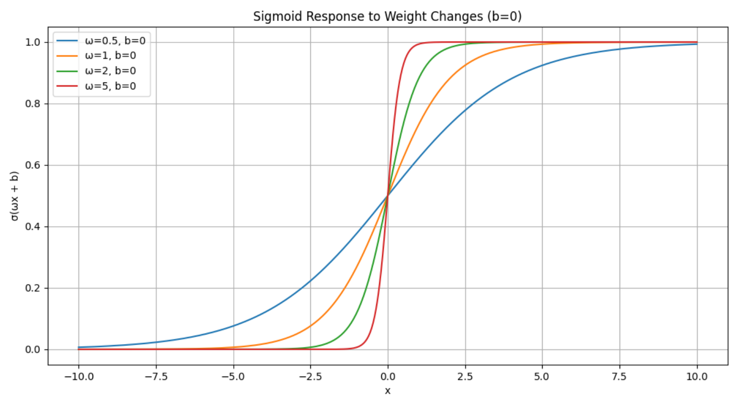

Figure 3: Sigmoid response to different weights (ω) with $b = 0$

Effect of Changing the Bias (b)

When $\omega = 1$, the sigmoid function becomes:

$$\sigma(x + b) = \frac{1}{1 + e^{-(x+b)}} \tag{2.3}$$

- Increasing b shifts the sigmoid curve **to the left**.
- Decreasing b shifts the sigmoid curve **to the right**.

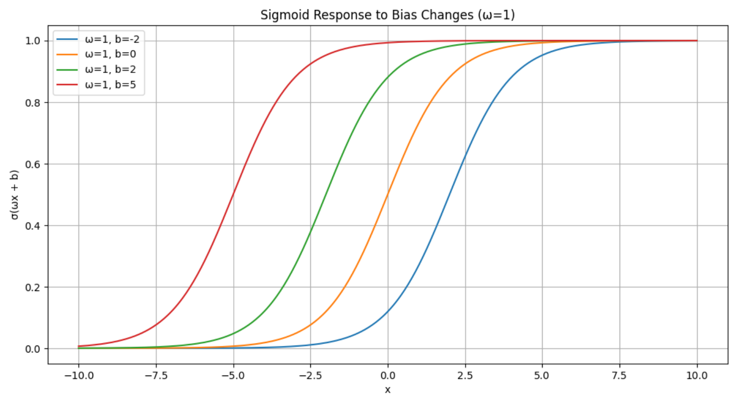

Figure 4: Sigmoid response to different biases (b) with $\omega = 1$

Conclusion

- The **weight** (ω) controls the *steepness* or *sensitivity* of the sigmoid function.
- The **bias** (b) controls the *horizontal shift* of the function.

3 Multi-layer 2D activation tower

3.1 Python code - multi-layer 2D activation tower

Visualize a multi-layer 2D activation tower.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

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

def plot_sigmoid_towers():
    # Create a 2D grid
    x1 = np.linspace(-5, 5, 100)
    x2 = np.linspace(-5, 5, 100)
    X1, X2 = np.meshgrid(x1, x2)

    # Define tower centers
    a1, b1 = -2, -2
    a2, b2 = 2, 2

    # Combine two sigmoid towers
    Z = sigmoid(-((X1 - a1)**2 + (X2 - b1)**2)) + sigmoid(-((X1 - a2)**2 + (X2 - b2)**2))

    # Plot
    fig = plt.figure(figsize=(10, 5))

    # Heatmap
    ax1 = fig.add_subplot(1, 2, 1)
    contour = ax1.contourf(X1, X2, Z, cmap='plasma')
    fig.colorbar(contour, ax=ax1)
    ax1.set_title("Sigmoid Towers - Heatmap")
    ax1.set_xlabel("X1")
    ax1.set_ylabel("X2")
```

```

    # 3D Surface
    ax2 = fig.add_subplot(1, 2, 2, projection='3d')
    ax2.plot_surface(X1, X2, Z, cmap='plasma')
    ax2.set_title("Sigmoid Towers - 3D Surface")
    ax2.set_xlabel("X1")
    ax2.set_ylabel("X2")
    ax2.set_zlabel("Activation")

    plt.tight_layout()
    plt.savefig("sigmoid_towers.png")
    plt.show()

# Call the tower plotting function
plot_sigmoid_towers()
```

Listing 2: Code

3.2 Output - multi-layer 2D activation tower

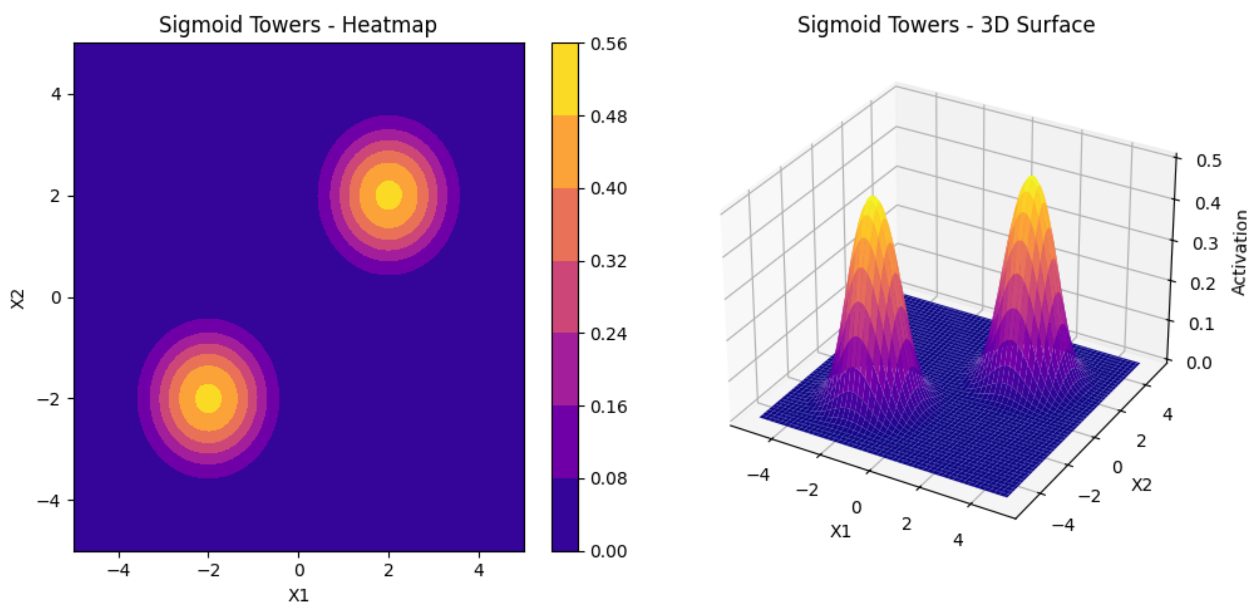

Figure 5: Sigmoid 2-d Towers

3.3 Analysis - multi-layer 2D activation tower

Concept	**What it Does**
`sigmoid(-distance^2)`	Creates a smooth, localized "tower" of activation
Two sigmoids added	Mimics a layer with two activated neurons
Heatmap	2D view of activation intensity
3D surface	Shape of activation across input space
Analogy	Simulates how a layer processes spatial input non-linearly

Table 1: Summary of key components in sigmoid tower visualization

This visualization demonstrates how two sigmoidal functions can be used to create localized "towers" of activation in a two-dimensional space. This is similar to how neurons can be tuned to respond strongly to specific regions of the input space.

Sigmoid Definition in 2D

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.1}$$

To create a 2D "bump" centered at (a, b), we use:

$$z = -((x_1 - a)^2 + (x_2 - b)^2) \tag{3.2}$$

$$\Rightarrow \sigma(z) = \frac{1}{1 + \exp\left((x_1 - a)^2 + (x_2 - b)^2\right)} \tag{3.3}$$

This results in a smooth, localized activation with a peak at $(x_1 = a, x_2 = b)$ and decaying smoothly in all directions.

Adding Two Sigmoid Towers

To create multiple activations, we can simply add two such sigmoid functions:

$$Z(x_1, x_2) = \sigma(-((x_1 - a_1)^2 + (x_2 - b_1)^2)) + \sigma(-((x_1 - a_2)^2 + (x_2 - b_2)^2)) \tag{3.4}$$

For example, setting:

$$(a_1, b_1) = (-2, -2), \quad (a_2, b_2) = (2, 2)$$

creates two "towers" centered at those points.

Visualization

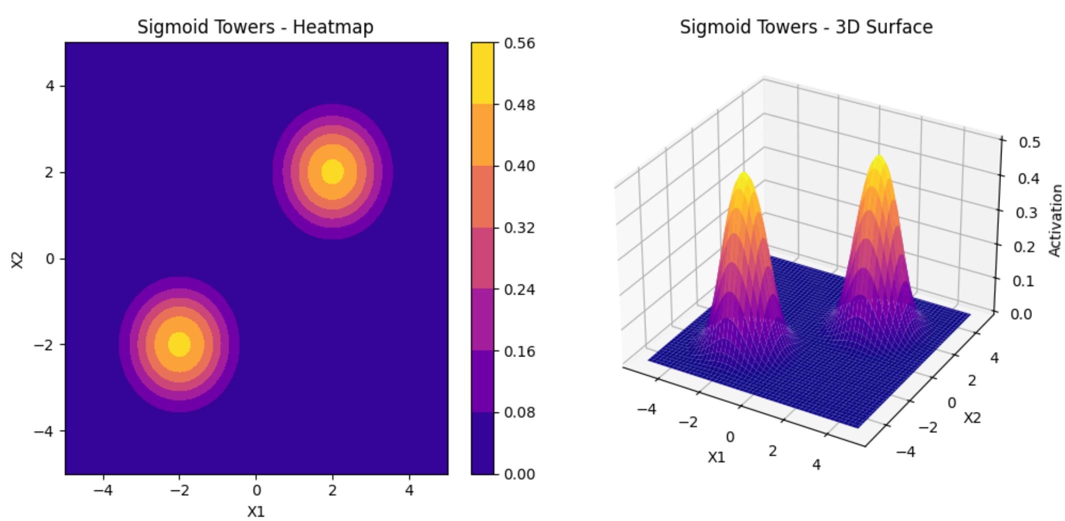

Figure 6: Heatmap and 3D surface of two added sigmoid towers centered at $(-2, -2)$ and $(2, 2)$

Interpretation

- Each sigmoid represents a localized response, with the peak activation at the center and smooth fall-off.
- Adding two sigmoids creates a composite surface with two high-activation regionssimilar to multiple neurons responding to different input features.
- This approach is a simplified version of how neural networks or RBF (Radial Basis Function) networks encode features across input space.