Activation Functions Analysis

This notebook explores various activation functions, their derivatives, and error metrics.

Task 1: Sigmoid and Tanh Activation Functions

- Input range: (-10, +10)
- Plot activation functions and their derivatives
- Observe behavior and characteristics
- Calculate MSE and MAE with sample predictions

Task 2: Tanh and ReLU Activation Functions

- Input range: (-5, +5)
- Plot activation functions and their derivatives
- Observe behavior and characteristics
- Calculate MSE and MAE with sample predictions

Task 3: Sigmoid, ReLU and Softmax Activation Functions

- Input range: (-10, +10)
- Plot activation functions and their derivatives
- Observe behavior and characteristics
- Calculate MSE and MAE with sample predictions

Import Libraries

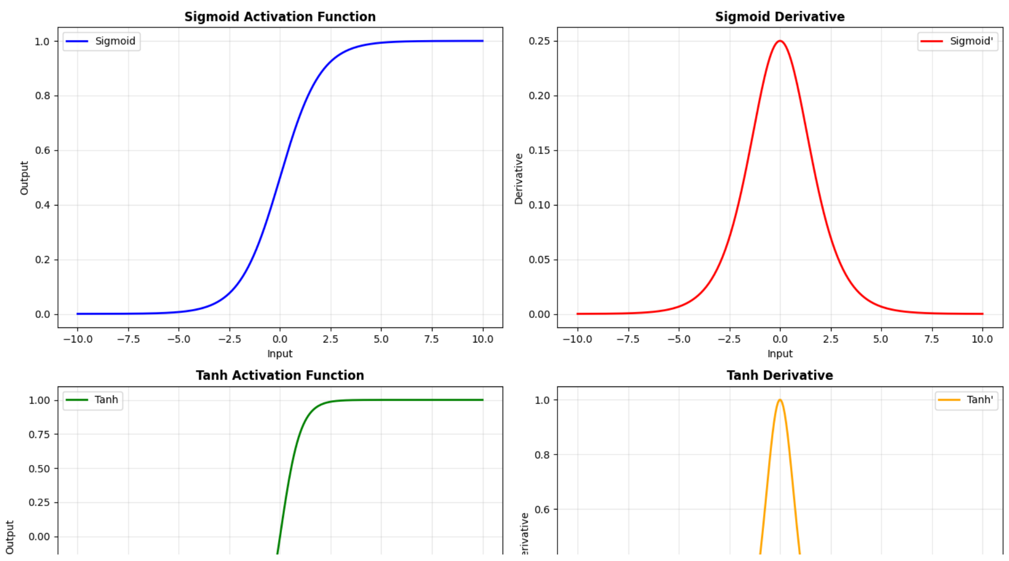
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
```

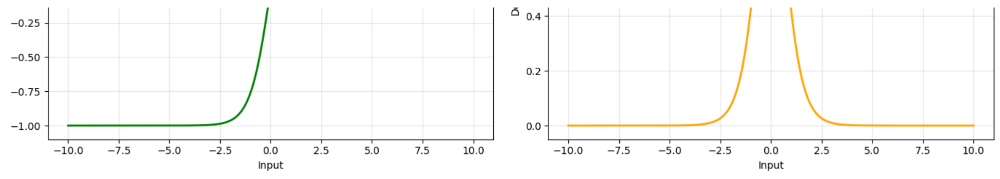
Define Activation Functions

In [2]:
```python
def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)

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

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

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

def relu_derivative(x):
    return (x > 0).astype(float)

def softmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / np.sum(exp_x, axis=0)

def softmax_derivative(x):
    s = softmax(x)
    return s * (1 - s)
```

Task 1: Sigmoid and Tanh Functions (-10 to +10)

In [3]:
```
x_task1 = np.linspace(-10, 10, 300)

sigmoid_output = sigmoid(x_task1)
sigmoid_deriv = sigmoid_derivative(x_task1)

tanh_output = tanh_function(x_task1)
tanh_deriv = tanh_derivative(x_task1)

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].plot(x_task1, sigmoid_output, 'b-', linewidth=2, label='Sigmoid')
axes[0, 0].set_title('Sigmoid Activation Function', fontsize=12, fontweight='bold')
axes[0, 0].set_xlabel('Input')
axes[0, 0].set_ylabel('Output')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].legend()

axes[0, 1].plot(x_task1, sigmoid_deriv, 'r-', linewidth=2, label="Sigmoid'")
axes[0, 1].set_title('Sigmoid Derivative', fontsize=12, fontweight='bold')
axes[0, 1].set_xlabel('Input')
axes[0, 1].set_ylabel('Derivative')
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].legend()

axes[1, 0].plot(x_task1, tanh_output, 'g-', linewidth=2, label='Tanh')
axes[1, 0].set_title('Tanh Activation Function', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Input')
axes[1, 0].set_ylabel('Output')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].legend()

axes[1, 1].plot(x_task1, tanh_deriv, 'orange', linewidth=2, label="Tanh'")
axes[1, 1].set_title('Tanh Derivative', fontsize=12, fontweight='bold')
axes[1, 1].set_xlabel('Input')
axes[1, 1].set_ylabel('Derivative')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].legend()

plt.tight_layout()
plt.show()

print("\nTask 1 - Sigmoid and Tanh Analysis:")
print("=" * 50)
print(f"Sigmoid output range: [{sigmoid_output.min():.4f},
{sigmoid_output.max():.4f}]")
print(f"Sigmoid derivative max: {sigmoid_deriv.max():.4f} at x =
{x_task1[sigmoid_deriv.argmax()]:.2f}")
print(f"\nTanh output range: [{tanh_output.min():.4f}, {tanh_output.max():.4f}]")
print(f"Tanh derivative max: {tanh_deriv.max():.4f} at x =
{x_task1[tanh_deriv.argmax()]:.2f}")
```





```
Task 1 - Sigmoid and Tanh Analysis:
==================================================
Sigmoid output range: [0.0000, 1.0000]
Sigmoid derivative max: 0.2499 at x = -0.03

Tanh output range: [-1.0000, 1.0000]
Tanh derivative max: 0.9989 at x = -0.03
```

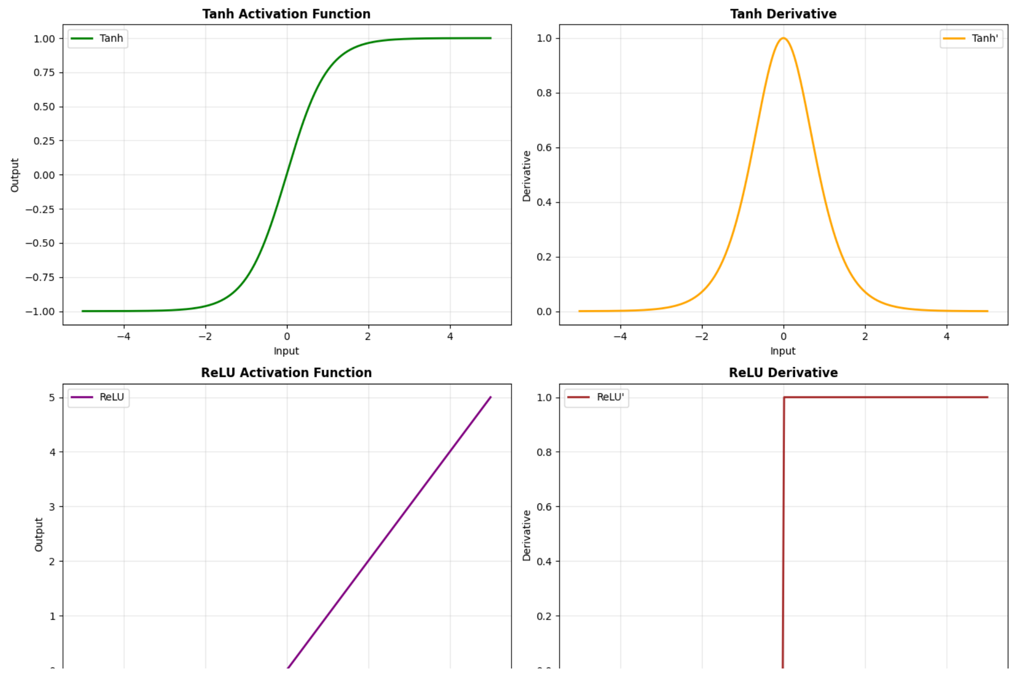
Task 1: Error Metrics (MSE and MAE)

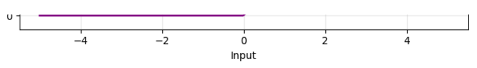
In [4]:
```
y_true_task1 = np.random.rand(50)
y_pred_sigmoid = sigmoid(np.random.uniform(-10, 10, 50))
y_pred_tanh = tanh_function(np.random.uniform(-10, 10, 50))

mse_sigmoid = mean_squared_error(y_true_task1, y_pred_sigmoid)
mae_sigmoid = mean_absolute_error(y_true_task1, y_pred_sigmoid)

mse_tanh = mean_squared_error(y_true_task1, y_pred_tanh)
mae_tanh = mean_absolute_error(y_true_task1, y_pred_tanh)

print("\nTask 1 - Error Metrics:")
print("=" * 50)
print(f"Sigmoid - MSE: {mse_sigmoid:.6f}, MAE: {mae_sigmoid:.6f}")
print(f"Tanh    - MSE: {mse_tanh:.6f}, MAE: {mae_tanh:.6f}")
```

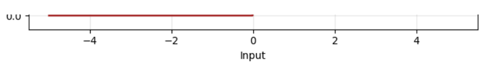
```
Task 1 - Error Metrics:
==================================================
Sigmoid - MSE: 0.287359, MAE: 0.452340
Tanh    - MSE: 1.010884, MAE: 0.833134
```

Task 2: Tanh and ReLU Functions (-5 to +5)

In [5]:
```
x_task2 = np.linspace(-5, 5, 300)

tanh_output_t2 = tanh_function(x_task2)
tanh_deriv_t2 = tanh_derivative(x_task2)

relu_output = relu(x_task2)
relu_deriv = relu_derivative(x_task2)

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].plot(x_task2, tanh_output_t2, 'g-', linewidth=2, label='Tanh')
axes[0, 0].set_title('Tanh Activation Function', fontsize=12, fontweight='bold')
axes[0, 0].set_xlabel('Input')
axes[0, 0].set_ylabel('Output')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].legend()

axes[0, 1].plot(x_task2, tanh_deriv_t2, 'orange', linewidth=2, label="Tanh'")
axes[0, 1].set_title('Tanh Derivative', fontsize=12, fontweight='bold')
axes[0, 1].set_xlabel('Input')
axes[0, 1].set_ylabel('Derivative')
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].legend()

axes[1, 0].plot(x_task2, relu_output, 'purple', linewidth=2, label='ReLU')
axes[1, 0].set_title('ReLU Activation Function', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Input')
axes[1, 0].set_ylabel('Output')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].legend()

axes[1, 1].plot(x_task2, relu_deriv, 'brown', linewidth=2, label="ReLU'")
axes[1, 1].set_title('ReLU Derivative', fontsize=12, fontweight='bold')
axes[1, 1].set_xlabel('Input')
axes[1, 1].set_ylabel('Derivative')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].legend()

plt.tight_layout()
plt.show()

print("\nTask 2 - Tanh and ReLU Analysis:")
print("=" * 50)
print(f"Tanh output range: [{tanh_output_t2.min():.4f}, {tanh_output_t2.max():.4f}]")
print(f"ReLU output range: [{relu_output.min():.4f}, {relu_output.max():.4f}]")
print(f"ReLU is zero for x < 0: {np.all(relu_output[x_task2 < 0] == 0)}")
```







```
Task 2 - Tanh and ReLU Analysis:
==================================================
Tanh output range: [-0.9999, 0.9999]
ReLU output range: [0.0000, 5.0000]
ReLU is zero for x < 0: True
```

Task 2: Error Metrics (MSE and MAE)

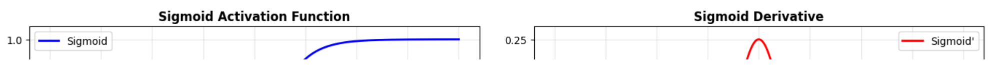
In [6]:

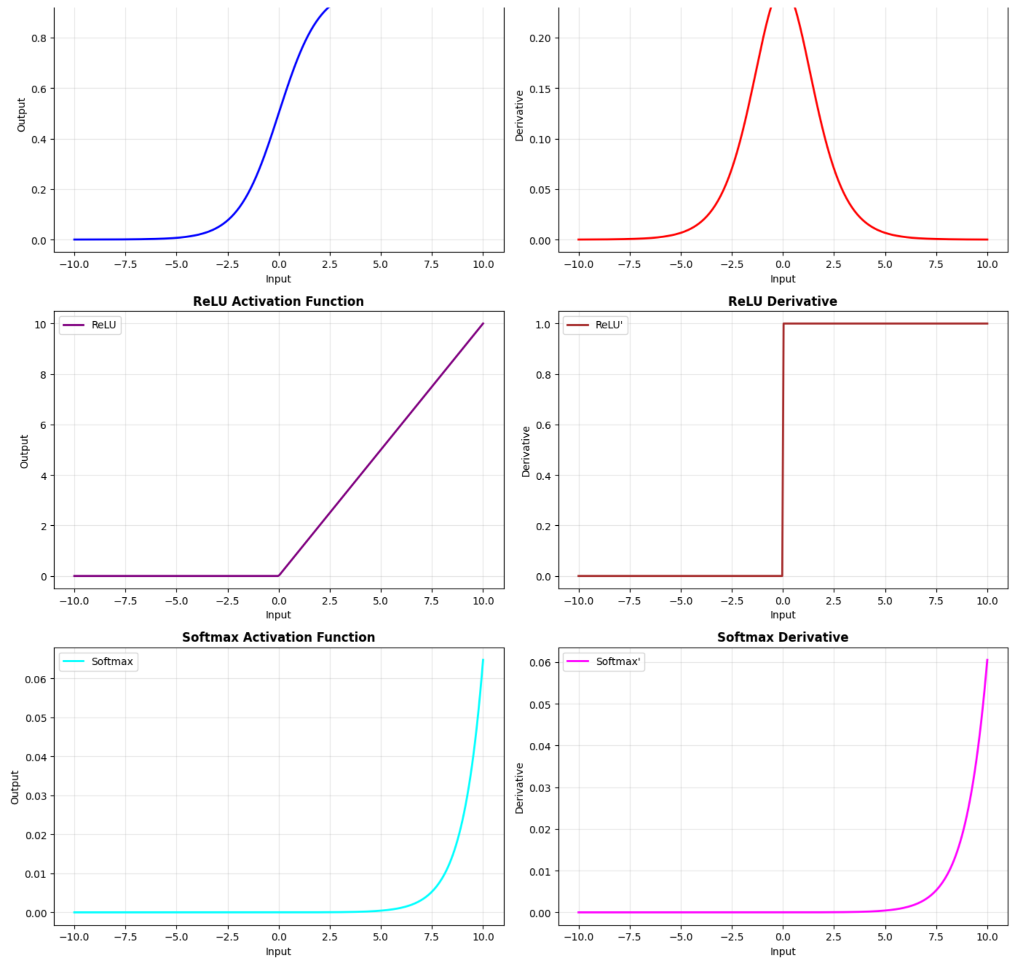
```
y_true_task2 = np.random.rand(50)
y_pred_tanh_t2 = tanh_function(np.random.uniform(-5, 5, 50))
y_pred_relu = relu(np.random.uniform(-5, 5, 50))

mse_tanh_t2 = mean_squared_error(y_true_task2, y_pred_tanh_t2)
mae_tanh_t2 = mean_absolute_error(y_true_task2, y_pred_tanh_t2)

mse_relu = mean_squared_error(y_true_task2, y_pred_relu)
mae_relu = mean_absolute_error(y_true_task2, y_pred_relu)

print("\nTask 2 - Error Metrics:")
print("=" * 50)
print(f"Tanh - MSE: {mse_tanh_t2:.6f}, MAE: {mae_tanh_t2:.6f}")
print(f"ReLU - MSE: {mse_relu:.6f}, MAE: {mae_relu:.6f}")
```

```
Task 2 - Error Metrics:
==================================================
Tanh - MSE: 1.250409, MAE: 0.960793
ReLU - MSE: 2.753757, MAE: 1.196055
```

Task 3: Sigmoid, ReLU and Softmax Functions (-10 to +10)

In [7]:
```
x_task3 = np.linspace(-10, 10, 300)

sigmoid_output_t3 = sigmoid(x_task3)
sigmoid_deriv_t3 = sigmoid_derivative(x_task3)

relu_output_t3 = relu(x_task3)
relu_deriv_t3 = relu_derivative(x_task3)

x_softmax = np.linspace(-10, 10, 300).reshape(-1, 1)
softmax_output = softmax(x_softmax).flatten()
softmax_deriv = softmax_derivative(x_softmax).flatten()

fig, axes = plt.subplots(3, 2, figsize=(14, 14))

axes[0, 0].plot(x_task3, sigmoid_output_t3, 'b-', linewidth=2, label='Sigmoid')
axes[0, 0].set_title('Sigmoid Activation Function', fontsize=12, fontweight='bold')
axes[0, 0].set_xlabel('Input')
axes[0, 0].set_ylabel('Output')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].legend()

axes[0, 1].plot(x_task3, sigmoid_deriv_t3, 'r-', linewidth=2, label="Sigmoid'")
axes[0, 1].set_title('Sigmoid Derivative', fontsize=12, fontweight='bold')
axes[0, 1].set_xlabel('Input')
axes[0, 1].set_ylabel('Derivative')
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].legend()

axes[1, 0].plot(x_task3, relu_output_t3, 'purple', linewidth=2, label='ReLU')
axes[1, 0].set_title('ReLU Activation Function', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Input')
axes[1, 0].set_ylabel('Output')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].legend()

axes[1, 1].plot(x_task3, relu_deriv_t3, 'brown', linewidth=2, label="ReLU'")
axes[1, 1].set_title('ReLU Derivative', fontsize=12, fontweight='bold')
axes[1, 1].set_xlabel('Input')
axes[1, 1].set_ylabel('Derivative')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].legend()

axes[2, 0].plot(x_task3, softmax_output, 'cyan', linewidth=2, label='Softmax')
axes[2, 0].set_title('Softmax Activation Function', fontsize=12, fontweight='bold')
axes[2, 0].set_xlabel('Input')
axes[2, 0].set_ylabel('Output')
axes[2, 0].grid(True, alpha=0.3)
axes[2, 0].legend()

axes[2, 1].plot(x_task3, softmax_deriv, 'magenta', linewidth=2, label="Softmax'")
axes[2, 1].set_title('Softmax Derivative', fontsize=12, fontweight='bold')
axes[2, 1].set_xlabel('Input')
axes[2, 1].set_ylabel('Derivative')
axes[2, 1].grid(True, alpha=0.3)
axes[2, 1].legend()

plt.tight_layout()
plt.show()

print("\nTask 3 - Sigmoid, ReLU and Softmax Analysis:")
print("=" * 50)
print(f"Sigmoid output range: [{sigmoid_output_t3.min():.4f},
{sigmoid_output_t3.max():.4f}]")
print(f"ReLU output range: [{relu_output_t3.min():.4f}, {relu_output_t3.max():.4f}]")
print(f"Softmax output range: [{softmax_output.min():.4f},
{softmax_output.max():.4f}]")
print(f"Softmax output sum: {np.sum(softmax_output):.6f}")
```





```
Task 3 - Sigmoid, ReLU and Softmax Analysis:
==================================================
Sigmoid output range: [0.0000, 1.0000]
ReLU output range: [0.0000, 10.0000]
Softmax output range: [0.0000, 0.0647]
Softmax output sum: 1.000000
```

Task 3: Error Metrics (MSE and MAE)

In [8]:
```
y_true_task3 = np.random.rand(50)
y_pred_sigmoid_t3 = sigmoid(np.random.uniform(-10, 10, 50))
y_pred_relu_t3 = relu(np.random.uniform(-10, 10, 50))
y_pred_softmax = softmax(np.random.uniform(-10, 10, 50).reshape(-1, 1)).flatten()

mse_sigmoid_t3 = mean_squared_error(y_true_task3, y_pred_sigmoid_t3)
mae_sigmoid_t3 = mean_absolute_error(y_true_task3, y_pred_sigmoid_t3)

mse_relu_t3 = mean_squared_error(y_true_task3, y_pred_relu_t3)
mae_relu_t3 = mean_absolute_error(y_true_task3, y_pred_relu_t3)

mse_softmax = mean_squared_error(y_true_task3, y_pred_softmax)
mae_softmax = mean_absolute_error(y_true_task3, y_pred_softmax)

print("\nTask 3 - Error Metrics:")
print("=" * 50)
print(f"Sigmoid - MSE: {mse_sigmoid_t3:.6f}, MAE: {mae_sigmoid_t3:.6f}")
print(f"ReLU    - MSE: {mse_relu_t3:.6f}, MAE: {mae_relu_t3:.6f}")
print(f"Softmax - MSE: {mse_softmax:.6f}, MAE: {mae_softmax:.6f}")
```

```
Task 3 - Error Metrics:
==================================================
Sigmoid - MSE: 0.296429, MAE: 0.474559
ReLU    - MSE: 16.294083, MAE: 2.650332
Softmax - MSE: 0.230787, MAE: 0.399618
```

Summary and Observations

In [9]:

```
summary = """
KEY OBSERVATIONS AND CHARACTERISTICS:

Sigmoid Function:
- Output range: (0, 1) - suitable for binary classification
- S-shaped curve with smooth transition
- Maximum derivative at x=0 (0.25)
- Suffers from vanishing gradient problem in deep networks
- Used in output layer for binary classification

Tanh Function:
- Output range: (-1, 1) - centered around zero
- S-shaped curve, smoother transitions than sigmoid
- Maximum derivative at x=0 (1.0) - stronger gradients
- Better for hidden layers than sigmoid
- Converges faster than sigmoid

ReLU Function:
- Output range: [0, ∞) - allows unbounded activation
- Linear for positive inputs, zero for negative
- Derivative: 1 for x > 0, 0 for x < 0
- Computationally efficient (simple comparison)
- Reduces vanishing gradient problem
- Most popular in modern deep networks

Softmax Function:
- Output: Probability distribution (sum = 1)
- Converts logits to probabilities
- Used in multi-class classification
- Output range: (0, 1) for each element
- Maintains relative ordering of inputs

ERROR METRICS INSIGHTS:
- MSE penalizes larger errors more (quadratic)
- MAE treats all errors equally (linear)
- Lower values indicate better predictions
- Choice depends on problem requirements and outlier sensitivity
"""

print(summary)
```

```
KEY OBSERVATIONS AND CHARACTERISTICS:

Sigmoid Function:
- Output range: (0, 1) - suitable for binary classification
- S-shaped curve with smooth transition
- Maximum derivative at x=0 (0.25)
- Suffers from vanishing gradient problem in deep networks
- Used in output layer for binary classification

Tanh Function:
- Output range: (-1, 1) - centered around zero
- S-shaped curve, smoother transitions than sigmoid
- Maximum derivative at x=0 (1.0) - stronger gradients
- Better for hidden layers than sigmoid
- Converges faster than sigmoid

ReLU Function:
- Output range: [0, ∞) - allows unbounded activation
- Linear for positive inputs, zero for negative
- Derivative: 1 for x > 0, 0 for x < 0
- Computationally efficient (simple comparison)
- Reduces vanishing gradient problem
- Most popular in modern deep networks

Softmax Function:
- Output: Probability distribution (sum = 1)
- Converts logits to probabilities
- Used in multi-class classification
- Output range: (0, 1) for each element
- Maintains relative ordering of inputs

ERROR METRICS INSIGHTS:
- MSE penalizes larger errors more (quadratic)
- MAE treats all errors equally (linear)
- Lower values indicate better predictions
- Choice depends on problem requirements and outlier sensitivity
```