Lab - 1 | 22 January 2026

↳ 6 cells hidden

Lab - 2 | 29 January 2026

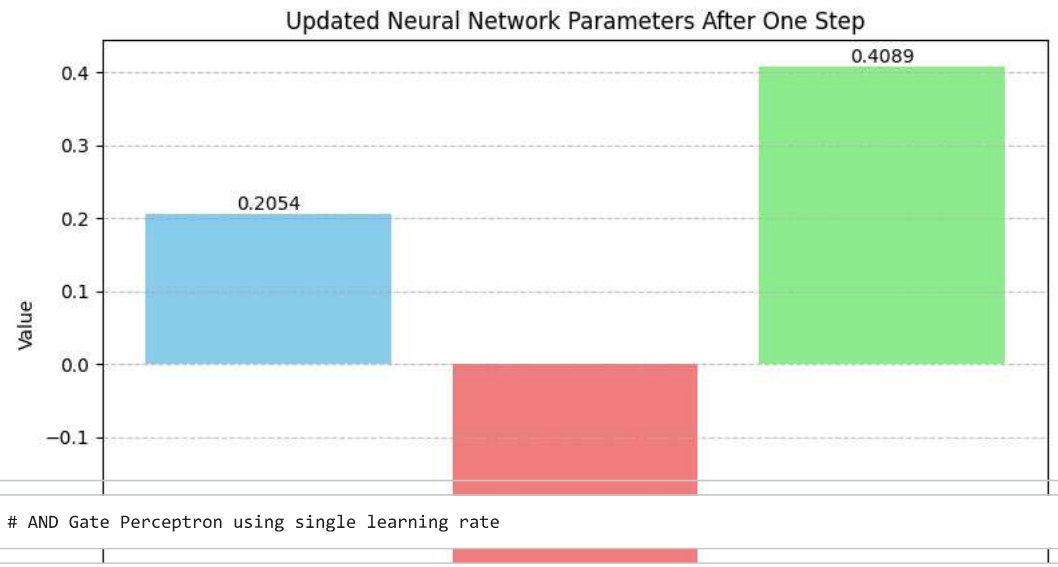
↳ 14 cells hidden

Lab - 3 | 05 February 2026

```
#  1) Updated Neural Network using gradient Descent and Backpropagation
```

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

x1, x2 = 0.6, 0.1
y = 1
w1, w2, b = 0.2, -0.3, 0.4
lr = 0.1

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

z = w1 * x1 + w2 * x2 + b
y_pred = sigmoid(z)

dL_dy_pred = y_pred - y

Dy_pred_dz = y_pred * (1 - y_pred)

dL_dz = dL_dy_pred * Dy_pred_dz

dL_dw1 = dL_dz * x1
dL_dw2 = dL_dz * x2
dL_db = dL_dz

w1 -= lr * dL_dw1
w2 -= lr * dL_dw2
b -= lr * dL_db

print(f"------------------------Update W1 is stated below:------------------------")
print(f"{w1:.4f}, w2: {w2:.4f}, b: {b:.4f}\n")
print(f"--------------------------------------------------------------------------------")

labels = ['Weight 1 (w1)', 'Weight 2 (w2)', 'Bias (b)']

values = [w1, w2, b]

plt.figure(figsize=(8, 5))
plt.bar(labels, values, color=['skyblue', 'lightcoral', 'lightgreen'])

plt.title('Updated Neural Network Parameters After One Step')
plt.ylabel('Value')
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, value in enumerate(values):
    plt.text(i, value, f'{value:.4f}', ha='center', va='bottom' if value >= 0 else 'top')

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

```
------------------------Update W1 is stated below:------------------------
0.2054, w2: -0.2991, b: 0.4089

--------------------------------------------------------------------------
```

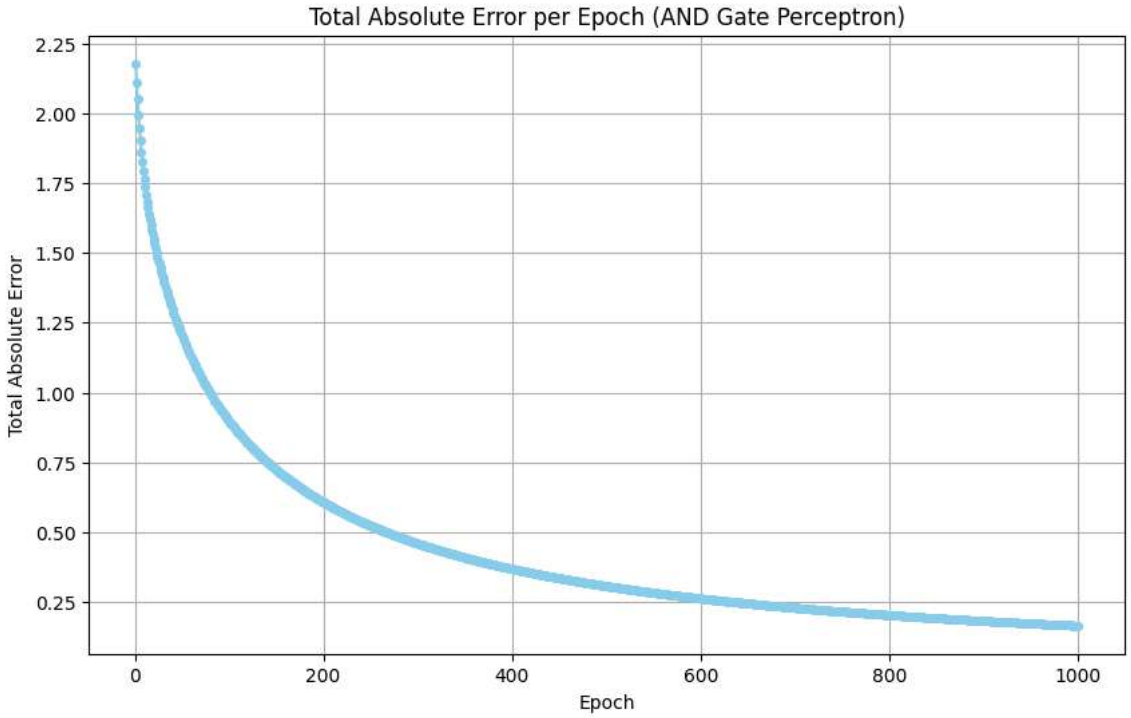



```
# AND Gate Perceptron using single learning rate
```

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

X = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])
y = np.array([0,0,0,1])

w = np.array([0.1, 0.2])
b = 0.3
lr = 0.1

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

errors_per_epoch = []

for epoch in range(1000):
    total_absolute_error = 0
    for i in range(len(X)):
        z = np.dot(X[i], w) + b
        y_pred = sigmoid(z)

        total_absolute_error += abs(y[i] - y_pred)

        dz = y_pred - y[i]

        dw = dz * X[i]
        db = dz

        w -= lr * dw
        b -= lr * db

    errors_per_epoch.append(total_absolute_error)

print("---------- Trained Perceptron Parameters ----------")
print(f"Final Weights: {np.round(w, 4)}")
print(f"Final Bias: {np.round(b, 4)}")
print("--------------------------------------------------")

epoch_numbers = range(1, len(errors_per_epoch) + 1)

plt.figure(figsize=(10, 6))
plt.plot(epoch_numbers, errors_per_epoch, marker='o', linestyle='-', color='skyblue', markersize=4)
plt.title('Total Absolute Error per Epoch (AND Gate Perceptron)')
plt.xlabel('Epoch')
plt.ylabel('Total Absolute Error')
```

```
plt.grid(True)
plt.show()
```

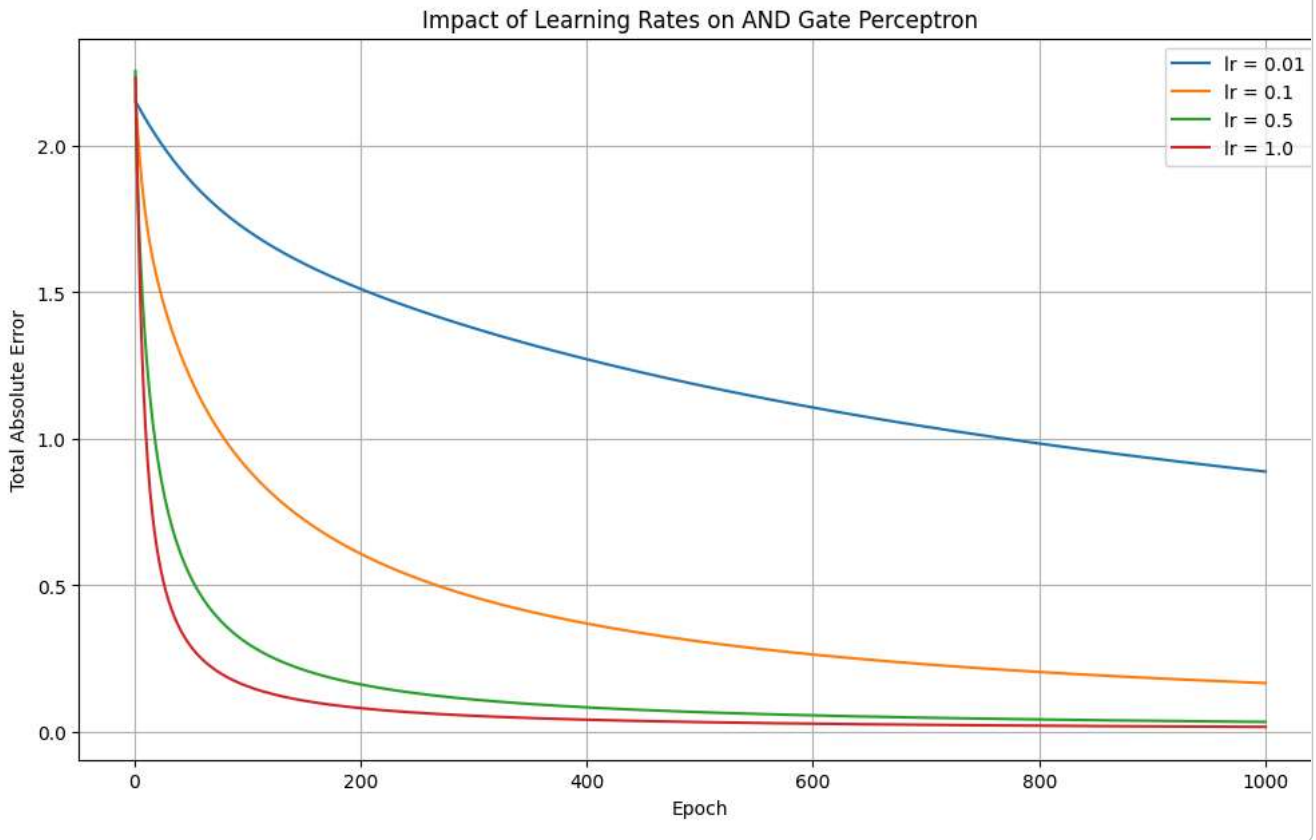
```
---------- Trained Perceptron Parameters ----------
Final Weights: [5.6015 5.5954]
Final Bias: -8.566
--------------------------------------------------
```




```
# Perceptron using sigmoid activation to learn the AND logic gate using gradient descent
```

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

# Dataset for AND Gate
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([0, 0, 0, 1])

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

# List of learning rates to test
learning_rates = [0.01, 0.1, 0.5, 1.0]
epochs = 1000

plt.figure(figsize=(12, 7))

for lr in learning_rates:
    # Reset weights and bias for each learning rate for a fair comparison
    w = np.array([0.1, 0.2])
    b = 0.3
    errors_per_epoch = []

    for epoch in range(epochs):
        total_absolute_error = 0
        for i in range(len(X)):
            z = np.dot(X[i], w) + b
            y_pred = sigmoid(z)

            error = y[i] - y_pred
            total_absolute_error += abs(error)

            # Gradient Descent
            dz = y_pred - y[i]
            dw = dz * X[i]
            db = dz

            w -= lr * dw
```

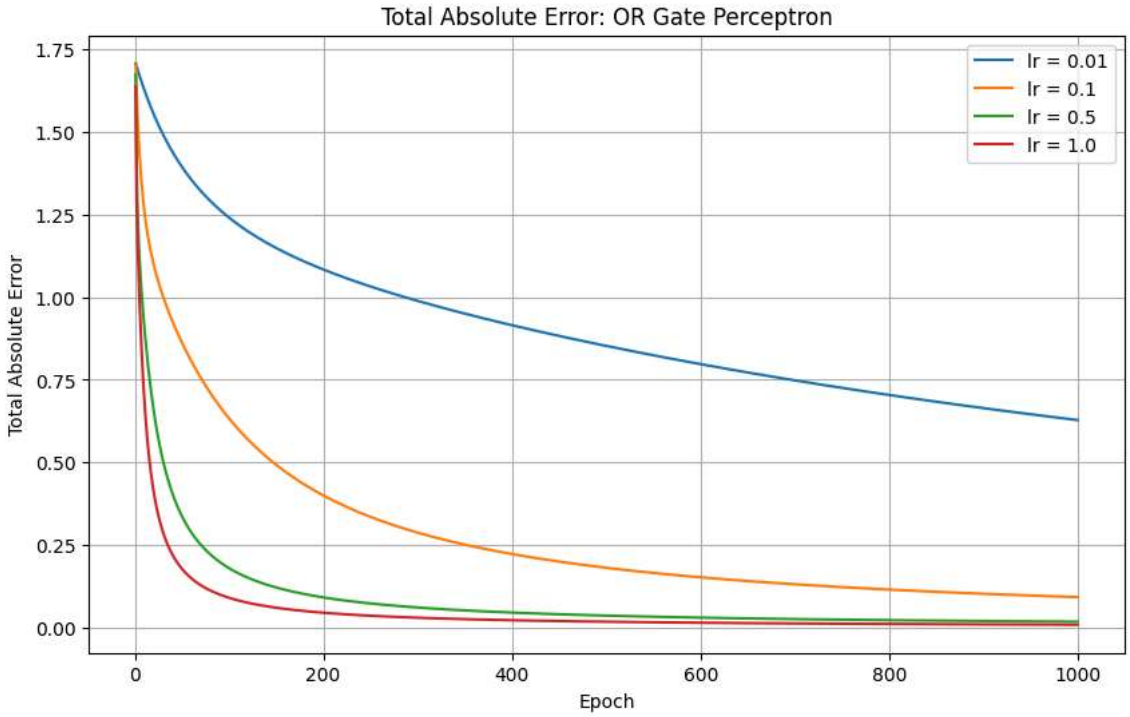
```
            b -= lr * db

        errors_per_epoch.append(total_absolute_error)

    # Plotting the results for this specific learning rate
    plt.plot(range(1, epochs + 1), errors_per_epoch, label=f'lr = {lr}')

plt.title('Impact of Learning Rates on AND Gate Perceptron')
plt.xlabel('Epoch')
plt.ylabel('Total Absolute Error')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Perceptron using sigmoid activation to learn the OR logic gate using gradient descent
```

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

# Dataset for OR Gate
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([0, 1, 1, 1]) # Updated for OR Gate logic

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

# Testing different learning rates
learning_rates = [0.01, 0.1, 0.5, 1.0]
epochs = 1000

plt.figure(figsize=(10, 6))

for lr in learning_rates:
    # Resetting weights/bias for each test
    w = np.array([0.1, 0.2])
    b = 0.3
    errors_per_epoch = []

    for epoch in range(epochs):
        total_absolute_error = 0
        for i in range(len(X)):
            # Forward pass
```

```
            z = np.dot(X[i], w) + b
            y_pred = sigmoid(z)

            # Tracking error
            total_absolute_error += abs(y[i] - y_pred)

            # Gradient Descent (Backpropagation)
            dz = y_pred - y[i]
            dw = dz * X[i]
            db = dz

            w -= lr * dw
            b -= lr * db

        errors_per_epoch.append(total_absolute_error)

    plt.plot(range(1, epochs + 1), errors_per_epoch, label=f'lr = {lr}')

plt.title('Total Absolute Error: OR Gate Perceptron')
plt.xlabel('Epoch')
plt.ylabel('Total Absolute Error')
plt.legend()
plt.grid(True)
plt.show()

# Print the final parameters for the last lr tested (1.0)
print(f"Final Weights for OR Gate: {np.round(w, 4)}")
print(f"Final Bias for OR Gate: {np.round(b, 4)}")
```




```
Final Weights for OR Gate: [11.5024 11.5064]
Final Bias for OR Gate: -5.2901
```