Double-click (or enter) to edit

```
import numpy as np

# Input and Output for AND gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs: (x1, x2)
Y = np.array([[0], [0], [0], [1]])             # Output: y

# Initialize weights and bias
w = np.array([0.0, 0.0])  # Two weights, one for each input
b = 0.0                   # Bias

# Step activation function
def step_activation(z):
    return 1 if z >= 0 else 0

# Hyperparameters
learning_rate = 1
epochs = 10

# Training loop
for epoch in range(epochs):
    total_loss = 0  # Track loss for logging

    for i in range(X.shape[0]):  # Loop through each training example
        x_i = X[i]  # Single input pair (e.g., [0, 1])
        y_i = Y[i]  # Corresponding target output (e.g., [0])

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = step_activation(z)  # Step activation function

        # Compute error
        error = y_i - y_pred  # Error: (target - predicted)
        total_loss += abs(error)  # Accumulate absolute error for logging

        # Backward pass (Weight and bias update)
        dw = learning_rate * error * x_i  # Gradient w.r.t weights
        db = learning_rate * error       # Gradient w.r.t bias

        # Update weights and bias
        w += dw
        b += db

    # Print loss for the epoch
    print(f"Epoch {epoch + 1}/{epochs}, Total Error: {total_loss}")

# Final weights and bias
print("Final weights:", w)
print("Final bias:", b)

# Test the neuron
print("\nTesting AND Gate:")
for i in range(X.shape[0]):
    x_i = X[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted Output: {y_pred}")
```

```
Epoch 1/10, Total Error: [2]
Epoch 2/10, Total Error: [2]
Epoch 3/10, Total Error: [1]
Epoch 4/10, Total Error: [0]
Epoch 5/10, Total Error: [0]
Epoch 6/10, Total Error: [0]
Epoch 7/10, Total Error: [0]
Epoch 8/10, Total Error: [0]
Epoch 9/10, Total Error: [0]
Epoch 10/10, Total Error: [0]
Final weights: [1. 1.]
Final bias: [-1.]

Testing AND Gate:
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 1
```

Input and Output for OR gate

```python
import numpy as np

# Input and Output for OR gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs: (x1, x2)
Y = np.array([[0], [1], [1], [1]])             # Output: y

# Initialize weights and bias
w = np.array([0.0, 0.0])  # Two weights, one for each input
b = 0.0                   # Bias

# Step activation function
def step_activation(z):
    return 1 if z >= 0 else 0

# Hyperparameters
learning_rate = 1
epochs = 10

# Training loop
for epoch in range(epochs):
    total_loss = 0  # Track loss for logging

    for i in range(X.shape[0]):  # Loop through each training example
        x_i = X[i]  # Single input pair (e.g., [0, 1])
        y_i = Y[i]  # Corresponding target output (e.g., [0])

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = step_activation(z)  # Step activation function

        # Compute error
        error = y_i - y_pred  # Error: (target - predicted)
        total_loss += abs(error)  # Accumulate absolute error for logging

        # Backward pass (Weight and bias update)
        dw = learning_rate * error * x_i  # Gradient w.r.t weights
        db = learning_rate * error       # Gradient w.r.t bias

        # Update weights and bias
        w += dw
        b += db

    # Print loss for the epoch
    print(f"Epoch {epoch + 1}/{epochs}, Total Error: {total_loss}")

# Final weights and bias
print("Final weights:", w)
print("Final bias:", b)

# Test the neuron
print("\nTesting AND Gate:")
for i in range(X.shape[0]):
    x_i = X[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted Output: {y_pred}")
```

```
Epoch 1/10, Total Error: [2]
Epoch 2/10, Total Error: [2]
Epoch 3/10, Total Error: [1]
Epoch 4/10, Total Error: [0]
Epoch 5/10, Total Error: [0]
Epoch 6/10, Total Error: [0]
Epoch 7/10, Total Error: [0]
Epoch 8/10, Total Error: [0]
Epoch 9/10, Total Error: [0]
Epoch 10/10, Total Error: [0]
Final weights: [1. 1.]
Final bias: [-1.]

Testing AND Gate:
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 1
```

Input and Output for NAND gate

```python
import numpy as np

# Input and Output for NAND gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs: (x1, x2)
Y = np.array([[1], [1], [1], [0]])             # Output: y
```

```
Y = np.array([[1], [1], [1], [0]])            # Output: y

# Initialize weights and bias
w = np.array([0.0, 0.0])  # Two weights, one for each input
b = 0.0                   # Bias

# Step activation function
def step_activation(z):
    return 1 if z >= 0 else 0

# Hyperparameters
learning_rate = 1
epochs = 10

# Training loop
for epoch in range(epochs):
    total_loss = 0  # Track loss for logging

    for i in range(X.shape[0]):  # Loop through each training example
        x_i = X[i]  # Single input pair (e.g., [0, 1])
        y_i = Y[i]  # Corresponding target output (e.g., [0])

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = step_activation(z)  # Step activation function

        # Compute error
        error = y_i - y_pred  # Error: (target - predicted)
        total_loss += abs(error)  # Accumulate absolute error for logging

        # Backward pass (Weight and bias update)
        dw = learning_rate * error * x_i  # Gradient w.r.t weights
        db = learning_rate * error       # Gradient w.r.t bias

        # Update weights and bias
        w += dw
        b += db

    # Print loss for the epoch
    print(f"Epoch {epoch + 1}/{epochs}, Total Error: {total_loss}")

# Final weights and bias
print("Final weights:", w)
print("Final bias:", b)

# Test the neuron
print("\nTesting AND Gate:")
for i in range(X.shape[0]):
    x_i = X[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted Output: {y_pred}")
```

```
Epoch 1/10, Total Error: [1]
Epoch 2/10, Total Error: [3]
Epoch 3/10, Total Error: [3]
Epoch 4/10, Total Error: [2]
Epoch 5/10, Total Error: [1]
Epoch 6/10, Total Error: [0]
Epoch 7/10, Total Error: [0]
Epoch 8/10, Total Error: [0]
Epoch 9/10, Total Error: [0]
Epoch 10/10, Total Error: [0]
Final weights: [-2. -1.]
Final bias: [2.]

Testing AND Gate:
Input: [0 0], Predicted Output: 1
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
```

Input and Output for NOR gate

```
import numpy as np

# Input and Output for NOR gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs: (x1, x2)
Y = np.array([[1], [0], [0], [0]])            # Output: y

# Initialize weights and bias
w = np.array([0.0, 0.0])  # Two weights, one for each input
b = 0.0                   # Bias
```

```
# Step activation function
def step_activation(z):
    return 1 if z >= 0 else 0

# Hyperparameters
learning_rate = 1
epochs = 10

# Training loop
for epoch in range(epochs):
    total_loss = 0  # Track loss for logging

    for i in range(X.shape[0]):  # Loop through each training example
        x_i = X[i]  # Single input pair (e.g., [0, 1])
        y_i = Y[i]  # Corresponding target output (e.g., [0])

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = step_activation(z)  # Step activation function

        # Compute error
        error = y_i - y_pred  # Error: (target - predicted)
        total_loss += abs(error)  # Accumulate absolute error for logging

        # Backward pass (Weight and bias update)
        dw = learning_rate * error * x_i  # Gradient w.r.t weights
        db = learning_rate * error       # Gradient w.r.t bias

        # Update weights and bias
        w += dw
        b += db

    # Print loss for the epoch
    print(f"Epoch {epoch + 1}/{epochs}, Total Error: {total_loss}")

# Final weights and bias
print("Final weights:", w)
print("Final bias:", b)

# Test the neuron
print("\nTesting AND Gate:")
for i in range(X.shape[0]):
    x_i = X[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted Output: {y_pred}")
```

```
Epoch 1/10, Total Error: [1]
Epoch 2/10, Total Error: [2]
Epoch 3/10, Total Error: [1]
Epoch 4/10, Total Error: [0]
Epoch 5/10, Total Error: [0]
Epoch 6/10, Total Error: [0]
Epoch 7/10, Total Error: [0]
Epoch 8/10, Total Error: [0]
Epoch 9/10, Total Error: [0]
Epoch 10/10, Total Error: [0]
Final weights: [-1. -1.]
Final bias: [0.]

Testing AND Gate:
Input: [0 0], Predicted Output: 1
Input: [0 1], Predicted Output: 0
Input: [1 0], Predicted Output: 0
Input: [1 1], Predicted Output: 0
```