```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and two features for simplicity
X = X[y != 2, :2]  # Use features 0 and 1, and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize weights and bias
w = np.zeros(2)  # Two weights for two features
b = 0.0          # Bias

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

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input pair (e.g., [feature1, feature2])
        y_i = y_train[i]  # Corresponding target output (0 or 1)

        # 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)

# Testing the model
print("\nTesting on test set:")
correct_predictions = 0
for i in range(X_test.shape[0]):
    x_i = X_test[i]
    y_i = y_test[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
    if y_pred == y_i:
        correct_predictions += 1

# Accuracy
accuracy = correct_predictions / X_test.shape[0]
```

```
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 7
Epoch 2/10, Total Error: 2
Epoch 3/10, Total Error: 0
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: [ 0.35797368 -0.16574933]
Final bias: 0.1

Testing on test set:
Input: [ 0.81130447 -0.80201291], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258], Predicted: 0, Actual: 0
Input: [-0.57291168  1.55055828], Predicted: 0, Actual: 0
Input: [-0.57291168  0.69507785], Predicted: 0, Actual: 0
Input: [-1.34192066  1.12281807], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.44362323], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818], Predicted: 0, Actual: 0
Input: [-0.57291168  0.90894796], Predicted: 0, Actual: 0
Input: [0.34989908 1.55055828], Predicted: 0, Actual: 0
Input: [-1.03431707  0.05346753], Predicted: 0, Actual: 0
Input: [ 0.96510626 -0.5881428 ], Predicted: 1, Actual: 1
Input: [0.04229549 2.40603872], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.01588301], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807], Predicted: 0, Actual: 0
Input: [ 2.04171882 -0.5881428 ], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258], Predicted: 0, Actual: 0
Input: [-0.1115063   0.69507785], Predicted: 0, Actual: 0
Accuracy: 100.00%
```

Two class classification with three parameters -

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and three features for simplicity
X = X[y != 2, :3]  # Use first 3 features, and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize weights and bias
w = np.zeros(3)  # Three weights for three features
b = 0.0          # Bias

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

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input pair (e.g., [feature1, feature2, feature3])
        y_i = y_train[i]  # Corresponding target output (0 or 1)

        # 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)

    # Testing the model
    print("\nTesting on test set:")
    correct_predictions = 0
    for i in range(X_test.shape[0]):
        x_i = X_test[i]
        y_i = y_test[i]
        z = np.dot(w, x_i) + b
        y_pred = step_activation(z)
        print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
        if y_pred == y_i:
            correct_predictions += 1

    # Accuracy
    accuracy = correct_predictions / X_test.shape[0]
    print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 4
Epoch 2/10, Total Error: 0
Epoch 3/10, Total Error: 0
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: [ 0.09228108 -0.1924831   0.2409292 ]
Final bias: 0.0

Testing on test set:
Input: [ 0.81130447 -0.80201291  1.5607252 ], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334  0.78124837], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764 1.34814061], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141], Predicted: 0, Actual: 0
Input: [-0.57291168  1.55055828 -0.70684376], Predicted: 0, Actual: 0
Input: [-0.57291168  0.69507785 -0.99028988], Predicted: 0, Actual: 0
Input: [-1.34192066  1.12281807 -1.34459753], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.44362323  0.63952531], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818 -0.99028988], Predicted: 0, Actual: 0
Input: [-0.57291168  0.90894796 -1.06115141], Predicted: 0, Actual: 0
Input: [ 0.34989908  1.55055828 -0.84856682], Predicted: 0, Actual: 0
Input: [-1.03431707  0.05346753 -0.91942835], Predicted: 0, Actual: 0
Input: [ 0.96510626 -0.5881428   1.27727908], Predicted: 1, Actual: 1
Input: [ 0.04229549  2.40603872 -1.06115141], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.01588301  1.06469449], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807 -1.06115141], Predicted: 0, Actual: 0
Input: [ 2.04171882 -0.5881428   1.34814061], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258  1.48986367], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141], Predicted: 0, Actual: 0
Input: [-0.1115063   0.69507785 -0.99028988], Predicted: 0, Actual: 0
Accuracy: 100.00%
```

Two class classification with four parameters -

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
```

```
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and all four features for simplicity
X = X[y != 2, :]  # Use all 4 features, and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1 (binary classification)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize weights and bias
w = np.zeros(4)  # Four weights for four features
b = 0.0          # Bias

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

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input pair (e.g., [feature1, feature2, feature3, feature4])
        y_i = y_train[i]  # Corresponding target output (0 or 1)

        # 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)

# Testing the model
print("\nTesting on test set:")
correct_predictions = 0
for i in range(X_test.shape[0]):
    x_i = X_test[i]
    y_i = y_test[i]
    z = np.dot(w, x_i) + b
    y_pred = step_activation(z)
    print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
    if y_pred == y_i:
        correct_predictions += 1

# Accuracy
accuracy = correct_predictions / X_test.shape[0]
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 2
Epoch 2/10, Total Error: 2
Epoch 3/10, Total Error: 0
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: [ 0.10766126 -0.14970908  0.21967074  0.27195297]
Final bias: 0.0

Testing on test set:
Input: [ 0.81130447 -0.80201291  1.5607252   1.44135075], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334  0.78124837  0.89744481], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764 1.34814061 1.80395471], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141 -0.91557501], Predicted: 0, Actual: 0
Input: [-0.57291168  1.55055828 -0.70684376 -0.73427302], Predicted: 0, Actual: 0
Input: [-0.57291168  0.69507785 -0.99028988 -1.09687699], Predicted: 0, Actual: 0
Input: [-1.34192066  1.12281807 -1.34459753 -1.09687699], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.44362323  0.63952531  0.53484084], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818 -0.99028988 -1.09687699], Predicted: 0, Actual: 0
Input: [-0.57291168  0.90894796 -1.06115141 -1.09687699], Predicted: 0, Actual: 0
Input: [ 0.34989908  1.55055828 -0.84856682 -0.91557501], Predicted: 0, Actual: 0
Input: [-1.03431707  0.05346753 -0.91942835 -1.09687699], Predicted: 0, Actual: 0
Input: [ 0.96510626 -0.5881428   1.27727908  0.71614283], Predicted: 1, Actual: 1
Input: [ 0.04229549  2.40603872 -1.06115141 -1.09687699], Predicted: 0, Actual: 0
Input: [ 0.04229549 -1.01588301  1.06469449  0.71614283], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807 -1.06115141 -1.09687699], Predicted: 0, Actual: 0
Input: [ 2.04171882 -0.5881428   1.34814061  1.07874679], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258  1.48986367  1.62265273], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141 -1.27817897], Predicted: 0, Actual: 0
Input: [-0.1115063   0.69507785 -0.99028988 -0.73427302], Predicted: 0, Actual: 0
Accuracy: 100.00%
```

Two class classification with two parameters using bipolar activation function -

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and two features for simplicity
X = X[y != 2, :2]  # Use only the first two features and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1 (binary classification)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert the target labels to bipolar form (0 -> -1, 1 -> 1)
y_train = np.where(y_train == 0, -1, 1)
y_test = np.where(y_test == 0, -1, 1)

# Initialize weights and bias
w = np.zeros(2)  # Two weights for two features
b = 0.0          # Bias

# Bipolar activation function (outputs 1 or -1)
def bipolar_activation(z):
    return 1 if z >= 0 else -1  # Outputs 1 if z >= 0, else -1

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input pair (e.g., [feature1, feature2])
        y_i = y_train[i]  # Corresponding target output (-1 or 1)

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = bipolar_activation(z)  # Bipolar 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)

# Testing the model
print("\nTesting on test set:")
correct_predictions = 0
for i in range(X_test.shape[0]):
    x_i = X_test[i]
    y_i = y_test[i]
    z = np.dot(w, x_i) + b
    y_pred = bipolar_activation(z)
    print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
    if y_pred == y_i:
        correct_predictions += 1

# Accuracy
accuracy = correct_predictions / X_test.shape[0]
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 14
Epoch 2/10, Total Error: 4
Epoch 3/10, Total Error: 0
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: [ 0.71594735 -0.33149867]
Final bias: 0.2

Testing on test set:
Input: [ 0.81130447 -0.80201291], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258], Predicted: -1, Actual: -1
Input: [-0.57291168  1.55055828], Predicted: -1, Actual: -1
Input: [-0.57291168  0.69507785], Predicted: -1, Actual: -1
Input: [-1.34192066  1.12281807], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.44362323], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818], Predicted: -1, Actual: -1
Input: [-0.57291168  0.90894796], Predicted: -1, Actual: -1
Input: [0.34989908 1.55055828], Predicted: -1, Actual: -1
Input: [-1.03431707  0.05346753], Predicted: -1, Actual: -1
Input: [ 0.96510626 -0.5881428 ], Predicted: 1, Actual: 1
Input: [0.04229549 2.40603872], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.01588301], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807], Predicted: -1, Actual: -1
Input: [ 2.04171882 -0.5881428 ], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258], Predicted: -1, Actual: -1
Input: [-0.1115063   0.69507785], Predicted: -1, Actual: -1
Accuracy: 100.00%
```

Two class classification with three parameters using bipolar activation function -

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and three features for simplicity
X = X[y != 2, :3]  # Use only the first three features and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1 (binary classification)
```

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert the target labels to bipolar form (0 -> -1, 1 -> 1)
y_train = np.where(y_train == 0, -1, 1)
y_test = np.where(y_test == 0, -1, 1)

# Initialize weights and bias
w = np.zeros(3)  # Three weights for three features
b = 0.0          # Bias

# Bipolar activation function (outputs 1 or -1)
def bipolar_activation(z):
    return 1 if z >= 0 else -1  # Outputs 1 if z >= 0, else -1

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input triplet (e.g., [feature1, feature2, feature3])
        y_i = y_train[i]  # Corresponding target output (-1 or 1)

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = bipolar_activation(z)  # Bipolar 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)

# Testing the model
print("\nTesting on test set:")
correct_predictions = 0
for i in range(X_test.shape[0]):
    x_i = X_test[i]
    y_i = y_test[i]
    z = np.dot(w, x_i) + b
    y_pred = bipolar_activation(z)
    print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
    if y_pred == y_i:
        correct_predictions += 1

# Accuracy
accuracy = correct_predictions / X_test.shape[0]
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 8
Epoch 2/10, Total Error: 0
Epoch 3/10, Total Error: 0
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: [ 0.18456215 -0.38496619  0.48185841]
Final bias: 0.0

Testing on test set:
Input: [ 0.81130447 -0.80201291  1.5607252 ], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334  0.78124837], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764 1.34814061], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141], Predicted: -1, Actual: -1
Input: [-0.57291168  1.55055828 -0.70684376], Predicted: -1, Actual: -1
Input: [-0.57291168  0.69507785 -0.99028988], Predicted: -1, Actual: -1
Input: [-1.34192066  1.12281807 -1.34459753], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.44362323  0.63952531], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818 -0.99028988], Predicted: -1, Actual: -1
Input: [-0.57291168  0.90894796 -1.06115141], Predicted: -1, Actual: -1
Input: [ 0.34989908  1.55055828 -0.84856682], Predicted: -1, Actual: -1
Input: [-1.03431707  0.05346753 -0.91942835], Predicted: -1, Actual: -1
Input: [ 0.96510626 -0.5881428   1.27727908], Predicted: 1, Actual: 1
Input: [ 0.04229549  2.40603872 -1.06115141], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.01588301  1.06469449], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807 -1.06115141], Predicted: -1, Actual: -1
Input: [ 2.04171882 -0.5881428   1.34814061], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258  1.48986367], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141], Predicted: -1, Actual: -1
Input: [-0.1115063   0.69507785 -0.99028988], Predicted: -1, Actual: -1
Accuracy: 100.00%
```

Two class classification with four parameters using bipolar activation function -

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 features)
y = iris.target  # Target classes (3 classes: 0, 1, 2)

# Select only two classes (linearly separable) and all four features for simplicity
X = X[y != 2, :]  # Use all 4 features, and exclude class 2
y = y[y != 2]      # Use only classes 0 and 1 (binary classification)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data for better gradient updates
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert the target labels to bipolar form (0 -> -1, 1 -> 1)
y_train = np.where(y_train == 0, -1, 1)
y_test = np.where(y_test == 0, -1, 1)

# Initialize weights and bias
w = np.zeros(4)  # Four weights for four features
b = 0.0          # Bias

# Bipolar activation function (outputs 1 or -1)
def bipolar_activation(z):
    return 1 if z >= 0 else -1  # Outputs 1 if z >= 0, else -1

# Hyperparameters
learning_rate = 0.1
epochs = 10

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

    for i in range(X_train.shape[0]):  # Loop through each training example
        x_i = X_train[i]  # Single input pair (e.g., [feature1, feature2, feature3, feature4])
        y_i = y_train[i]  # Corresponding target output (-1 or 1)

        # Forward pass
        z = np.dot(w, x_i) + b  # Linear combination
        y_pred = bipolar_activation(z)  # Bipolar 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)

    # Testing the model
    print("\nTesting on test set:")
    correct_predictions = 0
    for i in range(X_test.shape[0]):
        x_i = X_test[i]
        y_i = y_test[i]
        z = np.dot(w, x_i) + b
        y_pred = bipolar_activation(z)
        print(f"Input: {x_i}, Predicted: {y_pred}, Actual: {y_i}")
        if y_pred == y_i:
            correct_predictions += 1

    # Accuracy
    accuracy = correct_predictions / X_test.shape[0]
    print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/10, Total Error: 4
Epoch 2/10, Total Error: 4
Epoch 3/10, Total Error: 0
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: [ 0.21532251 -0.29941815  0.43934149  0.54390594]
Final bias: 0.0

Testing on test set:
Input: [ 0.81130447 -0.80201291  1.5607252   1.44135075], Predicted: 1, Actual: 1
Input: [ 0.04229549 -1.65749334  0.78124837  0.89744481], Predicted: 1, Actual: 1
Input: [0.65750267 0.26733764 1.34814061 1.80395471], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141 -0.91557501], Predicted: -1, Actual: -1
Input: [-0.57291168  1.55055828 -0.70684376 -0.73427302], Predicted: -1, Actual: -1
Input: [-0.57291168  0.69507785 -0.99028988 -1.09687699], Predicted: -1, Actual: -1
Input: [-1.34192066  1.12281807 -1.34459753 -1.09687699], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.44362323  0.63952531  0.53484084], Predicted: 1, Actual: 1
Input: [-0.1115063   1.33668818 -0.99028988 -1.09687699], Predicted: -1, Actual: -1
Input: [-0.57291168  0.90894796 -1.06115141 -1.09687699], Predicted: -1, Actual: -1
Input: [ 0.34989908  1.55055828 -0.84856682 -0.91557501], Predicted: -1, Actual: -1
Input: [-1.03431707  0.05346753 -0.91942835 -1.09687699], Predicted: -1, Actual: -1
Input: [ 0.96510626 -0.5881428   1.27727908  0.71614283], Predicted: 1, Actual: 1
Input: [ 0.04229549  2.40603872 -1.06115141 -1.09687699], Predicted: -1, Actual: -1
Input: [ 0.04229549 -1.01588301  1.06469449  0.71614283], Predicted: 1, Actual: 1
Input: [-0.72671348  1.12281807 -1.06115141 -1.09687699], Predicted: -1, Actual: -1
Input: [ 2.04171882 -0.5881428   1.34814061  1.07874679], Predicted: 1, Actual: 1
Input: [ 1.88791703 -0.16040258  1.48986367  1.62265273], Predicted: 1, Actual: 1
Input: [-1.03431707 -0.16040258 -1.06115141 -1.27817897], Predicted: -1, Actual: -1
Input: [-0.1115063   0.69507785 -0.99028988 -0.73427302], Predicted: -1, Actual: -1
Accuracy: 100.00%
```