2348045

Nishant Rodrigues

Deep Learning Lab 1

Perceptron

Perceptron is one of the simplest Artificial neural network architectures. It was introduced by Frank Rosenblatt in 1957s. It is the simplest type of feedforward neural network, consisting of a single layer of input nodes that are fully connected to a layer of output nodes. It can learn the linearly separable patterns. it uses slightly different types of artificial neurons known as threshold logic units (TLU). it was first introduced by McCulloch and Walter Pitts in the 1940s.

Types of Perceptron Single-Layer Perceptron: This type of perceptron is limited to learning linearly separable patterns. effective for tasks where the data can be divided into distinct categories through a straight line. Multilayer Perceptron: Multilayer perceptrons possess enhanced processing capabilities as they consist of two or more layers, adept at handling more complex patterns and relationships within the data.

Basic Components of Perceptron

A perceptron, the basic unit of a neural network, comprises essential components that collaborate in information processing.

Input Features: The perceptron takes multiple input features, each input feature represents a characteristic or attribute of the input data.

Weights: Each input feature is associated with a weight, determining the significance of each input feature in influencing the perceptron's output. During training, these weights are adjusted to learn the optimal values.

Summation Function: The perceptron calculates the weighted sum of its inputs using the summation function. The summation function combines the inputs with their respective weights to produce a weighted sum.

Activation Function: The weighted sum is then passed through an activation function. Perceptron uses Heaviside step function functions. which take the summed values as input and compare with the threshold and provide the output as 0 or 1.

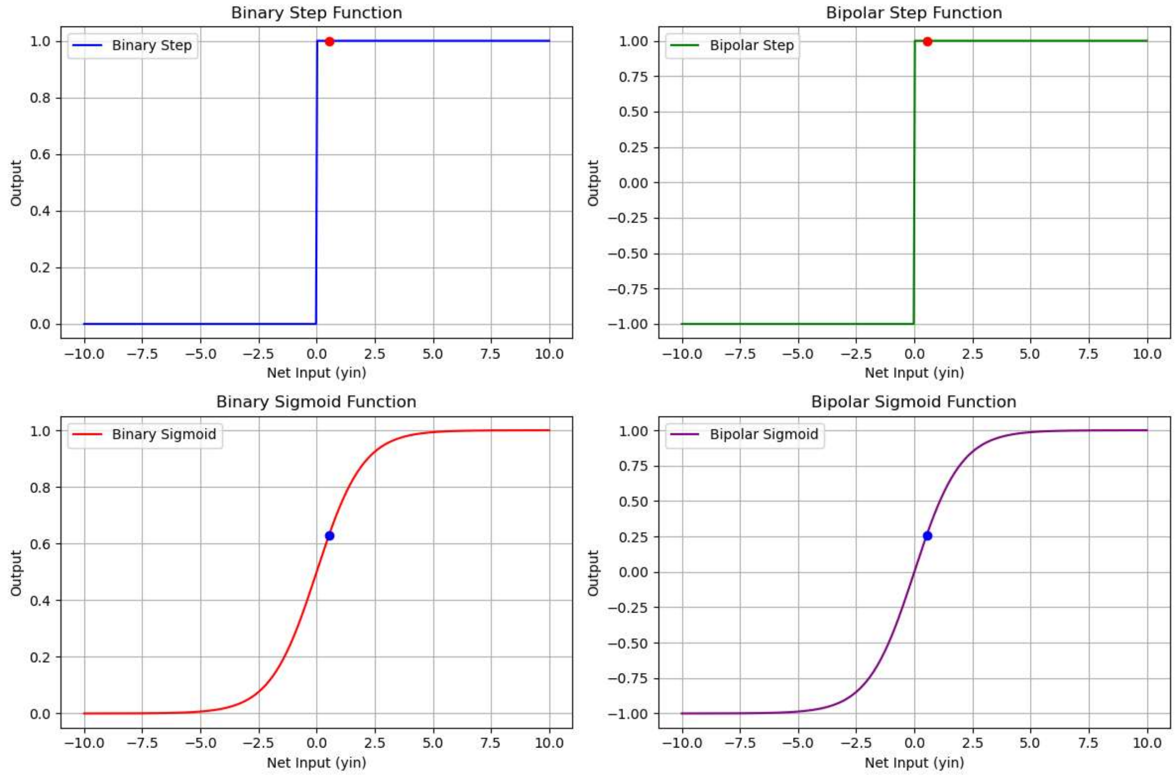
Output: The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).

Bias: A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.

Learning Algorithm (Weight Update Rule): During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Inputs and weights
x1 = 0.8
x2 = 0.6
x3 = 0.4
w1 = 0.1
w2 = 0.3
w3 = -0.2
b = 0.35

# Calculating net input
yin = b + (w1 * x1) + (w2 * x2) + (w3 * x3)

# Activation functions
def binary_step(yin, threshold=0):
    return 1 if yin >= threshold else 0

def bipolar_step(yin, threshold=0):
    return 1 if yin >= threshold else -1

def binary_sigmoid(yin):
    return 1 / (1 + np.exp(-yin))

def bipolar_sigmoid(yin):
    return (2 / (1 + np.exp(-yin))) - 1

# Generating a range of net input values for visualization
yin_values = np.linspace(-10, 10, 400)

# Computing activation function values
binary_step_values = [binary_step(y) for y in yin_values]
bipolar_step_values = [bipolar_step(y) for y in yin_values]
binary_sigmoid_values = binary_sigmoid(yin_values)
bipolar_sigmoid_values = bipolar_sigmoid(yin_values)

# Computing final activation function outputs for the calculated yin
binary_step_final = binary_step(yin)
bipolar_step_final = bipolar_step(yin)
binary_sigmoid_final = binary_sigmoid(yin)
bipolar_sigmoid_final = bipolar_sigmoid(yin)

# Plotting activation functions
plt.figure(figsize=(12, 8))

# Binary Step Function
plt.subplot(2, 2, 1)
plt.plot(yin_values, binary_step_values, label="Binary Step", color="blue")
plt.scatter([yin], [binary_step_final], color="red", zorder=5)
plt.title("Binary Step Function")
plt.xlabel("Net Input (yin)")
plt.ylabel("Output")
plt.grid(True)
plt.legend()

# Bipolar Step Function
```

```
plt.subplot(2, 2, 2)
plt.plot(yin_values, bipolar_step_values, label="Bipolar Step", color="green"
plt.scatter([yin], [bipolar_step_final], color="red", zorder=5)
plt.title("Bipolar Step Function")
plt.xlabel("Net Input (yin)")
plt.ylabel("Output")
plt.grid(True)
plt.legend()

# Binary Sigmoid Function
plt.subplot(2, 2, 3)
plt.plot(yin_values, binary_sigmoid_values, label="Binary Sigmoid", color="re
plt.scatter([yin], [binary_sigmoid_final], color="blue", zorder=5)
plt.title("Binary Sigmoid Function")
plt.xlabel("Net Input (yin)")
plt.ylabel("Output")
plt.grid(True)
plt.legend()

# Bipolar Sigmoid Function
plt.subplot(2, 2, 4)
plt.plot(yin_values, bipolar_sigmoid_values, label="Bipolar Sigmoid", color="
plt.scatter([yin], [bipolar_sigmoid_final], color="blue", zorder=5)
plt.title("Bipolar Sigmoid Function")
plt.xlabel("Net Input (yin)")
plt.ylabel("Output")
plt.grid(True)
plt.legend()

# Adjust layout and show plot
plt.tight_layout()
plt.show()

# Print results for the given yin
print("Net input:", yin)
print("Binary step:", binary_step_final)
print("Bipolar step:", bipolar_step_final)
print("Binary sigmoid:", binary_sigmoid_final)
print("Bipolar sigmoid:", bipolar_sigmoid_final)
```




```
Net input: 0.53
Binary step: 1
Bipolar step: 1
Binary sigmoid: 0.6294831119673949
Bipolar sigmoid: 0.25896622393478985
```

In [10]:
```
!pip install numpy scikit-learn
```

```
Requirement already satisfied: numpy in c:\users\user\anaconda3\lib\site-pac
kages (1.24.3)
Requirement already satisfied: scikit-learn in c:\users\user\anaconda3\lib\s
ite-packages (1.3.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\user\anaconda3\lib\s
ite-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\user\anaconda3\lib
\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anacond
a3\lib\site-packages (from scikit-learn) (2.2.0)
```

In [27]:

```
# Defining the input and target values
inputs = np.array([[1, 1], [1, -1], [-1, 1], [-1, -1]])
targets = np.array([1, -1, -1, -1])

# Initialising the weights and bias
w1 = 0
w2 = 0
b = 0

# Setting the learning rate
alpha = 1

# Defining the threshold
theta = 0

# Defining the activation function
def activation(net):
  if net >= theta:
    return 1
  else:
    return -1

# Training the perceptron
for i in range(len(inputs)):
  # Calculate the net input
  net = w1 * inputs[i][0] + w2 * inputs[i][1] + b

  # Calculating the output
  output = activation(net)

  # Updating the weights and bias
  w1 = w1 + alpha * (targets[i] - output) * inputs[i][0]
  w2 = w2 + alpha * (targets[i] - output) * inputs[i][1]
  b = b + alpha * (targets[i] - output)

# Printing the final weights and bias
print("Final weights and bias:")
print("w1:", w1)
print("w2:", w2)
print("b:", b)

# Testing the perceptron
for i in range(len(inputs)):
  # Calculate the net input
  net = w1 * inputs[i][0] + w2 * inputs[i][1] + b

  # Calculating the output
  output = activation(net)

  # Printing the input, target, and output
  print("Input:", inputs[i], "Target:", targets[i], "Output:", output)
```

```
Final weights and bias:
w1: 0
w2: 0
b: -4
Input: [1 1] Target: 1 Output: -1
Input: [ 1 -1] Target: -1 Output: -1
Input: [-1  1] Target: -1 Output: -1
Input: [-1 -1] Target: -1 Output: -1
```

In [37]:
```
!pip install mlxtend
```

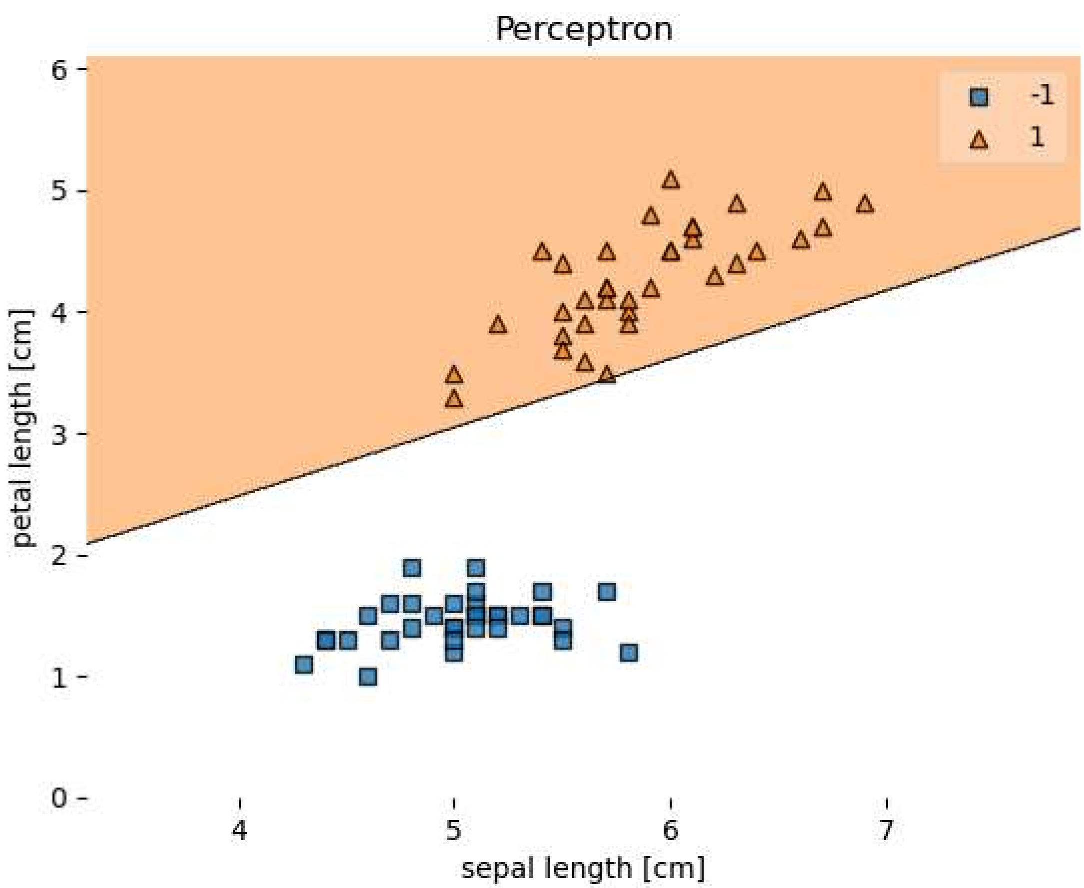
```
Collecting mlxtend
  Obtaining dependency information for mlxtend from https://files.pythonhost
ed.org/packages/1c/07/512f6a780239ad6ce06ce2aa7b4067583f5ddcfc7703a964a082c7
06a070/mlxtend-0.23.1-py3-none-any.whl.metadata (https://files.pythonhosted.
org/packages/1c/07/512f6a780239ad6ce06ce2aa7b4067583f5ddcfc7703a964a082c706a
070/mlxtend-0.23.1-py3-none-any.whl.metadata)
  Downloading mlxtend-0.23.1-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: scipy>=1.2.1 in c:\users\user\anaconda3\lib\s
ite-packages (from mlxtend) (1.11.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\user\anaconda3\lib
\site-packages (from mlxtend) (1.24.3)
Requirement already satisfied: pandas>=0.24.2 in c:\users\user\anaconda3\lib
\site-packages (from mlxtend) (2.0.3)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\user\anaconda
3\lib\site-packages (from mlxtend) (1.3.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\user\anaconda3
\lib\site-packages (from mlxtend) (3.7.2)
Requirement already satisfied: joblib>=0.13.2 in c:\users\user\anaconda3\lib
\site-packages (from mlxtend) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\user\anaconda3\l
ib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\s
ite-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\user\anaconda3
\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\user\anaconda3
\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\user\anaconda3\li
b\site-packages (from matplotlib>=3.0.0->mlxtend) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\user\anaconda3\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (10.0.1)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\user\anacon
da3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\user\anacond
a3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\user\anaconda3\lib\s
ite-packages (from pandas>=0.24.2->mlxtend) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\user\anaconda3\lib
\site-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anacond
a3\lib\site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\user\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Downloading mlxtend-0.23.1-py3-none-any.whl (1.4 MB)
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
    --------------------------------------- 0.0/1.4 MB 262.6 kB/s eta 0:00:0
6
   - -------------------------------------- 0.1/1.4 MB 363.1 kB/s eta 0:00:0
4
   --- ------------------------------------ 0.1/1.4 MB 655.8 kB/s eta 0:00:0
2
   --------- ------------------------------ 0.4/1.4 MB 1.4 MB/s eta 0:00:01
   -------------- ------------------------- 0.5/1.4 MB 1.8 MB/s eta 0:00:01
```

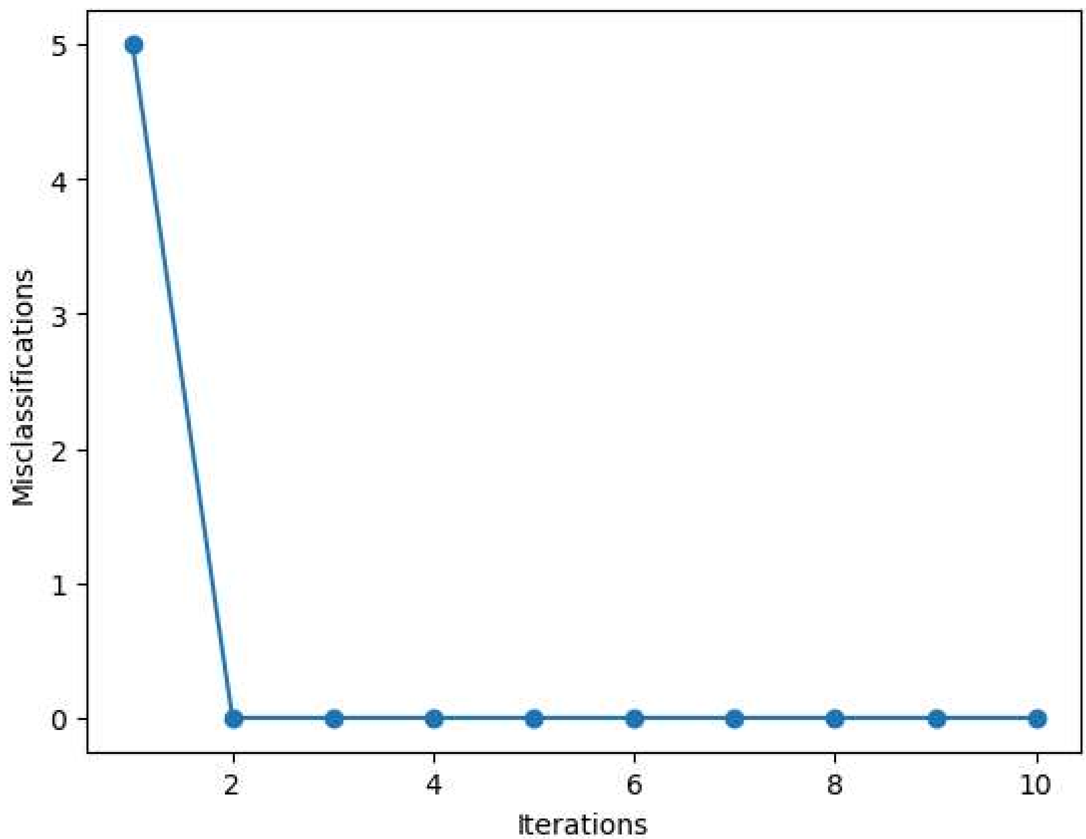
```
   -------------------- ----------------- 0.8/1.4 MB 2.2 MB/s eta 0:00:01
   ------------------------------ -------- 1.1/1.4 MB 2.9 MB/s eta 0:00:01
   ----------------------------------- --- 1.3/1.4 MB 3.1 MB/s eta 0:00:01
   ---------------------------------------- 1.4/1.4 MB 3.0 MB/s eta 0:00:00
Installing collected packages: mlxtend
Successfully installed mlxtend-0.23.1
```

In [4]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions

# Perceptron class definition
class Perceptron(object):
    def __init__(self, eta=0.01, epochs=50):
        self.eta = eta
        self.epochs = epochs

    def train(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.epochs):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)

# Loading the Iris dataset
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/i

# Selecting setosa and versicolor, set the target labels
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

# Selecting sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# Spliting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand

# Training the perceptron
ppn = Perceptron(epochs=10, eta=0.1)
ppn.train(X_train, y_train)

# Printing the weights
print('Weights: %s' % ppn.w_)

# Plot the decision regions
plot_decision_regions(X_train, y_train, clf=ppn)
plt.title('Perceptron')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.show()
```

```
# Ploting the misclassifications over epochs
plt.plot(range(1, len(ppn.errors_)+1), ppn.errors_, marker='o')
plt.xlabel('Iterations')
plt.ylabel('Misclassifications')
plt.show()

# Calculating accuracy on test set
y_pred = ppn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on test set: {accuracy * 100:.2f}%')
```

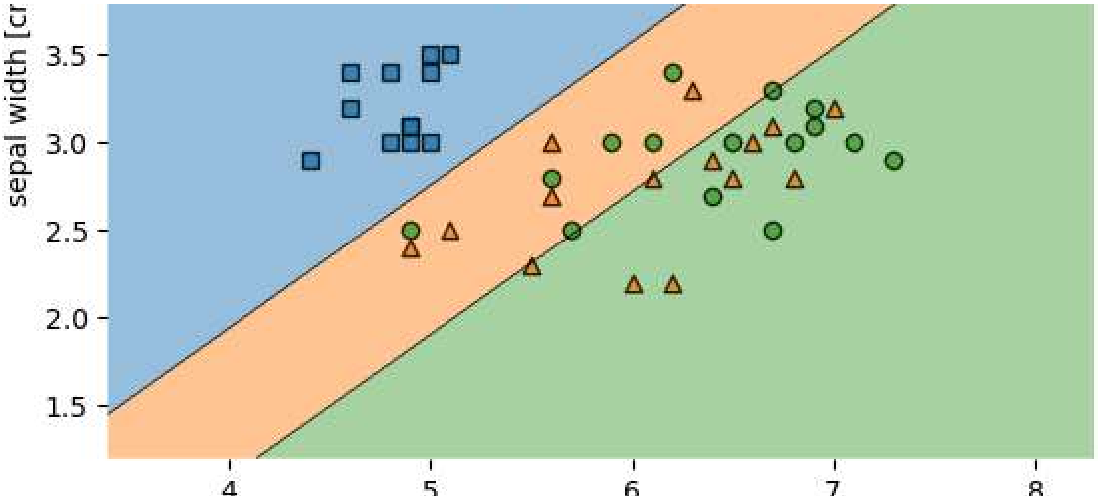
```
Weights: [-0.2  -0.52  0.92]
```



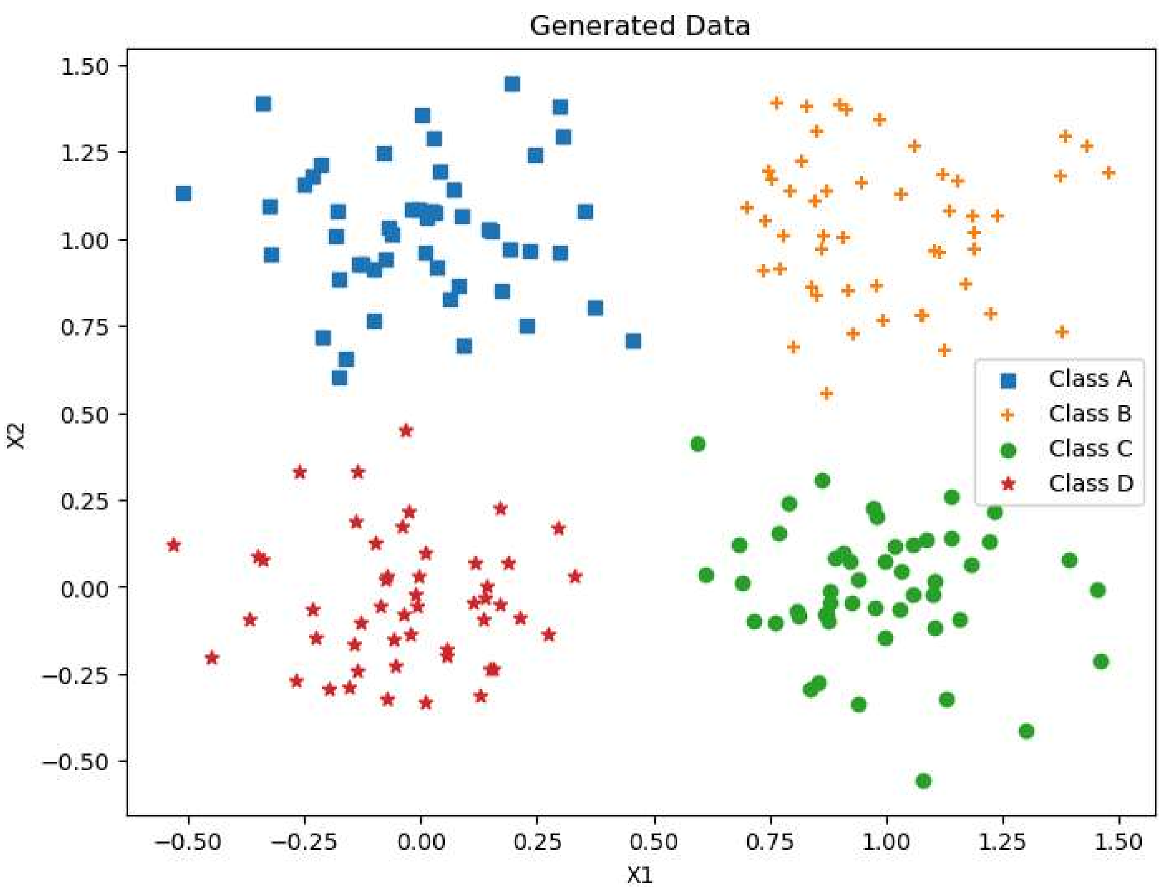



```
Accuracy on test set: 96.67%
```

In [10]:
```
from sklearn.neural_network import MLPClassifier


# Loading the Iris dataset
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/i
df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', '

# Encode\ing class labels to integers
y = df['class'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica'

# Selecting all four features
X = df.iloc[:, [0, 1, 2, 3]].values

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand

# Defining the multi-layer perceptron classifier
mlp = MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=1000, random_state=1)

# Trainong the classifier
mlp.fit(X_train, y_train)

# Predictong on the test set
y_pred = mlp.predict(X_test)

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on test set: {accuracy * 100:.2f}%')

# Plotting the decision regions
from mlxtend.plotting import plot_decision_regions

X_train_vis = X_train[:, [0, 1]]
X_test_vis = X_test[:, [0, 1]]
mlp_vis = MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=1000, random_stat
mlp_vis.fit(X_train_vis, y_train)

# Plot decision regions for training set
plot_decision_regions(X_train_vis, y_train, clf=mlp_vis, legend=2)
plt.title('Perceptron - Multi-class Classification (Training set)')
plt.xlabel('sepal length [cm]')
plt.ylabel('sepal width [cm]')
plt.show()

# Plot decision regions for test set
plot_decision_regions(X_test_vis, y_test, clf=mlp_vis, legend=2)
plt.title('Perceptron - Multi-class Classification (Test set)')
plt.xlabel('sepal length [cm]')
plt.ylabel('sepal width [cm]')
plt.show()
```

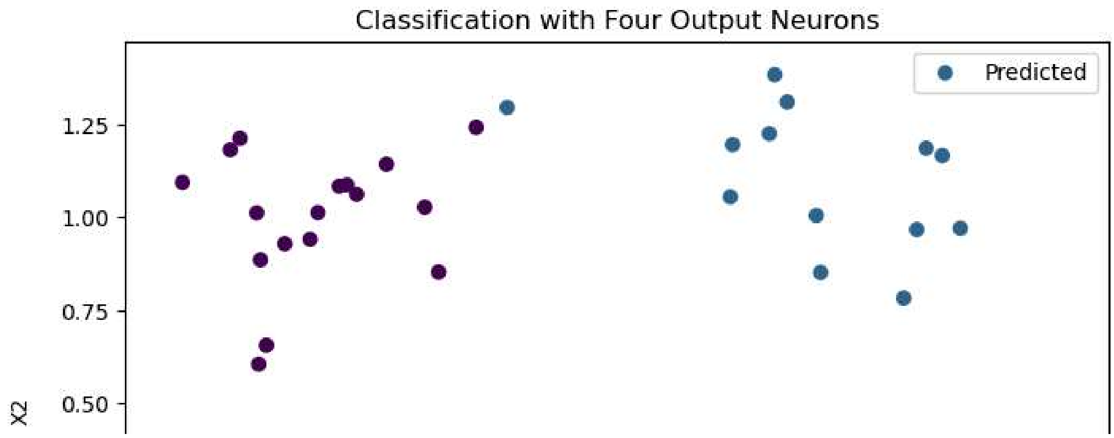

sepal width [cr
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generate data for 4 classes
np.random.seed(0)
n_samples = 50

class_A = np.random.randn(n_samples, 2) * 0.2 + [0.0, 1.0]
class_B = np.random.randn(n_samples, 2) * 0.2 + [1.0, 1.0]
class_C = np.random.randn(n_samples, 2) * 0.2 + [1.0, 0.0]
class_D = np.random.randn(n_samples, 2) * 0.2 + [0.0, 0.0]

X = np.vstack((class_A, class_B, class_C, class_D))
y = np.array([0]*n_samples + [1]*n_samples + [2]*n_samples + [3]*n_samples)

# Plot the generated data
plt.figure(figsize=(8, 6))
plt.scatter(class_A[:, 0], class_A[:, 1], marker='s', label='Class A')
plt.scatter(class_B[:, 0], class_B[:, 1], marker='+', label='Class B')
plt.scatter(class_C[:, 0], class_C[:, 1], marker='o', label='Class C')
plt.scatter(class_D[:, 0], class_D[:, 1], marker='*', label='Class D')
plt.legend()
plt.title('Generated Data')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



In [13]:
```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand

# Define the multi-layer perceptron classifier with four output neurons
mlp_four_outputs = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000, rand

# Train the classifier
mlp_four_outputs.fit(X_train, y_train)

# Predict on the test set
y_pred_four = mlp_four_outputs.predict(X_test)

# Calculate accuracy
accuracy_four = accuracy_score(y_test, y_pred_four)
print(f'Accuracy with four output neurons: {accuracy_four * 100:.2f}%')

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_four, cmap='viridis', marker
plt.title('Classification with Four Output Neurons')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()

# Combine classes into two groups: (A, B) and (C, D)
y_combined = np.array([0 if i in [0, 1] else 1 for i in y])

# Split the dataset into training and test sets
X_train, X_test, y_combined_train, y_combined_test = train_test_split(X, y_co

# Define the multi-layer perceptron classifier with two output neurons
mlp_two_outputs = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000, rando

# Train the classifier
mlp_two_outputs.fit(X_train, y_combined_train)

# Predict on the test set
y_combined_pred = mlp_two_outputs.predict(X_test)

# Calculate accuracy
accuracy_two = accuracy_score(y_combined_test, y_combined_pred)
print(f'Accuracy with two output neurons: {accuracy_two * 100:.2f}%')

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_combined_pred, cmap='coolwarm', m
plt.title('Classification with Two Output Neurons')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(

Accuracy with four output neurons: 85.00%
```



In [ ]: