Neural Network and Deep Learning

Introduction to Neural Network

Dr.Sherif Saad

Learning Objectives

- Introduce the students to neural network
- Introduce the students to modern neural network techniques.

Outlines

Introduction to Neural Networks and its History

Single-layer Neural Networks (Perceptrons)

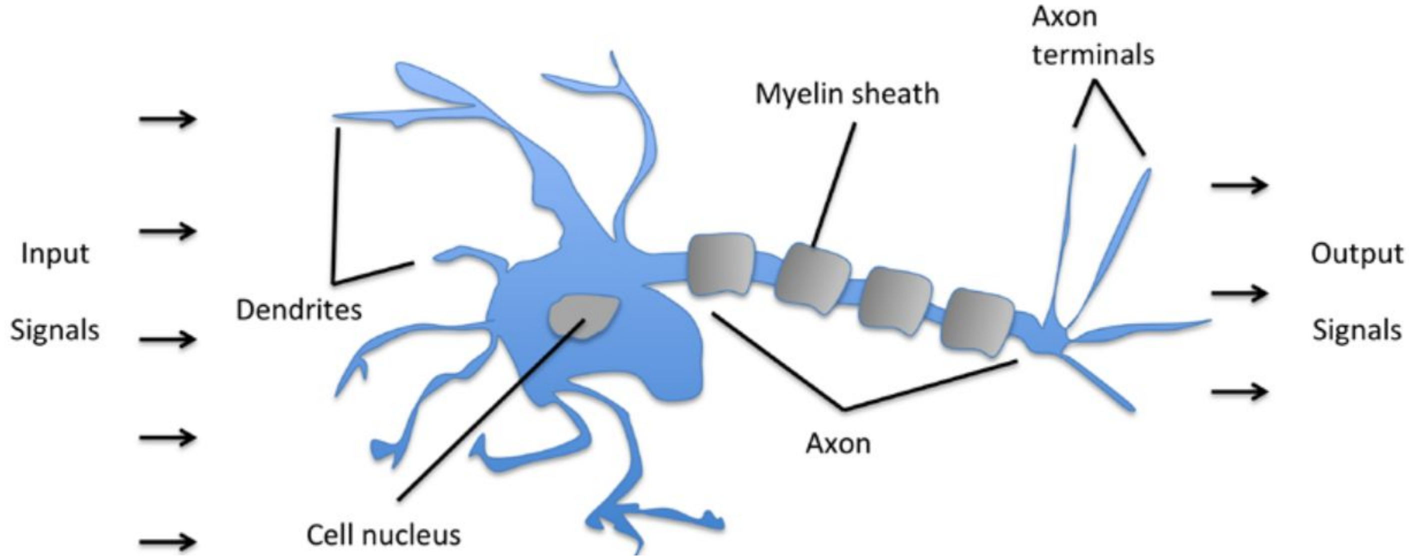
Perceptrons Algorithm

Perceptrons Implementation

Perceptrons with the Iris Flower Dataset

Artificial Neural Networks (ANN)

The basic building blocks of biological neural systems are nerve cells, referred to as neurons.



Artificial Neural Networks (ANN)

Neurons are interconnected nerve cells in the brain that are involved in the processing and transmitting of chemical and electrical signals.

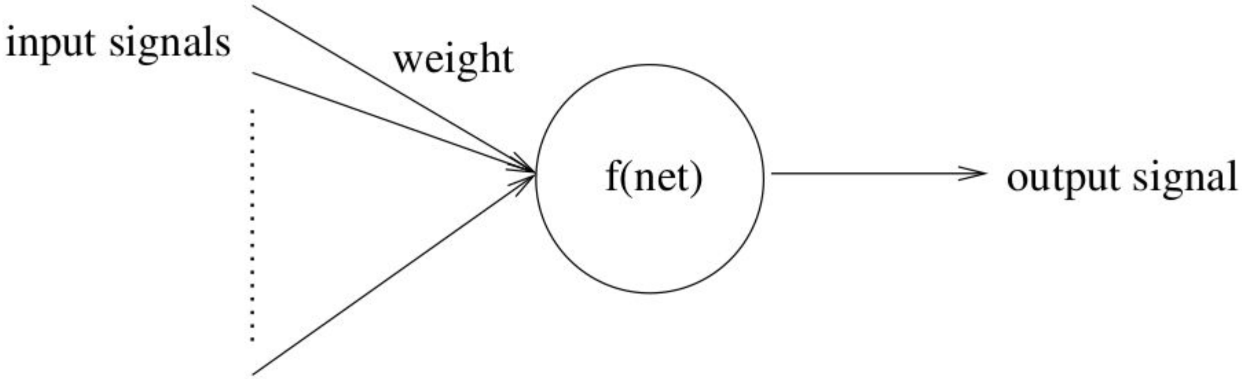
Signals propagate from the dendrites, through the cell body to the axon; from where the signals are propagated to all connected dendrites.

A signal is transmitted to the axon of a neuron only when the cell “fires”. A neuron can either inhibit or excite a signal.

Artificial Neural Networks (ANN)

An artificial neuron (AN) is a model of a biological neuron (BN).

Each AN receives signals from the environment, or other ANs, gathers these signals, and when fired, transmits a signal to all connected ANs.



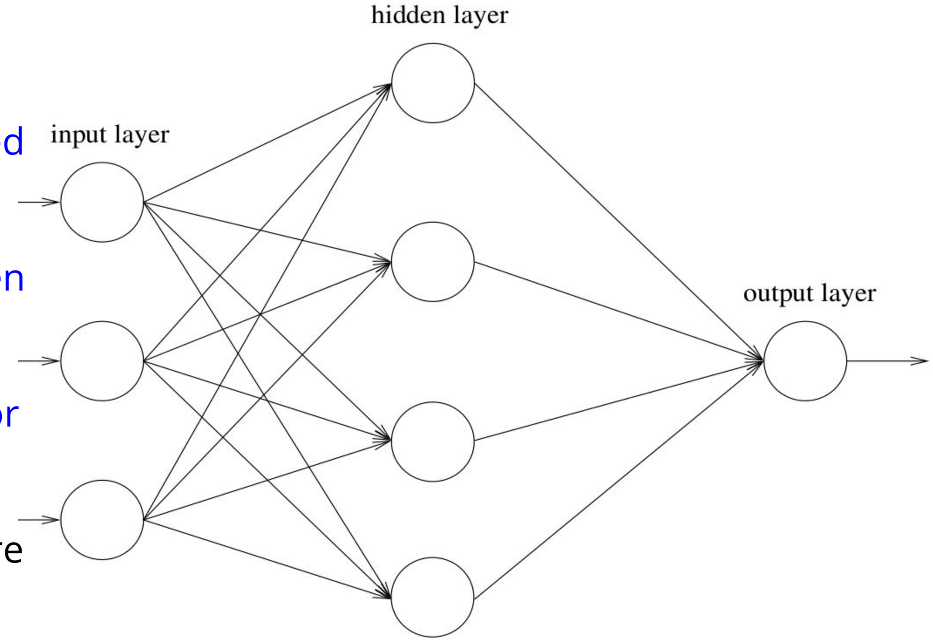
Artificial Neural Networks (ANN)

- Input signals are inhibited or excited through negative and positive numerical weights associated with each connection to the AN.
- The firing of an AN and the strength of the existing signal are controlled via a function, referred to as the activation function.
- The AN collects all incoming signals, and computes a net input signal as a function of the respective weights.
- The net input signal serves as input to the activation function which calculates the output signal of the AN.

Artificial Neural Networks Structure

What is an ANN?

- An artificial neural network (NN) is a layered network of ANs.
- An NN may consist of an input layer, hidden layers and an output layer.
- ANs in one layer are connected, fully or partially, to the ANs in the next layer.
- Feedback connections to previous layers are also possible.



Artificial Neural Networks (ANN)

Several different NN types have been developed:

- Single-layer NNs, such as the Hopfield network, Adaptive Linear Neuron, Perceptron
- Multilayer feedforward NNs (backpropagation NNs)
- Temporal NNs, such as the Elman and Jordan simple recurrent networks as well as time-delay neural networks.
- Self-Organizing NNs, such as the Kohonen self-organizing feature maps and the learning vector quantizer

Artificial Neural Networks (ANN)

Current successes in neural modeling are for small artificial NNs aimed at solving a specific task.

NN types have been used for a wide range of applications, including diagnosis of diseases, speech recognition, data mining, composing music, image processing, forecasting, robot control, credit approval, classification, pattern recognition.

Artificial Neural Networks (ANN)

In 1934, Warren McCullock and Walter Pitts published the first concept of a simplified brain cell. They called it MCP neuron

The MCP neuron is a simple logic gate with binary outputs; multiple signals arrive at the dendrites, are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

In 1957 Frank Rosenblatt published the first concept of the perceptron learning rule based on the MCP neuron model.

Single-layer Neural Networks (Perceptrons)

Rosenblatt, proposed an algorithm that would automatically learn the optimal weight coefficients that are then multiplied with the input features in order to make the decision of whether a neuron fires or not.

The perceptron algorithm represent a single layer neural network at at least has one MCP neuron.

We can use the perceptron in classification problems, mainly in binary classification. In this case we use the perceptron to predict if a sample belonged to one class or the other

Single-layer Neural Networks (Perceptrons)

- To model a binary classification problem using the perceptron algorithm we will use two classes: Positive Class = 1 and Negative Class = -1
- We will use a simple activation functions ***f(z),*** where z is the so-called net input ($z = w_1 x_1 + \dots + w_m x_m$)

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Single-layer Neural Networks (Perceptrons)

- If the activation of a particular sample x , that is, the output of $f(z)$, is greater than a predefined threshold θ , we predict class 1 and class -1,
- In other words the activation function is a simple piecewise unit step function

$$f(z) = \begin{cases} 1 & z \geqslant \theta \\ -1 & z < \theta \end{cases}$$

Single-layer Neural Networks (Perceptrons)

$$f(z) = \begin{cases} +1 & x_1w_1 + x_2w_2 + \ldots + x_mw_m \geqslant \theta \\ -1 & x_1w_1 + x_2w_2 + \ldots + x_mw_m \leqslant \theta \end{cases}$$

We can move threshold ***θ*** to the left side and rewrite z as

$$z = x_0w_0 + x_1w_1 + x_2w_2 + \ldots + x_mw_m$$

In this case $\boldsymbol{x_0 = 1}$ and $\boldsymbol{w_0 = \theta}$ and then ***f(z)*** is rewritten as follows

$$f(z) = \begin{cases} 1 & z \geqslant 0 \\ -1 & z < 0 \end{cases}$$

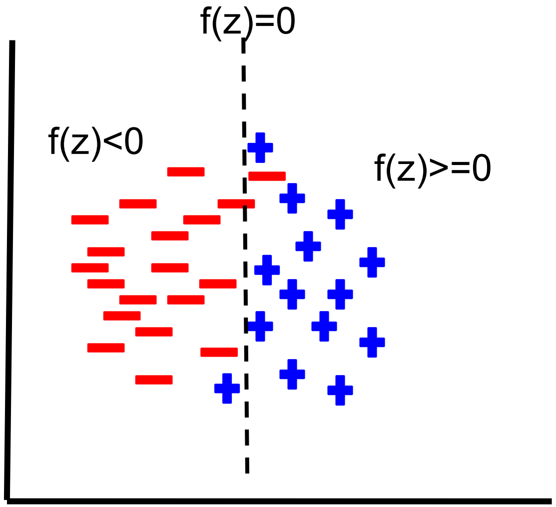
Single-layer Neural Networks (Perceptrons)

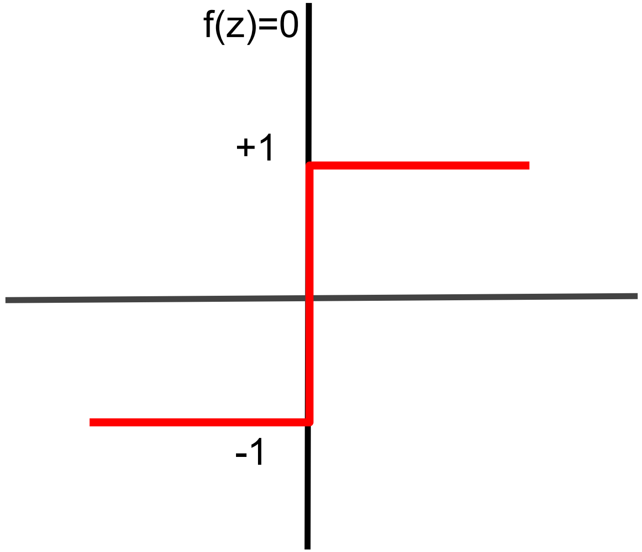
Then finally we can think of z as

$$z = x_0 w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_m w_m = \sum_{i=0}^{m} x_i w_i = w^T x$$

$$\underbrace{\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}}_{w} \times \underbrace{\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}}_{x} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32 .$$

Single-layer Neural Networks (Perceptrons)



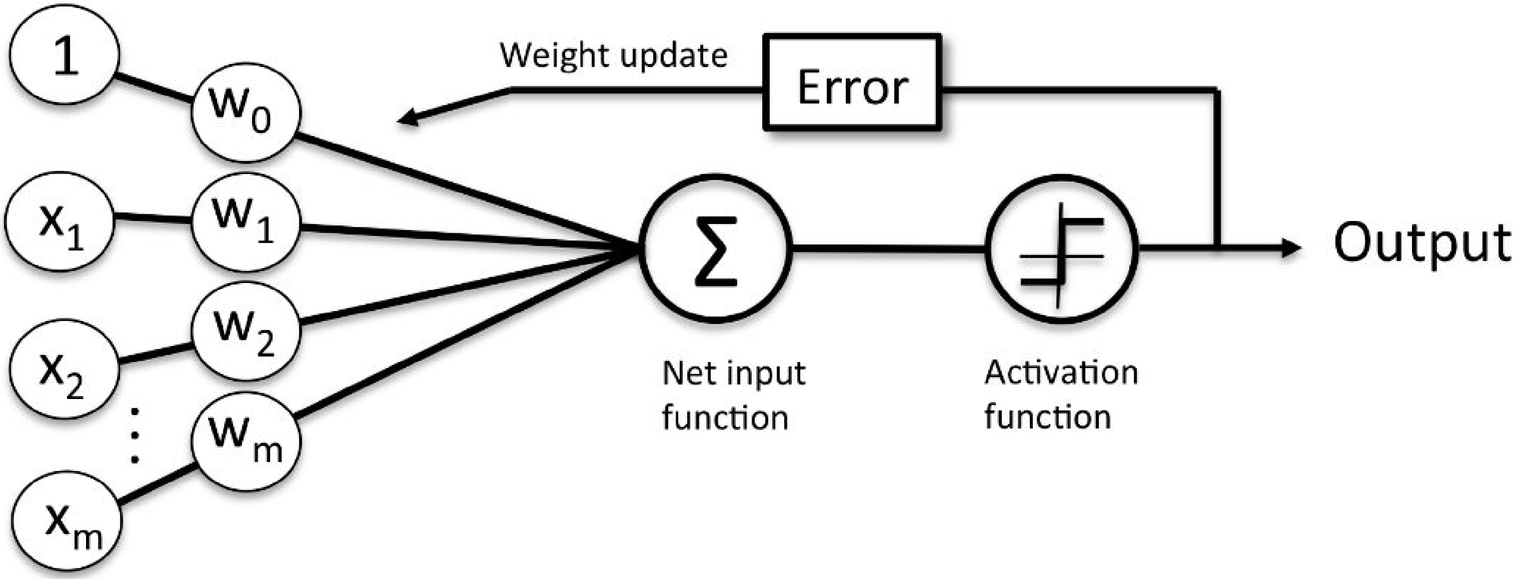


Single-layer Neural Networks (Perceptrons)

Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:

1. Initialize the weights to 0 or small random numbers
2. For each training sample x perform the following steps:
    a. Compute the output value y
    b. Update the weights based on the error in y

Single-layer Neural Networks (Perceptrons)



Single-layer Neural Networks (Perceptrons)

To update the weights

$$new(w_i) = wi + \Delta w_i$$
$$\Delta w_i = \alpha(l - p)x_i$$

Where w_i is the weight we want to update, alpha is the learning rate (preselected value) that controls how the weights are updated. L is the expected label and p is the predicate label and x_i is the feature associated with w_i

Single-layer Neural Networks (Perceptrons)

What happen to the weight if the predicate class is the same as the actual class?

$$\Delta w_i = \alpha((-1) - (-1))x_i$$

$$\Delta w_i = \alpha(1 - 1)x_i$$

Single-layer Neural Networks (Perceptrons)

However, in case we have a wrong prediction

$$\Delta w_i = \alpha(1 - (-1))x_i = \alpha(2)x_i$$

$$\Delta w_i = \alpha(-1 - 1))x_i = \alpha(-2)x_i$$

Single-layer Neural Networks (Perceptrons)

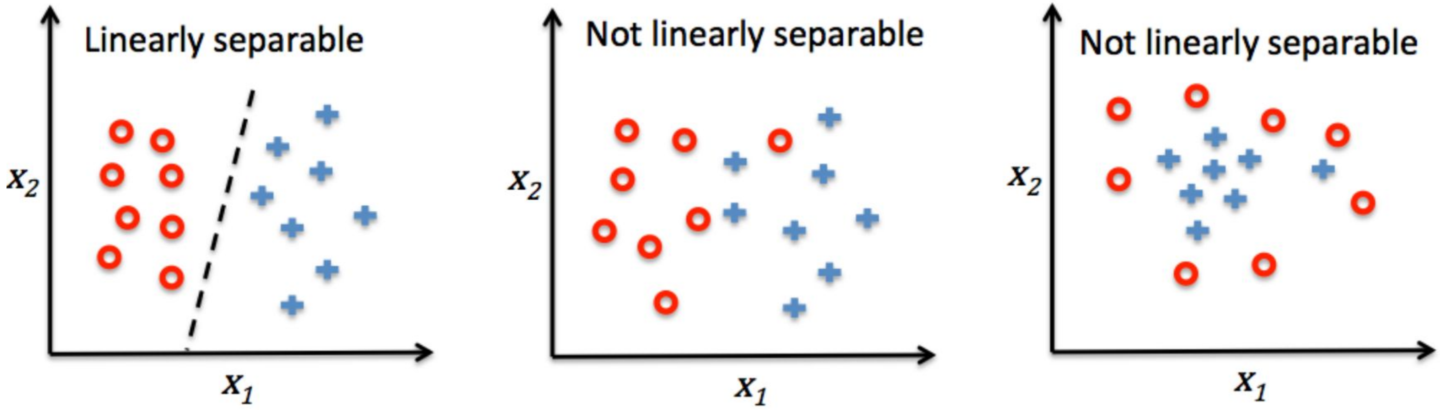
Example: let us assume L = +1 and p = -1 and learning rate alpha = 1 and the xi = 0.5 and wi = 0.2 Calculate the new wi

$$new(w_i) = 0.2 + 1(1 - -1))0.5 = 1.2$$

The weight update is proportional to the value of xi. For example let us assume that xi =2

$$new(w_i) = 0.2 + 1(1 - -1))2 = 4.2$$

Single-layer Neural Networks (Perceptrons)



The convergence of the perceptron is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small. If the two classes can't be separated by a linear decision boundary, we can set a maximum number of passes over the training dataset.

Implementing Perceptron Algorithm

One Class with Three methods:

fit(X,y): takes the training data and the labels, and return the perceptron classifier.

predict(x): takes a new sample and return the sample label

net_input(x): takes a sample and calculate the net input or **z**

```
import numpy as np

class Perceptron(object):

    def __init__(self, la=0.01, n_iter=10):

    def fit(self, X, y):

    def net_input(self, x):

    def predict(self, x):
```

Implementing Perceptron Algorithm

Initialize the perceptron object

```
class Perceptron(object):

    def __init__(self, la=0.01, n_iter=10):

        self.learning_rate = la

        self.num_of_iterations = n_iter

        self.weights = None

        self.errors = None
```

Implementing Perceptron Algorithm

Calculate the step function z

$$z = x_0 w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_m w_m = \sum_{i=0}^{m} x_i w_i = w^T x$$

```
    def net_input(self, x):
        """Calculate net input"""
        return np.dot(x, self.weights[1:]) + self.weights[0]
```

Implementing Perceptron Algorithm

Calculate | Predict the class label for

$$f(z) = \begin{cases} 1 & z \geqslant 0 \\ -1 & z < 0 \end{cases}$$

```python
    def predict(self, x):
        """Return class label after unit step"""
        return np.where(self.net_input(x) >= 0.0, 1, -1)

```

Implementing Perceptron Algorithm

```python
    def fit(self, X, y):
        # initialize the weights vector by zeros.
        # The size of the weights vector depend on the number of features
        self.weights = np.zeros(1 + X.shape[1])

        self.errors = []

        # start the iterative process to learn the best set of weights
        for _ in range(self.num_of_iterations):
            errors = 0

            for xi, target in zip(X, y):
                update = self.learning_rate * (target - self.predict(xi))

                self.weights[1:] += update * xi

                self.weights[0] += update

                errors += int(update != 0.0)

            self.errors.append(errors)

        return self
```

$$new(w_i) = wi + \Delta w_i$$

$$\Delta w_i = \alpha(l - p)x_i$$

Using Perceptron with the Iris Flower Dataset

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

df = pd.read_csv('https://archive.ics.uci.edu'
    +'/ml/machine-learning-databases/iris/iris.data', header=None)

    # we extract the first 100 class labels that correspond to
    # the 50 Iris-Setosa and 50 Iris-Versicolor flowers
    y = df.iloc[0:100, 4].values

    y = np.where(y == 'Iris-setosa', -1, 1)

    X = df.iloc[0:100, [0, 2]].values

    plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
    plt.scatter(X[50:100, 0], X[50:100, 1],  color='blue', marker='x', label='versicolor')

    plt.xlabel('sepal length')
    plt.ylabel('petal length')

    plt.legend(loc='upper left')
    plt.show()
```

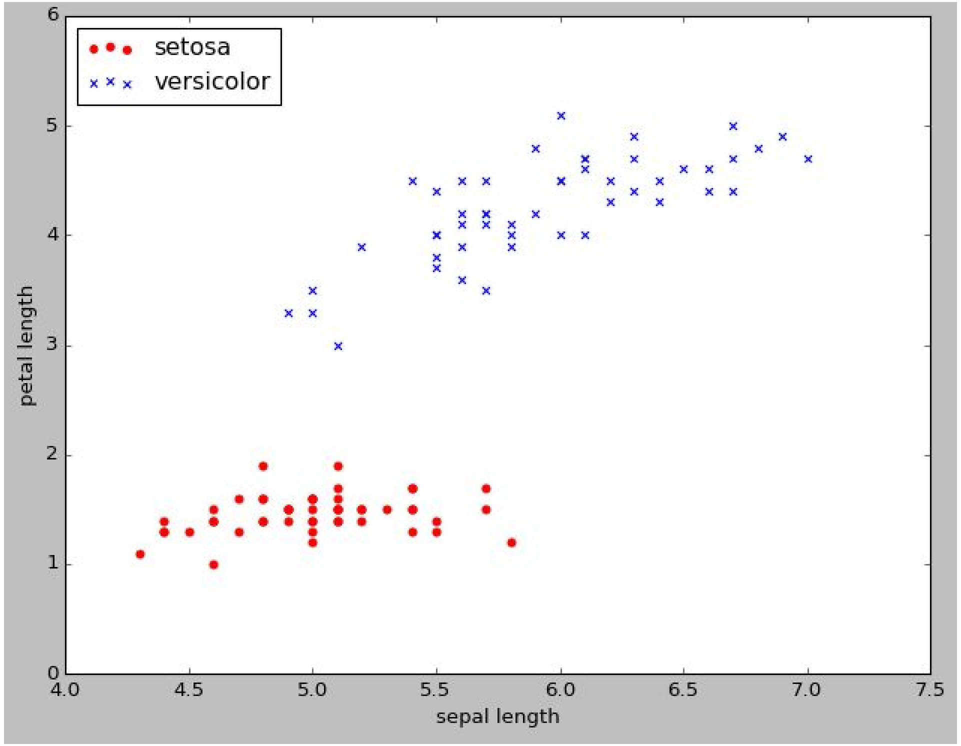
Using Perceptron with the Iris Flower Dataset

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

df = pd.read_csv('https://archive.ics.uci.edu'
     +'/ml/machine-learning-databases/iris/iris.data', header=None)

     # we extract the first 100 class labels that correspond to
     # the 50 Iris-Setosa and 50 Iris-Versicolor flowers
     y = df.iloc[0:100, 4].values

     y = np.where(y == 'Iris-setosa', -1, 1)

     X = df.iloc[0:100, [0, 2]].values
```

Using Perceptron with the Iris Flower Dataset

```
    plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
    plt.scatter(X[50:100, 0], X[50:100, 1],  color='blue', marker='x', label='versicolor')

    plt.xlabel('sepal length')
    plt.ylabel('petal length')

    plt.legend(loc='upper left')
    plt.show()
```

Using Perceptron with the Iris Flower Dataset

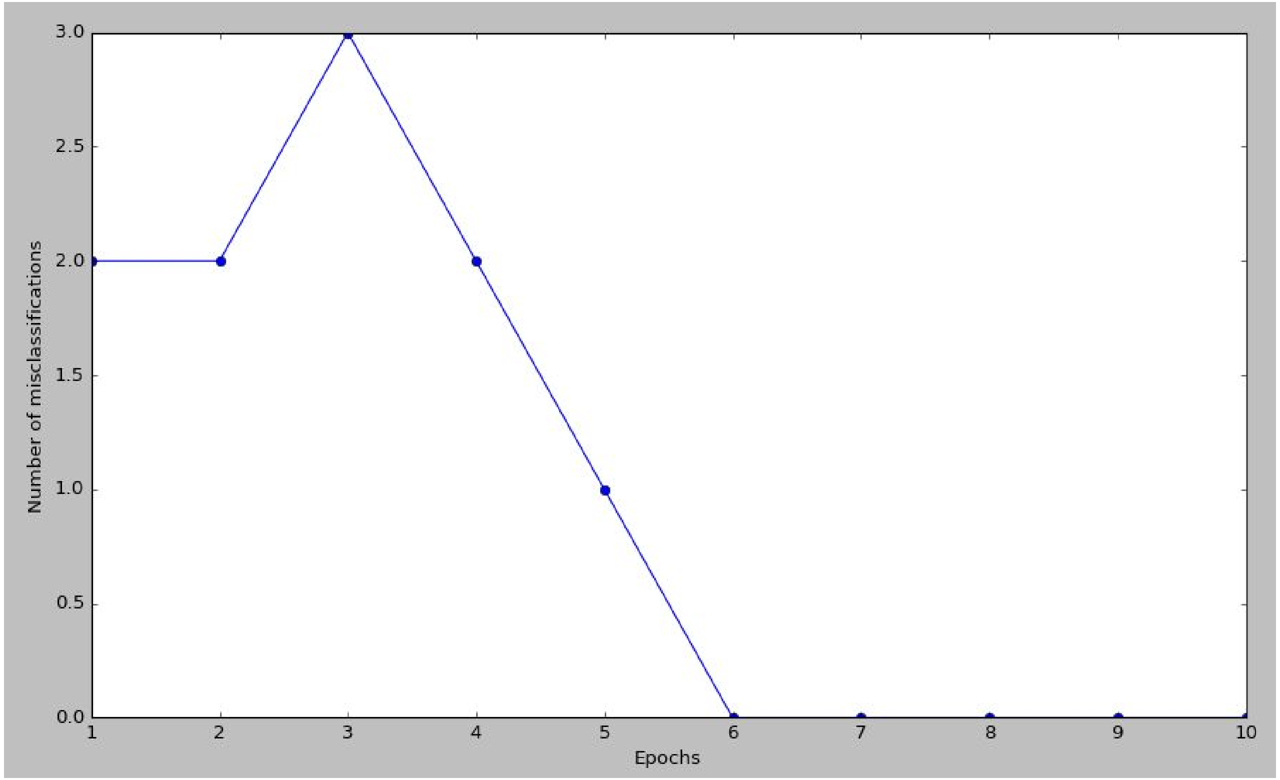


Using Perceptron with the Iris Flower Dataset

Create a perceptron instance and use the fit method to learn the perceptron rule

```
ppn = Perceptron(la=0.1, n_iter=10)

ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors) + 1), ppn.errors, marker='o')

plt.xlabel('Epochs')

plt.ylabel('Number of misclassifications')

plt.show()
```

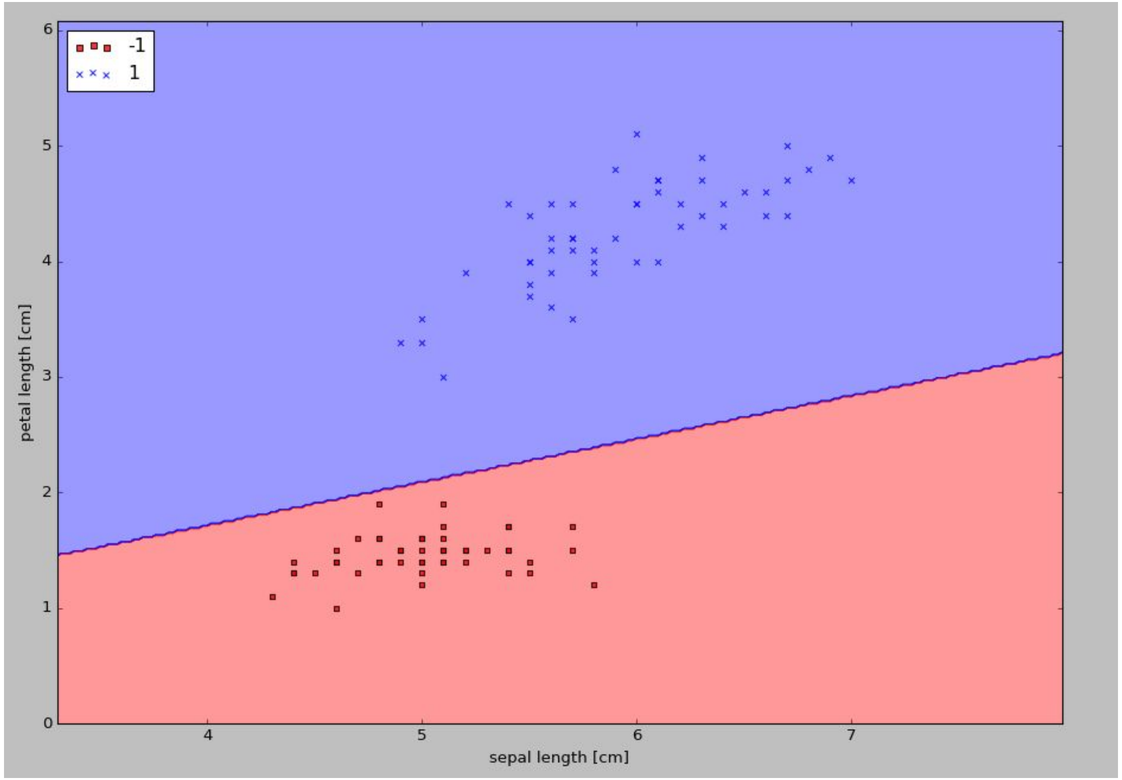
Using Perceptron with the Iris Flower Dataset



Using Perceptron with the Iris Flower Dataset

```python

plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()

```

Using Perceptron with the Iris Flower Dataset



Using Perceptron with the Iris Flower Dataset

```python

def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))

    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

```

Questions