George Mason University

Learning From Data Fall 2024

Computer Exercise #7

Assigned: November 03, 2024
Due Date: November 14, 2024

This computer exercise is concerned with multilayer perceptrons. The due date for this is **Thursday, November 14** at midnight. No submissions after this time will be accepted.

Computer Exercise 7.1 (Multilayer Perceptrons for Classification):

Multilayer perceptrons are capable of realizing arbitrarily complex decision surfaces and functions. However, since the cost functions that are minimized are not convex, training the network with a steepest gradient descent algorithm may lead to a local minimum. Deep networks may contain tens of thousands of weights that are to be learned, which leads to difficulties in training and the potential for overfitting. In spite of these issues, MLPs and the special architecture of the convolutional neural network have made them extremely useful in many applications. Here, we explore some simple shallow networks in order to understand their properties.

1. Getting Started

The first step in this exercise is to load the following classes,

```
#Common imports
import matplotlib.pyplot as plt
import numpy as np
#Classes for designing MLPs
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
```

The first two are standard classes that have been used in previous exercises. The last two are the classes that will be used to design a MLP for classification and for regression. Others are given in the jupyter notebook that will be used to create scatter plots and plots of decision boundaries either in 2-d or 3-d.

2. The XOR Data Set

We have seen how to design a NN to implement the logical operations of `AND`, `OR`, and `XOR` for binary inputs. While the first two can be designed using a single perceptron, the exclusive OR function requires a single hidden layer of two neurons.

Here we consider a more difficult problem of designing an exclusive OR function where the data samples may lie anywhere in each of four quadrants. More specifically, the data set consists of samples that lie within the square $-2 \leq x_1, x_2 \leq 2$, and these samples have target values given by:

$$y = \begin{cases} 1 & ; \text{ if } x_1 \cdot x_2 < 0 \\ -1 & ; \text{ if } x_1 \cdot x_2 > 0 \end{cases}$$

The four quadrants and the labels are illustrated in Fiig. 1

Create the Data Set

Generate a data set of 200 random samples over the square $[-2, 2] \times [-2, 2]$ and label those points in quadrants 1 and 3 as $y = -1$ and those in quadrants 2 and 4 as $y = +1$. This data set may be generated easily as follows:

```
X = np.random.uniform(low=-2, high=2, size=(200,2))
y=(X[:,0]*X[:,1] < 0).astype(int)
```

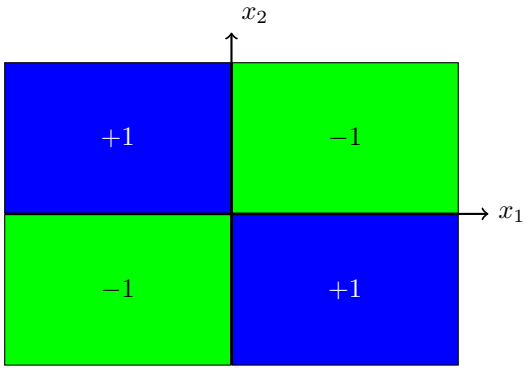
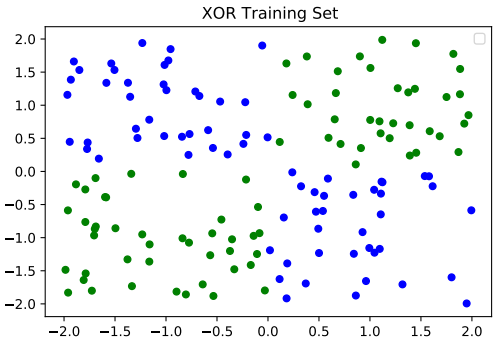


Figure 1: The feature space for the XOR classifier. Data samples lying in quadrants one and three have a target value of -1 and those in quadrants two and four have a target value of $+1$.

A scatter plot of the data set will look similar to that shown in the figure below.



In the following experiments, you are to design an MLP classifier for this data set using the `MLPClassifier` class. There are many parameters that may be set for this class including the activation function, the hidden layer sizes, and parameters associated with the stochastic gradient algorithm. An example is given below,

```
clf = MLPClassifier(solver='lbfgs',max_iter=20000,tol=10**(-16),hidden_layer_sizes=(5))
clf.fit(X_train, y_train)
```

Read the documentation on this class to become familiar with the parameters and how the network is trained?

Questions

(a) What is the default activation function for the hidden layers?

(b) What is the activation function that is used in the output layer?

Experiments

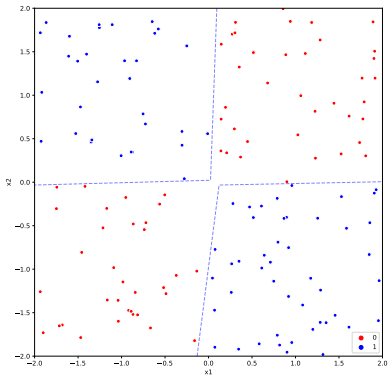
(a) Design a MLP with one hidden layer for the data set that you created. Begin with five neurons in the hidden layer. Remember that the network is learned using SGD, so different initializations of the algorithm may lead to different solutions. Run the classifier several times and note its performance. Comment on how many iterations are necessary for convergence and the accuracy of the classifier.

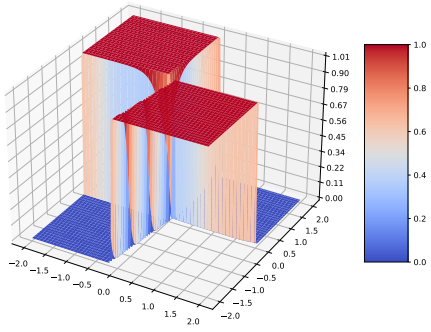
(b) Repeat part (a) using only four neurons. Are you able to train the network to get zero training error? If so, try three neurons. If not, is there a reason that you can give why it is impossible to design the classifier with only four neurons?

(c) What happens if you design a deeper network with two or three hidden layers?

(d) Does your design improve or get worse with other activation functions?

To help you with your experiments, you are given some code in your jupyter notebook to make plots of the decision surfaces in 2-d or 3-d similar to those shown below.





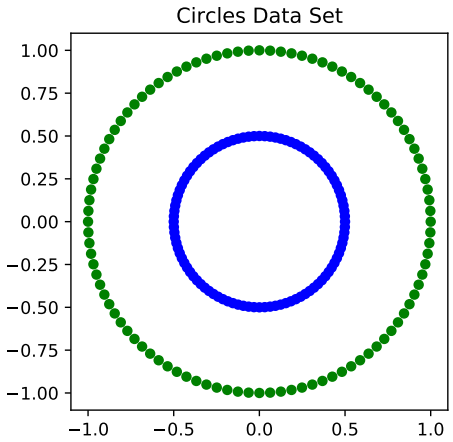


Figure 2: The circles data set.

3. The Circles Data Set

Next consider the data set generated using the `make_circles` class. Here the data samples are distributed around two concentric circles, with a separation set by the parameter `factor`, which must be number in the range $(0, 1)$. The outer circle has a radius of one, and the inner circle a radius equal to the value specified by `factor` (the default value is 0.8).

Create the Data Set

Generate a data set of 200 random samples using the `make_circles` class with `factor=0.5`,

```
X,y=make_circles(n_samples=200, shuffle=True, factor=0.5)
```

A scatter plot of the data set will look like that shown in Fig. 2.

Experiments

(a) Repeat the experiments you performed for the XOR data set.

(b) What is the minimum number of hidden layer neurons necessary to perfectly separate the data samples?

(c) Now create a new data set with `factor=0.8` and repeat your experiments. How many hidden neurons are necessary in a single hidden layer MLP in this case?

(d) Next try setting `factor=0.9` and repeat your experiments.