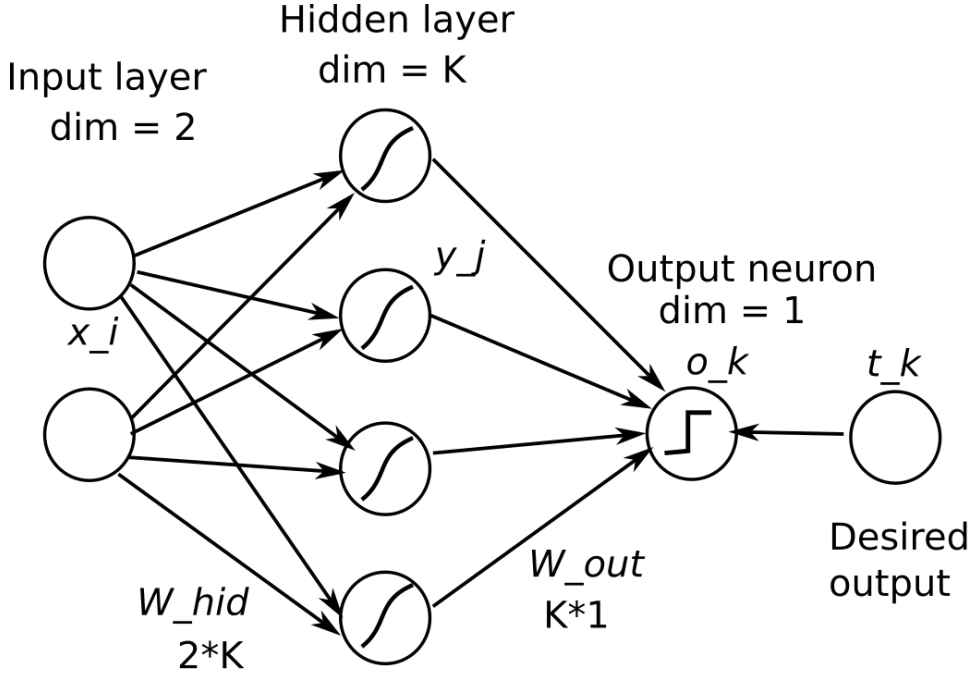
Machine Learning - Exercise 4

Multi-layer perceptron

Goal of the exercise

- The goal of this exercise is to implement a multi-layer perceptron to perform classification on the given files `nonlinear-classification.data` and `circular-classification.data`.
- The script `mlp.py` already provides the basis of the code.



Structure of the MLP

- The MLP is composed of 2 input neurons x_i, one output neuron (o_k) and K hidden neurons in a single layer (y_j).
- The output neuron sums its inputs with K weights W^{out} and a bias b^{out}. It uses a threshold transfer function:

$$o_k = \begin{cases} +1, & \text{if} \quad \sum_{j=1}^{K} W_j^{\text{out}} \cdot y_i + b^{\text{out}} \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- Each of the K hidden neurons receives 2 weights from the input layer, what gives a $2 \times K$ matrix W_{hid}, and has a bias b^{hid}.
- The hidden neurons use a logistic transfer function:

$$y_j = \frac{1}{1 + \exp(-\sum_{i=1}^{2} W_{i,j}^{\text{hid}} \cdot x_i - b^{\text{hid}})}$$

Implementation

```
import numpy as np
import matplotlib.pyplot as plt

# Load the data set
data = np.loadtxt('nonlinear_classification.data')

# Separate the input from the output
X = data[:, :2]
T = data[:, 2]
N, d = X.shape

# Parameters
eta = 0.05 # learning rate
K = 15 # Number of hidden neurons

# Weights and biases
max_val = 0.1
W_hid  = np.random.uniform(-max_val, max_val, (d, K))
b_hid  = np.random.uniform(-max_val, max_val, K)
W_out = np.random.uniform(-max_val, max_val, K)
b_out = np.random.uniform(-max_val, max_val, 1)
```

Feedforward pass

- The feedforward pass is already implemented in the method `feedforward()`.

```
# Logistic transfer function for the hidden neurons
def logistic(x):
    return 1.0/(1.0 + np.exp(-x))

# Threshold transfer function for the output neuron
def threshold(x):
    data = x.copy()
    data[data > 0.] = 1.
    data[data < 0.] = -1.
    return data

def feedforward(X, W_hid, b_hid, W_out, b_out):
    # Hidden layer
    Y = logistic(np.dot(X, W_hid) + b_hid)
    # Output layer
    O = threshold(np.dot(Y, W_out) + b_out)
    return Y, O
```

Backpropagation

- The goal is to implement the backpropagation algorithm by comparing the desired output t_k with the prediction o_k:

$$\delta_k = (t_k - o_k) \cdot f'(\text{net}_k)$$

$$\delta_j = f'(\text{net}_j) \cdot W_j^{\text{output}} \cdot \delta_k$$

$$\Delta W_{i,j}^{\text{input}} = \eta \cdot \delta_j \cdot x_i$$

$$\Delta W_j^{\text{output}} = \eta \cdot \delta_k \cdot y_j$$

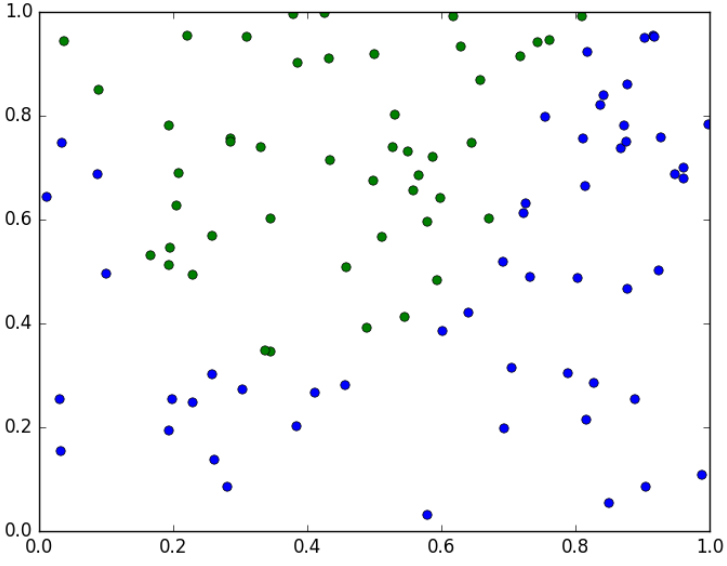
- You'll consider that the derivative of the threshold function is 1, while the derivative of the logistic function is given by:

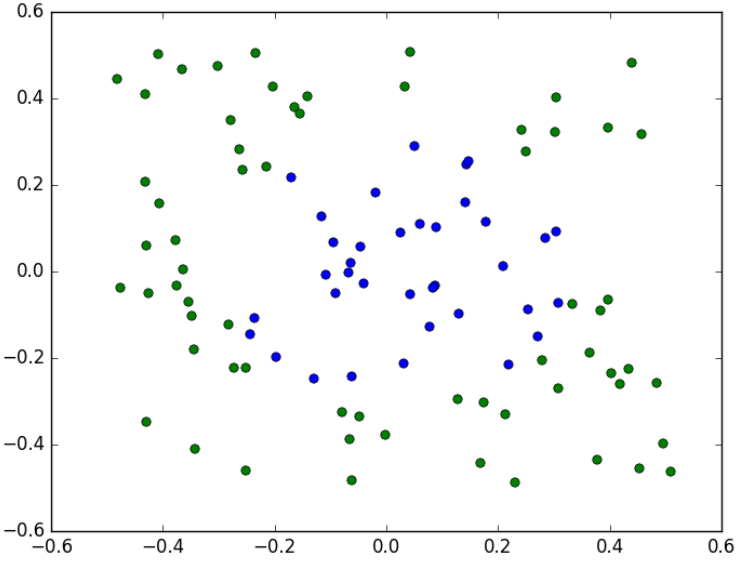
$$f'(x) = f(x) \cdot (1 - f(x))$$

Backpropagation

```
errors = []
for epoch in range(2000):
    nb_errors = 0
    for i in range(N):
        x = X[i, :]
        t = T[i]
        y, o = feedforward(x, W_hid, b_hid, W_out, b_out)
        if t != o:
            nb_errors += 1
        delta_out = (t - o)
        delta_hidden = 0.0 # TODO
        W_out += 0.0 # TODO
        b_out += 0.0 # TODO
        for k in range(K):
            W_hid[:, k] += 0.0 # TODO
        b_hid += 0.0 # TODO
    if nb_errors == 0:
        break
```

Questions





1. Complete the file `mlp.py` with the backpropagation algorithm to classify `nonlinear_classification.data` and `circular_classification.data`. The learning rate can be fixed to 0.05. Learning should stop when no more examples are miscorrectly classified, or when 2000 epochs have already been made. Try different values for the number of hidden neurons K (e.g. {2, 5, 10, 15, 20, 25}) and observe how correctness and speed of convergence evolve.

Questions

2. Observe more precisely the behaviour of the network during learning by calling the graphical function `plot_classification` every five epochs.

3. In the given file, the weights are initialized randomly between -1 and 1. Try to initialize them to 0. Does it work? Why?

4. For a fixed number of hidden neurons (e.g. $K = 15$), launch 10 times the same network. Does performance change? What is the mean and variance of the number of epochs needed? Vary the learning rate. How does performance evolve?

Questions

5. Instead of the logistic function, use a linear transfer function for the hidden neurons. How does performance evolve? Is the non-linearity of the transfer function important for learning?

6. Use this time the hyperbolic tangent function as a transfer function for the hidden neurons (method tanh() in NumPy). Does it improve learning? The derivative of the tanh function is given by:

$$f'(x) = 1 - f(x)^2$$

7. Use the Rectified Linear Unit (ReLU) transfer function:

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else.} \end{cases}$$

What does it change? Conclude on the importance of the transfer function for the hidden neurons.

Questions

8. In order to improve the convergence speed, try:

    1. to remove the mean value from the input,
    2. to randomize the order in which the examples are presented during one epoch (check the doc for the Numpy method `random.permutation()`).

    Does it improve convergence?

9. According to the article published in 2010 in AISTATS by Y.Bengio and X. Glorot called "Understanding the difficulty of training deep feedforward neural networks", the optimal initialization values for the weights between two layers of a MLP are uniformely taken in the range:

$$[-\frac{24}{\sqrt{N_{\text{in}}+N_{\text{out}}}};\frac{24}{\sqrt{N_{\text{in}}+N_{\text{out}}}}]$$

    where N_{in} is the number of neurons in the first layer and N_{out} the number of neurons in the second layer. Initialize both hidden- and output-weights with this new range. Does it help?