Data Mining: Practical Assignment #4

Due on Thu & Fri, May 22-23 2014,

Task 1

Get familiar with Python and numpy:

i.) Which expression generates the column vector $A = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$?

ii.) Given a matrix $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

Which of the following expressions would give the matrix $C = \begin{bmatrix} 2 & 3 \\ 5 & 6 \\ 8 & 9 \end{bmatrix}$?

1. `B[:,2]`
2. `B[:,1:3]`
3. `B[:,2:3]`
4. `B[2:3,:]`

iii.) Suppose you wish to generate a 3x1 vector D that contains the number 5 in every position. Which of the following expressions will accomplish this task?

1. `numpy.eye(3)*5`
2. `numpy.identity(3)*5`
3. `numpy.ones(3)*5`
4. `numpy.ones(3,1)*5`
5. `numpy.ones((3,1))*5`

iv.) Which expression allows to create a new matrix E by appending the column vector D to the matrix B?

v.) Suppose you wish to generate a 2x3 matrix F that contains only zeros. Which of the following expressions will achieve this goal?

1. `numpy.zeros(3)`
2. `numpy.zeros((2,3))`
3. `numpy.zeros(2,3)`
4. `numpy.zeros(3,2)`
5. `[0 0 0; 0 0 0]`
6. `[[0, 0, 0], [0, 0, 0]]`

Present and discuss your solutions.

Task 2

The task is to train an MLP that approximates the function as well as possible. Unfortunately, the given pseudo-code contains several errors. Correct the errors in the following sketch of a solution and do necessary completions.
Given data $\{\vec{x}^\alpha, y^\alpha\}$, $\alpha = 1, \ldots, 10000$. Let the components of the input vectors $\vec{x}^\alpha \in \mathbb{R}^5$ be from within the interval $[0, 100]$. The outputs $y^\alpha \in \mathbb{R}$ are within an interval $[0, 10]$ and have been produced by an unknown function with noise.

Proposed solution:

Definition of the network architecture: 5 inputs (layer 0), 3 hidden neurons (layer 1), 1 output neuron (layer 2). One bias neuron with activation "-1", which is connected with all neurons.

```
Initialisation of all weights $w_{ij}^{\lambda\,\lambda-1}$ with random values in the interval [0,1]

BEGIN {iterate}

    all temporary weights: $\Delta w_{ij}^{\lambda\,\lambda-1} = 0$

    training error: $E^{train} = 0$

    BEGIN {for each data point}

        activations of input neurons $i$: $S_i^0 = x_i^\alpha$

        for layers $\lambda = 1$ and then $\lambda = 2$: $S_i^\lambda = 1/(1 + \exp(-\sum_j w_{ij}^{\lambda\,\lambda-1} S_j^{\lambda-1}))$

        in layer $\lambda = 2$ for neuron $i$: $\delta_i^\lambda = S_i^\lambda \cdot (1 - S_i^\lambda) \cdot (y^\alpha - S_i^\lambda)^2$

        for layer $\lambda = 1$: $\delta_j^{\lambda-1} = S_j^{\lambda-1} \cdot (1 - S_j^{\lambda-1}) \cdot \sum_i \delta_i^\lambda w_{ij}^{\lambda\,\lambda-1}$

        for layers $\lambda = 1$ und $\lambda = 2$: $\Delta w_{ij}^{\lambda\,\lambda-1} = \Delta w_{ij}^{\lambda\,\lambda-1} + \delta_i^\lambda S_j^{\lambda-1}$

        with $i = 1$, $\lambda = 2$: $E^{train} = E^{train} + (y^\alpha - S_i^\lambda)$

    END {for each data point}

    for layers $\lambda = 1$ and $\lambda = 2$: $w_{ij}^{\lambda\,\lambda-1} = w_{ij}^{\lambda\,\lambda-1} + \Delta w_{ij}^{\lambda\,\lambda-1}$

END {iterate} STOP IF {$E^{train} = 0$}
```

Task 3

Perceptron, Multi-Layer Perceptron, and Learning.

a) Explain in detail the limits of linear separability with the XOR-problem.
Draw a sketch.

b) Design a perceptron at hand (pen & paper should suffice) with inputs x_A and x_B, which implements the boolean function $f(x_A, x_B)$, representing the propositional formula $A \wedge \neg B$.
Draw the network, indicating all relevant parameters.

Hint: Use the value 0 *for false and* 1 *for true, and the activation function* $\varphi(x) = \max(sign(x), 0)$*. Here, we are interested in the ideal network for the problem.*

c) Now, design a two-layer-perceptron, which implements a function, representing A XOR B. *(same hint)*

d) Let's dive into the code examples that are given in the `data4.zip` – you can choose between **Matlab** or **Python**. The given code considers a Multi-Layer-Perceptron solving the XOR problem, including the backpropagation algorithm for learning, as discussed in the lecture.
Assign the computational steps (forward pass, backward pass, etc.) into your figure of last exercise c.
With your ingredients, go through the code and check, where and how the steps have been realized.

e) Train the network.
Print out the weight vectors.
Sketch them on paper (which represents the input space), and the according decision boundaries of the hidden neurons.
To make the interpretation easier, set the biases to zero and re-train the network.

Task 4

Finally we want go get practical with the MLP. For this task you have the choice between working on an exercise designed for Matlab and an exercise designed for Python. For both options the procedure is a bit different, but we want to work on the same data here.

Matlab:
Matlab provides a huge Neural Network Toolbox for pattern recognition. In this task we want to rapid prototype a neural network for classification of wine data.
For the description of the data set type *help wine_dataset* in your Matlab command window.
Then, load the data with *load wine_dataset* into your Matlab path. Call the Neural Network toolbox with *nnstart* and provide the input and target for your network. Make some experiments by configuring the parameters, e.g. number of hidden neurons and check your results using the confusion matrix and the ROC curve.

Python:
Python is a nice tool for quickly implementing neural networks and for reasonably quick working on large data if efficient libraries (e.g. numpy) are used. In this task we want to see our neural network in action for a larger real data set.
For the description of the data have a look in the *README_task4.txt* within `data4.zip` file.
In the archive you also find a Python script for this task. View and comment the source code and then call the script. Make some experiments by configuring the parameters, e.g. number of hidden neurons, and learning parameters and check your results using the confusion matrix, and comparing development of the mean training error of different runs.

For **both options** also discuss the following issues:

- What is the purpose of a training, validation and test set?
- What would happen if the data set is reduced to only some samples, i.e. which consequence would that have in the context of learning?
- How are these sets created from the whole data set?
- What is your best network configuration to solve the problem?