MACHINE LEARNING FROM DATA
Fall 2018

Lab Session 5 – Neural Networks

1. Goal

The goal of this session is to

- Learn how to use the Neural Network pattern recognition tool GUI `nprtool`
- Learn to create, train and evaluate a Neural Network from the command line
- Use a Neural Network to solve the MNIST classification problem

2. Instructions

Getting the material:

- Download and uncompress the file ML_Lab5_soft.zip

Handling your work:

- Answer the questions in the document Lab5_report_yourname.pdf
- Save the report and Matlab code in a folder, and upload the compressed folder (zip, rar).

3. Previous study and nprtool GUI

Read the slides corresponding to lecture 4.3: neural networks.

In this section we will see how to train a network using the neural network pattern recognition tool GUI `nprtool`. This example uses the **cancer dataset** provided with Matlab Neural Network Toolbox. The dataset consists of 699 nine-element input vectors (d=9) corresponding to two classes (c=2, benign or malignant). Labels are one-hot encoded, using for each sample a vector of two elements, with a one in either the first of the second element: [1,0] for benign and [0,1] for malignant.

```
Dataset:
cancerInputs - a 9x699 matrix defining nine attributes of 699 biopsies.
1. Clump thickness
2. Uniformity of cell size
3. Uniformity of cell shape
4. Marginal Adhesion
5. Single epithelial cell size
6. Bare nuclei
7. Bland chomatin
8. Normal nucleoli
9. Mitoses

cancerTargets - a 2x966 matrix where each column indicates a correct
category with a one in either element 1 or element 2.
1. Benign
2. Malignant
```

Follow these steps:

1. Open the Neural Network Start GUI with the command `nnstart`
2. Click **Pattern Recognition Tool** to open the Neural Network Pattern Recognition Tool (you can also use the command `nprtool`)
3. Click **Next** to proceed. The Select Data window opens

4. Click **Load Example Data Set**. The Pattern Recognition Data Set Chooser window opens
5. Select **Breast Cancer** and click **Import**. You return to the Select Data window
6. Click **Next** to continue to the Validation and Test Data window
   Set validation and test datasets to 15% of the original data. With these settings, the input data and labels will be randomly divided into three datasets:
   - 70% used for training
   - 15% used to validate that the network is generalizing and to stop training before overfitting
   - 15% used as an independent test of network generalization
7. Click **Next**
   The standard network for pattern recognition is a two-layer feedforward network, with sigmoid activations in the hidden layer, and a softmax activation in the output layer. The default number of hidden layers is 10. You can increase this number if the network does not perform as well as expected. The number of output neurons is set to 2, which is equal to the number of classes. The Neural Network panel shows a graphical diagram of the network.
8. Click **Next**.
9. Click Train
   With the default setting, the training automatically stops when generalization stops improving, as indicating by an increase in the loss of the validation set.

10. Analyze the Neural Network Training window.
    Under the Plots pane, click **Confusion**. The figure shows the confusion matrices for training, testing and validation, and the three datasets combined. Green squares show the number of correct responses and red squares show the number of wrong responses. The lower right blue squares illustrate the overall accuracies and errors.
11. Under the Plots pane, click **Receiver Operating Characteristic** to visualize the ROC curves, plotting the true positive rate (= sensitivity = recall) versus false positive rate (1-specificity) as the threshold is varied.
12. In the **Neural Network Pattern Recognition Tool** window (nprtool), click Next to evaluate the network.

    At this point you can test the network on new data
    If you are not satisfied with the network performance on the original or new data, you can train the network again, increasing the number of neurons, or using a larger training set. If the performance on the training set is good but the performance on the test set is significantly worse, which can indicate overfitting, you can try reducing the number of neurons.
13. Click **Next**.
    You can use this panel to generate a Matlab function for the network. You can use it to better understand how the neural network computes outputs from inputs. You can also generate a graphical diagram.
14. Click **Next** and use the buttons on this screen to save results.
    You can create simple or advanced scripts, Matlab code that can be used to reproduce all the previous steps from the command line. Creating Matlab code can be helpful if you want to learn how to use the command line functionality of the toolbox to customize the training process. You can also save the network as net in the workspace. Click **Advanced Script** and save the file.
15. Click **Finish** to end.

Open the script `Lab5_cancer_net.m`. Identify the code sections: network setting (training function, loss or performance function, number of hidden layers, plot functions), data division into training, validation and test subsets, network training and evaluation.

Q1: List the main network features: training function, performance function, number of hidden layers, learning rate (for traingd), epochs, early stopping criterion. You will need to search Matlab help documentation to find the default values of some of the hyperparameters (ex. help traingd)

Q2: Train the network using gradient descent. Copy Performance and Confusion figures and discuss the results.

Q3: Repeat the experiment in Q2 changing the values of the learning rate (ex. 0.05 or 0.1). If necessary, change the default number of epochs so that the training stops when there are no improvements on the validation set. Discuss the results

Q4: Train the network using Levenberg-Marquadt (net.trainFcn='trainlm'). Compare with gradient descent in terms of convergence speed and performance.

4. MNIST dataset

4.1. Characteristics of the dataset

MNIST ("Modified National Institute of Standards and Technology") is a popular dataset in computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. The goal is to correctly identify digits from a dataset of tens of thousands of handwritten images.

The data file **mnist.csv** contain gray-scale images of hand-drawn digits, from zero through nine. There are 42.000 samples from the ten classes (c=10).

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels. Each pixel has a single value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The data set (mnist.csv) has 785 columns. The first column, the label, is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

4.2. Classification using a NN

Open the script `Lab5_mnist_net.m` and identify the different sections: dataset loading, image visualization, data preprocessing and network definition, training and evaluation.

Q5: Which are the network settings and hyperparameters? (training algorithm, performance function, number of epochs, etc.)

Q6: Compute confusion matrices and performance plots for the training, validation and test sets. Insert the figures in the report. Discuss results.

Q7: Try to improve the network performance by increasing the number neurons in the hidden layer. Edit the script, add the code to validate the number of neurons. Choose the number of neurons that minimize the classification error on the validation set. Try with the following numbers: 10, 50, 100, 150, 200, 250, 300. Compute the error on the test set.

Q8: Plot the network accuracy vs the number ~~of hidden layers~~
of neurons in the hidden layers