Lab 4: neural networks

Summary

Lab 4 covers neural networks, including Matlab's Neural Networks Toolbox and Keras. Students will experiment with feedforward multi-layer networks, autoencoders, and optionally, neural networks in Keras.

- contains hundreds of data sets of all types and sizes
  - Well-known datasets from UCI: Iris, 20 Newsgroups, Wine, Breast Cancer Wisconsin (Diagnostic), Human activity recognition using smartphones
- The NIST handwritten digits data set from the American "National Institute of Standards and Technology"
  - A choice of character images from NIST is includded in The MNIST handwritten character database that you already used.
- Kaggle hosts machine learning competitions and has many datasets available, including links to UCI

Task 0: Neural networks in Matlab

In this assignment you will use Matlab's own neural network tools, contained in a library called Neural Networks Toolbox

Matlab doc page: The Neural Networks Toolbox ("Shallow networks")

To get acquainted with the toolbox, first execute the introductory tutorial

Matlab tutorial: Fit Data with a Neural Network

One thing to be aware of: the library assumes a different convention for the data.

While we always had patterns as rows and variables as columns, here the functions expect patterns as columns and variables as rows. So you have to transpose data matrices if they follow our standard rather than Matlab's (e.g., Iris).

This is not a problem if you use the GUI (it lets you choose), but when using the functions directly it is important to be careful.

Task 1: Feedforward multi-layer networks (multi-layer perceptrons)

Execute the tutorial:

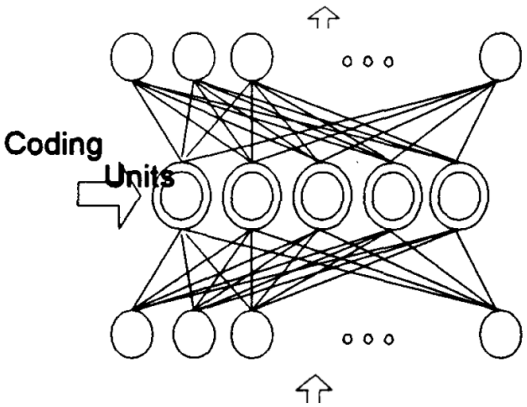
Matlab tutorial: Classify Patterns with a Neural Network

You can try with various data sets.

For the report, create result tables or use the confusion matrices automatically generated by the Matlab apps, obtaining experimental results for a couple different problems (data sets) and with different design choices (number of layers, number of units per layer).

Task 2: Autoencoder

The simplest **autoencoder network** is a multi-layer perceptron neural network which has one input layer, one hidden layer (n.hidden units<n.inputs), and one output layer (n.output units=n.inputs).



It is trained using **the same pattern as both the input and the target**. So for instance input pattern x(l, :) has target t(l, :)=x(l, :) (the same as the input).

Note that in this case we don't have any classes or other mapping to learn. This is a special case of **unsupervised training**. In fact, it is sometimes called "self-supervised", since the target we use is the input pattern itself.

Train a multilayer perceptron as an autoencoder for the MNIST data. Training an autoencoder only amounts to using a multi-layer perceptron neural network for **data prepared in a special way** (target = input); in its basic form it is **not a different neural network algorithm**. However, Matlab provides a separate function that is used as follows:

```
myAutoencoder = trainAutoencoder(myData,nh);
myEncodedData = encode(myAutoencoder,myData);
```

nh = size (number of units) in the hidden layer.

An autoencoder learns an **internal, compressed representation** for the data. The interesting output, therefore, is the **value of its hidden layer**. What we hope is that similar patterns will have similar representations; for instance, we hope that images representing a "1" will correspond to very similar representations, and quite similar to "7" but different from "0" or "8". In other words, that the network will **learn the classes**.

The workflow of the experiment is as follows, illustrated for the 10-class MNIST digits problem:

1. Split the data into subsets of different classes x1, x2, ... , x10. (Given that the dataset is relatively large, you may want to extract only some classes.)
2. Create a training set with only 2 classes (e.g., for MNIST you may choose 2 digits like 1 and 8). Experiment with different combinations; some will be easier to learn, some harder.
3. Train an autoencoder on the new, reduced training set (see how to call function "trainAutoencoder" above)
4. Encode the different classes using the encoder obtained
5. Plot the data using the "plotcl" function from the Lab resources Aulaweb section, folder "Matlab code snippets".
   When calling plotcl, be sure to transpose the data, to go back to our convention "observations in rows, variables in columns"

Optional task: Neural networks in Keras

Select one problem (a dataset) from the repositories mentioned at the beginning of this page. Select either classification or regression tasks. Download the data and train a neural network with the architecture of your choice using the Keras python library. Get guidance from the demo on the Puma Dynamics prediction task (downloadable from the main Aulaweb page).

Obviously, for this task you are supposed to install all relevant software pieces. Check the PumaDyn demo code for suggestions, then go on reading the Keras online documentation for more possibilities.