Computational Intelligence, SS11

http://www.igi.tugraz.at/lehre/CI

Gerhard Neumann [1]

4 Homework: Face Recognition with Neural Networks

[Points: 9 + 11* , Issued: 2010/05/03 , Deadline: 2011/05/20 , Tutor: Manuel Hofer [2]; Infohour: TBA , 15:30-16:30 , HS i11 ; Einsichtnahme: TBA , 15:30-16:30 , HS i11 ;]

3 Face Recognition with Neural Networks [9 points]

In this task you are asked to work with the dataset faces.mat[3] which contains face images. The dataset contains images of different persons, with different pose (straight/left/right/up), with/without sunglasses and showing different emotions. Download the matlab dataset. It contains 2 datasets: dataset1 (*input1, target1*) with 60 data points and dataset2 (*input2, target2*) with 564 data points. The *target* matrices contain the class informations. The first column codes the person, the second column the pose, the third column the emotion and the last column indicates wether the person is wearing sunglasses. In template_faces.m[4] you can find a script for training a sunglasses recognizer. This script can be used as template. Additionally you need to download the file confmat.m[5] which is needed to calculate the confusion matrix.

3.1 Pose Recognition

- Train a 2 layer feed-forward neural network with 6 hidden units for pose recognition. Use dataset2 for training, *trainscg* as training algorithm and train for 300 epochs. Do not use any test set.
- State the confusion matrix on the training set. Are there any poses which can be better separated than others?
- Plot the weights of the hidden layer for every hidden unit. Can you find particular regions of the images which get more weights than others? Do particular units seem to be tuned to particular features of some sort?

3.2 Face Recognition

- Train a 2 layer feed-forward neural network with 20 hidden units for recognizing the individuals. Use dataset1 for training, *trainscg* as training algorithm and train for 1000 epochs. Use dataset2 as test set.
- Repeat the process 10 times starting from a different initial weight vector. Plot the histogram for the resulting mean squared error (mse) on the training and on the test set.
- Interpret your results! Explain the variance in the results.
- Use the best network (with minimal mse on the test set) to calculate the confusion matrix for the test set and the mean classification error (not the mse !) on the test set. Plot a few missclassified images. Do they have anything in common?

[1] mailto:neumann@igi.tu-graz.ac.at
[2] mailto:manuel.hofer@student.tugraz.at
[3] http://www.igi.tugraz.at/lehre/CI/homework/data/faces.mat
[4] http://www.igi.tugraz.at/lehre/CI/homework/data/faces_template.m
[5] http://www.igi.tugraz.at/lehre/CI/homework/data/confmat.m

3.2.1 Hints

- Normalize your input data using *mapstd* (in older Matlab versions (< 7.5) this function is called *prestd*)
- In the template script you can find the code for plotting an image and plotting the weights of a hidden neuron
- Be aware that the template script only covers the 2 class classification case !
- Use the functions *full* and *ind2vec* to get from the standard class coding to a 1 out of n coding.

3.2.2 Remarks

- Hand-in your matlab code as print-outs (no emails !!).
- Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortlessly.

4 Neural Networks as Feature Generator [5 *points]

Show that the hidden units of a network may find meaningful features on the following optical digit recognition problem.

- Let your input space consist of a 8x8 pixel grid. Generate a 100 training patterns for a digit-8 category in the following way: Start with a block letter representation of 8, where black pixels have values 0 and the white pixels +1. Generate 100 different versions of this prototype by adding independent random noise to each pixel. Let the distribution of the noise be uniform between -0.5 and 0.5. Repeat the above procedure for the digits 0 and 3 by removing some black pixels from the original (without noise) version of 8. This gives you a dataset of 300 training patterns.
- Train a 2-layer network with 2 hidden units for this classification task.
- Display the input to hidden weights as 8x8 images seperately for each hidden unit.
- Can you find any useful features in the weight patterns (features are in this case areas with the same weight value)? Interpret your results, in particular discuss why such a feature representation has been chosen by the hidden layer.

5 Decision Trees [6 *points]

In this example you are asked to implement the information gain calculation in order to build a decision tree. Download and unzip the decision tree framework for matlab trees.zip[6]. This framework consists of several functions for building a decision tree from a given data set, classify new data points with a given decision tree and several other evaluation functions. Please refer to the *trees_template.m* file in order to see how these functions work.
The function *getScore* calculates a score for splitting a certain attribute at node v. As input, it gets the attribute values of dimension j of all training examples in L_v and also the class labels of all examples in L_v. The attribute values can be any integer value, the class labels are either 1 or 2. At each new node, the function *getScore* is called for each dimension j. The dimension with the highest score is used for splitting. By now, the *getScore* function only returns random values, the correct implementation of these function is your task.

[6] http://www.igi.tugraz.at/lehre/EW/homework/data/trees.zip

- Use the dataset *treedataset1.mat*, which is already contained in the zip file. The file contains a *trainingSet* and a *testSet* variable. The last column of these matrices always contains the class label (1 or 2), all other columns contain nominal attribute values. Build the tree with the already existing random *getScore* function and determine the depth, the number of leaves and the error of the tree by performing a leave-one-out cross validation.
- Implement the information gain in the *getScore* function. Again evaluate the error, the depth and the number of leaves of the tree by performing a leave-one-out cross validation.
- Implement the gain ratio criterium in the *getScore* function. Again evaluate the error, the depth and the number of leaves of the tree by performing a leave-one-out cross validation.
- Compare the results of the three tree-building strategies. Can you see any difference in the complexity of the trees as well as in the performance of the three tree-building strategies?

5.0.1 Hints

- In order to determine all possible values for a given attribute the function *unique* is very useful.