Cooperating Intelligent Systems Lab 2
Digit recognition

August 28, 2012

我们的任务是构造一个分类识别手写数字。机器学习算法可以应用到这样的问题，如果我们能够提供的例子（观测）。为了设计这样的算法的一个需要调查如何最好地代表输入/输出模式和学习参数/配置，是最适合于特别的问题。要验证的算法，设计了一个正确的方式，你需要把它通过最终测试能够正确分类新的（看不见的）数字。

1 Task

The task is to construct a working classifier for recognizing hand written digits. Machine learning algorithms can be applied to such a problem if we can provide examples (observations). In order to design such an algorithm one needs to investigate how to best represent input/output patterns and what learning parameters/ configuration that is best suited for the particular problem. To verify that the algorithm has been designed in a correct way, you will need to put it through the ultimate test: Being able to classify new (unseen) digits correctly.

2 Details

Using a graphical interface the user jots a single digit. It could be on some sophisticated handheld device but in our case the writing functionality is supplied by a simple Java application (you are provided with this fully working application, see the 'aidigit.zip' file). The input is a 32 by 32 bitmap. In the test application the user presses a classify button which directs the application (see AppDigits.java) to a function which you should modify (see Classifier.java). Currently there is a fully working neural network (a single layer network) operating in the provided classifier. You will find a fully working example of how to train a neural network for this purpose in AppTrain.java. You can use it, and/or modify it in whatever way you like. In your study you should make use of a neural network, and tweak some aspect of it (e.g. change the learning rate, and/or try different number of

hidden nodes). At your disposal there is a fully working implementation of a single-layer neural network nn.NN and a multi-layer neural network nn.NN2. You have access to a large data set with handwritten digits, available in the archive as digits.dat (have a look at the file in a text editor, each row corresponds to an image, and each row contains the 0s and 1s in a 32 by 32 bitmap plus at the end of the row is a number that tells what number the image corresponds to). User input was slightly modified so that the digit appeared centralized on the bitmap. The data originally comes from the UCI machine learning repository. As different ways of presenting inputs and targets may work better than others, a crucial factor of creating efficient machine learning applications is to understand which input/output encoding serves our purpose best. For the suggested domain, it has been shown that pre-processing can increase recognition accuracy considerably. As an example, dividing the original 32 by 32 bitmap into 4 by 4 regions which translates into a value between 0 and 16 depending on how many of the bits that are set, decreases input dimensionality and increases reliability (for help, and full implementation on this transformation, see to8x8 in Classifier.java). A good pre-processing strategy (which exploits domain-specific properties, consider e.g. gradients, stroke lengths and orientation) may give you the upper hand on more naive approaches. Note that to provide an idea of how well a machine learning algorithm performs you need to divide data into a training set and a test set.

3 Grading

Your results are graded as either u,3,4 or 5. These are the requirements for the grades:
3. Show that a neural network be used to classify handwritten digits for at least two different configurations of your neural network. A configuration is different if you use a different learning rate and/or if you use a different number of hidden nodes. How well does your system perform on training data? Divide the original data set into at least two halves (a training and a test set). How well does your system perform on test data? Can you verify that pre-processing in accordance with the 8x8 input approach described above increases accuracy for handwritten digit recognition?
4. Which learning parameters are the most effective? (i.e. which are the optimal values). Are there any alternative architectures that can outperform

the one supplied in the code?
5. Come up with an alternative pre-processing (or regularization) strategy that at least can be thought of as performing better on test data compared with the above.