Project 5:

Image Classification with Neural Networks

Antes que cualquier cosa: indicarme en un "readme" los miembros del equipo (2 por equipo máximo). En este proyecto se utilizará solo el ambiente de desarrollo Netbeans, ningún otro se aceptará. El proyecto tiene que compilar PERFECTAMENTE; ademas se usara una ruta relativa para leer los archivos, cosa que el programa pueda leerlos sin problemas independientemente de en qué directorio se corra el programa. Si estas condiciones no son cumplidas, eso influirá en su nota de una manera negativa.

The purpose of this project is to learn to use neural networks for the purpose of classification. In this case we are going to classify handwritten digits using the MNIST database (Modified National Institute of Standards and Technology database).This database has a training set of 60,000 examples, and a test set of 10,000 examples. Each example is an image of a handwritten digit and a corresponding label. The images have been centered, but the pixel image values should be normalized to real values by using this formula:

$$\frac{pixel\ intensity}{127.5} - 1.0$$

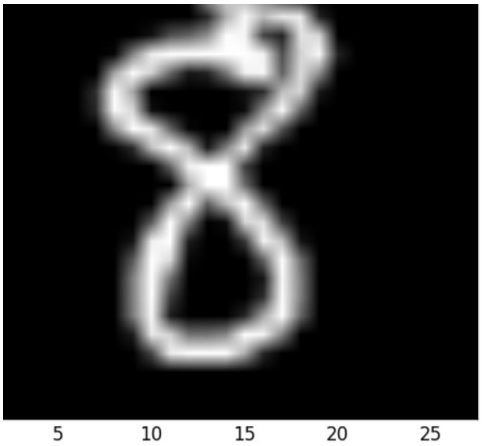
Each image is 28 x 28 pixels. The labels consist of a vector of 10 elements. Each position in the vector corresponds to a digit, digits from zero to 9. For example a vector $[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]^T$ corresponds to digit 9, a vector $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ corresponds to the digit 0. All the 10 decimal digits are represented following the same pattern.

Each position in the digit vector represents an output neuron in the neural network. The inputs of the neural network are the 28 x 28 or 784 pixels of each image of a handwritten digit. Each pixel is an input neuron.

The MNIST database of handwritten digits consist of four files, the first two files are for training the neural network, and the last two files are for testing the neural network. The training set consists of 60,000 handwritten images with the corresponding 60,000 labels. The testing set consists of 10,000 handwritten images with their corresponding 10,000 labels.

(1) train-images-idx3-ubyte: training set images
(2) train-labels-idx1-ubyte: training set labels
(3) t10k-images-idx3-ubyte: test set images
(4) t10k-labels-idx1-ubyte: test set labels

Here is a sample image of the MNIST database:



Our neural network overall design:

1. 28 x 28 input nodes. One for each pixel associated with a given digit.
2. 10 output nodes, one for each digit.
3. One inner layer.
4. We will use backpropagation for training the neural network.

You will be supplied with the code in Java that comes with our text book; Artificial Intelligence: A Modern Approach (3rd Edition) by Stuart Russell and Peter Norvig. Use the code I am attaching as part of the project; **do not use any other code**. You are to modify this code to define a neural network per the above specifications, and to successfully train it with the training examples to be able to predict handwritten digits captured in images with the already specified format. You are then to test the trained neural network with the test set examples. Such that good results can be obtained with respect to the error rate on the test set, you are to experiment with the following parameters:

1. The number of neurons in the hidden layer
2. The activation functions used in the hidden and output layers
3. The learning and momentum rates used during learning
4. The number of epochs to train the neural network

A class to read both the training and test set examples from the files provided has been provided. The name of the class is MNISTReader. You are to integrate this class to the code.

The code has two parts: the aima-core and the aima-gui. The aima-gui part references the aima-core part. You can start to analyze the code by looking at the class aima.gui.demo.learning.LearningDemo. You need to integrate the examples obtained from reading the files with the mechanism in the code for training and testing a backpropagation neural network.

The backpropagation algorithm in the code updates weights and also biases.

Two types of gradient descent are **batch gradient descent** and **stochastic gradient descent**. Batch gradient descent updates the weights only after all the training examples have been processed. That means for instance that for the first weight update iteration we take the partial derivative with respect to each weight and then add those derivatives evaluated at each example using the initial weights. Once again, only after all the examples have been processed we update each weight. Depending on the error function batch gradient descent can converge to a local minimum.

Stochastic gradient descent (SGD) updates the weights for each training example. SGD has a tendency to overshoot and may have problems converging in a minimum. One way to deal with this is to slowly decrease the learning rate as a function of the number of iterations.

The code given to you uses stochastic gradient descent.

Momentum is another term added to the weight updates. The idea is that momentum will avoid local minima by increasing the weight update in the previous update direction:

$$\Delta w_{ij}(n+1) = \varepsilon * \text{Gradient} + \alpha * \Delta w_{ij}(n)$$

The analogy used is a ball going downhill gathering enough momentum to go over local minima. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

VERY IMPORTANT:

1. **You can only use and modify the code I provided**
2. **You are to turn in a Netbeans project that compiles and runs without any problem whatsoever**
3. **You must experiment and research such that the neural network parameters give you at least an error rate in the test set ≤ 10%**