COGS 125 / CSE 175
Introduction to Artificial Intelligence
Specification for Programming Assignment #4

David C. Noelle

Due: 11:59 P.M. on Wednesday, December 13, 2017

Overview

This programming assignment has two main learning goals. First, the assignment will provide you with an opportunity to practice your skills in developing programs in Java™ using the Eclipse integrated development environment. Second, this assignment will provide you with some experience in implementing a key component of the most widely used artificial neural network learning procedure, namely the *backpropagation of error* learning algorithm. Your efforts in implementing this procedure are intended to produce a deeper understanding of machine learning methods using artificial neural networks. At minimum, completing this assignment will require an understanding of how error information can be propagated backward in a multi-layer perceptron in order to support the modification of connection weights in a manner that improves the performance of the network.

You will be provided with Java™ source code that implements a basic version of the backpropagation algorithm. This code will be complete except for the methods that calculate the *delta* (error) values for both output processing units and hidden processing units. You will be required to complete this implementation by supplying these methods with bodies, allowing the program to appropriately learn from a set of input-output training patterns.

Submission for Evaluation

To complete this assignment, you must modify only a single Java™ class source file, called "`Layer.java`". A template for this file will be provided to you, along with additional supporting source code. This file is to implement a layer of artificial neural network processing elements — a collection of units that all receive inputs from a common set of units and that all send their activation outputs to a common set of other units. As provided, this source code file is *not* complete, however. It is missing implementations for two methods: `computeOutputDelta` and `computeHiddenDelta`. The first of these methods is to fill in the vector of unit "delta" (unit

error) values for a layer of output units. The second is to fill in the "delta" values for a layer of hidden units. Your task is to provide bodies for these two methods by modifying the "`Layer.java`" source code file. You should *not* modify this file except to provide these two method implementations. Your implementations should work well with the rest of the provided source code, without any other modifications to that code. Thus, your modified version of "`Layer.java`" is the only file that you should submit for evaluation.

To submit your completed assignment for evaluation, log onto the class site on CatCourses and navigate to the "Assignments" section. Then, locate "Programming Assignment #4" and select the option to submit your solution for this assignment. Provide your program file as an attachment. Do not upload any other files as part of this submission. Comments to the teaching team should appear as header comments in your Java™ source code files.

Submissions must arrive by 11:59 P.M. on Wednesday, December 13th. Please allow time for potential system slowness immediately prior to this deadline. You may submit assignment solutions multiple times, and only the most recently submitted version will be evaluated. As discussed in the course syllabus, late assignments will not be evaluated.

If your last submission for this assignment arrives by 11:59 P.M. on Friday, December 8th, you will receive a 10% bonus to the score that you receive for this assignment. This bonus is intended to encourage you to try to complete this assignment early.

Activities

You are to provide Java™ source code that calculates unit delta values in a multi-layer perceptron according to the backpropagation of error learning algorithm. Your code is to be implemented as two methods of the provided `Layer` Java™ class. Your provided Java™ source code must be compatible with a collection of provided Java™ classes that implement a basic version of the backpropagation algorithm. Indeed, your assignment solution will be evaluated by combining your submitted modified `Layer` class file with copies of the provided artificial neural network utility files and testing the resulting complete program against a variety of test cases. In other words, *your solution must work with the provided utilities, without any modifications to these provided files*.

More specifically, you are to provide the following two public methods, all appearing in the "`Layer.java`" source code file:

- `computeOutputDelta`

  This method takes no arguments and returns no value. It is intended to fill in the `delta` vector instance variable of this layer according to the backpropagation algorithm. This method assumes that the layer is an output layer, so its `delta` vector can be based directly on external target values. Note that the Layer object contains both a vector of target values and a vector of net input values, as well as the layer's current activation values, and these should be used to calculate the appropriate unit delta values.

- `computeHiddenDelta`

  This method takes no arguments and returns no value. It is intended to fill in the `delta` vector instance variable of this layer according to the backpropagation algorithm. This method assumes that the layer is a hidden layer. This means that there is no valid external target information available to use in calculating this layer's `delta` vector values. Instead, the `delta` values must be computed as a function of the `delta` values of layers that are downstream from this one. The downstream layers can be found by examining the projections in this layer's `outputs` instance variable.

The rest of the backpropagation algorithm is already provided for you. In particular, the method called `computeDelta` on the `Layer` class checks to see if a layer is either an output layer or a hidden layer and calls the appropriate one of the two methods that you are implementing.

Many useful utilities are available in the provided source code files. This is particularly true of the files supporting the `Vector` and `Matrix` classes. If these utilities are properly used, the implementations of your delta-computation methods can be kept quite concise — certainly less than a dozen lines of code each. Making proper use of these utilities will require you to exercise your knowledge of linear algebra, however, thinking about the calculation in terms of vectors and matrices instead of individual scalar values. **The best solutions to this assignment will take this approach.**

If these two methods are properly implemented, the `main` method in the provided `Pfour` class should correctly train a specified three-layer network on a given collection of input-output examples. This program requires you to specify the number of inputs and the number of outputs in each example, as well as some number of hidden units to use in a single hidden layer. Training examples are provided to the program by specifying the name of a file containing those example patterns. This pattern set file is a plain text file that contains (1) the number of patterns in the file, (2) the number of inputs in each pattern, (3) the number of outputs in each pattern, and (4) one pattern per line for the remainder of the file, with input values listed before target output values. Numbers appearing in this file are to be separated by whitespace. The `Pfour` program also asks for the name of a file containing testing set patterns. Usually, testing set patterns are different than training set patterns, and they are used to test how well the trained network can generalize to new inputs. You may, however, provide the same file as both the training set and testing set, in which case no test of generalization will be possible, but you will still be able to check to make sure that the network is learning to produce the correct outputs for the patterns on which it is trained. Before asking for the pattern set file names, the `Pfour` program requests values for a number of training parameters. These include the learning rate, the maximum number of times (epochs) that each pattern should be presented to the network during training, and the level of sum-squared error (SSE) which, if attained, should cause training to stop. With all this information in hand, the `Pfour` program constructs a three-layer artificial neural network with the appropriate number of units in each layer and trains it on the training patterns, stopping when either the SSE criterion is reached or the maximum number of training epochs are completed. The program then displays the trained network's performance on each pattern in the testing set, displaying the network's outputs alongside targets and indicating the amount of remaining sum-squared error.

The Java™ utility classes that you are required to use are provided in a ZIP archive file called "`PA4.zip`" which is available in the "Assignments" section of the class CatCourses site. These utilities include:

- `Vector` — This object encodes a vector of real numbers. A wide variety of linear algebra functions are provided as methods on this class, including such things as the inner product (dot product) and outer product. Functions of particular use in artificial neural networks, such as applying a logistic sigmoid function to all of the elements in a vector, are also provided.

- `Matrix` — This object encodes a two-dimensional matrix of real values. As with `Vector`, many useful linear algebra routines are provided. Most noteworthy is the ability to multiply a matrix and a vector to produce another vector. (Hint: The matrix transpose method is also useful for this assignment!)

- `Pattern` — This is a single input-output pattern. It represents a single example, indicating that the given target vector should be produced at the network's output layer when the given input vector is presented to the network's input layer.

- `PatternSet` — This object encodes a collection of patterns. Methods are provided for reading and writing pattern sets from files.

- `Layer` — This object encodes a single layer of artificial neural network processing units. A layer can be an *input layer*, receiving input activity from external patterns, an *output layer*, receiving output target values from external patterns, or a *hidden layer*, with no direct connection to the external world. A layer can receive activity from several other layers, specified by its `inputs` instance variable. A layer can also send its activation output to several other layers, specified by its `outputs` instance variable. In simple three-layer networks, however, each layer has at most one layer providing it with input and at most one layer to which it provides its output. Still, the provided code works in the general case, and *your code should work in the general case*, as well! The `Layer` class contains methods for both forward propagation of activation and backward propagation of error (delta) values.

- `Projection` — `Layer` objects are connected through `Projection` objects. Each projection encodes a complete set of connections between all of the units in one layer and all of the units in another layer. Each `Projection` object keeps track of the layer on its input side and the layer on its output side, as well as the corresponding matrix of connection weight values.

- `Network` — This object encodes a complete backpropagation network. Layers are stored in the network in the order in which they need to be visited in order to properly update activation levels when a new input is presented. In other words, the input layer should be the first layer in this list and the output layer should be the last. Operationally, however, this order is determined by the order in which the user creates the layers, so care must be taken to create network layers in order, from inputs to outputs. This class includes methods for propagating activity through the network, from inputs to outputs, and propagating delta values back through the network, from outputs to inputs.

- `BP` — This object provides the top-level control of the backpropagation algorithm. Learning algorithm parameters, such as the learning rate, are stored as instance variables of this class. Methods are provided for performing one training pass (i.e., one epoch) over a set of training patterns, updating the network weights appropriately, and for testing network performance on a set of testing patterns.

- `Pfour` — This object provides a top-level driver that tests your implementation of the backpropagation algorithm by building a simple three-layer network and training it on a specified set of training patterns. Your code must allow `Pfour` to produce correct output for a variety of test cases.

The contents of these Java™ utility files will be introduced during a laboratory session, and inline comments in these files should assist in your understanding of the provided code. Questions are welcome, however, and should be directed to the teaching team.

Your implementation will be evaluated primarily for accuracy, with efficiency being a secondary consideration. Your source code *will* be examined, however, and the readability and style of your implementation will have a substantial influence on how your assignment is evaluated. As a rough rule of thumb, consider the use of good software writing practices as accounting for approximately 10% to 20% of the value of this exercise. Note also that, as discussed in the course syllabus, submissions that fail to successfully compile under the laboratory Eclipse Java™ IDE (i.e., they produce "build errors") will not be evaluated and will receive *no credit*.

As for all assignments in this class, submitted solutions should reflect the understanding and effort of the individual student making the submission. Not a single line of computer code should be shared between course participants. If there is ever any doubt concerning the propriety of a given interaction, it is the student's responsibility to approach the instructor and clarify the situation *prior* to the submission of work results. Also, helpful conversations with fellow students, or any other person (including members of the teaching team), should be explicitly mentioned in submitted assignments (e.g., in comments in the submitted source code files). Failure to appropriately cite sources is called *plagiarism*, and it will not be tolerated!

The members of the teaching team stand ready to help you with the learning process embodied by this assignment. Please do not hesitate to request their assistance.