CS 540: Introduction to Artificial Intelligence

HW #4: Neural Networks and SVMs

Assigned: November 7
Due: November 19

Late Policy:

Homework must be handed in by noon on the due date and electronically turned in by this same time.

1. If it is 0 – 24 hours late, including weekend days, a deduction of 10% of the maximum score will be taken off in addition to any points taken off for incorrect answers.
2. If it is 24 – 48 hours late, including weekend days, a deduction of 25% of the maximum score will be taken off in addition to any points taken off for incorrect answers.
3. If it is 48 – 72 hours late, including weekend days, a deduction of 50% of the maximum score will be taken off in addition to any points taken off for incorrect answers.
4. If it is more than 72 hours late, you will receive a '0' on the assignment.
5. In addition, there are 2 'late days' you may use any time throughout the semester. Each late day has to be used as a whole – you can't use only 3 hours of it and "save" 21 hours for later use.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TA, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

2. not explicitly tell each other the answers
3. not to copy answers or code fragments from anyone or anywhere
4. not to allow your answers to be copied
5. not to get any code or help on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: [12] Representing Boolean Functions using Neural Networks

For each of the following, define a Perceptron with one output unit that computes the desired output, if possible. Let 1 correspond to true and 0 false. Use a simple threshold activation (LTU) function at the output unit. Label the weights and the thresholds used in each network. If it is not possible to define the function using a Perceptron, say why not and then give a feed-forward network with one hidden layer (using LTUs) that implements it.

[a] [3] $A \Rightarrow B$
[b] [3] $A \Leftrightarrow \neg A$
[c] [3] $\neg$(A XOR B)
[d] [3] A four-input, one-output parity detector, where the output is 1 if the number of "1" inputs is even; otherwise the output is 0.

Question 2: [10] Neural Networks and Back-Propagation

Consider a learning task where there are three real-valued input features, A, B and C, and one binary output, D. You decide to use a 2-layer feed-forward neural network with two hidden units, H1 and H2, defining the hidden layer. The activation function is the sigmoid function defined in Figure 20.16(b) in the textbook. Each input unit is connected to every hidden unit, and each hidden unit is connected to the output unit.

[a] [5] Draw this neural network and initialize all the weights into and out of H1 to 0.5 and all of the weights into and out of H2 to -0.2. Initialize the "bias" of each hidden unit and output unit to 0.2 (i.e., the bias weight 0.2 is connected to a fixed input of -1. In practice, these weights would be set to small, randomly chosen, numbers but for grading simplicity we are specifying the initial values).

[b] [5] Consider the following training example and show how the back-propagation algorithm (Figure 20.25 in the textbook) would change your network's weights and biases. Using a learning rate of $\alpha = 0.3$. Only perform the "repeat…until" loop *once*. Clearly list all the variables and their values as you go through the algorithm.

$$A = 0.3, B = 0.8, C = 0.1, D = 1$$

Question 3: [8] Support Vector Machines

Exercise 20.12 in the textbook.

Question 4: [35] Face Detection using Neural Networks and SVMs

For this part you will be using a toolkit called Weka which contains feed-forward neural networks and SVMs.

The Data

Weka takes input from specially formatted .arff files. A description of the file format can be found at http://www.cs.waikato.ac.nz/ml/weka/ We have provided an .arff data file `faceData.arff` for you. Feel free to try some experiments on it to get used to using Weka. The dataset focuses on the task of detecting the existence of human faces in grayscale images. The images used in this study were created by the MIT Center for Biological and Computation Learning. The original CBCL Face Database #1's training set has 2,429 faces and 4,548 non-faces, and its testing set has 472 faces and 23,573 non-faces. A subset (250) of the training set and testing set is provided in the example file. The images' resolution has been reduced so they are all of size 19 × 19 pixels in order to keep training times manageable while still retaining enough information for adequate good classification performance. Consequently, each example will have 361 (= 19 × 19) input attributes, one for each pixel. The data file can be found on the course webpage.

Using Weka

Weka is a package of machine learning algorithms written at the University of Waikato, New Zealand. See `http://www.cs.waikato.ac.nz/ml/weka/` for more information. It has been installed on the instructional Linux machines at

```
        /s/weka/bin/weka
```

To use it, add the following line in your `.cshrc.local` file at the end of the section that specifies your paths:

```
        set path = ( $path  /s/weka/bin )
```

Go to the Weka web page if you want to install it on your own machine; there are Linux and Windows versions available.

To use Weka, execute the command: `weka`

6. You should see a small GUI with four options. Choose '`Explorer`'
7. This will open '`Weka Explorer`'. The starting tab will allow you select your data file. Choose '`Open file...`' and select the file that was provided. Feel free to examine the options in this tab, but nothing else needs to be done here.
8. Switch to the '`Classify`' tab. This is where you will be doing most of your work.
9. Under '`Classifier`' select '`Choose`'. You should see a large number of available classifiers. The two we will be using are '`functions > SMO`' and '`function > Multilayer Perceptron`'.
10. Once you've selected a classifier, you can set options by left clicking in the text box next to the '`Choose`' button. This will bring up a window allowing you to set options that are different for each classifier.
11. For all the examples in this assignment, you will be using five-fold cross validation to produce accuracy results ('`Test Options > Cross-`

validation > Folds = 5'). This will automatically divide the data into five partitions, build and test five models, and then aggregate the accuracy (thus you will only report one accuracy for each run)

12. Click 'Start' to begin your run. The results will appear in the window to the right.

If you get an "out of memory" message when running Weka, start it again with an increased memory size by executing the following command line on an instructional Linux machine:

```
  java –Xmx<amount>m -jar /s/weka/src/weka-3-4-13/weka.jar
```

where `<amount>` is the amount of memory in megabytes. The default value is 64 MB. For example, to increase memory to 256 MB do:

```
  java -Xmx256m -jar weka.jar
```

Part 1 - Support Vector Machines

For the first part of the assignment, you will be using Support Vector Machines to classify your data. Under 'Classifier', choose 'Functions > SMO'

1. Train a Support Vector Machine on the data using the default options
   (a) How long did the model take to build?
   (b) What percentage of the examples were classified correctly?
   (c) Report the confusion matrix produced.
   (d) Did the SVM tend to err in one direction?
2. Train a second SVM, but set the 'exponent' to 2
   (a) How long did the model take to build?
   (b) What percentage of the examples were classified correctly?
   (c) Did the results improve or get worse? Why?
3. Train a third SVM, but use an 'RBF function'. Set 'gamma' to 0.09.
   (a) What percentage of the examples were classified correctly?
   (b) Was there a small or significant difference between the results in 1 and 2? Why?

Part 2 – Multi-layer, Feed-Forward Neural Network

For the second part of the assignment you will be using a 2-layer, feed-forward neural network, containing one hidden layer, and an output layer with one unit for each output class. Under 'Classifier', choose 'Functions > Multilayer Perceptron'

1. Set the training time to 100, and reset to 'false'. Train six neural networks using combinations of the following settings: 'learning rate' to {0.1, 0.3} and 'hidden layers' (specifying the number of units in the hidden layer) to {5, 10, 20}.
   (a) For each of the six runs, report the time to train the network, and the percentage correctly classified.
   (b) Which of the six runs performed the best? Explain why you think the results are the way they are.

2. Set the training time to 300, and the hidden layers to 10. Train two networks using the two learning rates {0.1, 0.3}
   (a) Report the time to train and the percentage correctly classified for each of the runs.
   (b) Compare the results to those in question 1(a) with hidden layers set to 10. Did increase the training time increase performance? Will this always be the case?
   (c) Create another network with training time = 300, hidden layers = 5, and learning rate = 0.3. Compare this result to the analogous result in 1(a). Did it do better or worse?
3. Set the training time to 500, hidden layers to 10, and learning rate to 0.3. Set the validation set size to 10, and the validation threshold to 20. To understand what this is doing, click '`more`' and look for these properties. Set GUI to '`true`'. Run five cross-validation folds and observe what happens in order to answer the following questions.
   (a) Report the time to train and the percentage of correctly classified examples.
   (b) Report the approximate number of epochs each fold required for training. Were these all about the same?
   (c) Compare the accuracy percentage to the runs you did before with hidden layers set to 10 and learning rate set to 0.3.
   (d) What can we learn from questions (b) and (c) about the number of epochs required for the 'best' network?
4. Create your own experiment. Choose one of the parameters (hidden layers, epochs, learning rate, validation threshold, etc) to experiment with. Set the other parameters to constant values, and perform ten (10) runs, changing the parameter you chose to a new value each time.
   (a) Describe your setup, listing the constants you chose and the parameter you experimented with.
   (b) Describe the results you expected to see.
   (c) For each run, record the percentage correctly classified.
   (d) Create a graph of your parameter value vs correct classification percentage.
   (e) What do the results tell you about the parameter you chose?

Question 5: [35] Implement a Perceptron

Implement in Java the PERCEPTRON-LEARNING algorithm (Figure 20.21) with a single linear threshold unit (LTU). Use a fixed input value of -1 for the bias weight, w_0. Note the threshold activation function outputs 0 for input value 0 (Figure 20.16(a)).

Input: Your program will be run as follows:

```
java Perceptron training_file alpha
```

where `training_file` is the name of a file that contains the training examples, and `alpha` is the learning rate (a positive real number). The `training_file` has the following format:

- The first line has the number of training examples
- The second line has the number of feature dimensions
- Each remaining line lists (in this order): the features of a training example, and then the corresponding output y. These numbers are space separated.

An example training file, `Q5.txt`, can be found on the class webpage.

Initial weights: Before learning, please set all weights to 0.

Stopping criterion: Your algorithm should stop if and only if it no longer makes mistake on any of the training examples.

Output: Your program should print out the Perceptron weights w_j, $j = 0, \dots n$, in this order, separated by whitespace, on a single line. Do not print out anything else in the program that you hand in.

[a] Run your program on `Q5.txt` with `alpha` = 1. What are the final weights of your Perceptron?

[b] Draw the decision boundary corresponding to the trained Perceptron in [a], in the two-dimensional feature space. Mark which side of the decision boundary gets what label. Also show the training points and their labels.

[c] How does the learning rate, `alpha`, affect the learning of LTU Perceptrons?

[d] Train a (linear) SVM using Weka with the default options and the same training set (check '`Use training set`' button in the '`Test options`' box, then click '`Start`'). The file `Q5.txt` has been converted for use in Weka to a file called `Q5.arff` As in [b], show the training points, their labels, and manually draw the decision boundary of the ideal linear SVM solution. Compare the differences between this boundary and the boundary found in [b].

[e] Find *one more* training example and its label, such that when this new example is added to `Q5.txt`, your Perceptron training program loops forever, i.e., no matter how many epochs it trains, there is always some training error.

What to Turn In

1. On paper: Hand in in class a hard copy of your answers to Questions 1, 2, and 3. For Question 4, hand in the required results (test set accuracy) of all your test runs, and answers to the questions. For Question 5, answers to the questions. Put this all together, stapled, with your name, login and the date on the top of the first page.
2. Electronic content: hand in your code for Question 5 using the `handin` program. Copy it along with any other java files necessary to compile and run your program on the Linux machines into your own private handin directory, say called hw4-handin. Then, execute the following command from the directory containing your own handin directory:

   ```
   handin –c cs540-SECTION –a hw4 –d hw4-handin
   ```

   where SECTION is 1 or 2, depending on your CS540 section. For more information on how to electronically hand in your code, see the course web page. Be sure your code will compile and run on the departmental Linux machines; if it does not, we will not be able to grade it.