CS 4641: Machine Learning
Assignment 4

Due: November 17, 2016 11:59pm

Instructions

Read all instructions in this section thoroughly.

Collaboration: Make certain that you understand the course collaboration policy, described on the course website. You must complete this assignment **individually**; you are **not** allowed to collaborate with anyone else. You may *discuss* the homework to understand the problems and the mathematics behind the various learning algorithms, but **you are not allowed to share problem solutions or your code with any other students.** You must also not consult code on the internet that is directly related to the programming exercise. We will be using automatic checking software to detect academic dishonesty, so please don't do it.

You are also prohibited from posting any part of your solution to the internet, even after the course is complete. Similarly, please don't post this PDF file or the homework skeleton code to the internet.

Formatting: This assignment consists of two parts: a problem set and program exercises.

For the problem set, you must write up your solutions electronically and submit it as a single PDF document. We will not accept handwritten or paper copies of the homework. Your problem set solutions must use proper mathematical formatting. For this reason, we **strongly** encourage you to write up your responses using LaTeX. (Alternative word processing programs, such as MS Word, produce very poorly formatted mathematics.)

Your solutions to the programming exercises must be implemented in python, following the precise instructions included in Part 2. Portions of the programing exercise may be graded automatically, so it is imperative that your code follows the specified API. Several parts of the programming exercise ask you to create plots or describe results; these should be included in the same PDF document that you create for the problem set.

Homework Template and Files to Get You Started: The homework zip file contains the skeleton code and data sets that you will require for this assignment. **Please read through the documentation provided in ALL files before starting the assignment.**

Citing Your Sources: Any sources of help that you consult while completing this assignment (other students, textbooks, websites, etc.) ***MUST*** be noted in the your `README` file. This includes anyone you briefly discussed the homework with. If you received help from the following sources, you do not need to cite it: course instructor, course teaching assistants, course lecture notes, course textbooks or other readings.

Submitting Your Solution: You will be submitting only the following files, which you created or modified as part of homework 4:

- `README`
- `hw4-GTACC.pdf` (a PDF of your homework writeup)
- `textShowdown20News.py`
- `nn.py`
- `naiveBayes.py` (optional; if you're using your naive Bayes code from HW3)

Please follow the naming conventions exactly, and do not submit additional files including the test scripts or data sets. Your PDF writeup of Homework 4 should be named `hw4-GTACC.pdf`, where "GTACC" is your own Georgia Tech account name (for example, my file would be named "hw4-bboots3.pdf").

Submission Instructions: Please place all of your files into a single folder file named hw4-GTACC and compress the folder as a zip file with the same name. Upload the zip file as an attachment on T-Square. Again, please follow the naming conventions exactly, and do not submit additional files including the test scripts or data sets. For example my file would be named "hw4-bboots3.zip"

You may resubmit multiple times, only your last submission will be counted.

PART I: PROBLEM SET

Your solutions to the problems will be submitted as a single PDF document. Be certain that your problems are well-numbered and that it is clear what your answers are. Additionally, you will be required to duplicate your answers to particular problems in the `README` file that you will submit.

1 Logical Functions with Neural Networks (15pts)

For each of the logical functions below, draw the neural network that computes the function, and give a truth table showing the inputs, the value of the logical function, and the output of the neural network verifying that the neural network is correct. Show your work for the computations of the neural network's output.

(a) The NAND of two binary inputs

(b) The parity of three binary inputs

2 Backpropagation with Momentum (15pts)

[Adapted from Mitchell Sect. 4.5.2.1 and Ex. 4.7] One of the most common modifications to backpropagation is to alter the weight update rule to make the weight update on the nth iteration partially dependent on the update during the previous iteration. The modified weight update rule for the nth epoch is given by:

$$\underbrace{\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}(n)}_{\text{Standard Update Rule}} + \underbrace{\mu D_{ij}^{(l)}(n-1)}_{\text{Momentum Term}} \quad ,$$

where $D_{ij}^{(l)}(n)$ is the average regularized gradient computed at the nth epoch. The first few terms are exactly the same as the standard weight update rule, the last term is new and is governed by a parameter $0 \leq \mu < 1$ that is called the *momentum*. Essentially, the momentum term includes some fraction of the update from the previous epoch, which will enable the update to bounce out of small local minima or keep moving through flat regions where the search would stop if there were no momentum. It also has the effect of gradually increasing the step size of the search in regions where the gradient does not change, thereby speeding convergence.

Consider a two-layer feed-forward neural network with two inputs x_1 and x_2, one hidden unit h and one output unit y. This network has five weights $\left(\Theta_{x_1,h}^{(1)}, \Theta_{x_2,h}^{(1)}, \Theta_{0,h}^{(1)}, \Theta_{h,y}^{(2)}, \Theta_{0,y}^{(2)}\right)$, where $\Theta_{0,z}^{(l)}$ represents the threshold weight for unit z at level l. Initialize these weights to be $(0.1, 0.1, 0.1, 0.1, 0.1)$ and give their values after each of the first two training epochs of batch backpropagation with momentum. Assume a learning rate of $\alpha = 0.3$, momentum $\mu = 0.9$, and the following two training examples: $(x_1 = 1, x_2 = 0, y = 1)$ and $(x_1 = 0, x_2 = 1, y = 0)$. Be sure to show your computations for full credit.

3 (Extra Credit) TANH Neural Networks (15pts)

[Adapted from Bishop, Exercise 5.1] In a two-layer neural network (one hidden layer) with sigmoid activations, the outputs are given by:

$$y_k(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\sum_{j=1}^{M} \Theta_{jk}^{(2)} \sigma\left(\sum_{i=1}^{d} \Theta_{ij}^{(1)} x_i + \Theta_{0j}^{(1)}\right) + \Theta_{0k}^{(2)}\right) \quad ,$$

where $\sigma(a) = \frac{1}{1+\exp(-a)}$. This equation simply combines all the stages of the network into a single equation. Instead of the sigmoid function, we could use $\tanh(a)$ functions instead.

(a) (5 pts) What is the advantage of using the tanh function instead of the sigmoid in a neural network?

(b) (10 pts) Consider the two-layer neural network with sigmoid activations described above. Show that there exists an equivalent network, which computes exactly the same function, but with *hidden unit activation functions* given by $\tanh(a)$. Hint: begin by re-writing the equation above with tanh hidden unit activations, then find the relation between $\sigma(a)$ and $\tanh(a)$, and show that the parameters of the two networks differ by linear transformations.

PART II: PROGRAMMING EXERCISES

1 Text Classification (30 points)

In this section, we'll apply naive Bayes and support vector machines to the problem of document classification. Sklearn provides a number of utilities that make text processing easy! For a tutorial, see http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html. Read through the tutorial first, and then come back to this document.

Welcome back! In class, we discussed using two algorithms for text classification: (a) naive Bayes and (b) SVMs with a cosine similarity kernel. Write a python program that reads and evaluates both of these classifiers on the 20 newsgroups data set and outputs a number of performance metrics. Here are the requirements for the program:

- Name your program `textShowdown20News.py`, and ensure the entire program is contained in this file.
- Your program should load both the training and testing portions of the 20 newsgroups data set (see http://scikit-learn.org/stable/datasets/twenty_newsgroups.html)
- Your program will process the text to create TF-IDF feature vectors, train models on the training data (optimizing parameters as needed), and evaluate performance on the test data. Note that we're only using one training/testing split.
- You may use your own naive Bayes implementation from the last assignment, or you may use sklearn's built in `MultinomialNB` classifier. You should use the SVM and cosine similarity kernel implementations provided with sklearn.
- Your program must output a table of the following metrics for both classifiers: (a) train & test accuracy (b) train & test precision (c) train & test recall (d) training time. Ensure the table is neat and clear.
- Ensure that your program does not produce any other output when it is run. It is fine to add debugging or status output while you're developing the program, but remove this output before submission.

Recall from class that for document classification, we often do better if we use TF-IDF features instead of the bag of words representation (i.e., a feature vector of raw word counts). For the text processing aspects of this problem, be sure to follow these guidelines:

- Lowercase all terms and remove stop words (using the default english stop word list in `CountVectorizer`),
- Compute the dictionary and inverse document frequency over the training data only, then use the same preprocessing values for the test data,
- Use log-scaled term counts for the term frequency, and
- Normalize the final TF-IDF feature vectors to have unit length.

Run your program on the 20 newsgroups data set. Include your table of performance statistics in your PDF writeup. Which classifier is better? Write 2-3 sentences justifying your answer, discussing the results you obtained.

While writing this program, you may find it useful to reference the following websites:

- http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity

2 Neural Networks (40 pts)

Relevant Files in the Homework Skeleton

- **nn.py**
- digitsVisualization.tiff
- data/digitsX.dat
- data/digitsY.dat

In this problem, you will implement an artificial neural network (NN) classifier that learns via backpropagation. Instead of choosing the architecture of the network at implementation time, you should implement your neural net in a more general manner; the specific architecture of the network will be given to the constructor. Recall from class that we can specify the architecture of a basic neural net via a vector **s**, where each entry in the vector gives the number of nodes in that layer and the length of the vector is the total number of layers in the network. We will assume that within each layer, each node receives input from every other node in the previous layer.

Implement the neural network in a class called `NeuralNet` in `nn.py`, following the API given below:

- `__init(layers, epsilon = 0.12, learningRate, numEpochs)__` : constructor

  `layers` – a vector of $L - 2$ positive integers, where the number of layers in the network is L. The value contained in `layers[i]` specifies the number of nodes in the i^{th} hidden layer. Note that the specification of `layers` is a bit different from the **s** vector; in particular, it does not include s_0 and s_{L-1}, which are initialized from the training data in `fit()`.

  `epsilon` – one half the interval around zero for the initial weights (defaults to 0.12)

  `learningRate` – the learning rate for back-propagation

  `numEpochs` – the number of epochs for backpropagation

- `fit(X,Y)`: train the neural network model via backpropagation
- `predict(X)`: use the trained neural network model for prediction via forward propagation
- `visualizeHiddenNodes(filename)`: (Extra Credit Only) outputs an image representing hidden layers

While implementing the neural net, I recommend you follow the steps outlined below.

2.1 Network Structure and Initialization

Complete the constructor, which simply saves the arguments for use later. Since the number of units in the first and last layer are specified by the training data, we cannot initialize the network architecture and weight matrices until `fit()`. After finishing `__init()__`, start writing `fit()`.

The first thing you should do in `fit()` is to create all of the weight matrices $\Theta^{(1)}, \ldots, \Theta^{(L-1)}$. Initialize all of the weights to be uniformly chosen from $[-\epsilon, \epsilon]$, where ϵ is specified by the `epsilon` argument to the constructor.[1] Then, unroll the weight matrices $\Theta^{(1)}, \ldots, \Theta^{(L-1)}$ into a single long vector $\boldsymbol{\theta}$ that contains all parameters for the neural net.

2.2 Forward-propagation

Implement a private function to perform forward-propagation. This method will be useful for both backpropagation and the `predict()` method. This private function should take in a vector of parameters (e.g., $\boldsymbol{\theta}$) for the neural network and an instance (or instances) and return the neural network's outputs.

2.3 Backpropagation

Finish the `fit()` method to use backpropagation to minimize $J(\boldsymbol{\theta})$. Details for implementing backpropagation are in the lecture slides. At a minimum, you'll need to compute $J(\boldsymbol{\theta})$ and $\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta})$.

To confirm that your gradient computations are correct, I recommend that you implement the following gradient checking procedure to numerically estimate the gradient. Form two vectors:

$$\boldsymbol{\theta}_{i+c} \leftarrow \boldsymbol{\theta}; \text{ then } \theta_i \leftarrow \theta_i + c \qquad\qquad \boldsymbol{\theta}_{i-c} \leftarrow \boldsymbol{\theta}; \text{ then } \theta_i \leftarrow \theta_i - c \ . \tag{1}$$

[1] According to Andrew Ng, an alternative (and effective) strategy for choosing ϵ is to choose a different value for each layer's weights, with the ϵ for $\Theta^{(l)}$ as $\frac{\sqrt{6}}{\sqrt{s_l + s_{l+1}}}$, where **s** is the vector containing the number of nodes in each layer.

In other words, $\boldsymbol{\theta}_{i+c}$ (or $\boldsymbol{\theta}_{i-c}$) is the same as $\boldsymbol{\theta}$, but the i^{th} element has been incremented (or decremented) by c. Then, we can numerically estimate the gradient and verify that our gradient computations are correct by confirming that

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{J(\boldsymbol{\theta}_{i+c}) - J(\boldsymbol{\theta}_{i-c})}{2c} \quad . \tag{2}$$

Setting $c = 10^{-4}$, you should find that the computed gradient and the estimated gradient should agree to at least four significant digits.

This gradient checking procedure is very expensive, and so should only be used for small networks (small numbers of parameters). **Make absolutely certain to disable the gradient checking procedure once you're certain that your gradient implementation is correct.** Use the gradient checking procedure only for debugging purposes; be sure to disable it before running your learning algorithm.

Once your gradient computations are confirmed to be correct, finish `fit()` to run backpropagation for the number of epochs specified in the constructor to train the neural net.

2.4 Complete the Prediction Function

Your prediction function should call the forward-propagation method with the neural net parameters $\boldsymbol{\theta}$ trained via backpropagation in `fit()`.

2.5 Apply Your Neural Network to Digit Recognition

Write a test script named `testNeuralNetDigits.py` to apply your neural network classifier to the problem of digit recognition. The homework skeleton contains a data set of 5,000 20 × 20 digit images (see `digitsVisualization.tiff` for a visualization). We can represent each image as a 400-dimensional vector of pixel intensities. The features for the digits are provided in the `data/digitsX.dat` file and their corresponding labels are in `data/digitsY.dat`.

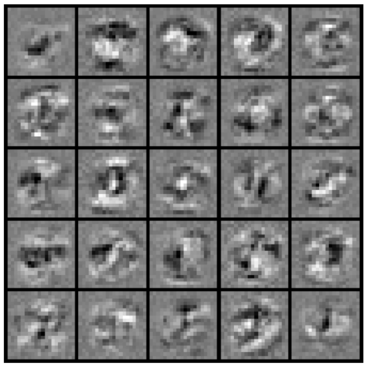
Train your neural network on the digits data with one hidden layer of 25 nodes over 100 epochs. Choose a small value for the regularization parameter in the neural network (e.g., start with something like 0.001 and tune it up or down as needed by hand – no need to tune it via cross-validation).

Tune the learning rate for the neural network. You should be able to get a training accuracy of approximately 95.3% or higher if your implementation is correct (±1% due to random initialization). If you're having trouble obtaining this accuracy with only 100 epochs, try using more epochs. It is fine if your implementation requires a few hundred more epochs to obtain higher accuracy. Report your optimal learning rate, regularization parameter, and the maximum training performance you obtained in your PDF writeup and README.

2.6 (Extra Credit) Visualizing the Hidden Layers (15 pts)

Complete the `visualizeHiddenNodes(filename)` function to visualize the hidden units in the network. For the neural network you trained above, note that the i^{th} row of $\Theta^{(1)}$ is a 401-dimensional vector that specifies the parameters for the i^{th} hidden unit. Discarding the bias term yields a 400-dimensional vector that we can reshape into an 20 × 20 matrix via the `numpy.reshape` command. If we remap[2] the values of this matrix to lie in 0 . . . 255, we can visualize the weights as a greyscale image.

Use the Python Image Library to create an image that visualizes all of the hidden layers. Separate the layers into blocks (e.g., you can visualize a 25 unit layer as an 5 × 5 grid, where each grid entry is the 20 × 20 greyscale image). You might find it useful to consult `http://en.wikibooks.org/wiki/Python_Imaging_Library/Editing_Pixels`. Save this image to the filename given as an argument to `visualizeHiddenNodes()`.



You should find that the hidden units correspond to different stroke detectors and other patterns in the input. For an example of the hidden unit visualization, see the image to the right. Yours will likely look slightly different from this one due to randomization. Include your output image visualizing the hidden layers of your network in your PDF writeup.

[2] I suggest you either map -1 to 0 and +1 to 255, or the min value over all units to 0 and the max value over all units to 255 – whichever gives you the nicer picture.