CIS 419/519 Introduction to Machine Learning
Assignment 4

Due: November 20, 2017 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 processors, 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 will be graded automatically, so it is imperative that your code follows the specified API. A few parts of the programming exercise asks 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: We will post instructions for submitting your solution approximately one week before the assignment is due. Be sure to check Piazza then for details.

CIS 519 ONLY Problems: Several problems are marked as "[CIS 519 ONLY]" in this assignment. Only students enrolled in CIS 519 are required to complete these problems. However, we do encourage students in CIS 419 to read through these problems, although you are not required to complete them.

All homeworks will receive a percentage grade, but CIS 519 homeworks will be graded out of a different total number of points than CIS 419 homeworks. Students in CIS 419 choosing to complete CIS 519 ONLY exercises will not receive any credit for answers to these questions (i.e., they will not count as extra credit nor will they compensate for points lost on other problems).

Assignment Version 20171102a

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 (10pts)

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 (10pts)

[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 TANH Neural Networks (CIS 519 ONLY – 10pts) (10pts)

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

(b) (7 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 and ROC (20 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 evaluates both of these classifiers on the 20 newsgroups data set, outputting a number of performance metrics and an ROC plot of both classifiers. 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 should use the `MultinomialNB` classifier, 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.
- Your program should also produce an ROC plot that contains curves for both classifiers, and output that plot to the file `graphTextClassifierROC.pdf`. Plot ROC curves for only the following classes: ['comp.graphics', 'comp.sys.mac.hardware', 'rec.motorcycles', 'sci.space', 'talk.politics.mideast']. The plot should show 10 ROC curves: 5 for naive Bayes and 5 for the SVM.
- 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 and your ROC plot 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/auto_examples/plot_roc.html
- http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity

2 Neural Networks (30 pts)

In this problem, you will apply artificial neural networks to the problems of digit and object recognition.

2.1 Digit Recognition with a Multi-Layer Perceptron (10 pts)

Scikit-learn already includes an implementation of the multi-layer perceptron (`MLPClassifier`, http://scikit-learn.org/stable/modules/neural_networks_supervised.html) that we will train via back-propagation.

Write a test script named `MLPDigits.py` to apply a 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 the 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). Use 'adam' as the solver with default values for the beta parameters.

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 the neural net requires a few hundred more epochs to obtain higher accuracy. Try different activation functions to see which works the best.

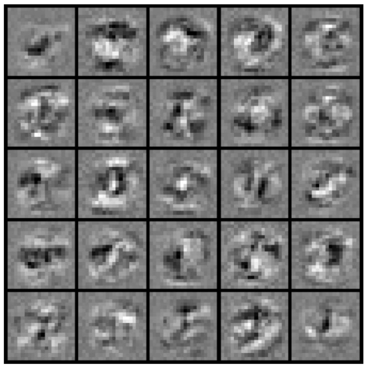
Report your optimal learning rate, regularization parameter, activation function, and the maximum training performance you obtained in your PDF writeup and README. The final `MLPDigits.py` you submit should either contain your optimal model and output the training performance; you do not need to submit your tuning code.

2.2 Visualizing the Hidden Layers (519 ONLY – 7 pts)

For the neural network you trained above, note that each hidden unit is governed by a 401-dimensional parameter vector. 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[1] the values of this matrix to lie in $0 \ldots 255$, we can visualize the weights as a greyscale image.

You may find it useful to consult http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py, which shows you how to extract the parameters of each hidden unit.

Modify your `MLPDigits.py` to 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 `visualizeHiddenNodesDigits.ppm`.



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.

[1] 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.

2.3 Digit Recognition via Deep Learning (20 pts)

Digit recognition, specifically using the MNIST dataset, remains one of the benchmarks for deep learning. In this exercise, you will train a deep net, examining the effect of different activation functions and techniques such as dropout. We will use a high-level interface to TensorFlow (https://www.tensorflow.org/), one of the main toolkits for deep learning. This high-level interface integrates well with scikit-learn.

Installing and Activating TensorFlow

Start by installing TensorFlow. If you are working on your own computer, follow the instructions listed at https://www.tensorflow.org/install. Please note as always, we cannot support installation problems on your own computer.

To install TensorFlow into your SEAS account, ssh into eniac or log onto another SEAS computer and issue these commands:

1.) Make sure you're running bash as your shell.

2.) Create a virtualenv environment. In this case we're creating the virtualenv environment as /tensorflow, but you can change this path if you want to install it to a different location.

```
virtualenv --system-site-packages ~/tensorflow
```

3.) Active the virtualenv environment

```
source ~/tensorflow/bin/activate
```

The prompt should change to `(tensorflow)$`.

4.) Upgrade pip

```
pip install --upgrade pip
```

5.) Install TensorFlow

```
pip install --upgrade tensorflow
```

These instructions pretty much follow https://www.tensorflow.org/install/install_linux, which has far more detail, so see it if you have any problems.

Once TensorFlow is installed, you must activate the virtualenv environment each time you want to develop using TensorFlow by re-running the command in Step 3 above. Once you are finished using TensorFlow, you can deactivate the virtualenv environment via the command `deactivate`.

Examine the Sample TensorFlow Script

The sample script `dnn_iris_custom_model.py`, provided in the skeleton, uses TensorFlow to train a deep net with ReLU (rectified linear unit) activation functions and dropout on the iris dataset. This sample is one of the examples provided with TensorFlow, but we have annotated it with additional comments to help you understand it. Read through it, and be sure that you understand how it works.

TensorFlow does not work exactly like a standard procedural language. The basic idea is that you construct a TensorFlow graph to represent your model (this is what the `my_model()` function does), specifying the different layers of the graph, and things like the loss function you will optimize (cross-entropy, in this case, which is the same as logistic loss), and the optimization method (Adagrad, in this case). Then, TensorFlow uses this graph during the training and prediction phases, handling things like automatic differentiation to compute the gradients for optimization, etc. TensorFlow is almost like a language onto itself, and we could use it to construct many different types of machine learning models. However, we won't go into the specifics of TensorFlow in this course. But, using the high-level TensorFlow interface, we can easily construct and train a deep net in a manner similar to scikit-learn.

Construct Your Own Deep Net

Building off of the sample script, create a new file called `DNNDigitsExperiment.py` to train a deep net for the digits dataset that we used before. Experiment with different activation functions (https://www.tensorflow.org/versions/r0.12/api_docs/python/nn/activation_functions_), including ReLU, tanh, and sigmoid. Also, experiment with varying the dropout rate at different layers (or eliminating it all together), and trying different network architectures (numbers of layers and hidden units).

Create plots showing how training performance varies for:

- different activation functions
- different dropout rates for different layers (519 only)
- different numbers of layers and hidden units

You should automate the experiment as much as possible in `DNNDigitsExperiment.py`. Although I recommend that you construct the plots within this script, you can use an external plotting program (e.g., Excel, etc.) if you wish.

Since `eniac` processes can only run for 20 minutes, I would highly recommend that you use `biglab` (`http://www.seas.upenn.edu/cets/answers/biglab.html`) or your own computer to run this experiment. Everyone enrolled in CIS 419/519 should already have access to `biglab`.

Based on your findings, figure out the best network architecture and parameter settings for the digits dataset. Duplicate `dnn_iris_custom_model.py` into a new file called `MyDNNDigits.py`, and modify it to train your best deep net on the digits dataset and compute its training accuracy. `MyDNNDigits.py` should only train one deep net.

In your PDF writeup:

- include the plots showing how performance varies, and write a paragraph summarizing your findings;
- describe the settings for the best deep net you found on the digits dataset; and
- report the training accuracy of your best deep net on the digits dataset.