CSE 398/498: Deep Learning

Programming Assignment 1

Due on Thursday September 14, by midnight

In this assignment, you will get familiar with the learning procedure for perceptrons and linear neurons. You will start with the example code "Perceptron.py" posted on course site, and test the code using "iris.txt" and "iris2.txt" datasets. Your tasks are listed as follows.

Task 1

1.1. Answer the question: in the original "Perceptron.py" implementation, what parameter-updating strategy is used to update the weights using weight derivatives? Choose among: Online (i.e. update after each training case), Full batch (update after a full sweep through the training data), Mini-batch (update after a small sample of training cases).

1.2. Do research on stochastic gradient descent (SGD). Write one or more paragraphs to explain what is SGD, how SGD is implemented, and the benefits of using SGD.

Task 2

2.1. Set up your own Python + NumPy + matplotlib programming environment. Useful tips:

- Install Python, SciPy (including NumPy), and a lot of other packages using Anaconda (https://docs.continuum.io/anaconda/install/)
- You can run a Python shell in your terminal, or you can install a Python IDE, such as Eclipse (https://eclipse.org/ide/) + PyDev(http://www.pydev.org/), Spyder (https://github.com/spyder-ide). Or, you can try out the web application Jupyter Notebook (http://jupyter.readthedocs.io/en/latest/install.html)
- If you were frustrated with buggy debugging tools in IDEs, you can always use print statements to check on variables, examine the behavior of your code, and much more.

2.2. Write a script to load in data "iris.txt", display the data points in a figure, import the *Perceptron* module, and call its *fit* and *predict* functions to learn a linear classifier for the data. Print out the error rate (or # of misclassified cases) for each iteration. Print out the converged weight vector for the classifier.

2.3. (Extra credits) Visualize the trained linear classifier's decision boundary (as a line) in the figure where data points are plotted. Make an animation to show the decision boundary after each iteration.

Task 3

3.1. Create three separate modules: Perceptron_online.py, Perceptron_fullbatch.py, Perceptron_minibatch.py, to implement the three parameter updating strategies---online, full batch and mini-batch, respectively. Alternatively, you can use one module, but multiple different *fit* functions for the module.

3.2. Write a script to test and compare the three modules above. Show the different convergence properties (e.g. by displaying the evolution of error across iterations)

3.3. (Extra credits) Write a script to display results when different learning rates are used. Try to adaptively adjust the learning rate based on error evolution and convergence behavior. Try to use different learning rates for different input dimensions.

3.4. (Extra credits) Modify the code to experiment with different initialization schemes for weight parameters. Compare results and summarize your observations.

Task 4

4.1. Write a script to load in data "iris2.txt". Again, import the *Perceptron* module, and call its *fit* and *predict* functions to learn a linear classifier for the data. Print out the error rate (or, # of misclassified cases) for each iteration, using full batch. Print out the converged weight vector for the classifier.

4.2. Do research on cross validation. Write a paragraph to explain what is cross validation and how to conduct a 10-fold cross validation.

4.3. (Extra credits) Implement a *k*-fold (*k*=3,5,or 10) cross validation using the iris2 dataset. Print out the resulting weight vector for each fold. Report the average weight vector for the *k* folds. Report the error rate (on validation data) for each fold. Report the average error rate for the *k* folds.

4.4. (Extra credits) Write a new module for a logistic neuron. Change the target (label) values for iris2 data: -1 change to 0.0, 1 change to 1.0. Use the logistic module to classify the revised iris2 data. Print out the error rate for each iteration, using full batch. Compare results from the logistic neuron to those by the linear neuron.

What to hand in

1. Your source code implementing all the required tasks.
2. A README file, to (1) answer questions, (2) summarize which scripts complete which tasks, and (3) briefly explain what you did if you implemented any extra credit requirements.

Your completed work should be submitted via Course Site.