CSC246 Machine Learning
Homework 4: Neural Networks

Overview

The purpose of this project is to give you experience with the backprop algorithm and feedforward neural networks for classification.

You are asked to complete an implementation of a multilayer perceptron, and to perform some experiments. To receive credit, you will need to submit your source code and a writeup (with figures and explanations, PDF format only, named `hw4_NETID.pdf` where NETID is your login) by 1159PM, Saturday, March 14th. Submit your code and writeup via running the ∼cs246/TURN_IN script on your assignment directory.

If you have any questions about the assignment, please post them to Piazza.

Datasets and Programming Languages

As with the previous assignment, your project must be completed using Python3 and Numpy. You are strongly encouraged to develop your solution using the Department of Computer Science instructional network, accessible via ssh: cycle{1,2,3}.csug.rochester.edu.

The starter code and dataset is available on the CSUG network at: ∼cs246/pub/neuralnets/

- `mlp.py` – the skeleton file for the MLP class.
- `train_mlp.py` – a script for training MLPs
- `test_mlp.py` – we will use this during grading
- `dataproc.py` – utility functions for data processing

The dataset you will use is similar to the previous assignment (plain text, one observation per line, features separated by spaces, label in the rightmost column). In this case, the target labels have been rewritten as being either 0 or 1 (for the perceptron assignment, we used -1 and +1). Multilayer perceptrons provide an extremely general purpose solution to classification problems, so your implementation should expect 1-hot encodings for the class labels.

Implementation Requirements

You are asked to implement a multilayer perceptron with an arbitrary number of hidden units, tanh activation for the hidden layer, and softmax activation for the output layer.

Complete the implementation of `mlp.py` by writing the methods for evaluating a network, calculating the gradient of cross-entropy error via backprop, and applying a single step of gradient descent.

Note that although you will be using softmax and cross-entropy, you should not write any code to directly calculate the derivatives of either of these functions. Recall that, assuming softmax outputs and cross-entropy error, you were asked to derive the following fact in Homework 3:

$$\frac{\partial E}{\partial a_k} = y_k - t_k$$

where the softmax function y_k is defined by

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j exp(a_j(\mathbf{x}, \mathbf{w}))}$$

and cross entropy error E is defined by

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

For further details, see sections 5.1-5.3 of the primary textbook.

Experiments

You should implement stochastic gradient descent for training with an adjustable learning rate and a batch size of one (i.e., process one sample at a time.)

For this project, we have sampled a small portion of a much larger physics-based dataset. Each observation has 28 floating-point features and a binary class label (0 or 1). The dataset has been partitioned into three subsets for training, development, and testing. You will not be provided the test data until after the submission window has closed; however, we will use it to grade your submission.

You should experiment freely with various learning rates and numbers of hidden units until you are comfortable with your results. As a sanity check, you are again provided linearSmoke and xorSmoke which should quickly converge to 100% accuracy with 1 and 2 hidden units, respectively, and a learning rate of 1e-1.

In machine learning it is important to develop robust proceses, so you are asked to discover and describe a training process which reliably produces good results. How would you characterize your best recommendations for training? At a minimum, this should include a specific learning rate, a number of epochs, and a number of hidden units. Additionally, you must describe the accuracy (on both training and development data) you observed using your method. You should also indicate whether your model always underfits or if you were ever able to detect overfitting.

You are welcome to experiment with alternative methods of initialization, stop conditions (e.g., by inspecting accuracy on development data), regularization (e.g., weight decay), non-constant learning rates, or momentum.

Grading

Your submission will be graded according to your implementation, your experiments, and your results. The training script by default saves a model to a file named "modelFile". You must submit your best trained model along with your source code and your writeup. We will evaluate your model on the datasets ourselves while grading your submission and a portion of your grade will be determined by your model's accuracy on the test data.

Your submission will be graded according to the following approximate rubric:

- 40% – program correctness (`eval`, `grad`, and `sgd_step`).
- 40% – experiments and writeup
- 20% – model quality