CSC246 Machine Learning
Project 2: Neural Networks

Overview

The purpose of this project is to give you experience with the backprop algorithm and feedforward neural networks for binary 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) by 1159PM EST, Thursday, March 18th. Your grade will be based on the correctness of your program, the quality of your writeup, and the quality of your final trained model.

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.

This assignment is being distributed as a zipfile via blackboard. It includes starter code and several datasets, described below:

- `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
- `analysis.py` – utility functions for analyzing performance
- `linearSmoke.dat` - a linearly separable dataset for testing
- `xorSmoke.dat` - the xor data from the previous project

- `htru2.train` - a training dataset
- `htru2.dev` - a development dataset

Implementation Requirements

You are asked to implement a multilayer perceptron with a single layer of (an arbitrary number of) hidden units with tanh activations capable of binary classification.

Complete the implementation of `mlp.py` by writing the methods for evaluating a network, calculating the gradient of softmax with cross-entropy loss via backprop, and application of stochastic gradient descent.

Specifically, you must complete the following methods from the starter code:

- eval - apply the network in a forward pass to a set of inputs
- grad - calculate the loss gradient via backprop and return
- sgd_step - apply a single step of stochastic gradient descent

Math Hints

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 and the lecture video and slides from February 16th.

Experiments

You should implement stochastic gradient descent for training with an adjustable learning rate and batch size.

You should experiment freely with various learning rates, numbers of hidden units, and batch sizes until you begin to see patterns in the results. More hidden units should *theoretically* increase the learning capacity of your network; however, in practice, increasing the number of hidden units can lead to more "violent" gradients which make learning more difficult and subject to the variance involved in intialization.

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. If you find that your network does not seem to successfully learn, it is most likely a mistake in your implementation of backprop (or possibly in the forward calculation.) One debugging trick is to implement a *finite differences* based calculation of the gradient, and compare your backprop results to the value returned by finite differences. For more information, see `https://en.wikipedia.org/wiki/Finite_difference`.

The simplest initialization method is to use uniform random sampling in the interval $(0, 1)$. A variety of more sophisticated techniques exist. For this project, you can simply use uniform initialization.

Regarding program time performance, the instructor's solution took approximately 10 minutes to process 200 epochs training with 10 hidden units.

Lastly, recall our earlier discussions of generalization vs overfitting. IN THEORY, if you choose a high enough number of hidden units, it may be possible to fit the training data with 100% accuracy; however, if there is any noise (and there likely is), then such an approach may overfit and perform poorly on the test data. In order to incentivize thinking about this, *the students with the top-5 highest accuracies on the test set will receive 5 bonus points.*

Dataset Specifics and Evalution

The dataset supplied with this project is a real astronomy dataset which is constructed by surveying the sky with a radio telescope, extracting features, and manually identifying pulsar stars. In theory, robust results on this learning task could aid astonomers in the search for more pulsars, helping to shed light on the structure of our universe. The data consists of 17898 observations of 8 numerical features, and an integer class label indicating whether the region of the sky in question contains a pulsar.

More information can be found here: `https://archive.ics.uci.edu/ml/datasets/HTRU2`

Do these numbers sound familliar? They should be! This dataset is challenge0.dat from the perceptron assignment, so you should already have some results. Your neural network should be able to reach a higher level of accuracy on this data than your perceptron implementation.

One unusual aspect of this data is that it is very *unbalanced*, meaning roughly 90% of the samples are negative, and only 10% are in fact pulsars. This means a program which simply outputs "no" without inspecting the data would achieve a classification accuracy of 90%!

The F_1 score is an alternative metric which is well-suited to unbalanced datasets. It is calculated as $2PR/(P+R)$, where P is the *precision* and R is the *recall*, defined by

- P = #true positives / (#true positives + #false positives)
- R = #true positives / (#true positives + #false negatives)

Effectively the precision identifies the average quality of a model's positive predictions, and the recall identifies how many true positives get "missed" by a model. Since a majority negative baseline would never predict positive class labels, on this dataset such an approach would have an F_1 score of zero - in contrast to an accuracy of 90%!

For 5 bonus points, you can modify your perceptron code to calculate the F_1 score include a plot of perceptron-based F_1 score over time along with your neural network results.

Writeup

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.

The ultimate goal in this project is that you learn about neural networks. We have constructed the assignment in a way that we believe will lead you to develop an intuitive, practical understanding of these powerful tools. In your writeup, you should include a "What I Learned" section which conveys, well, what you learned! There are no right or wrong responses (other than a blank one.)

In summary, the ideal writeup would contain the following:

- Your recommendation for an effective number of hidden units, learning rate, number of epochs, and batch size in order to solve this problem.
- Graphical results of training under the conditions you identified above. This should be in the form of plots of accuracy and F_1 per epoch on both training and development datasets.
- A "What I Learned" section where you identify any interesting observations or discoveries you made while working on this project. Have writer's block? Try starting by describing a bug that you fixed and how it impacted your process.

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

Revisions

1. March 10 - formulas for precision and recall were inverted. Fixed.