CSC246 — Project 2 – MLPulsars

Deadline: Sunday, March 13th by 1159pm.

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 multi-layer 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 the deadline. Your grade will be based on the correctness of your program, the quality of your writeup, and the quality of your results.

Datasets and Programming Languages

For this project the math is a bit more involved. You are strongly encouraged to consider working with Numpy and Python3. Matlab, Octave, Java, C, and C++ are also acceptable - but you should not assume any "frameworks" are allowed without checking first. My intention is that you take as a given certain matrix and vector primitives, but that you implement all the mathematics of backprop yourself.

This assignment includes some test data available on blackboard:

- `linearSmoke` - a linearly separable dataset for testing
- `xorSmoke` - the xor data from the previous project
- `htru2.train` - a training dataset
- `htru2.dev` - a development dataset

Basic Requirements

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

I recommend starting by designing an object to represent an MLP. In particular, you will need to represent the weights from the inputs to the hidden layer, and the weights from the hidden layer to the output. If you are careful, you can design your program in such a way that you can easily adjust the number of hidden units, the initialization method, the dimension of data inputs and outputs, and even the activation functions.

I strongly recommend writing a method to calculate the gradient of your loss function on an arbitrary set of training samples. You can then feed the results of your gradient method into one for implementing SGD (i.e., a separate method that updates the parameters and applies the learning rate, weight decay, etc.)

Lastly, in order to facilitate comparison across submissions, each submission must include an "inference engine" which is a program which loads your best trained model and classifies input from standard input, line-by-line. You should describe how to compile and execute your inference engine in your writeup.

Flourish Requirements and Collaboration

You may complete this project alone, with a partner, or in a team of up to four people. Individuals are not required to complete any flourishes. Teams of size two must complete at least one flourish. Teams of size three or four must attempt both flourishes. It is possible that these flourishes may be computationally very difficult to achieve. If you try and are unable to make it work, significant (and well documented) partial effort will be considered.

- Obtain meaningful results for 1000+ hidden units. Meaningful results mean at least a few epochs of gradient descent with a clear improvement in accuracy. In theory adding hidden units should increase accuracy (possibly at the cost of overfitting). In practice as you add more hidden units things become more difficult to train, so you may need to experiment with different learning rates.
- All the math we did in class covers the general case of arbitrarily many hidden units. Extend your work to support two layers of 100 hidden units each. This will also likely require tweaking your learning rate.

The Work

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 *theoret-*

ically 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 initialization. For this project, the key result I am asking you to find is the *best possible fit* using 100 hidden units and `tanh` activations. You will need to discover a reasonable learning rate and number of epochs to train on your own.

As a sanity check, you are 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`.

A simple (but effective) initialization method is to simply choose normally distributed reals centered around zero. A variety of more sophisticated techniques exist. For this project, you can simply use this simple initialization.

Through the previous project, you gained a deeper understanding of overfitting vs generalization. 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 practice, you may actually find that your model underfits the data. Nevertheless, you should use the supplied dev data to watch out for overfitting.

Data and Metrics

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 astronomers 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`

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%!

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 Problem Set 2:

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

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 Bishop book.

Writeup

In machine learning it is important to develop robust processes, 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 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. You should also include the total time required to run your algorithm.
- An explanation of how to use your program to train a new model, as well as instructions on how to compile and execute your inference engine using your best model. Clean and simple interfaces are strongly preferred.
- If you are working in a group, you must describe your attempted flourish(es), including what you tried and how well it worked. Additionally you should include a collaboration section which details the contributions of individual members of your team.
- A "What I/We 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% – train program effectiveness (i.e., efficiency and accuracy for backprop)
- 20% – inference engine accuracy and efficiency
- 30% – project writeup and results
- 10% – model quality – you must achieve performance significantly above majority baseline in order to receive full credit.