PyTorch Linear Classification:

By Joshua Feldman

In recent years, PyTorch has become a popular AI tool. There are essentially infinite ways to apply this python library, but for this report, I wanted to test some of its more basic functionalities and applications. One such use is linear classification, a tool which has many real-world applications such as diagnosing heart diseases. At its most basic level, binary classification is given a set of linearly separable data, trains on this data, and finds a method of classifying future data based on its training. Binary classification finds a halfspace in the data. If a new data point is above the halfspace, then the data point is positive. If the data point is below, then the data point is negative. This method, of course, is not foolproof as there can be problems with noise and faulty data, as well as computational limits. Methods such as Support Vector Machines can be used to compensate for these inaccuracies, but this only goes so far.

To test PyTorch, I decided to implement gradient descent with Mean Squared Loss, Cross Entropy Loss, and SoftMax loss. While implementing these functions, I created 100 two dimensional vectors in PyTorch, with their corresponding labels of either (0,1) or (-1,1). Then I would train each algorithm with this data. I arbitrarily set some parameters on the training, so that my algorithms would run in a reasonable amount of time, with a reasonable amount of accuracy on my machine. After some limited testing, I found that a learning rate of 0.01 was acceptable. To prevent the algorithm from running too long, I set an arbitrary breaking tolerance of 0.1, meaning once a loss of .1 was reached or surpassed, the algorithm would terminate training. The program I wrote has a functionality of graphing the loss rates over iterations for each algorithm.

A graph of different colored lines

AI-generated content may be incorrect.

As is evidently clear, the Least Squares Loss function performed the best, followed by SoftMax loss and Cross Entropy loss in terms of iterations. In terms of time, Least Squares-loss spent an average of 0.0037 seconds, SoftMax spent 0.1547 seconds and Cross entropy spent 1.4424 seconds of training per 100 vectors. Therefore, if speed is the only concern, least-squares loss is the best.

I then modified the program to train and test each algorithm 100 times, and find how many values were misclassified. On average, Least Squares Loss misclassified .11% of all values, while the other two algorithms misclassified a much better .02% of all values. While this might seem like a small difference in terms of values, in the real world, speed might not be the priority for accuracy. When it comes to classifying heart disease, decimal points could mean the difference between life and death. There are many reasons why the least square would be less efficient. One reason is that it is quite sensitive to outliers. Another is that it assumes that all values are continuous and does not consider the probabilistic nature of data.

To thoroughly test PyTorch, I also implemented a perceptron algorithm. Perceptron’s are a vital part of machine learning, as they are a vital part of neural nets. My perceptron algorithm took a similar data set to the first three algorithms, and plotted the halfspace of the data correctly, but did not use any aid such as a Support Vector Machine. As this perceptron only had to deal with 100 points, after running 1000 times, misclassified 0 times, with an average training time of 0.036696 seconds, making it both faster and more accurate than all of the other three algorithms.

A diagram of a line with red and blue dots

AI-generated content may be incorrect.

To compare PyTorch to another common AI tool, I used SciPy to find a halfspace of some data. Scipy linprog uses a simple algorithm known as a simplex to find a halfspace of some data, similar to a perceptron. While both algorithms were able to correctly label all points in every case, SciPy did it in .039 seconds, which is significantly slower than my perceptron. This is odd as a simplex does not gradually change weights as a perceptron does, but rather finds pivots to change weights to maximize or minimize a function, so it should run faster. Therefore I can conclude that PyTorch is an excellent tool for machine learning.