SGDBabysitter

Joshua Catalano – 28675650

Abstract

We created a stochastic gradient descent algorithm that dynamically adjusts learning rate and batch size. To make decisions, the algorithm considers the change in validation error over time and the angle between gradients. We're hoping to exploit our knowledge of the behavior of stochastic gradient descent in order to create an implementation that attains lower validation errors. We consider our algorithm a success if it can attain a lower validation error than the best constant learning rate or learning rates determined by a predetermined series of numbers (e.g. $\frac{1}{t}$ or $\frac{1}{\sqrt{t}}$). In order to test our algorithm, we use it to fit a basic neural network to various real and generated datasets.

Introduction

As we learned in class, Stochastic Gradient Descent (SGD) is a popular optimization algorithm in Machine Learning that allows for fitting various machine learning models to massive datasets. It can be used to fit logistic regression, ordinary least squares, neural networks, and more. While SGD is less computationally costly than gradient descent, it demonstrates unsavory behavior due to the fact that it assesses the gradient using a random subsample of the data. We constructed our own unique implementation that accounts for this strange behavior in the hopes of improving the performance of models that are fit using SGD.

Making a more accurate implementation of stochastic gradient descent is important because it could be used to fit more accurate machine learning models. Since many academic disciplines and the private sector use SGD to fit machine learning models, a more accurate implementation could improve the performance of prediction in all types of applications.

Theory

In order to create this algorithm, we utilized the concepts we learned in class. In this section, we will discuss our understanding of the theory.

When far away from the minimizer, stochastic gradient descent, with an appropriate learning rate, moves us closer to the minimizer with high probability. As it gets closer to the minimizer, the angles between the gradients assessed at all the different values of X become wider and wider. This increases the probability that the next iteration will actually move away from the minimizer. In class, we called this phenomenon the "ball of confusion."

The size of this ball of confusion is a function of the learning rate. The smaller the learning rate, the smaller the radius of this ball. Starting with a very tiny learning rate, however, can make the algorithm take far too long. On the other hand, having too large of a learning rate can give you divergent results. So ideally, the learning rate would be large in the beginning but just small enough so that the algorithm doesn't diverge. Then as we approach the minimizer, this learning rate should decay so that we can get closer to the minimizer within the ball of confusion.

When selecting batch size, the trade off is between lower computation costs and a more accurate estimate of the gradient. When far away from the minimizer, a small batch size is ideal because the

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

angle between the gradients evaluated at an of the X values should be relatively small, so taking an average of gradients is less beneficial but still increases computation cost. As we approach the minimizer, the angles between these gradients increase, and averaging will be more beneficial as more gradients are likely to be pointing in the wrong direction.