A Method for Detecting Convergence of Machine Learning Algorithms with Cauchy-Style Estimates

Galen Novello

August 22, 2017

Abstract

In this paper I propose a method for detecting the convergence of the method of learning algorithms with "Cauchy style" estimates. Brief overviews of background information including Cauchy sequences and various learning algorithms are provided. As an explicit example, the method is applied to the convergence of gradient descent, the method of max DJ is presented and it is conjectured that this learning algorithm has converged on iteration k if $\sum_{p=k-2n}^{k} \left(\max_{i\in\{0,\ldots,n\}} |DJ_{i_p}|\right) < 1$. Empirical evidenceleading to this conjecture is presented. It is shown how the method of max DJ is then applied to learning algorithms whose solutions must be minimized over non-convex sets and this extension is applied to a neural network.

1 Cauchy Sequences

Cauchy Sequences are a well known tool for analyzing convergence of sequences. Most people first encounter them in the context of real numbers in an introductory course on real analysis or topology but the equivalence of being a Cauchy Sequence and being a convergent sequence is so important that it gets its own name and we call a topological space with such an equivalence complete. You can find rather exotic examples of complete spaces in the literature, but for the purposes of this conversation it will suffice to point out that closed metric spaces, and in particular $\mathbb{R}^n$, are complete.

A sequence θ_i is said to be a Cauchy sequence if for any $\epsilon > 0$ there is some N for which $|\theta_i - \theta_j| < \epsilon$ for all $i, j > N$. The idea of this paper is to adjust this definition in two ways to create a method to detect the convergence of the coefficients of gradient descent

- By changing the condition "for all $i, j > N$" to "for all i such that $k - k_0 < i < k$ " where k is the index of the most recent iteration and k_0 is a "window size" that depends on the number of variables in the model.
- By finding a bound on ϵ that will ensure (at least a high probability of) convergence of the model.

2 ApplyingThe Method to Gradient Descent

2.1 Gradient Descent

The method of gradient descent is a regression algorithm that works to minimize a cost function by using the gradient to iteratively adjust the model. A good introduction to the method as well as an outline of it's justification can be found in open source resources from Andrew Ng's Machine Learning Course, and it was these materials which guided me to creating the gradient descent algoritm used in this paper.

To produce a model for quantity of interest y as a function of variables $x_1, ..., x_n$ the method seeks coefficents $\theta_0, \theta_1, ..., \theta_n$ of the line

$$y = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n.$$

The error of a model $\Theta = (\theta_0, \theta_1, \ldots, \theta_n)$ is given by the cost function

$$J = \frac{1}{2n} \sum (y - X \cdot \Theta)^2$$

where the sum is taken over all points $(y, x_1, x_2, \ldots, x_n)$ in training data set and for each point we set $X = (1, x_1, x_2, ...x_n)$ for notational convenience.

On each iterateration the method takes a hypothesis $\Theta = (\theta_0, \theta_1, \ldots, \theta_n)$ and computes the gradient of the cost function of using this hypothesis, DJ. A new hypothesis is then produced by taking $\Theta_{\text{new}} = \Theta - \alpha DJ$ where $\alpha \in \mathbb{R}$, known as the step size, is a suitable constant. In this way a sequence θ_{i_k} is produced for each coefficient θ_i and it is clear that convergence of the model is equivalent to convergence of each of these sequences pointwise.

2.2 Initial Data, visual estimates of convergence time for different numbers of variables.

2.2.1 How pseudo random data for these tests is generated.

Convergence time of the method of gradient descent is affected by the spread of the data and the step size α. In fact, the method can diverge if α is chosen to be too large relative to the spread of the data. Data is generated for these test by first generating a vector, M, of constants uniformly from $[-1, 1]$ - this is the target function, the "answer" that the method will try and reconstruct.

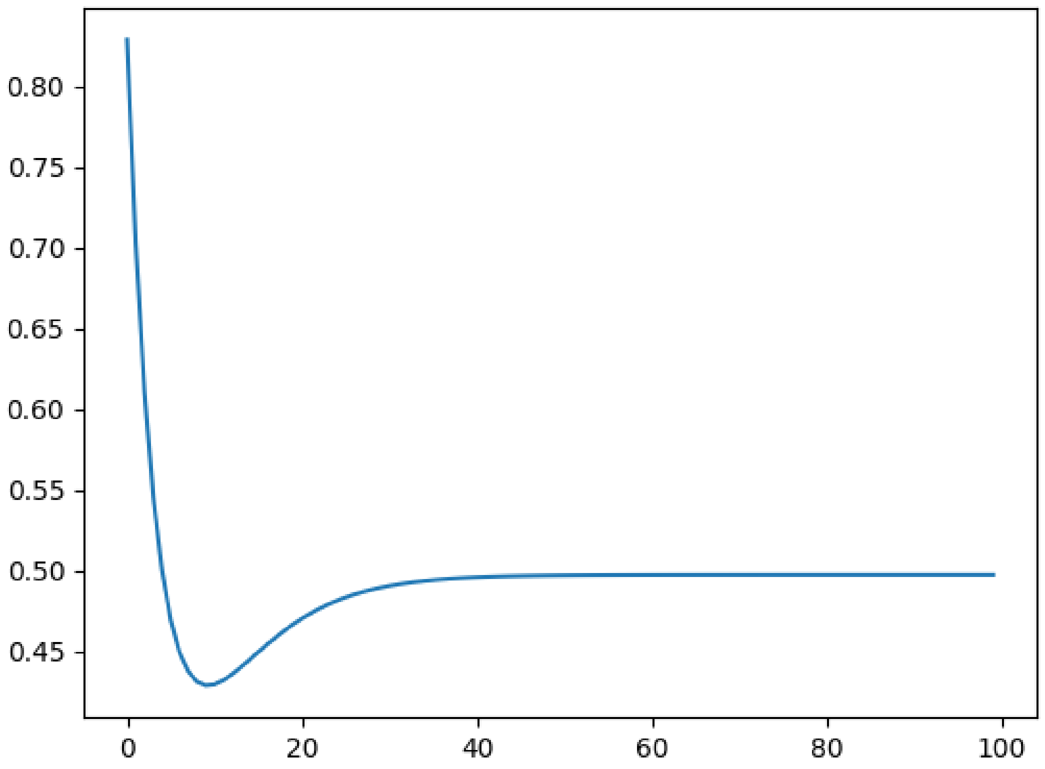
Each data point $(y, x_1, x_2, ..., x_n)$ is then generated by picking each x_k uniformly from $[-1, 1]$ and computing y through the formula $y = M \cdot X$ where $X = (x_1, ..., x_n)$.

Convergence of the algorithm is affected if the interval $[-1, 1]$ is changed in either case. It is the suspicion of the author that the ratio of the lengths of these intervals is relevant to choosing the step size α, and perhaps it is worth noting that in this case that ratio is 1.

A set of 500 data points is generated in this way, 50 of them are removed for a final test set and the remaining 450 form the training set for the data. α is choosen as .01.

2.2.2 Data on the rate of convergence at randomness level blank for different numbers of variables

The convegence of the algorithm can be visualized by plotting the values of the θ_i vs the number of iterations. For example, the convergence of θ_1 for a 9 variable gradient descent algorithm is given



With this particular model converegence can further be determined by the target function to the model which in this case gave 7 digits of accuracy after 100 iterations ($\theta_{real} = 0.48810335753783995$, $\theta_{predicted} = 0.48810336800164367$). From the graph it is also clear that almost as good an approximation would have been obtained with half as many (or fewer) iterations.

Running 30 tests for each number of variables from 2 to 11 and analyzing the convergence of θ_1 (the algorithm is symmetric in θ_i so there is no harm in this) leads to the following estimates for the number of iterations by which we expect convegence to begin.

# of vars	estimate
2	does not converge with this alpha
3	10
4	12
5	15
6	20
7	25
8	30
9	35
10	40
11	40

These are, of course rough estimates, based on a small sample, but they appear to indicate a pretty linear growth and lead to the initial hypothesis to look for convergence after $4n$ iterations where n is the number of variables.

Another look at the graph shows that the distance between the local min on the curve and the value to which it converges is less than .1. This was true for all 270 models that converged, so we expect that if all points start to tend to be within .1 of eachother the curve is on its way to convergence. In this case α is .01 so our bound of .1 can be though of as 10α. We give a silightly more conservative initial estimate: that we can choose ϵ to be 5 α.

Putting it together we conjecture that the method of gradient descent has converged on iteration k if

$$\forall\ k - 4n \leq j, m \leq k, \quad |\theta_{i_j} - \theta_{i_m}| < 5\alpha$$

2.3 The Cost of Implementing the Condition

To implement this condition it will at worst be necessary to keep a matrix of the n+1 values of each coefficient over 4n iterations and checking them which means the cost of implementing the proceduce is on the order n^2 checks per iteration in particular this is constant with respect to the size of the training set. Each iteration of the gradient descent method requires computations at least on the order of n times the number of data points in the training set, so unless the number of variables is greater than the number of data points (which probably wouldn't converge anyway), implementing the test required above will lower the computational cost of running the algorithm (significantly).

2.4 Empirical Data

To gauge the effectiveness of this method to detect convergence, 30 more simulations for each number of variables from 3 to 11 were performed, and the result of the Gradient Descent after 100 iterations and after the Cauchy cutoff were compared. On average the theta from the Cauchy estimate agreed with the real value of the coefficient to within .0001 for low numbers of variables and .001 for larger. The following table gives an idea of the averge number of computations for caucy convergence and to what accuracy:

# of vars	average iterations to converge	Cauchy accuracy
3	15	.0001
4	21	.001
5	27	.001
6	33	.001
7	38	.001
8	45	.001
9	49	.001
10	55	.001
11	65	.001

The accuracy of the coefficients from 100 iterations continued to be within .00001 at 11 variables. So there was a loss of a couple places of accuracy, but about a 33% savings in computation. Probably not a bad trade off in most situations.

The entirety of the data referred to above as well as the source code used to generate it and a copy of this paper can be found on GitHub .

2.5 The method of max DJ

Recall that the method of gradient descent produces a new hypothesis by taking

$$\Theta_{\text{new}} = \Theta - \alpha DJ.$$

Formally one may write $\Theta_{j+1} = \Theta_j - \alpha DJ_j$ and $\theta_{i_j} - \theta_{i_{j+1}} = \alpha DJ_{i_j}$. Furthermore,

$$|\theta_{i_j} - \theta_{i_m}| = |\alpha \sum_{p=j}^{m-1} DJ_{i_p}| \leq$$

$$\alpha \sum_{p=j}^{m-1} |DJ_{i_p}| \leq \alpha \sum_{p=j}^{m-1} \max_{i \in \{0,\ldots,n\}} |DJ_{i_p}|.$$

Then we can reformulate the satement in section 3 and suspect convergence if

$$\alpha \sum_{p=k}^{k+4n-1} \max_{i \in \{0,\ldots,n\}} |DJ_{i_p}| < 5\alpha$$

A more elegant refinement of the conjecture is that the method of gradient descent on n variables has converged on iteration k if

$$\sum_{p=k-2n}^{k} \left(\max_{i \in \{0,\ldots,n\}} |DJ_{i_p}| \right) \; < \; 1$$

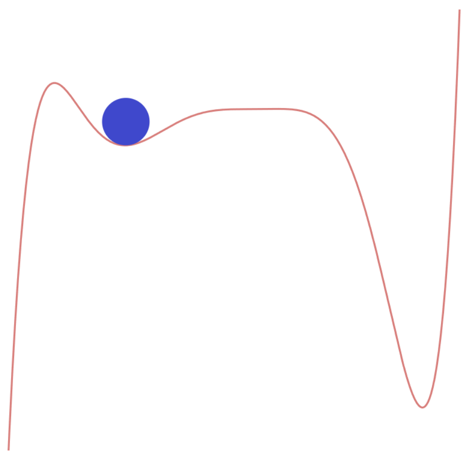


Figure 1: Getting Stuck at a Local Minimum

2.5.1 Computational savings of max DJ method

To implement the max DJ method the algorithm must perform at most n checks on the DJ vector to obain the max and then a set amount of operations to maintain a list of the most recent DJs and replace values in the current sum. In particular this is linear with respect to the number of variables and still constant with respect to the size of the data set.

3 Applying the Method to neural networks

The effective of this method on gradient descent begs the question of wheather it can be applied to more complicated learning techniques. Happily the answer is yes, with a slight modifaction. The major complication comes from the fact that linear models produce convex sets over which a minimum must be obtained, while more general methods, such as neural networks, produce with local minimums. One of the challenges in maximizing the effectiveness of a learning method is to avoid gettting caught in a local minimum. In order to avoid this, the idea is to perturb the values of the weights when a minimum is obtained. One way of doing this that has been suggested is to as a bit of randomness to each iteration in hopes of "jiggling" your way out of a shallow local min. This issue with this is how much to jiggle, too much and you fly out of the local min, not enough and you never escape. The method of max DJ, when applied to the weights of the neural network can be used to instead smoothly (and thus more quickly) descend upon a local minimum and then make an informed bump.

3.1 Neural Networks

There are many different types of neural networks, cost, functions, and methods for training them, but all neural netowrks consisst of layers of nodes, with each node having it's own weight. Each weight can change on each training example, so in principal, each weight should carry 3 indices : $w_k l j$ where k is the iteration number, l is the layer and j the position within the layer. In a similar fashion to the convergence of gradient descent, the convergence of a neural network essentially amounts to the stabilization of these weights. One major difference that weights are updated after each training examples instead of an entire epoch. As such, the change in a weight after an iteration is no longer dictated by the graditent of the cost function, but computed through different methods (which depend on the specific type of neural network being implemented.

3.2 Tweaking Max DJ for NNs

In some ways, the method needs very little tweaking: We can still make a vector DJ_k whose components are the changes in each weight after iteration k and we can still consider the sum

$$\sum_{p=k-2n}^{k} \left(\max_{i \in \{0,\ldots,n\}} |DJ_{i_p}| \right)$$

where n is the total number of weights. When this sum is small enough the algorithm has found a local minimum.

As mentioned above the interesting part is to avoid getting caught in a local minimum. When a local minimum is found we will store that set of weights as a hypothesis h and then vary the weights appropriately in hopes of climbing out of a local minimum. The idea move away from the direction which has been changing most quickly in the weight space. So suppose that we have decided that $\sum_{p=k-2n}^{k} \left(\max_{i \in \{0,\ldots,n\}} |DJ_{i_p}| \right)$ is small enough and we have reached a local minimum, h_t, we can construct the vector $DJ_{h_t} = \sum_{p=k-2n}^{k} DJ_p$. The coordinates of this vector are simply the sum in the changes of each weight. We can now perturb h by changing weight w_i by $ce^{-|DJ_p^i}$ where c is an appropriately chosen constant. In this way we move away from the direction in which we were decending most steeply and are more likely to "get over the hill" from the new starting point from which to decend we have created $H_{0_t} = h_t - e^{DJ_{h_t}}$. If we find T such that $h_T = h_{T+1}$ then we have generated a possible set of best hypotheses $h_1, h_2, ...h_T$ which can then be tested against validation data to determine which is most effective. It should be noted that this list may not be exhaustive of all local minimums and it will probably still be valueable to reiterate this whole process from a different initial hypothesis.

4 Proposals for Future Research

- Refine Estimate of computation savings
- Provide analytic justification/refinement of estimates for k_0 and ϵ.
- Make precise connection between size of alpha, intensity of randomness, and convergence