A Fast Algorithm for Neural Networks

Neural network "training" is the problem of determining network parameters by minimizing non-convex cost functions under severe computational limitations. This is a difficult problem that requires heavy computational effort.

I am an expert in the field of numerical minimization algorithms. I hold B.Sc., M.Sc., and D.Sc. degrees in Applied Mathematics from the Technion, Israel Institute of Technology. I worked for more than 5 years on (military) applied mathematical algorithms before completing my graduate degrees in mathematical optimization, and I have held positions at the University of Texas at Austin (Mathematics), York University (Business), Dalhousie University (Business), the Technical University of Nova Scotia (Director, School of Computer Science), and Dalhousie University (Industrial Engineering). My proprietary algorithm, B67, is the result of a theoretical breakthrough after more than 35 years of work on this problem. This fast algorithm is not a heuristic, nor a variant of existing algorithms.

The problem of minimizing a neural network's cost function is difficult. In general, minimization algorithms may not converge, may converge to a local minimum, or to a saddle point (a multi-dimensional inflection point) that is neither a minimum nor a maximum of the cost function. See the *Notes on Optimization* in the PAFA paper at http://scientificmetrics.com/publications.html

Algorithms that use second order derivatives cannot be used for large minimization problems. As a result, the only possible algorithms *in the literature* are the Steepest Descent or the Conjugate Gradients algorithms. The Conjugate Gradients algorithm works well on quadratic, convex, non-singular problems, but the training problem is none of the above. Here the Conjugate Gradients method does not terminate in a finite number of steps, the gradients are not conjugate, and the method may fail because division by zero may occur. It follows that the only applicable algorithm is the steepest descent method. This method requires a line search to determine a step size (how far to go in the [negative] gradient direction). The generally poor performance of this method is compounded by the use of fixed step sizes which is dictated by the high cost of line searches for the training problem. To emphasize, for this problem there is no alternative *in the optimization literature* to the use of the steepest descent method with a fixed step size.

A Computational Example

- An important practical sub-problem of computer handwriting recognition is the problem of recognizing the ten digits 0-9. For example, cheques' numerical amounts are "read" by banks' computers rather than by humans. The standard database for testing the performance of neural networks that recognize numerical digits is the MNIST database (Modified National Institute of Standards and Technology database) that contains 60,000 training images of handwritten digits.

- The simplest problem of this kind is to compute the 784 weights of a network that recognizes a single digit, with no intermediate ("hidden") layers, and no non-linear functions. To compute these weights it is necessary to minimize a so-called "cost function" of the 60,000 training images. The standard method of solving this minimization problem is the steepest descent algorithm with a fixed step size.
- For this problem, the steepest descent algorithm diverges for step sizes 0.1 up to 0.00001 and converges for the step size 0.000001. With this step size, after 10 million iterations of this algorithm, taking 1374 seconds (more than 22 minutes), 3 precision-solution-digits are attained while my B67 algorithm yields the solution at this precision in less than 0.092 seconds – that is, my algorithm is about 14900:1 faster.
- Both algorithms were implemented in the same version of Matlab and executed on the same computer with an initial estimate of equal weights.

Similar behaviour is observed on large-scale problems. The reason is this: the total computational effort depends on the effort *per step* and the *number of steps* needed. The effort *per step* for both algorithms does depend on the problem's size, but this effort is the same since both algorithms require the computation of the gradient of the cost function in each step. The *number of steps* depends on the algorithm's rate of convergence which is independent of the problem's size (the rate for Newton's method is 2 regardless of size; other algorithms depend on the problem's condition number which, again, is independent of size). For this reason my algorithm's fast convergence results in savings of orders of magnitude in computational effort for the training problem *regardless of its size.* This also means that it is easy to construct very small neural networks of the simplest structure for which standard training algorithms take an exceedingly large number of steps to converge.

One of my companies owns this intellectual property and my preference is to sell this company. Large corporations and smaller ones, invest heavily in Artificial Intelligence (AI) technology. They would assign a high value to the IP, deploy it in-house, and prefer not to see it in the hands of their competitors. The implementation of minimization algorithms is straightforward and they do not require user interface. No further development is needed, there is no need for a team of developers, and no need for investment (and one does not need to build an entire car to demonstrate a better battery). The IP can be sold *as is at this time.*

To re-emphasize, regardless of which neural network model is employed, network training is a minimization problem. My minimization algorithm represents a major competitive advantage to a potential buyer.

Technical Notes

- Neural network training requires a great computational effort to *minimize* a cost function. Our minimization tool is much more powerful, and much cheaper, than existing tools.
- The computational effort is reflected in the number of steps needed to minimize the cost function. The number of steps depends on the problem's structure, *not on its size*. (For example, the rate of convergence of Newton's method is 2 regardless of the problem's size.)
- Even if only linear functions are used, training a deep network amounts to determining the coefficients of a high-degree non-convex polynomial in many variables.
- In general, the steepest descent algorithm's convergence characteristics are poor and with a fixed step-size it is very poor. This is particularly the case when the cost function is ill-conditioned which may be the case even for a quadratic cost.

Jonathan Barzilai, D.Sc.
Professor
Dalhousie University

barzilai@ScientificMetrics.com
T: +902-444-3454