InFoMM
Industrially Focused
Mathematical Modelling

UNIVERSITY OF
OXFORD

Optimisation Challenges in Machine Learning

Constantin Octavian Puiu

Numerical Optimisation is the study of algorithms which find an approximate solution to a minimisation problem. In plain English, a minimisation problem is one where we are trying to find the lowest point of a given function. A real-world example of this is imagining being on a mountain, blindfolded, and given only some very local information about landscape (such as the slope, the height, and possibly the local curvature), we need to take steps in such a way as to reach the lowest valley.

Machine Learning (ML) is the study of computer algorithms which aim to gain generalization abilities after having learned on a set of data. In plain English, generalization abilities mean that they can (almost always) correctly predict an ouput for a given input. A typical example of this is a classification task: labeling an image as a dog or a cat. The process of learning involves using *optimisation algorithms* to find *the parameters of the model* which minimise the *loss*. The *loss* essentially tells us how far off we are from correctly mapping every single given input data point to its corresponding output. The *parameters* can simply be thought of as a collection of real numbers. If we order this collection, we then refer to it as *the parameter vector*, and its *dimension* is the number of real numbers it contains. This relates to the *mountain-example* that we have previously seen in the following way. The height of the mountain at each particular point represents the *loss* for a particular value of the *parameter vector*. The horizontal coordinates represent two different parameters. Because we can only see three dimensions, we can only visualize the loss when the dimension of the parameter vector is 2. However, in practice, the dimension is much larger.

Minimising the *loss* is computationally cheaper if this is *convex*. Loosely speaking, the loss is convex if its associated "*mountain*" looks like a paraboloid which points downwards, no matter which pair of parameters we select out of our *paramater vector* (see Figure 1). Deep Nets are a particular case of ML models which mathematically construct an ensemble of neurons. In the case of Deep Nets, the *loss* is NOT *convex*, making the optimisation procedure much more complicated. The complication is further exacerbated by the fact that for Deep Nets, the *dimension* of the parameter vector can be in the order of 1 million, which is much larger than for other ML models. We focus on Deep Nets here.

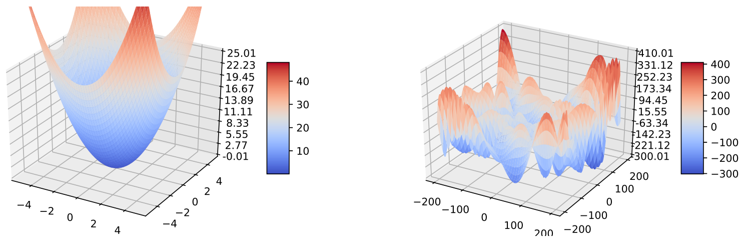

Figure 1 – Convex Function (Left) vs Non-Convex Function (Right)

Notion of Budget

We have seen that optimisation algorithms use local information to minimise the *loss*. However, obtaining this local information comes at a computational cost (waiting time), which can make the algorithms very slow in practice, up to the point where they become infeasible. Furthermore, in ML, the slope, height, and curvature are "*measured*" at each location with relatively high error. We can reduce this error by increasing what is known as the *batch-size* (n), but this comes at increased computation cost. We define the notion of **budget**, which is a measure of computation time. For a *batch-size* of n and parameter dimension d, it takes us nd, $2nd$, and $4nd^2$ units of budget to compute the height, the slope, and the local curvature respectively.

First and Second Order Methods

First Order Methods (FOM) use only height and slope information to perform a step. These have been the state of art in Deep Nets due to their cheap cost per iteration (budget cost of $3nd$ per iteration;

Engineering and
Physical Sciences
Research Council

as they do not require computing the curvature). The most popular algorithms are *Stochastic Gradient Descent (SGD)* and *Adaptive Momentum (ADAM)*. ADAM tends to outperform SGD in practice because it ingeniously uses the slope information to infer approximate curvature information.

Second Order Methods (SOM) also employ *curvature information*, on the top of height and slope. Because of this, each iteration offers much more progress than an iteration of a *first order method*. When d is low, and the local information can be accurately measured, SOM outperform FOM in terms of computational time. However, in the case of Deep Nets, d is so large that computing the curvature is infeasible (as $d^2 \gg d$). Thus, approaches in the literature generally resort to computing what is known as *Hessian-vector products*, which essentially give the curvature in only one direction, at a budget cost of $4nd$. The *Hessian-vector products* can be used to reconstruct the curvature in a chosen subspace. This essentially means that we only obtain part of the curvature information at a significantly reduced cost. The idea is that if the subspace is selected ingeniously, then we could achieve the same progress that we would if we had the complete curvature information, but at a much more reduced cost. If we managed to do this, then SOM would outperform FOM. However, selecting such a subspace is far from trivial, and represents our main challenge here. This challenge is further aggravated due to the *loss* landscape not being convex.

Our Proposed Algorithm and Results

The algorithm we propose is a Trust-Region (TR) algorithm. A TR algorithm is one which builds a model of the landscape at the current location based on the slope and (subspace) curvature, and uses it to take a step. The model is only “trusted” within a ball, whose radius is updated at each iteration based on the progress achieved.

Our algorithm draws upon the ones in the literature, but has the following differences: (1) the dimension of the subspace is much smaller: 3 - 10 (it is 200 in the literature); (2) The subspace selection is based on the cheaply available ADAM step; (3) The first order information used in the TR model is, loosely speaking, averaged over the past few values (unlike just the local one as is typical in the literature). Point (1) is meant to save on computation cost. However, because the subspace is of smaller dimension than typical, it is harder to select it to get meaningful second order information. By meaningful second order information we mean information which will improve upon the progress made when just first order information is employed. If the second order information is not meaningful, then we are essentially wasting budget. Point (2) is meant to improve the subspace selection and thus compensate for the drawbacks of Point (1). Point (3) is meant to help navigating potential ripples in the landscape better. We name our proposed algorithm *NLTR*.

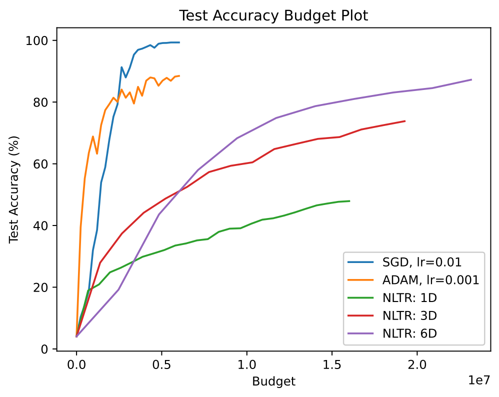

Figure 2 – ***NLTR vs SGD/ADAM: Varying the Subspace Dimension.***

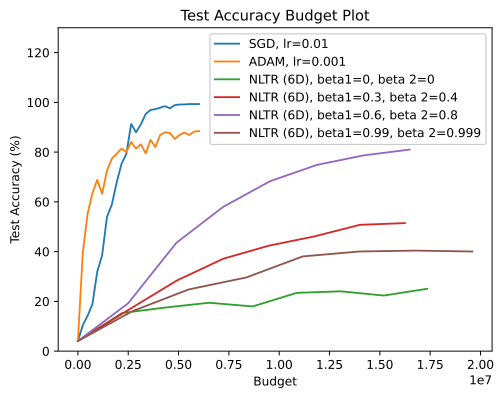

Figure 3 – ***NLTR vs SGD/ADAM: Varying the Momentum Parameters.***

In Figures 2 and 3, we see plots of *test-accuracy* vs *budget* for *NLTR* with different parameter values, compared against the state of art: ADAM and SGD. We see that for best parameter choices, *NLTR* underperforms SGD and ADAM, being about 8 times slower. This suggests that in order to make SOM competitive, we must come up with a better subspace selection technique. NLTR does however outperfom other approaches in the literature, who are typically 10 times slower. Finally, we believe we can enhance our method using concept of "momentum" with TR and this will be further investigated.

Mr Jan Fiala, *Product Manager for Optimization)*, at NAG said:

> *“Training Deep Nets is a highly nontrivial problem which attracts a lot of attention all over the world and any improvement will have a significant impact. We are very proud to take part in this research and we are excited about the progress Constantin has made so far. Success doesn’t come easy and it will be our pleasure to explore this topic further during the research project.“*

Engineering and Physical Sciences Research Council