Saddle-Corrected Naïve Krylov Subspace Method

Jeffrey Negrea
Department of Statistical Sciences
University of Toronto
Toronto, Ontario
`negrea@utstat.toronto.edu`

Yuxiang Gao
Department of Statistical Sciences
University of Toronto
Toronto, Ontario
`ygao@utstat.toronto.edu`

Abstract

We propose an optimisation algorithm based on Krylov subspaces and hessian-vector products and provide an implementation of the algorithm using Python and `Autograd`. The algorithm is compared to `ADAM` and we find that it is less efficient in the time domain but more efficient in the epoch domain. We propose possible efficiencies that may be used to improve the algorithm in later work, and ways in which the additional information produced by the algorithm may be used to provide visualisation of the objective function locally at each training iteration.

1 Introduction

The loss surface of a deep neural net is highly complicated, with ravines, saddle points and local minima proliferated across the parameter space. Different network architectures and data sets often require the tuning of metaparameters and/or use of different training techniques to achieve the desired performance. The current practice in training deep nets is to use first order descent methods that using a noisy estimate of the gradient. Stochastic gradient descent (`SGD`) and `ADAM` [KB14] are examples of such first order stochastic optimization techniques. A common problem for first order methods is that critical points are usually surrounded by plateaus of small curvature ([DPG$^+$14]), and first order stochastic optimisation methods tend to under perform in such scenarios.

Competing methods, in comparison to first order stochastic optimization methods, are Newton or Quasi-Newton methods. Such approaches require calculation of the Hessian matrix of the loss function with respect to the parameter vector, and in high dimensions this calculation becomes infeasible. In this work, we'll derive an algorithm based on the fast implementation of the Hessian-vector product in `Autograd` [MDJA15]. We form a basis for the Krylov subspace generated by the Hessian and the gradient of the loss function at each iteration that will be used to approximate the loss function and the action of the Hessian on a lower dimensional subspace of the parameter space. The use of Hessian-vector products avoids directly calculating or approximating the full Hessian of our objective function during each step.

2 Related Work

Previous work using Krylov Subspace methods involved approximating $(H + \lambda I)^{-1}$ where H is the Hessian of a neural net [VP12] and this was further extended into an approximate saddle-free Newton's method [DPG$^+$14]. It's worth noting that our derived algorithm is very similar to the approximate saddle-free Newton's method proposed by Dauphin *et al*, however the motivating ideas that drive our derivation are different from Dauphin's. Our approach stems from the idea that we're interested in approximating the action of the Hessian using a Krylov subspace basis for computational efficiency, whereas Dauphin's work was originally motivated by the desire to construct a Newton's method variant that would escape saddle points with high probability, then defaulting to an approximation based on Krylov subspaces as the derived method is intractible for real problems.

Another key difference is that in [VP12] and in [DPG$^+$14] the optimisation within the subspace is carried out using an auxiliary optimisation routine applied to the true loss function, while our method takes steps which exactly optimise the approximating quadratic on the lower dimensional subspace.

3 Methodology

The core idea underlying the algorithm is described in subsection 3.1. The actual implementation builds upon this to improve numerical stability and correct for non-convex objectives. The tools needed to achieve this are described in the subsection 3.3.

3.1 Theoretical Motivation

Consider an objective function to be minimised, $l(\theta)$. At any point θ_0, one may consider the second order approximation to this function:

$$l(\theta_0 + h) - l(\theta_0) = g'_{\theta_0} h + \frac{1}{2} h' H_{\theta_0} h$$

Where g_{θ_0} and H_{θ_0} are the gradient and hessian of l at the point θ_0 respectively. These will be abbreviated by g and H respectively from here onwards.

The Krylov Subspace of order $r + 1$ generated by g and H is the space

$$K_{r+1}(g, H) = \text{span}(g, Hg, H^2 g, ..., H^r g)$$

Newton's method aims to minimize the approximating quadratic

$$\tilde{l}(h) = g' h + \frac{1}{2} h' H h$$

exactly at every step, which requires inverting a $p \times p$ matrix (where p is the dimensionality of the parameter space). We instead search for the optimal $h \in K_{r+1}(g, H)$ for $\tilde{l}$ which will only involve inverting an $(r + 1) \times (r + 1)$ matrix.

Let $V = [g, Hg, H^2 g, ..., H^r g] \in \mathbb{R}^{p \times (r+1)}$. Suppose that $a \in \mathbb{R}^{(r+1)}$ and that $h = Va \in \mathbf{R}^p$. Then we may parametrise $\tilde{l}$ in terms of a as

$$\tilde{l}(a) = g' V a + \frac{1}{2} a' V' H V a$$

Supposing that $V' H V$ is strictly positive definite, the optimal solution to this loss fucntion occurs at

$$a = -(V' H V)^{-1} V' g$$

As $V' H V$ is a $(r + 1) \times (r + 1)$ matrix, the solution for a is relatively fast to compute (assuming the the Krylov Subspace dimension chosen, $r + 1$, is not too large). We may then set h, the optimisation step taken for the original problem, to be Va.

Notice that in the $r = 0$ case the solution returns a simple quadratic line-search method, which optimises the univariate quadratic approximation to the loss function in the direction of the gradient. In fact the method may be viewed as a direct generalization of the naïve quadratic line-search. It is also interesting to note that if the subspace is chosen to have the full dimension of the parameter space then the method is equivalent to the Newton's method.

3.2 Quadratic line search as a special case

We use the $r = 0$ case to illustrate the main idea of the method. Note that when $r = 0$, we have that $V = g$. This results in $a = -\frac{g^T g}{g^T H g}$ and thus the update step is

$$\theta_{t+1} := \theta_t + Va = \theta_t - \frac{g^T g}{g^T H g} g$$

A toy example using the update rule when $r = 0$ is applied to the Rosenbrock function

$$r(x, y) := (1 - x)^2 + 100(y - x^2)^2$$

It's a well-known fact that that the Rosenbrock function is non-convex, and exhibits a valley that contains the global minimum. Most optimization methods have no trouble descending to the valley, but traversing the valley to the global minimum is a hard task to accomplish. The global minimum for $r(x, y)$ occurs at $(x, y) = (1, 1)$. Quadratic line search is able to make large steps within the valley where the objective function is almost flat. Methods which do not incorporate any second order information would require many iterations to converge within or escape such regions.

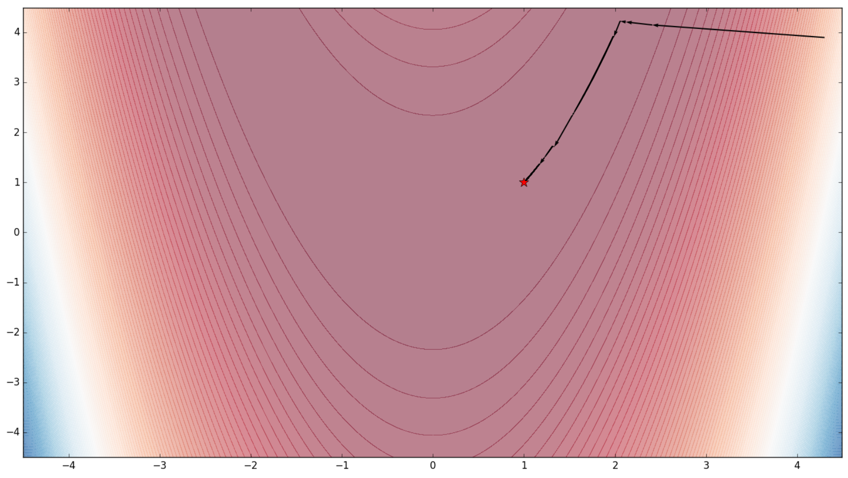

Figure 1: Quadratic line search as a special case, $\epsilon = 10^{-8}$

3.3 Practicalities

3.3.1 Ill-conditioning of subspace basis system

The matrix V is likely to be poorly conditioned for two reasons.

Firstly, the magnitude of its columns will likely increase or decrease geometrically causing issues in solving the linear system involved in determining a. This can fixed by renormalizing the basis components on each application of H. The pseudocode for this is given below, in Algorithm 1.

Algorithm 1 Normalised basis construction

```
1: $W_0 \leftarrow g$
2: $V_0 \leftarrow W_0 / \|W_0\|$
3: for $i = 0$ to $r + 1$ do
4:     $W_{i+1} \leftarrow HV_i$
5:     $V_{i+1} \leftarrow W_{i+1} / \|W_{i+1}\|$
6: end for
7: return $V_{0:r}$, $W_{1:(r+1)}$
```

Note that storing the W matrix will prove useful later when computing $V'HV = V'W$.

Secondly, even after normalisation, the basis may be ill-conditioned as the power iteration defining the normalized basis components converges to the dominant eigenvector of H with high probability. Hence, the angle between consecutive basis components becomes small for later terms in the basis. This causes the matrix V to be ill-conditioned. One can solve this problem by applying the Gram-Schmidt procedure while building up the basis. The revised algorithm would generate an orthonormal basis which also spans the Krylov subspace, but does not have the conditioning problem for large r. The pseudocode for this is given in Algorithm 2.

Algorithm 2 Ortho-Normalised basis construction

```
1: $W_0 \leftarrow g$
2: $V_0 \leftarrow W_0/\|W_0\|$
3: for $i = 0$ to $r + 1$ do
4:     $W_{i+1} \leftarrow HV_i$
5:     $V_{i+1} \leftarrow [W_{i+1} - \text{proj}_{V_{0:i}}(W_{i+1})]/\|W_{i+1} - \text{proj}_{V_{0:i}}(W_{i+1})\|$
6: end for
7: return $V_{0:r}$, $W_{1:(r+1)}$
```

3.3.2 Saddle Point and Ravine Corrections

The procedure described in Section 3.1 is valid if the matrix $V'HV$ is strictly positive definite. In many applications, especially in deep learning, this assumption would not hold. To address this issue we make use of the saddle point correction from [DPG$^+$14]. The correction entails replacing the true second order term, $V'HV$, by its rectified version, $|V'HV|$; if an SVD of $V'HV$ is $V'HV = E'DE$ then by definition $|V'HV| = E'|D|E$ where $|D|$ is the diagonal matrix of component-wise absolute values of D.

The algorithm is particularly suited to this type of correction as the dimension of the matrix $V'HV$ is assumed to be relatively small, so one may compute its eigendecomposition using standard methods for symmetric matrices. To avoid numerical stability issues when solving for a, we also inflate the diagonal part of the SVD by a fixed small multiple of the largest absolute eigenvalue; hence the true second order term used is based on $|\widetilde{V'HV}| = E'(|D| + \epsilon \max(|D|))E$, effectively putting a hard upper bound on the condition number of the system.

3.3.3 Divergence from quadratic behaviour and instability of higher order terms

The updates described thus far would work perfectly if the loss surface was truly a quadratic form. However, in practice the the loss is not perfectly quadratic. In fact, the higher order effects can be quite pronounced, especially when increasing the dimension of the Krylov subspace. To avoid taking large, detrimental steps in directions in which the surface deviates significantly from the approximating quadratic form, we introduce a learning rate, $\alpha \leq 1$ and take steps of the form $\theta_{t+1} = \theta_t + \alpha Va$ where Va is determined as described above. This is in contrast to the update $\theta_{t+1} = \theta_t + Va$ that the theoretical derivation would suggest is optimal.

4 Implementation

We implement the procedure in Python within the `Autograd` framework [MDJA15]. This allows us to compute gradients and hessian-vector products relatively inexpensively, which we require in order to build the basis for the Krylov subspace. The implementation code is shown in subsection 4.1. In subsection 4.2 we discuss the profiler output for SCNaKS, demonstrating that the slowest steps involved are inherent to the current implementation of Hessian-vector product in `Autograd`. Finally in subsection 4.3 we describe known limitations of the implementation.

4.1 Python Code

We format the procedure similarly to the implementation of other optimisation procedures supported by `Autograd` (SGD, RMSProp, ADAM). The three required arguments are (in order) the flattened gradient function, the flattened Hessian-vector product function and the initial parameter vector. The optional argument `r` is dimension of the Krylov subspace used (minus 1). The optional argument `gs` indicates whether to build the Krylov subspace basis using sequential Gram-Schmidt (default, `gs=True`, algorithm 2) or not (`gs=False`, algorithm 1)

Listing 1: SCNaKS Implementation Code

```
import autograd.numpy as np


def norm_orth(u0, u, gs):
    if gs: u0 = u0 - np.dot(u, np.dot(u.T, u0))
    return u0 / np.linalg.norm(u0)


def scnaks(grad, hvp, parms, r=2, num_iters=100, gs=True,
            step_size=1, eps=2**-8, callback=None):
    v, hv = np.zeros((parms.size, r+1)), np.zeros((parms.size, r+1))
    for curr_iter in range(num_iters):
        if callback: callback(parms, curr_iter)
        g = grad(parms, curr_iter)
        v[:, 0] = g/np.linalg.norm(g)
        hv[:, 0] = hvp(parms, curr_iter, v[:, 0])
        for j in range(r):
            v[:, j+1] = norm_orth(hv[:, j], v[:, 0:(j+1)], gs)
            hv[:, j+1] = hvp(parms, curr_iter, v[:, j+1])
        vhv, vg = v.T.dot(hv), v.T.dot(g)
        d, e = np.linalg.eigh(vhv)
        vhv_inv = np.dot(1/(np.abs(d)+eps*np.max(np.abs(d)))*e, e.T)
        parms -= step_size * v.dot(vhv_inv.dot(vg))
    return parms
```

4.2 Profiling

The profiler output for the slowest components of the algorithm are given in table 1. A very significant portion of the run time is spent in the `__call__` routine. `__call__` is defined as a method for the primitive class within `"autograd\core.py"` and is being executed repeatedly by `forward_pass` and `backward_pass` according to the profiler call tree.

Table 1: Profiler Output for Calls with (Own Time > 5%)

Name	Call Count	Time (%)	Own Time (%)
__call__	75505	72%	43%
<numpy.core.multiarray.array>	178197	20%	20%
<numpy.core.multiarray.dot>	6228	14%	14%

Hence, the most obvious way in which to speed up the SCNaKS procedure would be to investigate whether the `__call__` bottleneck can be addressed. This may be possible by incorporating hessian-vector product based Krylov subspace generation into core autograd in order to reduce the overhead required.

4.3 Limitations

There are four known limitations of the current implementation. Firstly, this procedure only works when the parameters are structured as a single long vector. Since `Autograd` fully supports differentiation with respect to parameters formatted as non-trivial data structures, we intend to extend this procedure to act on such structures as well.

Secondly, as this is intended to be used as a stochastic optimisation procedure, a momentum based approach, as in [SMDH13] or as in `ADAM` [KB14], would help to stabilize the results. We intend to implement such an approach in the next cycle.

Thirdly, the update steps have erratic behaviour when the learning rate is too high relative to the dimensionality of the subspace used. We do not have an automated method to suggest a learning rate in accordance with the dimensionality in a way that will yield reliable results. Currently the learning rate is set by trial and error. For the algorithm to work as an effective black box, we will need to establish such guidelines.

Lastly, as we will see in the following section, the algorithm is currently slower (in the time domain) than modern first-order methods.

5 Results

We provide plots to compare `SCNaKS` to `ADAM` [KB14] using the implementation built into `Autograd`. In figure 2 we show the test set error over time, while in figure 3 which show the test set error by training epoch. We find that, on our prototype model, SCNaKS takes more time to achieve the same level of accuracy as `ADAM`, but it does so in fewer epochs.

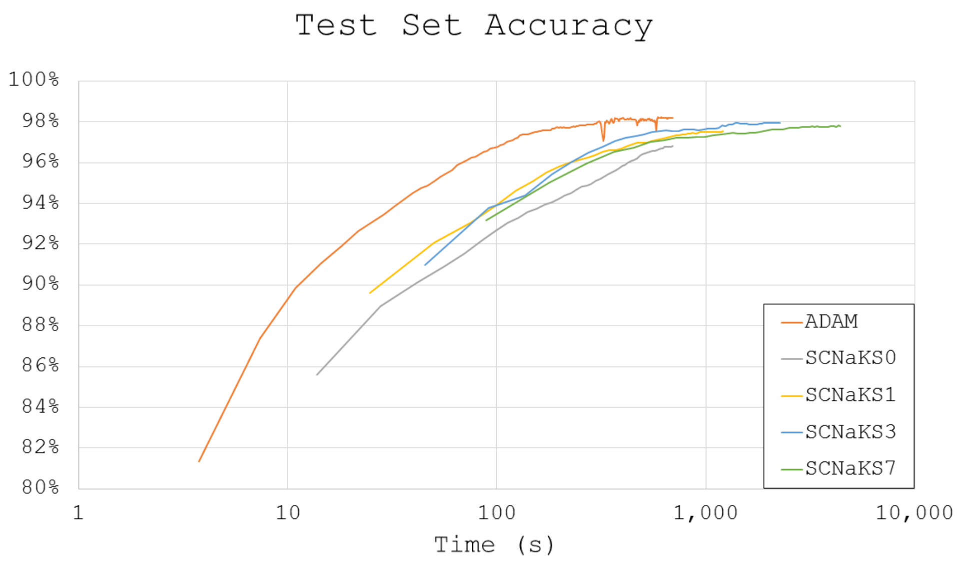

Figure 2: Test Set Prediction by Time, ADAM vs. SCNaKS

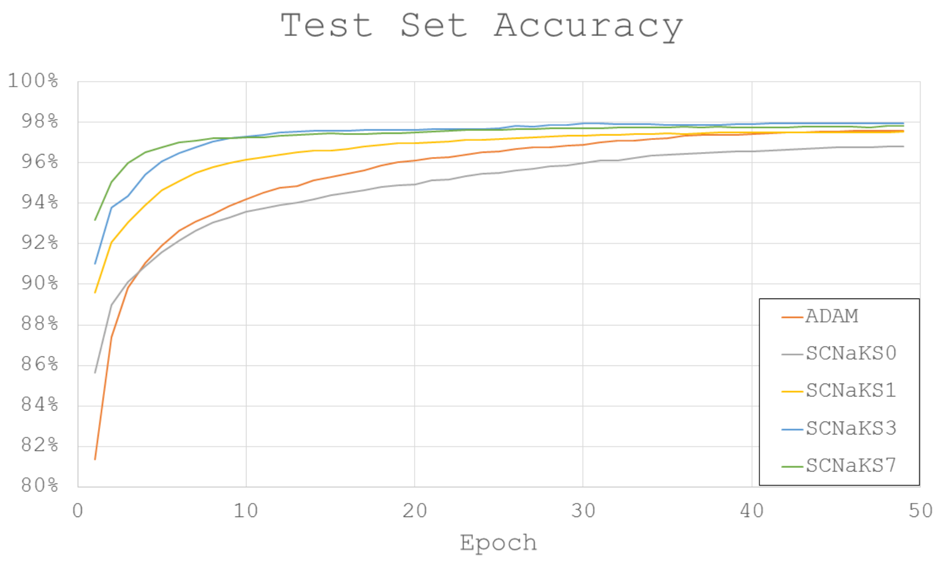

Figure 3: Test Set Prediction by Epoch, ADAM vs. SCNaKS

Our prototype model is a neural net with two hidden layers with 200 and 100 nodes respectively. The hidden layers have hyperbolic tangent activation function. The output layer is a 10-node softmax layer. The prototype model is applied to the MNIST data set. The architecture of our net can be summarized in the figure below.

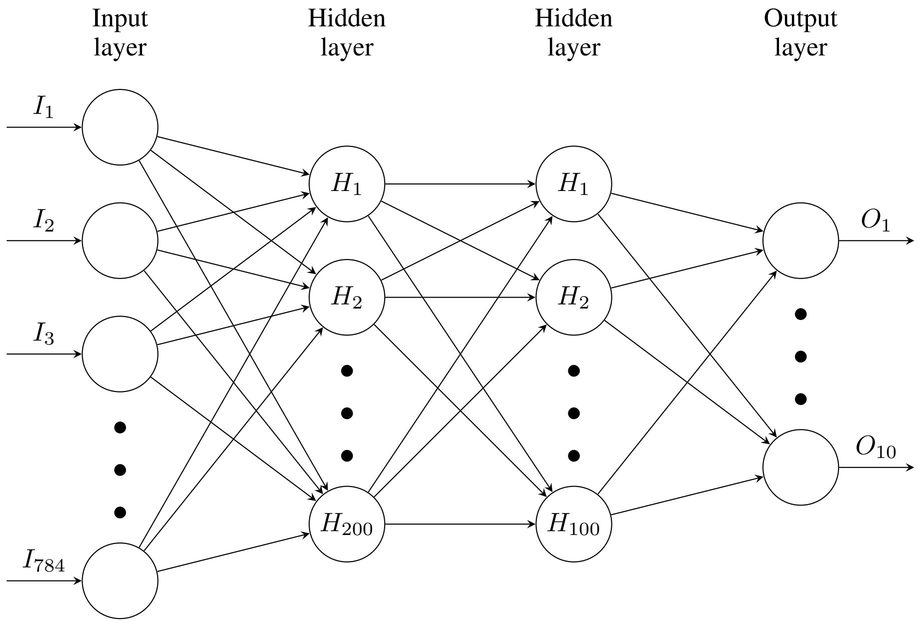

Figure 4: Prototype model architecture

6 Conclusions and Future Research Goals

We find that SCNaKS does not outperform modern first order methods in the time domain for the prototype model. We note, however that this may be due to the relatively simple model/data set used. In order to better understand the relative positioning of the algorithms we must further contrast their performance on more complicated data sets and neural net architectures.

We also remark that, at no additional computational cost, one may characterize the local shape of the loss function surface by tracking the signs and magnitudes of the components of the spectrum of the restriction of the hessian of the loss function to Krylov Subspace. This data is already computed by the algorithm (in the vector `d` on line 21 of Listing 1). The spectral decomposition can be used to determine whether the local surface characterization is hyperbolic or elliptical. Such information could be used to dynamically modify the search method, for example by increasing or decreasing the learning rate in accordance with the local topography. Spectral information could also aid in visualizations of model training. The eigenvalues could be tracked across epochs to get real-time (subspace-versions) of the plots shown in [SBL16] or of the Eigenscape plots proposed by Duvenaud in [Duv16]. In the $r = 1$ case, making use of the restriction of the gradient as well, one could also display an animation of the surface plots of the quadratic approximations to the loss surface and the update steps taken.

References

[DPG+14] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[Duv16] David Duvenaud. Hyperspatial topography. Unpublished, shared via email, 2016.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[MDJA15] Dougal Maclaurin, David Duvenaud, Matthew Johnson, and Ryan P. Adams. Autograd: Reverse-mode differentiation of native Python, 2015.

[Møl93] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[SBL16] Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

[SMDH13] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. 2013.

[VP12] Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *AISTATS*, pages 1261–1268, 2012.