Improving SGD convergence by tracing multiple promising directions and estimating distance to minimum

Jarek Duda
Jagiellonian University, Golebia 24, 31-007 Krakow, Poland, Email: dudajar@gmail.com

Abstract—Deep neural networks are usually trained with stochastic gradient descent (SGD), which optimizes $\theta \in \mathbb{R}^D$ parameters to minimize objective function using very rough approximations of gradient, only averaging to the real gradient. To improve its convergence, there is used some state representing the current situation, like momentum being local average of gradients - newly calculated gradients update this state, which directly governs the update of $\theta$ parameters. This high dimensional optimization might have more than one direction worth tracing, also the used methods usually do not try to estimate distance to the minimum for adapting the step size. There will be discussed using low parameter second order parametrization approximating local behavior in a smaller number of directions, continuously optimized as more promising - which have accumulated statistical trends from calculated gradients. In every step this parameterizations is updated to reduce difference between its gradient and calculated one, to better describe local situation. Then the proper optimization step shifts $\theta$ accordingly to this local parametrization - in every considered direction, attracts or repulses correspondingly to situation in this direction, distance from its extremum.

Keywords: optimization, stochastic gradient descent, convergence, deep learning, Hessian

I. INTRODUCTION

In many machine learning situations we want to find parameters $\theta \in \mathbb{R}^D$ minimizing some objective function $F : \mathbb{R}^D \to \mathbb{R}$ averaged over given dataset (of size $n$):

$$F(\theta) = \frac{1}{n} \sum_{i=1}^{n} F_i(\theta)$$  \hspace{1cm} (1)

where $F_i(\theta)$ is objective function for $i$-th object of dataset. We would like to minimize $F$ through gradient descent, however, dataset can be very large, and calculation of $\nabla_\theta F_i$ is often relatively costly, for example in backpropagation of neural network. Therefore, we would like to work on gradients calculated from a subset of dataset (mini-batch), or even single objects. Additionally, objective function can have some intrinsic randomness.

In stochastic gradient descent (SGD) framework \cite{1}, we can ask for some approximation of gradient for a chosen point $\theta^t$ in successive times $t$:

$$g^t = \nabla_\theta F^t(\theta^t) \quad (\approx \nabla_\theta F(\theta^t))$$  \hspace{1cm} (2)

where $F^t$ corresponds to average over a subset (mini-batch) or even a single element ($F^t = F_i$) - we can assume that $g^t$ averages to the real gradient over some time.

Therefore, we need to exploit statistical trends from $g^t$ sequence to optimize evolution of $\theta^t$ - providing fast convergence to local minimum. As calculation of $g^t$ is relatively costly and such convergence is often very slow, improving it with more accurate (and costly) modelling might provide significant benefits especially for deep learning applications.

A natural basic approach is working on momentum \cite{2}: shift toward some (e.g. exponential moving) average of $g^t$. There are also many other approaches, for example popular ADAM \cite{3} estimates both average gradient and coordinate-wise squared gradient. Such standard simple methods leave two basic opportunities for improved exploitation of statistical trends from $g^t$ sequence, there will be presented a general approach for using them in a reasonable cost:

1) They do not try to estimate distance from the minimum, what could allow to optimize the crucial choice of step size. A direct approach to do it is estimating second derivative in this chosen direction, however, it is problematic due to knowing only noisy stochastic gradients. We will focus on indirect approach: parametrize local behavior with second order polynomial (of known extremum), and update its parameters to improve agreement with calculated $g^t$.

2) They focus only on tracing a single most promising direction, erasing all information about the remaining ones. In high dimensional optimization problems it might be worth to simultaneously trace multiple statistically promising directions from the $g^t$ sequence. Intuitively, calculating PCA: covariance matrix of approximated gradients using e.g. exponential moving average, beside single dominant eigenvector, there are often multiple eigenvectors with close eigenvalues - also worth to include into considerations.

Both these opportunities could be exploited if trying to model the entire Hessian, but it is usually much too costly: requiring $O(D^2)$ memory and computations in every step.
However, we can focus on some chosen number \( d \leq D \) of looking most promising directions (as modelled eigenvectors), reducing this memory and computational cost to \( O(dD) \).

Such considered \( d \) dimensions can vary (in rotation and size) throughout the optimization to find the best (local) trade-offs between modelling cost and convergence improvement. For \( d = 1 \) we should get similar method as using only momentum, but additionally trying to estimate distance from extremum in this direction: attracting if minimum, repelling if maximum, with step size depending on this estimated distance. For larger \( d \) we perform such behavior in simultaneously \( d \) directions, e.g. attracting in some and repulsing in others for predicted saddle. Finally, for \( d = D \) we should get modelling of the entire Hessian.

Optimizing local parametrization should repair issues of other methods trying to exploit restricted Hessian:

1) L-BFGS \[^4\] uses differences of gradients to approximate Hessian. However, they are very noisy in stochastic setting. Statistical trends can be exploited from this noise with slowly evolving parameters of local approximation.

2) TONGA \[^5\] trying to recreate non-centered covariance matrix of \( g^t \) with exponential moving average. Such averaging does not include evolution of gradients with position, what is included in evolution of parameters of local approximation.

3) K-FAC \[^6\] uses block approximation of Hessian - what is usually much more costly, and the real gradients can mix such blocks, what is excluded in such models.

The current version of this article describes work in progress: general ideas leaving large freedom. The difficult choice of details is planned to be explored on examples in near future.

II. LOCAL SECOND ORDER PARAMETRIZATION

We would like to model \( d \ll D \) looking most promising eigendirections (should be orthonormal \( v_i \cdot v_j = \delta_{ij} \)) of Hessian of objective function \( F \) in current point \( \theta^t \):

\[
\begin{bmatrix}
\frac{\partial^2 F}{\partial \theta_i \partial \theta_j}(\theta^t)
\end{bmatrix}_{i,j=1...D} =: H^t \approx \tilde{H}^t := \frac{1}{2} \sum_{i=1}^{d} \lambda_i^t v_i^t (v_i^t)^T
\]  

(3)

We also need to parameterize e.g. positions of extremums in these \( d \) directions: denote their coordinates as \( (p_1, \ldots, p_d) \). Also its absolute height should be parameterized \( (h) \), however, it can be neglected if working only on gradients (unless exploiting absolute values of objective function). Finally, in time \( t \), the used model of local behavior of objective function around \( \theta^t \) has the following parametrisation and gradient:

\[
f^t(\theta) = h^t + \frac{1}{2} \sum_{i=1}^{d} \lambda_i^t (\theta \cdot v_i^t - p_i^t)^2
\]  

(4)

\[
\nabla_\theta f^t(\theta) = \sum_{i=1}^{d} \lambda_i^t (\theta \cdot v_i^t - p_i^t) v_i^t
\]  

(5)

The \( d \) directions should describe recently statistically dominating trends in \( g^t \) sequence. Then \( \lambda_i, p_i \) parameters describe parabola in \( i \)-th direction \( v_i \). Assuming objective function indeed decreases on considered \( \theta^t \) sequence, decrease of gradients in \( v_i \) direction suggests approaching its minimum \( (\lambda_i > 0) \). In contrast, increase of gradients in \( v_i \) suggests going away from its maximum \( (\lambda_i < 0) \). Estimated \( \lambda_i, p_i \) parameters allow to estimate distance from such extremum - allow to choose attraction or repulsion, and shift correspondingly to estimated distance.

In every \( t \to t+1 \) step we would like to update parameters to reduce residue \( r^t \): difference between calculated gradient and gradient of our parametrization:

\[
r^t = g^t - \nabla_\theta f^t(\theta^t) = g^t - \sum_{i=1}^{d} \lambda_i^t (\theta^t \cdot v_i^t - p_i^t) v_i^t
\]  

(6)

Such update reducing residue should maintain at least approximated orthonormality of considered set of directions \( v \). This way we should extract \( d \) locally dominant directions in statistics of \( g^t \) sequence, neglecting oscillations in the remaining directions.

There might be also other criteria worth to be included in minimization step, like \(-\sum_{i=1}^{d} \lambda_i^t (\theta \cdot v_i - p_i)^2\) reduction of value using current parametrization (lowest local valley) considered in the first version of this article. However, we should also have in mind negative eigenvalues, especially close to a saddle - this reduction could be optimized only over directions with positive \( \lambda_i^t \). For simplicity it is omitted in this version of article.

A. Proper optimization step

After reducing residue with parameter update step we will discuss in the following subsections, we can perform the proper optimization step for \( \theta \) based on current parametrization, for example for some \( \alpha \in (0, 1] \) learning rate:

\[
\theta^{t+1} = \theta^t + \alpha \sgn(\lambda_i^{t+1}) \sum_{i=1}^{d} (p_i^{t+1} - \theta^t \cdot v_i^{t+1}) v_i^{t+1}
\]  

(7)

Thanks to considering the sign of estimated eigenvalue, it has separate behaviors for attracting and repulsive directions:

1) In attractive directions \( (\lambda_i > 0) \) it reduces distance to estimated minimum \( 1 - \alpha \) times, allowing for exponential decrease \( (1 - \alpha)^{\Delta t} \) of distance if its estimated position would remain unchanged.

2) In repulsive directions \( (\lambda_i < 0) \) we would analogously get \( (1 + \alpha)^{\Delta t} \) exponential increase of distance.

It might be worth to use more sophisticated update step, like replacing sign with some activation function like \( \tanh(\lambda) \): smooth and being 0 for \( \lambda = 0 \), it additionally can be asymmetric: e.g. giving stronger repulsion than attraction. Finally, being able to evaluate also certainty of estimation, we could use larger steps in more certain directions.
B. Optimizing \( \lambda \) and \( p \) for distance estimation

For each chosen direction \( v_i \), we would like to update \( \lambda_i \) and \( p_i \) to each contribute of residue in this direction:
\[
r_i^t := v_i^t \cdot g_i^t = g_i^t - \lambda_i(t) \theta_i^t - p_i^t
\]
(8)
where \( g_i^t := g_i^t \cdot v_i^t \), \( \theta_i^t := \theta_i^t \cdot v_i^t \).

Minimization of \( (r_i^t)^2 \) gives minus gradient:
\[
\Delta \lambda_i^t = - \frac{\partial (g_i^t - \lambda_i(t) \theta_i^t - p_i^t)^2}{\partial \lambda_i} (\lambda_i^t, p_i^t) = 2r_i^t(g_i^t - \lambda_i(t) \theta_i^t - p_i^t)
\]
\[
\Delta p_i^t = - \frac{\partial (g_i^t - \lambda_i(t) \theta_i^t - p_i^t)^2}{\partial p_i} (\lambda_i^t, p_i^t) = -2r_i^t \lambda_i^t
\]
(9)

Suggesting to use gradient descent for some \( \beta > 0 \):
\[
\forall i \quad \lambda_i^{t+1} = \lambda_i^t + \beta \Delta \lambda_i^t \quad p_i^{t+1} = p_i^t + \beta \Delta p_i^t
\]
(10)

However, as we know the desired value: \( (r_i) = 0 \), we can try to optimize the step size in analogy to Newton’s method. In linear approximation, (10) will reduce \( (r_i^{t+1})^2 \) by \( \beta ((\Delta \lambda_i^t)^2 + (\Delta p_i^t)^2) \). It suggests to use instead:
\[
\lambda_i^{t+1} = \lambda_i^t + \beta \frac{(r_i^t)^2}{(\Delta \lambda_i^t)^2 + (\Delta p_i^t)^2} \Delta \lambda_i^t
\]
\[
p_i^{t+1} = p_i^t + \beta \frac{(r_i^t)^2}{(\Delta \lambda_i^t)^2 + (\Delta p_i^t)^2} \Delta p_i^t
\]
(11)

were \( \beta = 1 \) would correspond to \( (r_i^t)^2 \approx 0 \) as in Newton’s method. In SGD \( \beta \in (0, 1] \) should be chosen correspondingly to representativeness of mini-batch in the entire dataset.

The discussed gradient descent assumes that changes of \( \lambda \) and \( p \) have the same weights, modifying this balance might allow for additional optimization. Alternative approach to optimize these parameters, based on least square linear regression, is discussed in Appendix.

C. Optimizing considered directions

While optimization of \( \lambda \) and \( p \) used only the considered directions of residue, especially the remaining should suggest rotation of \( (v_1, \ldots, v_d) \) to extract local statistical trends from recently calculated \( g_i \) values, cancelling noise in less important remaining directions.

Its main tool is reduction of \( r^t = g^t - \nabla f_i(\theta_i^t) \) residue, what should lead to accumulation of residue in currently not considered directions, replacing recently less frequent ones.

Denoting by \( j = 1, \ldots, D \) the original coordinates, we get the following minus gradient of \( \| r^t \|^2 = \sum_{j=1}^{D} (r_j^t)^2 \) in \( j \)-th coordinate in \( v_i \):
\[
\Delta v^t_{ij} = - \frac{\partial \sum_{j=1}^{D} (g_j^t - \sum_{i=1}^{d} \lambda_i(\theta_i^t \cdot v_i - p_i^t) v_j^t)^2}{\partial v_{ij}} (v^t)
\]
\[
\Delta v^t_{ij} = 2r_j^t (\lambda_i^t(\theta_i^t \cdot v_i^t - p_i^t) + \lambda_j^t \theta_j^t v_{ij}^t)
\]
(12)

We would like to update \( v \) in this direction. Its first term corresponds to shifting directions toward the residue. Its second term corresponds to modification of previous parametrization - might be worth weakening or completely removing to assume some smoothness in the not traced directions.

A basic update choice, for some parameter \( \gamma > 0 \), is:
\[
\forall ij \quad v_i^{t+1} = v_i^t + \gamma \Delta v^t_{ij}
\]
(13)

As previously, in analogy to Newton’s method, we can use the fact that we know the desired value: perfect agreement \( (r_j) = 0 \) for \( j = 1, \ldots, D \). In linear approximation neglecting second \( (\Delta v^t)^2 \) term, (13) will reduce \( (r_j^t)^2 \) by \( \gamma \sum_{i=1}^d (\Delta v^t_{ij})^2 \), suggesting to use update step:
\[
v_i^{t+1} = v_i^t + \gamma' (r_i^t)^2 \sum_{j=1}^d (\Delta v^t_{ij})^2
\]
(14)

for some \( \gamma' \in (0, 1] \) describing how representative are chosen mini-batches for the entire sample, which can be chosen as \( \beta' \) in (11), finally suggesting that two hyperparameters might be sufficient: \( \alpha \) in (7) proper optimization step and \( \beta' = \gamma' \) for updating gradient agreement.

1) Maintaining orthonormalization: The discussed update of considered directions would rater not maintain orthonormalization: \( V V^T = I_d \) for \( d \times D \) matrix \( V = (v_1, \ldots, v_d) \). Taking time derivative of this equation:
\[
V V^T = I_d \quad \Rightarrow \quad V V^T = 0_d
\]
(15)

In considered discrete case we would like \( (\Delta V)V^T = 0_d \) : gradients for all vectors being orthogonal to all \( (v_1, \ldots, v_d) \). Using only the first term in (12), \( \Delta V \) for all \( d \) vectors are in direction of residue \( r^t \). Hence we can use instead:
\[
\beta^t = r^t - \sum_{i=1}^{d} (r_i^t \cdot v_i^t) v_i^t \quad \Delta v^t_{ij} = 2r_j^t \lambda_i^t(\theta_i^t \cdot v_i^t - p_i^t)
\]
(16)

Due to time discreetness, satisfying \( (\Delta V) V^T = 0_d \) condition is not sufficient not to slowly lose orthonormality, accumulation of distortions from it might lead to some unwanted behavior. Hence, e.g. every some number of steps, there should be performed some orthonormalization step, especially if considering both terms in (12).

A standard choice is Gram-Schmidt orthonormalization every some number of steps, however, it depends on order of directions. Symmetric approach, but only attracting orthonormal set, is first calculating \( u_i^t = v_i^t - \sum_{j \neq i} (v_i^t \cdot v_j^t) v_j^t \) for all \( i \), then replace directions with normalized: \( v_i^t = u_i^t / \| u_i^t \|_2 \).

D. Initialization

The main loop of optimization algorithm should perform update of parameters: for example (10) or (11) for eigenvalues and positions, and (13) or (12) for directions (additionally performing orthonormalization every some number of steps), then use new parameters to update \( \theta \) with (7), and finally increase time \( t \rightarrow t + 1 \), until satisfying some convergence criterion.

Remaining question is initialization of parameters. Choosing \( \theta^0 \) depends on the problem. A natural initial choice
for eigenvalues is $\lambda^0 = 0$, of positions to make $\theta^0$ in the
minimum: $p_0^i = \theta^0 \cdot v_0^i$. However, adding some bias at the
beginning due to no information might be worth considering.

The choice of initial orthonormal directions $v^0$ seems
difficult, but can be made completely random if eigenvalues
are initialized with 0 - they will be learned from the first
gradients, it might be worth increasing learning rate $\gamma$ and
decreasing $\alpha$ in the beginning.

Alternatively, we can try to start with $d = 1$ considered
directions, for example using the first calculated gradient
(maybe on a larger mini-batch), or starting with some number
of steps of standard e.g. momentum method choosing a single
looking promising direction $v_1$.

Then we can gradually increase the number of considered
directions $d$. Using current $d$ directions, we can average calculated
gradients with subtracted the already used components:

$$v_{d+1} = \gamma'' v_{d+1} + \left(1 - \gamma''\right) \left(g^t - \sum_{i=1}^d (g^t \cdot v^i_t) v^i_t\right)$$

finally normalizing it while including in the considered di-
rrections and increasing $d$. We can also try to automatically
reduce $d$, for example when some eigenvalue remains close to
zero.

III. CONCLUSIONS AND FURTHER WORK

There was presented a general concept for exploiting op-
portunities missed by standard SGD convergence methods:
estimation of distance to extremum and trying to simultane-
ously model multiple promising directions. While optimizing
the details will require a lot of experimental work, the poten-
tial benefits from convergence improvements might be
significantly larger than the cost of such more sophisticated
modelling, especially for deep learning models.

There are a lot of details to choose, like used mini-batches,
hyperparameters (can be evolving in time), maybe better
optimization criteria. Their choice might depend on the task
and finally should be made automatically - what is planned
for future work on concrete examples.

The discussed approach updates parametrization exploiting
only calculated gradients. Calculating more information to
improve this update might be also worth considering, like
value of objective function, or some second derivatives, e.g.
in the $v^i_t$ directions.

The $d = D$ case might be also worth considering: trying
to model the entire Hessian, but not directly from second
derivatives or differences of gradients, but by slowly updating
parametrization being local approximation. This way should
better handle stochasticity, exploit statistical trends.

A tempting simplification is putting eigenvalues into eigen-
vec tors: work on $u_i = \sqrt{\lambda_i} v^i_t$ instead, what would allow to
remove eigenvalues from considerations and enforcement of
unitary $v$ vectors. They were separated for better presentation,
and to separate rotational optimization.

APPENDIX

To estimate $\lambda$, $p_t$ parameters, alternative approach might
be based on least squares linear regression.

Let us start with one-dimensional static situation: basing
on $\{\theta^t, g^t\}_{t=1..N} \subset \mathbb{R}^2$ sequence, we would like to find
parameters of $f(\theta) = h + \frac{1}{2} \lambda(\theta - p)^2$ approximation of
objective function, to minimize $g^t - f'(\theta^t)$ distances, let say
MSE: $\sum (g^t - \lambda(\theta^t - p))^2$.

Necessary condition of zeroing derivatives becomes:

$$\sum_{t=1}^N (\theta^t - p)(g^t - \lambda \theta^t + \lambda p)) = 0 = \sum_{t=1}^N (g^t - \lambda \theta^t - p)$$

$$\overline{g} - \lambda \overline{\theta}^2 + 2 \lambda p \overline{g} - p - N \lambda p^2 = 0 = \overline{g} - \overline{\theta} + N \lambda p$$

for

$$\overline{g} = \frac{1}{N} \sum_{t} g^t, \overline{\theta} = \frac{1}{N} \sum_{t} \theta^t, \overline{\theta}^2 = \frac{1}{N} \sum_{t} (\theta^t)^2$$

(17)

Their solution is (least squares linear regression):

$$\lambda = \frac{\overline{\theta} \overline{g} - n \overline{g}}{n \overline{\theta}^2 - \overline{\theta}^2}, \quad p = \frac{\overline{\theta} g - \overline{\theta} \overline{g}}{n \overline{\theta} - \overline{\theta}^2 \overline{\theta}}$$

(18)

However, objective function is not exactly parabola - should
be only locally approximated this way. Additionally, consid-
ered directions can rotate.

Above least-squares regression would be suitable if di-
rections $v$ were fixed and objection function was just a
paraboloida. Generally, the four averages (17) should be
be used in (18) separately for $i = 1, \ldots, d$ coordinates.

They should also be calculated in adaptive way, e.g. using
exponential moving average:

$$\theta^t_{i+1} = \beta \theta^t_{i} + (1 - \beta) \theta^t_t \cdot v^t_i$$

and analogously for $\overline{g}, \overline{\theta} g, \overline{\theta}^2$. While it might be the most
accurate approach, it can also lead to issues like zeroing of
denominator.

REFERENCES

[1] H. Robbins and S. Monro, “A stochastic approximation method,” in
Herbert Robbins Selected Papers. Springer, 1985, pp. 102–109.
[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning represen-
tations by back-propagating errors,” nature, vol. 323, no. 6088, p. 533,
1986.
[3] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,”
arXiv preprint arXiv:1412.6980, 2014.
[4] D. C. Liu and J. Nocedal, “On the limited memory bfgs method for
large scale optimization,” Mathematical programming, vol. 45, no. 1-3,
pp. 503–528, 1989.
[5] N. L. Roux, P.-A. Manzagol, and Y. Bengio, “Topnoumoute online
natural gradient algorithm,” in Advances in neural information pro-
systems, 2008, pp. 849–856.
[6] J. Martens and R. Grosse, “Optimizing neural networks with kronecker-
factored approximate curvature,” in International conference on machine
learning, 2015, pp. 2408–2417.