Practical Gauss-Newton Optimisation for Deep Learning

Aleksandar Botev Hippolyt Ritter David Barber

University College London

Alan Turing Institute

ICML, 2017
Presenter: Bargav Jayaraman

Outline

Outline

First Order Methods

Pros:

- Easy to implement, scale to large models and datasets, and can handle noisy gradients

Cons:

- Suitable initial learning rate and decay schedule needs to be selected
- Requires many runs of training with different hyper-parameters
- Pure SGD struggles to escape 'valleys' in error surface

Outline

Second Order Methods

Pros:

- Perform updates of the form $\delta = H^{-1}g$ where H is the Hessian or some approximation, and g is the gradient of the error function
- Makes more progress per step using the curvature information

Cons:

- Explicit calculation and storage of Hessian is infeasible for deep networks

Outline

Contributions

- Recursive block-diagonal approximation
- Observation that piece-wise linear transformation functions have no differentiable strict local minima
- Relation with KFAC (block diagonal approximation to Fisher matrix)
- Experiments on benchmark dataset to show the superiority of second order methods over tuned first order methods

Related Work

- Efficient computation of full GN matrix-vector products using a form of automatic differentiation; used to approximately solve the linear system $\bar{G}\delta = \nabla f$ using conjugate gradients to find the parameter update . CGD - too slow
- KFAC in which the Fisher matrix is used as the curvature matrix. The Fisher matrix is another PSD approximation to the Hessian that is used in natural gradient descent. In general Fisher matrix and GN matrix are different.

Preliminaries

- Feed forward neural network is defined as:

$$h_\lambda = W_\lambda a_{\lambda-1}$$

$$a_\lambda = f_\lambda(h_\lambda)$$

  for $1 \leq \lambda < L$
- Error function is defined as $E(h_L, y)$
- Hessian is defined as:

$$[H]_{ij} = \frac{\partial^2}{\partial\theta_i \partial\theta_j} E(\theta)$$

Block Diagonal Hessian

- Gradient w.r.t. layer λ is defined with chain rule:

$$\frac{\partial E}{\partial W^{\lambda}_{a,b}} = \sum_i \frac{\partial h^{\lambda}_i}{\partial W^{\lambda}_{a,b}} \frac{\partial E}{\partial h^{\lambda}_i} = a^{\lambda-1}_b \frac{\partial E}{\partial h^{\lambda}_a}$$

- Differentiating again will give the sample Hessian:

$$[H_\lambda]_{(a,b),(c,d)} = \frac{\partial^2 E}{\partial W^{\lambda}_{a,b} \partial W^{\lambda}_{c,d}} = a^{\lambda-1}_b a^{\lambda-1}_d [\mathcal{H}_\lambda]_{(a,c)}$$

- In matrix form:

$$H_\lambda = (a_{\lambda-1} a^T_{\lambda-1}) \otimes \mathcal{H}_\lambda$$

where $\otimes$ denotes the Kronecker product and $\mathcal{H}_\lambda$ is the pre-activation Hessian.

Block Hessian Recursion

In order to calculate the sample Hessian, the pre-activation Hessian is first evaluated recursively as:

$$\mathcal{H}_\lambda = B_\lambda W_{\lambda+1}^T \mathcal{H}_{\lambda+1} W_{\lambda+1} B_\lambda + D_\lambda$$

where the diagonal matrices are:

$$B_\lambda = \text{diag}(f'_\lambda(h_\lambda))$$

$$D_\lambda = \text{diag}(f''_\lambda(h_\lambda)\frac{\partial E}{\partial a_\lambda})$$

Non Differentiable Local Maxima

- For piecewise linear functions like ReLU, second derivative is zero and hence D_λ is zero
- Thus if $\mathcal{H}_\lambda$ is PSD, pre-activation matrices are PSD everywhere
- If we fix all parameters except W_λ, objective function is convex w.r.t W_λ where it is twice differentiable
- Hence, there can be no saddle points or local maxima of objective function w.r.t the parameters within a layer
- Except, where the transfer function changes regimes

Non Differentiable Local Maxima

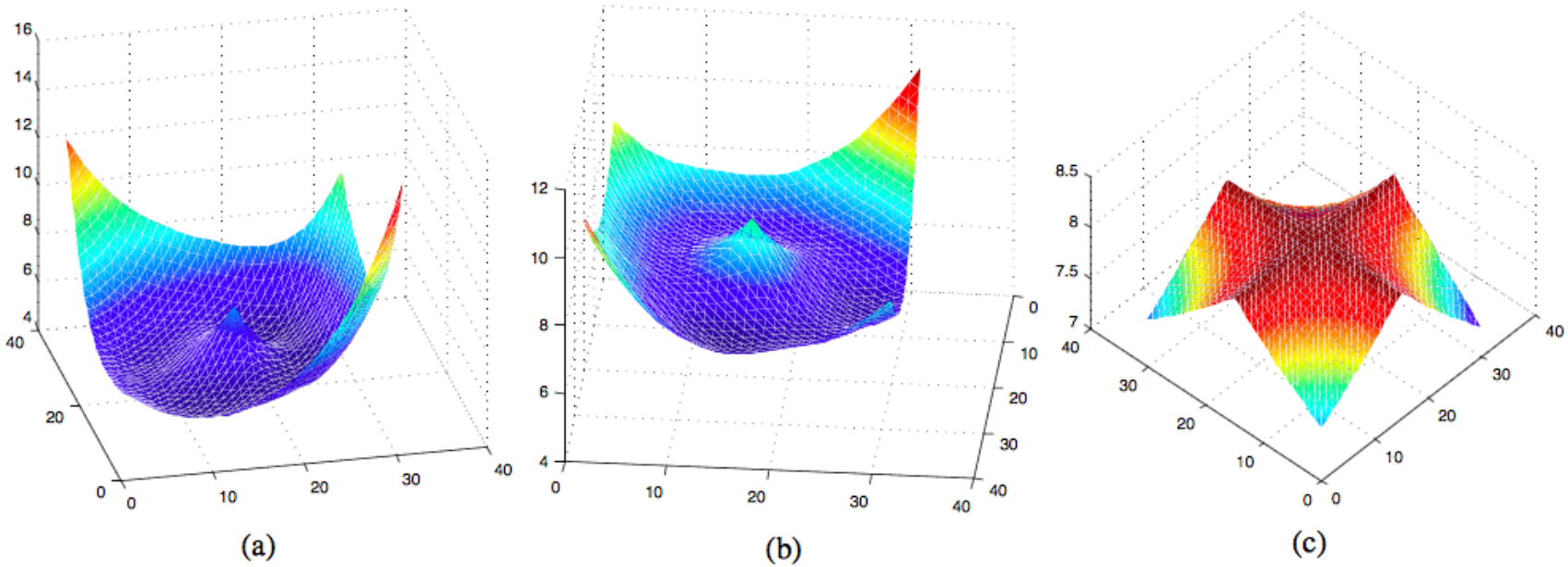


Figure 1. Two layer network with ReLU and square loss. (a) The objective function E as we vary $W_1(x, y)$ along two randomly chosen direction matrices U and V, giving $W_1(x, y) = xU + yV$, $(x, y) \in \mathbb{R}^2$. (b) E as a function of two randomly chosen directions within W_2. (c) E for varying jointly $W_1 = xU$, $W_2 = yV$. The surfaces contain no smooth local maxima.

Approximate Gauss-Newton Method

- Besides being intractable for large neural networks, the Hessian is not guaranteed to be PSD. A Newton update $H^{-1}g$ could therefore lead to an increase in the error
- A common PSD approximation to the Hessian is the Gauss-Newton (GN) matrix
- For an error $E(h^L(\theta))$, the sample Hessian is given by:

$$\frac{\partial^2 E}{\partial \theta_i \partial \theta_j} = \sum_k \frac{\partial E}{\partial h_k^L} \frac{\partial^2 h_k^L}{\partial \theta_i \partial \theta_j} + \sum_{k,l} \frac{\partial h_k^L}{\partial \theta_i} \frac{\partial^2 E}{\partial h_k^L \partial h_l^L} \frac{\partial h_l^L}{\partial \theta_j}$$

Assuming $\mathcal{H}_L$ is PSD, GN method forms a PSD approximation by neglecting the first term. GN matrix is given as:

$$G \equiv {J_\theta^{h_L}}^T \mathcal{H}_L J_\theta^{h_L}; \bar{G} \equiv \mathbb{E}[{J_\theta^{h_L}}^T \mathcal{H}_L J_\theta^{h_L}]_{p(x,y)}$$

GN Matrix as Khatri-Rao Product

- Block of matrix corresponding to layers λ and β is given as:

$$\bar{G}_{\lambda,\beta} = \mathbb{E}[{J_{W_\lambda}^{h_\lambda}}^T {J_{h_\lambda}^{h_L}}^T \mathcal{H}_L J_{h_\beta}^{h_L} J_{W_\beta}^{h_\beta}]$$

- Defining pre-activation GN matrix between layers as:

$$\mathcal{G}_{\lambda,\beta} = {J_{h_\lambda}^{h_L}}^T \mathcal{H}_L J_{h_\beta}^{h_L}$$

and using the fact that $J_{W_\lambda}^{h_\lambda} = a_{\lambda-1}^T \otimes I$, we obtain

$$\bar{G}_{\lambda,\beta} = \mathbb{E}[(a_{\lambda-1} a_{\beta-1}^T) \otimes \mathcal{G}_{\lambda,\beta}]$$

Thus GN matrix can be written as the expectation of the Khatri-Rao product $\bar{G} = \mathbb{E}[\mathcal{Q} * \mathcal{G}]$

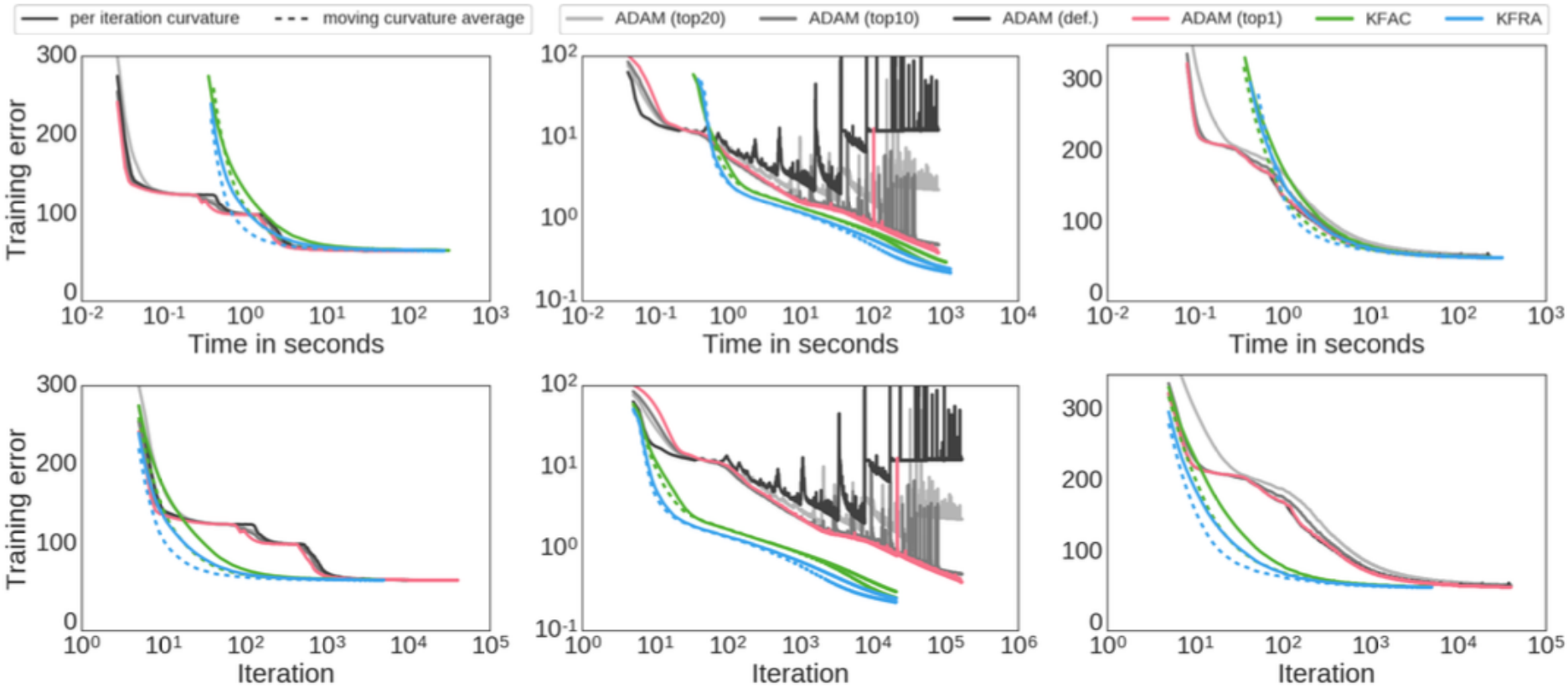
Approximating GN Diagonal Blocks

- Layer-wise GN block can be represented as: $G_\lambda = \mathcal{Q}_\lambda \otimes \mathcal{G}_\lambda$ and the expected GN blocks can be calculated as $\mathbb{E}[G_\lambda]$, however this requires storing values across all data points which can be prohibitive
- Proposed approach approximates the blocks as: $\mathbb{E}[G_\lambda] \approx \mathbb{E}[\mathcal{Q}_\lambda] \otimes \mathbb{E}[\mathcal{G}_\lambda]$, where $\mathbb{E}[\mathcal{Q}_\lambda] = \frac{1}{N} A_{\lambda-1} A_{\lambda-1}^T$ is the uncentered covariance of activations
- $\mathbb{E}[\mathcal{G}_\lambda] = \mathbb{E}[\sum_k C_\lambda^k C_\lambda^{k\,T}]$ where $C_\lambda^k = B_\lambda W_{\lambda+1}^T C_{\lambda+1}^k$
- This reduces the memory footprint to $K \times D \times N$, where K is the rank of GN matrix. This method is called Kronecker Factored Low Rank (KFLR).

Experimental Setup

- Datasets - MNIST, CURVES, FACES
- Comparison of Second order methods like KFRA and KFCA with first order methods like SGD, Nesterov Accelerated Gradient, Momentum and ADAM
- First order methods ran with 40,000 parameter updates for MNIST and CURVES and 160,000 updates for FACES. Second order methods ran only for 5,000 / 20,000 updates.

Results



(a) CURVES (b) FACES (c) MNIST

Figure 2. Comparison of the objective function being optimised by KFRA, KFAC and ADAM on CURVES, FACES and MNIST. GPU benchmarks are in the first row, progress per update in the second. The dashed line indicates the use of momentum on the curvature matrix for the second-order methods. Errors are averaged using a sliding window of ten.

Summary

- Derivation of block-diagonal structure of Hessian matrix
- For networks with piece-wise linear transfer functions and convex loss the objective has no differentiable local maxima
- With respect to the parameters of a single layer, the objective has no differentiable saddle points
- Second order methods are feasible and perform better than first order methods for neural networks

Extra Slide : Derivation of Block Diagonal Hessian Recursion

$$\mathcal{H}_\lambda = \frac{\partial}{\partial h_b^\lambda}\frac{\partial E}{\partial h_a^\lambda} = \frac{\partial}{\partial h_b^\lambda}\sum_i \frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial h_i^{\lambda+1}}{\partial h_a^\lambda} = \sum_i \frac{\partial}{\partial h_b^\lambda}\left(\frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial h_i^{\lambda+1}}{\partial a_a^\lambda}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\right)$$

$$= \sum_i W_{i,a}^{\lambda+1}\frac{\partial}{\partial h_b^\lambda}\left(\frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\right) = \sum_i W_{i,a}^{\lambda+1}\left(\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_b^\lambda \partial h_i^{\lambda+1}} + \frac{\partial E}{\partial h_i^{\lambda+1}}\frac{\partial^2 a_a^\lambda}{\partial h_a^\lambda \partial h_b^\lambda}\right)$$

$$= \sum_i W_{i,a}^{\lambda+1}\left(\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\sum_j \frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}}\frac{\partial h_j^{\lambda+1}}{\partial h_b^\lambda} + \frac{\partial E}{\partial h_i^{\lambda+1}}\delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\right)$$

$$= \delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\left(\sum_i W_{i,a}^{\lambda+1}\frac{\partial E}{\partial h_i^{\lambda+1}}\right) + \sum_{i,j} W_{i,a}^{\lambda+1}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}}W_{j,b}^{\lambda+1}\frac{\partial a_b^\lambda}{\partial h_b^\lambda}$$

$$= \delta_{a,b}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\frac{\partial E}{\partial a_a^\lambda} + \sum_{i,j} W_{i,a}^{\lambda+1}\frac{\partial a_a^\lambda}{\partial h_a^\lambda}\frac{\partial^2 E}{\partial h_j^{\lambda+1}\partial h_i^{\lambda+1}}\frac{\partial a_b^\lambda}{\partial h_b^\lambda}W_{j,b}^{\lambda+1}$$

Hence the pre-activation Hessian can be written in matrix notation as

$$\mathcal{H}_\lambda = B_\lambda W_{\lambda+1}^\mathsf{T}\mathcal{H}_{\lambda+1}W_{\lambda+1}B_\lambda + D_\lambda$$

where we define the diagonal matrices

$$[B_\lambda]_{a,a'} = \delta_{a,a'}\frac{\partial a_a^\lambda}{\partial h_a^\lambda} = \delta_{a,a'}f'(h_a^\lambda)$$

$$[D_\lambda]_{a,a'} = \delta_{a,a'}\frac{\partial^2 a_a^\lambda}{\partial {h_a^\lambda}^2}\frac{\partial E}{\partial x_a^\lambda} = \delta_{a,a'}f''(h_a^\lambda)\frac{\partial E}{\partial x_a^\lambda}$$