Project Final Report

Course: Optimization

(18-460/18-660)

Date: 5th May 2023

Reported by:

Atharva Anand Joshi (atharvaa@andrew.cmu.edu)

Ketan Ramaneti (kramanet@andrew.cmu.edu)

Motivation

The Newton method is a well-known optimization algorithm that leverages second-order information by incorporating the Hessian of the objective function in addition to the gradient during the update step. This results in a second-order method that can achieve very fast quadratic convergence in terms of the number of iterations required. Furthermore, in its pure phase, the Newton method's convergence is independent of the condition number, making it theoretically superior to gradient-based methods.

Despite its theoretical advantages, in practice, the Newton method often does not perform as well as expected. This is due to the significant computational and memory resources required to compute the Hessian and its inverse, particularly for high-dimensional problems. Additionally, the objective function must be twice differentiable, and the Hessian must be positive definite for the method to be applicable.

As a result, it is not used as widely as gradient-based methods. Our objective in this project has been to study and evaluate **Quasi-Newton** approaches that mitigate some of these problems. The approach we are primarily exploring is an efficient stochastic version of the BFGS method, which can be used with deep neural networks.

Complexity per iteration	Newton Method	Quasi Newton (L-BFGS)	Gradient Descent
Memory	$O(n^2)$	O(n)	O(n)
Computation	$O(n^3)$	$O(n^2)$	O(n)

Background

Quasi-Newton methods essentially provide an effective and practical way to approximate the Hessian matrix. We can formulate general update equation that can be expressed as $x(k+1) = x(k) - \alpha B_k^{-1} \nabla f(x(k))$. Here B_k can be any positive definite matrix. If B_k is the identity matrix, the update equation reduces to that of gradient descent, which is the most popular algorithm used for convex as well as non-convex problems. The identity matrix however does not bear any information about the curvature of the function. As a result, gradient-based methods cannot achieve convergence rates better than linear.

Another option for B_k is the true Hessian of the objective function, which is what is used in the Newton method. An important property of the Hessian is that it globally encodes the curvature information about the function. As a result we get a very fast quadratic convergence in the pure phase. However, as we have discussed earlier, computing the Hessian and inverting it is very expensive. We are therefore looking for an alternative approximation of the Hessian that has the following properties:

Local curvature information: The approximated gradient computed for x(k) in the neighborhood of the current x(k+1) should match the actual gradient of x(k). Note that this does not have to be true for all the points in the domain of f. Let us consider a quadratic approximation of the function $f(x(k) + d) \sim m_K(d) = f(x(k)) + \nabla f(x(k))^T d + ½ * d^T B_k d$. We require that $\nabla m_K(\alpha B_k^{-1} \nabla f(x(k))) = \nabla f(x(k))$.

Computation: The approximation can be computed significantly faster than the Hessian.

Invertibility: The approximation computed at each iteration must be symmetric and positive definite. We would be using the inverse of the approximation in the update equations.

BFGS Algorithm:
The Broyden–Fletcher–Goldfarb–Shanno algorithm is one of the most popular quasi newton methods. We can formulate the above-mentioned properties as a matrix optimization problem:

$$\begin{aligned} \min_{B_{k+1}^{-1}} &||B_{k+1}^{-1} - B_k^{-1}|| \\ \text{subject to } &{B_{k+1}^{-1}}^T = B_{k+1}^{-1} \\ \text{and } &\Delta \mathbf{x}_k = B_{k+1}^{-1} \mathbf{y}_k. \end{aligned} \qquad ||A||_F = \sqrt{\sum_i^m \sum_j^n |a_{ij}|^2}$$

Note that we are directly optimizing for the inverse of B_k^{-1}. This is to avoid the inversion operation, which is often the bottleneck computation. It is assumed that the B_k^{-1} would not change much in every iteration. Therefore, the objective function is minimizing the distance between B_{k+1}^{-1} and B_k^{-1}. The distance metric used here is the Frobenius norm. The first constraint requires B_{k+1}^{-1} to be symmetric. The second constraint is known as the secant equation. Here $\Delta x_k = (x_{k+1} - x_k)$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. This constraint corresponds to the property of encoding local curvature information and is derived from the same. A solution obtained for this problem is as follows:

$$B_+^{-1} = \left(I - \frac{\Delta \mathbf{x} \mathbf{y}^T}{\mathbf{y}^T \Delta \mathbf{x}}\right) B^{-1} \left(I - \frac{\mathbf{y} \Delta \mathbf{x}^T}{\mathbf{y}^T \Delta \mathbf{x}}\right) + \frac{\Delta \mathbf{x} \Delta \mathbf{x}^T}{\mathbf{y}^T \Delta \mathbf{x}}$$

Note that the solution B_{k+1}^{-1} is symmetric and positive semi definite only if B_k^{-1} is symmetric and positive semi definite. Also, the inner product of Δx_k and y_k should be non negative, which is true for convex problems. In case of non-convex problems, a few adjustments are made to meet this condition as we shall see in the next section.

Main Idea

The paper we followed for our project implementation is [1]. The authors considered a feed-forward Deep Neural Network(DNN) with L layers, defined by weight matrices W_i, activation functions ϕ_l for $l \in \{1, 2, \ldots L\}$ and a Loss function $\mathcal{L}$. For a given data-point (x, y), the loss $\mathcal{L}(a_L, y)$ between the output a_L of the DNN and y is a non-convex function of $\theta = [vec(W_1), \ldots, vec(W_L)]$. Algorithm 1 describes the network's forward and backward pass for a single input data point (x, y).

The authors used Kronecker-factored approach to store the gradients and the Hessians from the feed-forward DNN. Considering a dataset of I points indexed from $i = 1, \ldots, I$, we have

$$\mathbb{E}_i[\nabla^2 f_l(i)] \approx \mathbb{E}_i \left[\mathbf{a}_{l-1}(i)(\mathbf{a}_{l-1}(i))^\top\right] \otimes \mathbb{E}_i \left[G_l(i)\right] := A_l \otimes G_l$$

Based on the Kronecker-factored structural approximation, $H^l = H_a^l \bigotimes H_g^l$ as the QN approximation to $\mathbb{E}_i[\nabla^2 f_l(i)]^{-1}$, where H_a^l and H_g^l are positive definite approximations to A_l^{-1} and G_l^{-1}, respectively. Using this layer-wise block-diagonal approximation to the Hessian, a step in the algorithm for each layer l is computed as

$$\text{vec}(W_l^+) - \text{vec}(W_l) = -\alpha H^l \text{vec}\left(\widehat{\nabla \mathbf{f}}_l\right) = -\alpha(H_a^l \otimes H_g^l)\text{vec}\left(\widehat{\nabla \mathbf{f}}_l\right) = -\alpha \text{vec}\left(H_g^l \widehat{\nabla \mathbf{f}}_l H_a^l\right)$$

where $\widehat{\nabla f_l}$ denotes the estimate to $\mathbb{E}_i[\nabla f_l(i)]$ and α is the learning rate. After computing W_l^+ and performing another forward/backward pass, the method computes or updates H_a^l and H_g^l as follows:

1. For H_g^l, we use a damped version of BFGS (or L-BFGS) (See Section 3) based on the $(\mathbf{s}, \mathbf{y})$ pairs corresponding to the average change in $\mathbf{h}_l(i)$ and in the gradient with respect to $\mathbf{h}_l(i)$; i.e.,

$$\mathbf{s}_g^l = \mathbb{E}_i[\mathbf{h}_l^+(i)] - \mathbb{E}_i[\mathbf{h}_l(i)], \qquad \mathbf{y}_g^l = \mathbb{E}_i[\mathbf{g}_l^+(i)] - \mathbb{E}_i[\mathbf{g}_l(i)].$$

2. For H_a^l we use the "Hessian-action" BFGS method described in Section 4. The issue of possible singularity of the positive semi-definite matrix A_l approximated by $(H_a^l)^{-1}$ is also addressed there by incorporating a **Levenberg-Marquardt (LM)** damping term.

Algorithm 2 High-level summary of K-BFGS / K-BFGS(L)

Require: Given initial weights θ, batch size m, learning rate α
1: **for** $k = 1, 2, \dots$ **do**
2: Sample mini-batch of size m: $M_k = \{\xi_{k,i}, i = 1, \dots, m\}$
3: Perform a forward-backward pass over the current mini-batch M_k (see Algorithm 1)
4: **for** $l = 1, \dots, L$ **do** $p_l = H_g^l \widehat{\nabla} \mathbf{f}_l H_a^l$; $W_l = W_l - \alpha \cdot p_l$
5: Perform another forward-backward pass over M_k to get $(\mathbf{s}_g^l, \mathbf{y}_g^l)$
6: Use damped BFGS or L-BFGS to update H_g^l $(l = 1, ..., L)$ (see Section 3, in particular Algorithm 3)
7: Use Hessian-action BFGS to update H_a^l $(l = 1, ..., L)$ (see Section 4)

BFGS and L-BFGS for Gl

Damped BFGS Updating. It is well-known that training a DNN is a non-convex optimization problem. This non-convexity manifests in the fact that $G_l \succ 0$ often does not hold. Thus, for the BFGS update of H_l , the approximation to G^{-1}, to remain positive-definite, it is important to ensure $(s_g^l)^T y_g^l > 0$. Due to the stochastic setting, ensuring this condition by line-search, as is done in deterministic settings, is impractical. In addition, due to the large changes in curvature in DNN models that occur as the parameters are varied, large changes to H_g^l as it is updated are also to be supressed. To deal with both of these issues, a double damping (DD) procedure (Algorithm 3) is proposed, which is based upon Powell's damped-BFGS approach, for modifying the (s_g^l, y_g^l) pair. To motivate Algorithm 3, consider the formulas used for BFGS updating of B and H :

$$B^+ = B - \frac{B\mathbf{s}\mathbf{s}^\top B}{\mathbf{s}^\top B\mathbf{s}} + \rho \mathbf{y}\mathbf{y}^\top, \quad H^+ = (I - \rho \mathbf{s}\mathbf{y}^\top) H (I - \rho \mathbf{y}\mathbf{s}^\top) + \rho \mathbf{s}\mathbf{s}^\top,$$

where $\rho = \frac{1}{\mathbf{s}^\top \mathbf{y}} > 0$. If we can ensure that $0 < \frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \le \frac{1}{\mu_1}$ and $0 < \frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \le \frac{1}{\mu_2}$, then we can obtain the following bounds:

$$\begin{aligned} \|B^+\| &\le \|B - \frac{B\mathbf{s}\mathbf{s}^\top B}{\mathbf{s}^\top B\mathbf{s}}\| + \|\rho \mathbf{y}\mathbf{y}^\top\| \le \|B\| + \|\frac{B^{1/2} H^{1/2} \mathbf{y}\mathbf{y}^\top H^{1/2} B^{1/2}}{\mathbf{s}^\top \mathbf{y}}\| \\ &\le \|B\| + \|B\| \frac{\|H^{1/2}\mathbf{y}\|^2}{\mathbf{s}^\top \mathbf{y}} \le \|B\| \left(1 + \frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}}\right) \le \|B\| \left(1 + \frac{1}{\mu_1}\right) \end{aligned}$$

and

$$\begin{aligned} \|H^+\| &\le \|H^{1/2} - \frac{\mathbf{s}\mathbf{y}^\top H^{1/2}}{\mathbf{s}^\top \mathbf{y}}\|^2 + \|\frac{\mathbf{s}\mathbf{s}^\top}{\mathbf{s}^\top \mathbf{y}}\| \le \left(\|H^{1/2}\| + \frac{\|\mathbf{s}\| \|H^{1/2}\mathbf{y}\|}{\mathbf{s}^\top \mathbf{y}}\right)^2 + \frac{\|\mathbf{s}\|^2}{\mathbf{s}^\top \mathbf{y}} \\ &\le \left(\|H^{1/2}\| + (\frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}})^{1/2} (\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}})^{1/2}\right)^2 + \frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \le \left(\|H^{1/2}\| + \frac{1}{\sqrt{\mu_1 \mu_2}}\right)^2 + \frac{1}{\mu_2}. \end{aligned}$$

Algorithm 3 Double Damping (DD)

1: **Input:** $\mathbf{s}, \mathbf{y}$; **Output:** $\tilde{\mathbf{s}}, \tilde{\mathbf{y}}$; **Given:** H, μ_1, μ_2
2: **if** $\mathbf{s}^\top \mathbf{y} < \mu_1 \mathbf{y}^\top H \mathbf{y}$ **then** $\theta_1 = \frac{(1-\mu_1)\mathbf{y}^\top H \mathbf{y}}{\mathbf{y}^\top H \mathbf{y} - \mathbf{s}^\top \mathbf{y}}$ **else** $\theta_1 = 1$
3: $\tilde{\mathbf{s}} = \theta_1 \mathbf{s} + (1-\theta_1) H \mathbf{y}$ {Powell's damping on H}
4: **if** $\tilde{\mathbf{s}}^\top \mathbf{y} < \mu_2 \tilde{\mathbf{s}}^\top \tilde{\mathbf{s}}$ **then** $\theta_2 = \frac{(1-\mu_2)\tilde{\mathbf{s}}^\top \tilde{\mathbf{s}}}{\tilde{\mathbf{s}}^\top \tilde{\mathbf{s}} - \tilde{\mathbf{s}}^\top \mathbf{y}}$ **else** $\theta_2 = 1$
5: $\tilde{\mathbf{y}} = \theta_2 \mathbf{y} + (1-\theta_2)\tilde{\mathbf{s}}$ {Powell's damping with $B = I$}
6: **return** $\tilde{s}, \tilde{y}$

Levenberg-Marquardt Damping for A_l . Since $A_l = \mathbb{E}_i[(a_{l-1}(i)(a_{l-1}(i))^T)] \succeq 0$ may not be positive definite, or may have very small positive eigenvalues, an Levenberg-Marquardt (LM) damping term is added to make the Hessian-action BFGS stable; i.e., $A_l + \lambda_A I_A$ is used instead of A_l, when H_a^l is updated. Specifically, Hessian action BFGS for A_l is performed as

1. $A_l = \beta \cdot A_l + (1-\beta) \cdot \mathbb{E}_i \left[\mathbf{a}_{l-1}(i)\mathbf{a}_{l-1}(i)^\top\right]$; $A_l^{\text{LM}} = A_l + \lambda_A I_A$.
2. $\mathbf{s}_a^l = H_a^l \cdot \mathbb{E}_i[\mathbf{a}_{l-1}(i)]$, $\mathbf{y}_a^l = A_l^{\text{LM}} \mathbf{s}_a^l$; use BFGS with $(\mathbf{s}_a^l, \mathbf{y}_a^l)$ to update H_a^l.

The pseudo-code for the entire algorithm is as follows.

Algorithm 4 Pseudocode for K-BFGS / K-BFGS(L)

Require: Given initial weights $\theta = \left[\text{vec}(W_1)^\top, \ldots, \text{vec}(W_L)^\top\right]^\top$, batch size m, learning rate α, damping value λ, and for K-BFGS(L), the number of $(\mathbf{s}, \mathbf{y})$ pairs p that are stored and used to compute H_g^l at each iteration
1: $\mu_1 = 0.2$, $\beta = 0.9$ {set default hyper-parameter values}
2: $\lambda_A = \lambda_G = \sqrt{\lambda}$ {split the damping into A and G}
3: $\widehat{\nabla \mathbf{f}}_l = 0$, $A_l = \mathbb{E}_i\left[\mathbf{a}_{l-1}(i)\mathbf{a}_{l-1}(i)^\top\right]$ by forward pass, $H_a^l = (A_l + \lambda_A I_A)^{-1}$, $H_g^l = I$ $(l = 1, ..., L)$ {Initialization}
4: **for** $k = 1, 2, \ldots$ **do**
5: Sample mini-batch of size m: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$
6: Perform a forward-backward pass over the current mini-batch M_k to compute $\overline{\nabla \mathbf{f}}_l$, $\mathbf{a}_l$, $\mathbf{h}_l$, and $\mathbf{g}_l$ $(l = 1, \ldots, L)$ (see Algorithm 1)
7: **for** $l = 1, \ldots, L$ **do**
8: $\widehat{\nabla \mathbf{f}}_l = \beta \widehat{\nabla \mathbf{f}}_l + (1-\beta)\overline{\nabla \mathbf{f}}_l$
9: $p_l = H_g^l \widehat{\nabla \mathbf{f}}_l H_a^l$
10: {In K-BFGS(L), when computing $H_g^l \left(\widehat{\nabla \mathbf{f}}_l H_a^l\right)$, L-BFGS is initialized with an identity matrix}
11: $W_l = W_l - \alpha \cdot p_l$
12: Perform another forward-backward pass over M_k to compute $\mathbf{h}_l^+$, $\mathbf{g}_l^+$ $(l = 1, \ldots, L)$
13: **for** $l = 1, ..., L$ **do**
14: {Use damped BFGS or L-BFGS to update H_g^l (see Section 3)}
15: $\mathbf{s}_g^l = \beta \cdot \mathbf{s}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{h}_l^+} - \overline{\mathbf{h}_l}\right)$, $\mathbf{y}_g^l = \beta \cdot \mathbf{y}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{g}_l^+} - \overline{\mathbf{g}_l}\right)$
16: $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l) = \text{DD}(\mathbf{s}_g^l, \mathbf{y}_g^l)$ with $H = H_g^l$, $\mu_1 = \mu_1$, $\mu_2 = \lambda_G$ {See Algorithm 3}
17: Use BFGS or L-BFGS with $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l)$ to update H_g^l
18: {Use Hessian-action BFGS to update H_a^l (see Section 4)}
19: $A_l = \beta \cdot A_l + (1-\beta) \cdot \overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$
20: $A_l^{\text{LM}} = A_l + \lambda_A I_A$
21: $\mathbf{s}_a^l = H_a^l \cdot \overline{\mathbf{a}_{l-1}}$, $\mathbf{y}_a^l = A_l^{\text{LM}} \mathbf{s}_a^l$
22: Use BFGS with $(\mathbf{s}_a^l, \mathbf{y}_a^l)$ to update H_a^l

Implementation Details

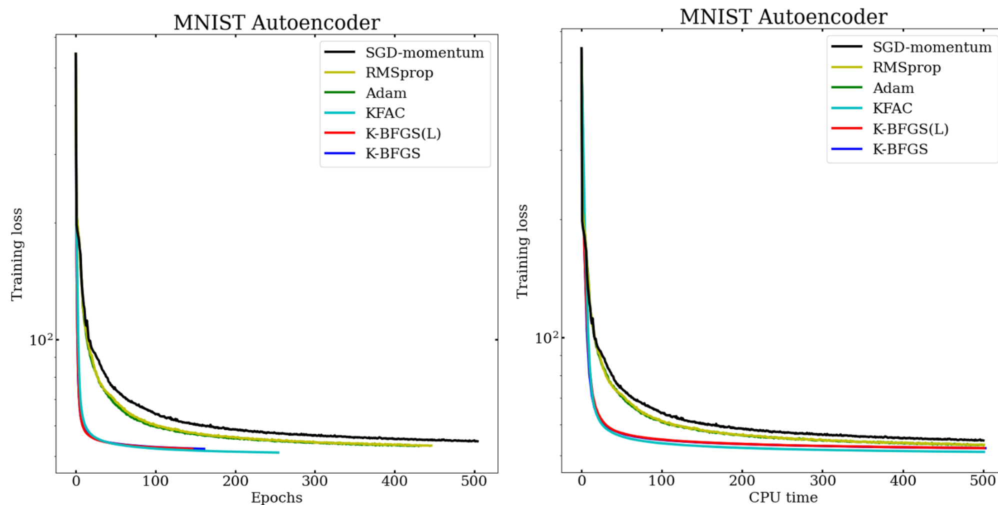
For the experimentation part, we used the autoencoders with the same dimensions and loss functions as those in the paper. Following are the details as given in the paper:

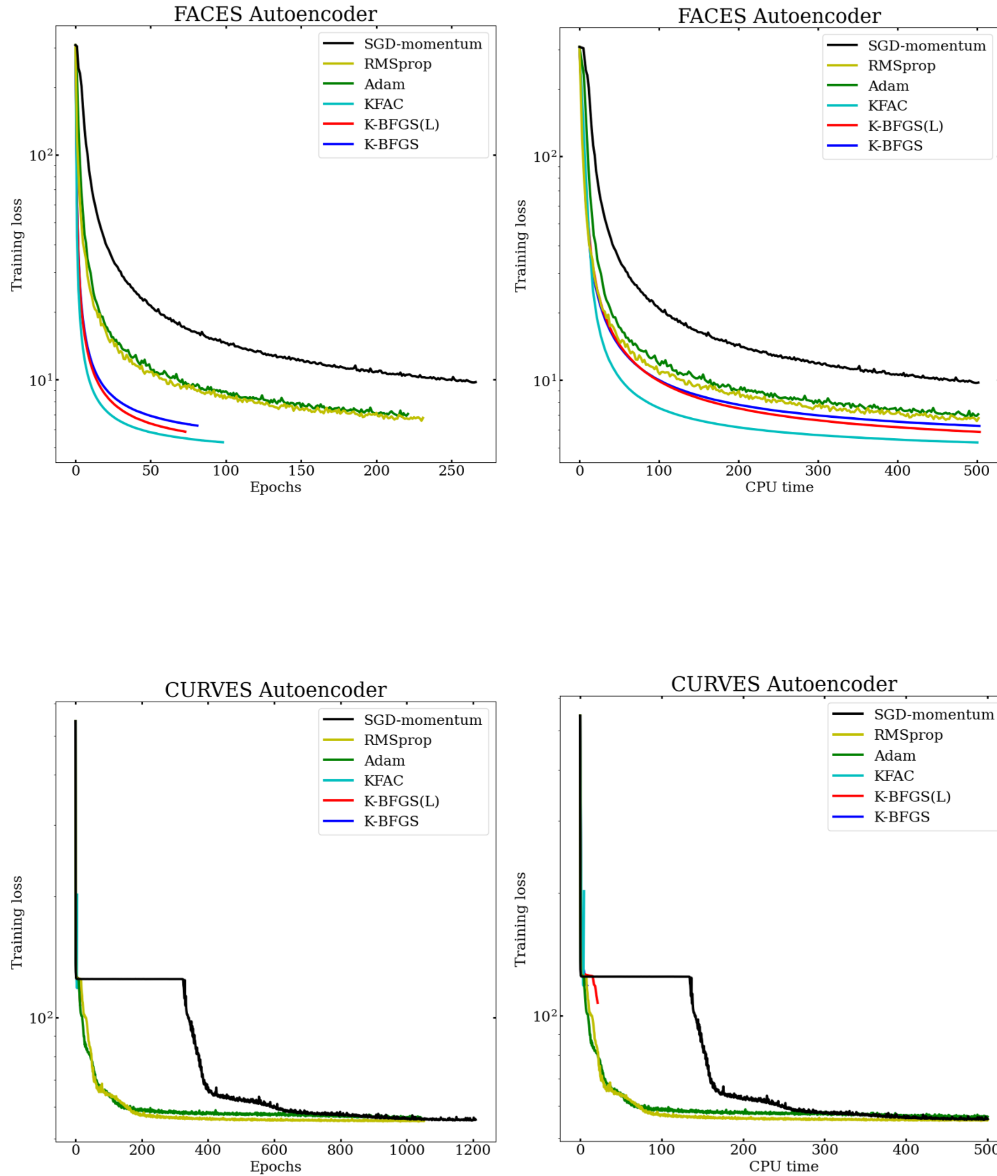
Dataset	Layer width & activation	Loss function
MNIST	[784, 1000, 500, 250, 30, 250, 500, 1000, 784] [ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, sigmoid]	binary entropy
FACES	[625, 2000, 1000, 500, 30, 500, 1000, 2000, 625] [ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, linear]	MSE
CURVES	[784, 400, 200, 100, 50, 25, 6, 25, 50, 100, 200, 400, 784] [ReLU, ReLU, ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, ReLU, ReLU, sigmoid]	binary entropy

The task here is to reconstruct the input of the model. For example, in the case of MNIST we input the flattened vector of pixel values and reconstruct these values in the output. The gradients are automatically calculated using the backward() function in PyTorch. The implementation uses these gradients to compute the rest of the expressions.

Results

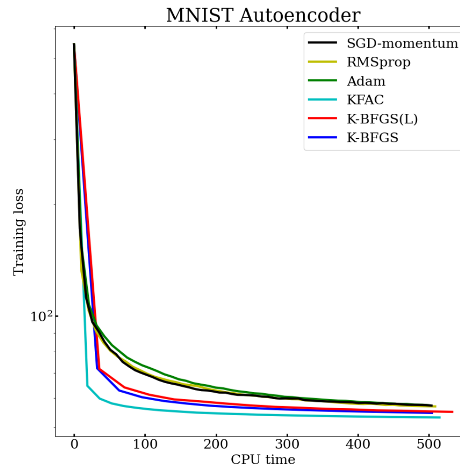
Our first task was to replicate the main results from the paper. We obtained the plots as follows: Each row corresponds to a dataset. The left column shows the training loss with respect to epochs and the right column shows the training loss with respect to CPU time. We bounded the time to 500 CPU seconds so that we could compare the performances for the same amount of compute. The hyperparameters are the same as those recommended in the paper.





These plots match very closely with those in the paper. There are a few minor differences but the trends and relative performances are exactly the same.

We also wanted to test the suitability of the second order methods in more “stochastic” situations, that is, when the mini batch size is much smaller. We reran the training with a mini batch size of 100 and the best set of hyperparameters for this batch size as mentioned in the appendix of the paper. Following is the resulting plot for the MNIST dataset.



This again matches closely with the authors' plots. It is evident from the plots that the second order methods are holding up well even at lower mini batch sizes. This is especially important from a practical perspective because for high dimensional datasets, it might be necessary to use much smaller mini batch sizes.

Open Questions

Here are some next steps to try:

We can study the convergence analysis of general Kronecker Factor-based methods. We are currently exploring one specific method but it can be generalized across multiple approaches.

Moreover, in the paper, the authors have focussed only on fully connected autoencoders. We can possibly extend and evaluate the implementation of this method to Convolutional Neural Networks and Recurrent Neural Networks.

Contribution

Atharva Anand Joshi: Ideation and Planning, Implementation, Experimentation, Literature Survey
Ketan Ramaneti: Theoretical Study, Evaluation, Experimentation, Literature Survey

References

1. Donald Goldfarb, Yi Ren, and Achraf Bahamou. 2020. Practical quasi-newton methods for training deep neural networks. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20). Curran Associates Inc., Red Hook, NY, USA, Article 201, proceedings.neurips.cc/paper/2020/file/192fc044e74dffea144f9ac5dc9f3395-Paper
2. A. Mokhtari and A. Ribeiro, "RES: Regularized Stochastic BFGS Algorithm," in *IEEE Transactions on Signal Processing*, vol. 62, no. 23, pp. 6089-6104, Dec.1, 2014, doi: 10.1109/TSP.2014.2357775.
3. https://www.jmlr.org/papers/volume14/hennig13a/hennig13a.pdf
   Philipp Hennig (MPI Intelligent Systems), Martin Kiefel (MPI for Intelligent Systems), "**Quasi-Newton Methods: A New Direction",** International Journal of Machine Learning Research
4. Broyden, C. G.. "Quasi-Newton methods and their application to function minimisation." Mathematics of Computation 21 (1967): 368-381.
5. A. S. Berahas, M. Jahani, P. Richtárik & M. Takáč (2022) Quasi-Newton methods for machine learning: forget the past, just sample, Optimization Methods and Software, 37:5, 1668-1704, DOI: 10.1080/10556788.2021.1977806