Trust-Region Minimization Algorithm for Training Responses (TRMinATR): The Rise of Machine Learning Techniques

Jacob Rafati
Electrical Engineering and Computer Science
University of California, Merced
Merced, CA 95343 USA

Omar DeGuchy
Applied Mathematics
University of California, Merced
Merced, CA 95343 USA

Roummel F. Marcia
Applied Mathematics
University of California, Merced
Merced, CA 95343 USA

***Abstract*—Deep learning is a highly effective machine learning technique for large-scale problems. The optimization of non-convex functions in deep learning literature is typically restricted to the class of first-order algorithms. These methods rely on gradient information because of the computational complexity associated with the second derivative Hessian matrix inversion and the memory storage required in large scale data problems. The reward for using second derivative information is that the methods can result in improved convergence properties for problems typically found in a non-convex setting such as saddle points and local minima. In this paper we introduce TRMinATR – an algorithm based on the limited memory BFGS quasi-Newton method using trust region – as an alternative to gradient descent methods. TRMinATR bridges the disparity between first order methods and second order methods by continuing to use gradient information to calculate Hessian approximations. We provide empirical results on the classification task of the MNIST dataset and show robust convergence with preferred generalization characteristics.**

***Index Terms*—Quasi-Newton methods, Limited-memory BFGS, Trust-region methods, Line-search methods, Deep learning.**

I. Introduction

Deep learning continues to emerge as a leading technique for solving problems in a variety of fields including computer vision, natural language processing and signal recovery [1], [2]. As the collection of large quantities of data become standard practice, the need to improve neural network performance in an increasing number of applications becomes imperative. While architecture and computational resources play an important role in the effectiveness of neural networks, the methodology used in training these networks leaves an opportunity to increase efficiency. Typically, gradient based optimization methods are used in order to minimize what is commonly referred to as the loss function in a supervised learning setting. The goal is to minimize the disparity between the outcome provided by the neural network and the intended result. This paper proposes the Trust-Region Minimization Algorithm for Training Responses (TRMinATR) as an alternative to gradient descent methods.

This research is supported by National Science Foundation Grants CMMI Award 1333326 and IIS 1741490.

A. Related Methods

Gradient descent based algorithms such as stochastic gradient descent (SGD) have emerged as popular methods for training deep neural networks [3]–[5]. Although they are preferred for their ease of implementation and their relatively low computational costs, they are not without their challenges. These types of methods are heavily dependent on the tuning and initialization of network parameters. As a result, complex algorithms have been developed to overcome these restrictions by incorporating adaptive learning rates, but continue to be sensitive to the structure of the data [4], [6], [7]. As an alternative to gradient descent, limited memory quasi-Newton algorithms with line search have been implemented in a deep learning setting [8]. These methods approximate second derivative information improving the quality of each training iteration and circumvent the need for application specific parameter tuning. The novelty of TRMinATR is in the use of the L-BFGS quasi-Newton method in a trust-region setting to train deep neural networks. TRMinATR solves the associated trust-region subproblem, which can be computationally intensive in large scale problems, by efficiently computing a closed form solution at each iteration.

II. Problem formulation

In this section we will summarize the formulation of the optimization problem involved in training deep neural networks. In particular we will discuss the need for the minimization of the cost function as the motivation for the methods presented in the next section.

We begin with a generalized construct of a feed forward deep neural network. The defining characteristic of this type of architecture is the number of hidden layers and the type of layers used (convolutional, fully connected, etc.) [1]. In order to describe the relationship between layers we adopt a more generalized convention than that presented in [10]. Each layer is described by the following expression $h^{(i)} = \theta^{(i)}(h^{(i-1)}, W^{(i)})$ for $i = 1 \dots n$, where n is the number of layers. Using this convention we define $h^{(i-1)}$ as the input to the current layer and $h^{(i)}$ as the output of the current layer. The operator $\theta^{(i)}$ corresponds to the architecture of the layer

ISBN 978-90-827970-1-5 © EURASIP 2018

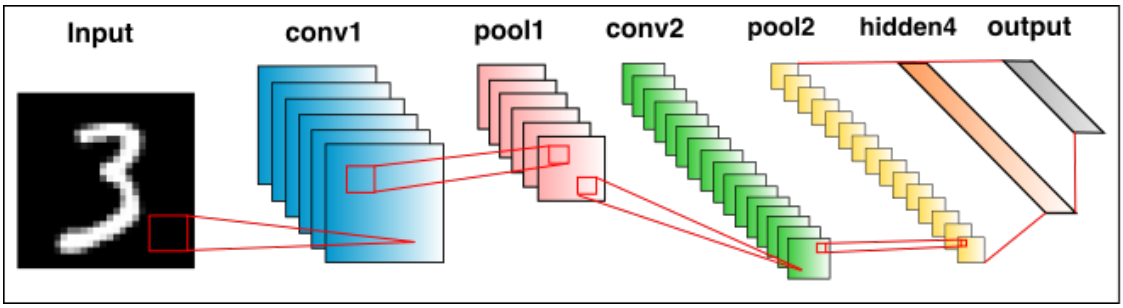

Fig. 1. A LeNet deep learning network inspired by the architecture found in [9] . The neural network is used in the classification of the MNIST dataset of hand written digits. The convolutional neural network (CNN) uses convolutions followed by pooling layers for feature extraction. The final layer transforms the information into the required probability distribution.

and $W^{(i)}$ corresponds to the parameters of the layer which are adjusted during training. We refrain from defining the dimensions of the layer components as they are particular to the task being completed. For example, the choice of layer architecture θ can include fully connected layers, convolutions, pooling layers and autoencoders [10]. Typically layers also include an activation function such as a rectified linear unit (ReLU), sigmoidal function or the softmax function [11], [12]. Having incorporating indices to the layer components we can represent a neural network with a depth of n layers as

$$\begin{aligned} h^{(1)} &= \theta^{(1)}(h^{(0)}, W^{(1)}), \\ h^{(2)} &= \theta^{(2)}(h^{(1)}, W^{(2)}), \\ &\vdots \\ h^{(n)} &= \theta^{(n)}(h^{(n-1)}, W^{(n)}), \end{aligned}$$

During training of the neural network we pass the information $h^{(0)}$ through the layers and obtain the network's approximation of the desired output $h^{(n)}$. We seek to adjust the weights $W^{(i)}$ in order to improve the quality of the approximate output $h^{(n)}$. Learning is accomplished by minimizing the loss function $\Phi(h, h^{(n)})$. Here the letter h without a superscript is the intended output. In image recovery this can be a clear image, in classification this is signified by a label. Once again the choice of loss function is determined by the task assigned to the neural network. In problems pertaining to image classification, the loss function is expressed as the cross entropy function

$$\Phi(h, h^{(n)}) = -\sum_i h_i \log(h_i^{(n)}), \tag{1}$$

where h and $h^{(n)}$ are distributions of the likelihood that the image is correctly classified [10]. Other applications such as denoising neural networks use the Mean Squared Error (MSE) to compare images with their reconstructions [13]. Because $h^{(n)}$ is a function of $W = (W^{(1)}, W^{(2)}, \ldots, W^{(n)})$ then we seek to

$$\min_W \ \Phi(h, h^{(n)}), \tag{2}$$

using backpropagation [14].

III. METHODOLOGY

In this section, we outline two methods used to solve the following unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} \Phi(x). \tag{3}$$

Both methods seek to minimize the objective function $\Phi(x)$ by defining a sequence of iterates $\{x_k\}$ which are governed by the search direction p_k. Each respective method is defined by its approach to computing the search direction p_k with minimizing the quadratic model of the objective function defined by

$$q_k(p) \triangleq g_k^T p + \frac{1}{2} p^T B_k p, \tag{4}$$

where $g_k \triangleq \nabla\Phi(x_k)$ and B_k is an approximation to $\nabla^2\Phi(x_k)$.

A. Line Search Method

Each iteration of the line search method computes search direction p_k by solving optimization subproblem

$$p_k = \arg\min_{p \in \mathbb{R}^n} q_k(p), \tag{5}$$

and then decides how far to move along p_k by choosing a *step length* α_k. The iteration x_k updates by following relation:

$$x_{k+1} = x_k + \alpha_k p_k. \tag{6}$$

Usually p_k is required to be a descent direction and $\alpha_k \in (0, 1]$ is chosen to satisfy the sufficient decrease and curvature conditions, e.g. Wolfe conditions [15]:

$$\Phi(x_k + \alpha_k p_k) \leq \Phi(x_k) + c_1 \alpha_k \nabla\Phi_k^T p_k, \tag{7a}$$

$$\nabla\Phi(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla\Phi(x_k)^T p_k, \tag{7b}$$

with $0 < c_1 < c_2 < 1$. The general pseudo-code for line search method is given in Algorithm 1 (see [15] for details).

B. Trust-Region Methods

The trust-region method solves (3) using the localized quadratic approximation of the objective function q_k defined in (4) at each iteration.

$$p_k = \arg\min_{p \in \mathbb{R}^n} q_k(p) \quad \text{subject to} \quad \|p\|_2 \leq \delta_k, \tag{8}$$

ISBN 978-90-827970-1-5 © EURASIP 2018

Algorithm 1 Line search method pseudo-code.

Input: starting point x_0, tolerance $\epsilon > 0$
  $k \leftarrow 0$
  repeat
    compute $g_k = \nabla\Phi(x_k)$
    update L-BFGS matrix B_k
    compute search direction p_k by solving (5)
    find α_k that satisfies Wolfe Conditions in (7)
    $k \leftarrow k + 1$
  until $\|g\| < \epsilon$ or k reached to max number of iterations

where δ_k denotes the radius of the trust region. There is a computational bottleneck associated with solving (8) in large-scale optimization. This is the type of problem associated with training neural networks. These computational costs will be addressed in a later section.

Solving the trust-region subproblem to high accuracy requires consideration of the problem's optimality conditions for a global solution. Methods such as those presented in [16]–[18] make use of the following theorem:

Theorem 1: Let δ be a positive constant. A vector p^* is a global solution of the trust-region subproblem (8) if and only if $\|p^*\|_2 \leq \delta$ and there exists a unique $\sigma^* \geq 0$ such that $B + \sigma^* I$ is positive semidefinite and

$$(B + \sigma^* I)p^* = -g \quad \text{and} \quad \sigma^*(\delta - \|p^*\|_2) = 0. \tag{9}$$

Moreover, if $B + \sigma^* I$ is positive definite, then the global minimizer is unique.

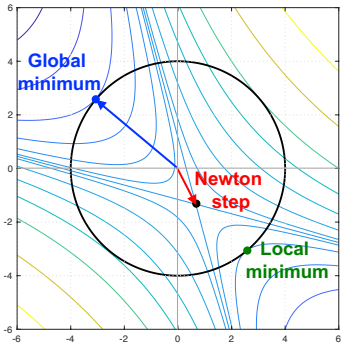

Fig 2. An illustration of trust-region methods. For indefinite matrices, the Newton step (in red) leads to a saddle point. The global minimizer (in blue) is characterized by the conditions in Eq. (9) with $B+\sigma^* I$ positive semidefinite. In contrast, local minimizers (in green) satisfy Eq. (9) with $B + \sigma^* I$ not positive semidefinite.

The general pseudo-code for trust region method is given in Algorithm 2. (see [15], [18] for details).

C. Quasi-Newton Methods

Methods that use $B_k = \nabla^2\Phi(x_k)$ for the Hessian in the quadratic model in (4) typically exhibit quadratic rates of convergence. However, there are several assumptions needed to ensure this approach is computationally feasible. First, solves

Algorithm 2 Trust region method pseudo-code.

Input: starting point x_0, tolerance $\epsilon > 0$, $\hat{\delta} > 0$,
    $\delta_0 \in (0, \hat{\delta})$, $\eta \in [0, 1/4)$
  $k \leftarrow 0$
  repeat
    compute $g_k = \nabla\Phi(x_k)$
    update L-BFGS matrix B_k
    compute search direction p_k by solving (8)
    $\rho_k \leftarrow (\Phi(x_k) - \Phi(x_k + p_k))/(q_k(0) - q_k(p_k))$
    update trust-region radius δ_k
    if $\rho_k > \eta$ **then**
      $x_{k+1} = x_k + p_k$
    else
      $x_{k+1} = x_k$
    end if
    $k \leftarrow k + 1$
  until $\|g_k\| < \epsilon$ or k reached to max number of iterations

with the matrix $\nabla^2\Phi(x_k)$ must be done efficiently. For large-scale problems, unless $\nabla^2\Phi(x_k)$ has structure that can easily be exploited, this is generally not the case. Second, $\nabla^2\Phi(x_k)$ must have the positive eigenvalues so that the resulting search direction p_k is guaranteed to be a descent direction. Third, $\nabla^2\Phi(x_k)$ must be computationally available. In applications such as inverse problems, computing $\nabla^2\Phi(x_k)$ requires solving a system of partial differential equations, which may or may not have an expression for the Hessian. In cases where using the Hessian matrix $\nabla^2\Phi(x_k)$ is not practical, quasi-Newton methods are viable alternatives because they exhibit superlinear convergence rates while maintaining memory and computational efficiency.

Perhaps the most well-known among all of the quasi-Newton methods is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update [15], [19], given by

$$B_{k+1} = B_k - \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T, \tag{10}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla\Phi(x_{k+1}) - \nabla\Phi(x_k)$. The matrices are defined recursively with the initial B_0 taken to be a $B_0 = \gamma I$, where the scalar $\gamma > 0$. In practice, only the m most-recently computed pairs $\{(s_k, y_k)\}$ are stored, where $m \ll n$, typically $m \leq 100$ for very large problems. This approach is often referred to as *limited-memory* BFGS, or L-BFGS. Because these updates are low-rank, the matrix B_{k+1} can be compactly represented as $B_{k+1} = B_0 + \Psi_k M_k \Psi_k^T$, for some $\Psi_k \in \Re^{n\times 2(k+1)}$ and $M_k \in \Re^{2(k+1)\times 2(k+1)}$. In particular,

$$\Psi_k = \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \text{ and } M_k = -\begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1},$$

where $S_k = [\, s_0 \;\; s_1 \;\; s_2 \;\; \cdots \;\; s_k \,] \in \Re^{n\times(k+1)}$, and $Y_k = [\, y_0 \;\; y_1 \;\; y_2 \;\; \cdots \;\; y_k \,] \in \Re^{n\times(k+1)}$, and L_k is the strictly lower triangular part and D_k is the diagonal part of the matrix

ISBN 978-90-827970-1-5 © EURASIP 2018

$S_k^T Y_k \in \Re^{(k+1)\times(k+1)}$, i.e., $S_k^T Y_k = L_k + D_k + U_k$, where U_k is a strictly upper triangular matrix (see [20] for details).

Given the compact representation of B_{k+1}, then both the line search problem (5) and the trust-region subproblem (8) can be efficiently solved when the L-BFGS matrix is used as the Hessian approximation. In particular, the solution to (4) is given by $p_k^* = -\frac{1}{\gamma}\left[I - \Psi_k(\gamma M_k^{-1} + \Psi_k^T\Psi_k)^{-1}\Psi_k^T\right] g_k$, using the well-known Sherman-Morrison-Woodbury formula. Similarly, the solution to the trust-region subproblem is obtained efficiently. First, the QR factorization of $\Psi_k = QR$ is formed. Then the eigendecomposition of the $2(k+1)\times 2(k+1)$ matrix $RM_kR^T = V\Lambda V^T$ is computed so that the partial eigendecomposition of $B_{k+1} = \gamma I + QV\Lambda V^T Q^T$ is obtained. This allows for a change in variables in (8) that yields a closed form expression for the solution p_k^* (see Algorithm 1 in [21] for details).

IV. Numerical Experiments

In this section we compare the line search L-BFGS optimization method with our proposed Trust-Region Minimization Algorithm for Training Responses (TRMinATR). The goal of the experiment is to perform the optimization necessary for neural network training. Both methods are implemented to train the LeNet-5 architecture with the purpose of image classification of the MNIST dataset. All simulations were performed on an AWS EC2 p2.xlarge instance with 1 Tesla K80 GPU, 64 GiB memory, and 4 Intel 2.7 GHz Broadwell processors. For the scalars c_1 and c_2 in the Wolfe line search condition, we used the typical values of $c_1 = 10^{-4}$ and $c_2 = 0.9$ [15]. All codes are implemented in TensorFlow and available at `https://github.com/root-master/lbfgs-tr`.

A. LeNet-5

The convolutional neural network known as LeNet-5 was mainly used for character recognition tasks such as reading zip codes and digits [9]. The architecture is given in Table I. The convolutional layers extract features from the input image and preserve spatial relationships between pixels using the learned information.

B. MNIST Dataset

The convolutional neural network was trained and tested using the MNIST Dataset [22]. The dataset consists of 70,000 examples of handwritten digits with 60,000 examples used as a training set and 10,000 examples used as a test set. The digits range from 0 - 9 and their sizes have been normalized to 28x28 pixel images. The images include labels describing their intended classification.

C. Results

The line search algorithm and TRMinATR perform comparably in terms of loss and accuracy. This remains consistent with different choices of the memory parameter m (see Fig. 4). The more interesting comparison is that of the training accuracy and the test accuracy, the two metrics follow each other closely. This is unlike the typical results using common gradient descent based optimization. Typically the test accuracy is delayed in achieving the same results as the train accuracy. This would suggest that the model has a better chance of being generalized beyond the training data.

We also report that the TRMinATR significantly improves on the computational efficiency of the line-search method when using larger batch sizes. This could be the result of the line-search method's need to satisfy certain wolfe conditions at each iteration. There is also an associated computational cost when verifying that the conditions for sufficient decrease are being met. When the batch size decreases, the trust-region method continues to outperform the line-search method. This is especially true when less information is used in the Hessian approximation (see Fig. 3).

TABLE I
Structure of the LeNet5 convolutional neural network trained on the MNIST dataset.

LeNet-5	
Layer	NVIDIAConnectivity
0: input	28×28 image
1	convolutional, 20 5×5 filters (stride=1), total 11520 neurons, followed by ReLU
2	max pool, 2×2 window (stride=2), total 2280 neurons
3	convolutional, 50 5×5 filters (stride=1), total 3200 neurons, followed by ReLU
4	max pool, 2×2 window (stride=2), total 800 neurons
5	fully connected, 500 neurons without dropout followed by ReLU
6: output	fully connected, 10 neurons without dropout followed by softmax
total of 431080 trainable parameters	

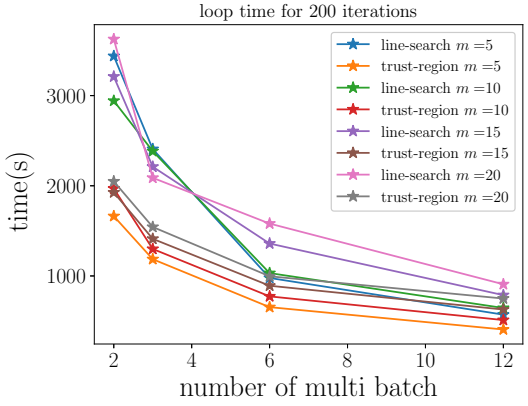

Fig 3. We compare the loop time for 200 iterations of the line-search and trust-region quasi-Newton algorithms for different batch sizes. As the number of multi batches increase, the size of each batch decreases. Both methods were tested using different values of the memory parameter m.

V. Conclusions

In this paper we present the limited memory quasi-Newton method known as L-BFGS as an alternative to the gradient descent methods used to train deep neural networks. In particular we develop the algorithm known as TRMinATR which minimizes the cost function of the neural network

ISBN 978-90-827970-1-5 © EURASIP 2018

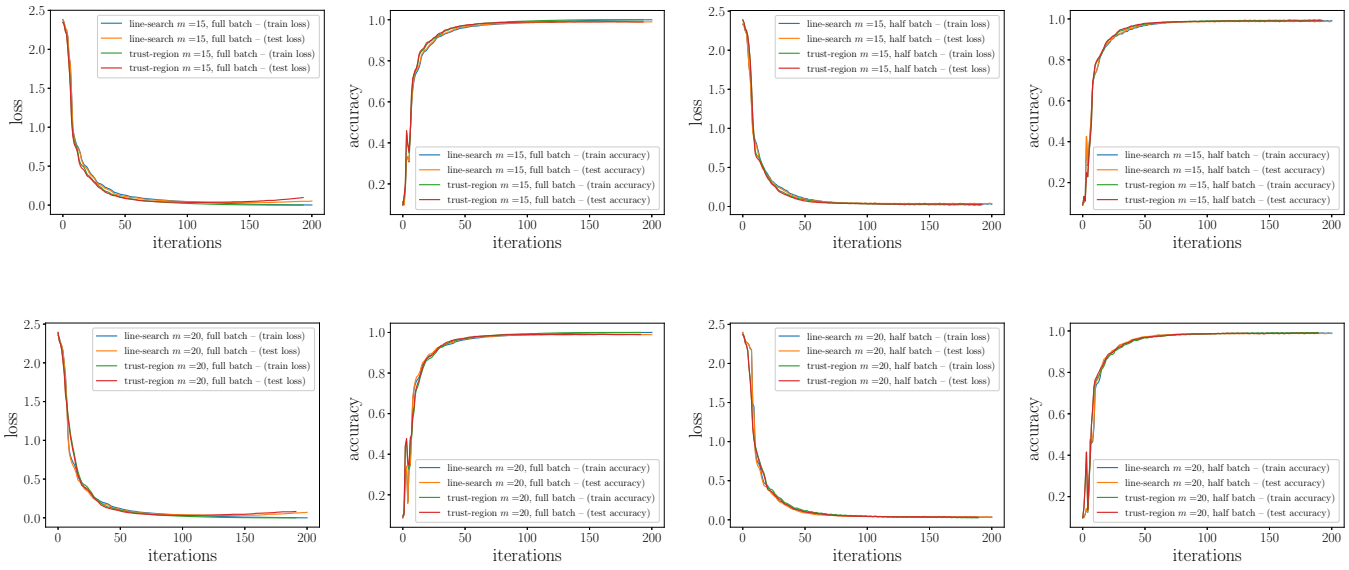

Fig 4. The behavior of the loss and accuracy for the training and test sets. (a) Training and testing using the full training set as a batch size. (b) Training using batch sizes that are half of the size of the full set. Results are shown using typical memory settings ($m = 15$ and $m = 20$) in an L-BFGS setting.

by efficiently solving a sequence of trust-region subproblems using low-rank Hessian approximations. The benefit of the method is that the algorithm is free from the constraints of data specific parameters seen in traditionally used methods. TRMinATR also improves on the computational efficiency of a similar line search implementation. In the future we hope to apply TRMinATR to deep learning architectures designed to complete tasks in applications such as cancer detection [23] or speech recognition [24].

Acknowledgments. The authors would like to thank Prof. Harish Bhat, whose course on the mathematics of deep learning help facilitate this research.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[3] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[4] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[5] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[7] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[8] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 265–272.

[9] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, p. 20, 2015.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press Cambridge, 2016.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[12] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[13] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[14] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[16] D. M. Gay, "Computing optimal locally constrained steps," *SIAM Journal on Scientific and Statistical Computing*, vol. 2, no. 2, pp. 186–197, 1981.

[17] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 3, pp. 553–572, 1983.

[18] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2000.

[19] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, pp. 503–528, 1989.

[20] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited-memory methods," *Math. Program.*, vol. 63, pp. 129–156, 1994.

[21] L. Adhikari, O. DeGuchy, J. B. Erway, S. Lockhart, and R. F. Marcia, "Limited-memory trust-region methods for sparse relaxation," in *Proc.SPIE*, vol. 10394, 2017, pp. 10 394 – 10 394 – 8. [Online]. Available: https://doi.org/10.1117/12.2271369

[22] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[23] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Mitosis detection in breast cancer histology images with deep neural networks," in *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer, 2013, pp. 411–418.

[24] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

ISBN 978-90-827970-1-5 © EURASIP 2018