Pushing Stochastic Gradient towards Second-Order Methods – Backpropagation Learning with Transformations in Nonlinearities

Tommi Vatanen[1], Tapani Raiko[1], Harri Valpola[1], and Yann LeCun[2]

[1] Department of Information and Computer Science
Aalto University School of Science
P.O. Box 15400, FI-00076, Aalto, Espoo, Finland
first.last@aalto.fi
[2] New York University
715 Broadway, New York, NY 10003, USA
firstname@cs.nyu.edu

Abstract. Recently, we proposed to transform the outputs of each hidden neuron in a multi-layer perceptron network to have zero output and zero slope on average, and use separate shortcut connections to model the linear dependencies instead. We continue the work by firstly introducing a third transformation to normalize the scale of the outputs of each hidden neuron, and secondly by analyzing the connections to second order optimization methods. We show that the transformations make a simple stochastic gradient behave closer to second-order optimization methods and thus speed up learning. This is shown both in theory and with experiments. The experiments on the third transformation show that while it further increases the speed of learning, it can also hurt performance by converging to a worse local optimum, where both the inputs and outputs of many hidden neurons are close to zero.

Keywords: Multi-layer perceptron network, deep learning, stochastic gradient.

1 Introduction

Learning deep neural networks has become a popular topic since the invention of unsupervised pretraining [3]. Some later works have returned to traditional back-propagation learning in deep models and noticed that it can also provide impressive results [5] given either a sophisticated learning algorithm [8] or simply enough computational power [2]. In this work we study back-propagation learning in deep networks with up to five hidden layers, continuing on our earlier results in [9].

In learning multi-layer perceptron (MLP) networks by back-propagation, there are known transformations that speed up learning [7, 10, 11]. For instance, inputs are recommended to be centered to zero mean (or even whitened), and nonlinear functions are proposed to have a range from -1 to 1 rather than 0 to 1 [7]. Schraudolph [11, 10] proposed centering all factors in the gradient to have zero mean, and further adding linear shortcut connections that bypass the nonlinear layer. Gradient factor centering changes the gradient as if the nonlinear activation functions had zero mean and zero slope on

M. Lee et al. (Eds.): ICONIP 2013, Part I, LNCS 8226, pp. 442–449, 2013.

average. As such, it does not change the model itself. It is assumed that the discrepancy between the model and the gradient is not an issue, since the errors will be easily compensated by the linear shortcut connections in the proceeding updates. Gradient factor centering leads to a significant speed-up in learning.

In this paper, we transform the nonlinear activation functions in the hidden neurons such that they have on average 1) zero mean, 2) zero slope, and 3) unit variance. Our earlier results in [9] included the first two transformations and here we introduce the third one. We explain the usefulness of these transformations by studying the Fisher information matrix and the Hessian, e.g. by measuring the angle between the traditional gradient and a second order update direction with and without the transformations.

It is well known that second-order optimization methods such as the natural gradient [1] or Newton's method decrease the number of required iterations compared to the basic gradient descent, but they cannot be easily used with high-dimensional models due to heavy computations with large matrices. In practice, it is possible to use a diagonal or block-diagonal approximation [6] of the Fisher information matrix or the Hessian. Gradient descent can be seen as an approximation of the second-order methods, where the matrix is approximated by a scalar constant times a unit matrix. Our transformations aim at making the Fisher information matrix as close to such matrix as possible, thus diminishing the difference between first and second order methods. Extended version of this paper with the experimental details can be found in arXiv [12] and Matlab code for replicating the experiments is available at

`https://github.com/tvatanen/ltmlp-neuralnet`

2 Proposed Transformations

Let us study a MLP-network with a single hidden layer and shortcut mapping, that is, the output column vectors $\mathbf{y}_t$ for each sample t are modeled as a function of the input column vectors $\mathbf{x}_t$ with

$$\mathbf{y}_t = \mathbf{A}\mathbf{f}\left(\mathbf{B}\mathbf{x}_t\right) + \mathbf{C}\mathbf{x}_t + \boldsymbol{\epsilon}_t, \tag{1}$$

where $\mathbf{f}$ is a nonlinearity (such as $\tanh$) applied to each component of the argument vector separately, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are the weight matrices, and $\boldsymbol{\epsilon}_t$ is the noise which is assumed to be zero mean and Gaussian, that is, $p(\epsilon_{it}) = \mathcal{N}\left(\epsilon_{it}; 0, \sigma_i^2\right)$. In order to avoid separate bias vectors that complicate formulas, the input vectors are assumed to have been supplemented with an additional component that is always one.

Let us supplement the $\tanh$ nonlinearity with auxiliary scalar variables α_i, β_i, and γ_i for each nonlinearity f_i. They are updated before each gradient evaluation in order to help learning of the other parameters $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. We define

$$f_i(\mathbf{b}_i\mathbf{x}_t) = \gamma_i\left[\tanh(\mathbf{b}_i\mathbf{x}_t) + \alpha_i\mathbf{b}_i\mathbf{x}_t + \beta_i\right], \tag{2}$$

where $\mathbf{b}_i$ is the ith row vector of matrix $\mathbf{B}$. We will ensure that

$$\sum_{t=1}^{T} f_i(\mathbf{b}_i\mathbf{x}_t) = 0, \ \sum_{t=1}^{T} f_i'(\mathbf{b}_i\mathbf{x}_t) = 0, \ \text{and} \tag{3}$$

$$\left[\sum_{t=1}^{T} \frac{f_i(\mathbf{b}_i\mathbf{x}_t)^2}{T}\right]\left[\sum_{t=1}^{T} \frac{f_i'(\mathbf{b}_i\mathbf{x}_t)^2}{T}\right] = 1 \tag{4}$$

by setting α_i, β_i, and γ_i to

$$\alpha_i = -\frac{1}{T}\sum_{t=1}^{T} \tanh'(\mathbf{b}_i\mathbf{x}_t) \tag{5}$$

$$\beta_i = -\frac{1}{T}\sum_{t=1}^{T} [\tanh(\mathbf{b}_i\mathbf{x}_t) + \alpha_i\mathbf{b}_i\mathbf{x}_t] \tag{6}$$

$$\gamma_i = \left\{\frac{1}{T}\sum_{t=1}^{T} [\tanh(\mathbf{b}_i\mathbf{x}_t) + \alpha_i\mathbf{b}_i\mathbf{x}_t + \beta_i]^2\right\}^{1/4}\left\{\frac{1}{T}\sum_{t=1}^{T} \left[\tanh'(\mathbf{b}_i\mathbf{x}_t) + \alpha_i\right]^2\right\}^{1/4}. \tag{7}$$

One way to motivate the first two transformations in Equations (3a) and (3b), is to study the expected output $\mathbf{y}_t$ and its dependency of the input $\mathbf{x}_t$:

$$\frac{1}{T}\sum_t \mathbf{y}_t = \mathbf{A}\frac{1}{T}\sum_t \mathbf{f}(\mathbf{B}\mathbf{x}_t) + \mathbf{C}\frac{1}{T}\sum_t \mathbf{x}_t \tag{8}$$

$$\frac{1}{T}\sum_t \frac{\partial \mathbf{y}_t}{\partial \mathbf{x}_t} = \mathbf{A}\left[\frac{1}{T}\sum_t \mathbf{f}'(\mathbf{B}\mathbf{x}_t)\right]\mathbf{B}^T + \mathbf{C}. \tag{9}$$

We note that by making nonlinear activations $\mathbf{f}(\cdot)$ zero mean in Eq. (3a), we disallow the nonlinear mapping $\mathbf{A}\mathbf{f}(\mathbf{B}\cdot)$ to affect the expected output $\mathbf{y}_t$, that is, to compete with the bias term. Similarly, by making the nonlinear activations $\mathbf{f}(\cdot)$ zero slope in Eq. (3b), we disallow the nonlinear mapping $\mathbf{A}\mathbf{f}(\mathbf{B}\cdot)$ to affect the expected dependency of the input, that is, to compete with the linear mapping $\mathbf{C}$. In traditional neural networks, the linear dependencies (expected $\partial\mathbf{y}_t/\partial\mathbf{x}_t$) are modeled by many competing paths from an input to an output (e.g. via each hidden unit), whereas our architecture gathers the linear dependencies to be modeled only by $\mathbf{C}$. We argue that less competition between parts of the model will speed up learning.

Transformations can also be motivated by observing that they make the non-diagonal parts of the Fisher information matrix closer to zero [9] and keep the diagonal of the Fisher information matrix similar in scale [12].

The goal of Equation (4) is to normalize both the output signals (similarly as data is often normalized as a preprocessing step – see,e.g., [7]) and the slopes of the output signals of each hidden unit at the same time. This is motivated by observing that the diagonal of the Fisher information matrix contains elements with both the signals and their slopes. By these normalizations, we aim pushing these diagonal elements more

similar to each other. As we cannot normalize both the signals and the slopes to unity at the same time, we normalize their geometric mean to unity.

The effect of the first two transformations can be compensated exactly by updating the shortcut mapping $\mathbf{C}$ by

$$\begin{aligned}\mathbf{C}_{\text{new}} = \mathbf{C}_{\text{old}} &- \mathbf{A}(\boldsymbol{\alpha}_{\text{new}} - \boldsymbol{\alpha}_{\text{old}})\mathbf{B} \\ &- \mathbf{A}(\boldsymbol{\beta}_{\text{new}} - \boldsymbol{\beta}_{\text{old}})\,[0 \;\; 0 \ldots 1]\,,\end{aligned} \tag{10}$$

where $\boldsymbol{\alpha}$ is a matrix with elements α_i on the diagonal and one empty row below for the bias term, and $\boldsymbol{\beta}$ is a column vector with components β_i and one zero below for the bias term. The third transformation can be compensated by

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}}\boldsymbol{\gamma}_{\text{old}}\boldsymbol{\gamma}_{\text{new}}^{-1}, \tag{11}$$

where $\boldsymbol{\gamma}$ is a diagonal matrix with γ_i as the diagonal elements.

Schraudolph [11, 10] proposed centering the factors of the gradient to zero mean. It was argued that deviations from the gradient fall into the linear subspace that the shortcut connections operate in, so they do not harm the overall performance. Transforming the nonlinearities as proposed in this paper has a similar effect on the gradient. Equation (3a) corresponds to Schraudolph's *activity centering* and Equation (3b) corresponds to *slope centering*.

3 Empirical Comparison to a Second-Order Method

Here we investigate how linear transformations affect the gradient by comparing it to a second-order method, namely Newton's algorithm with a simple regularization to make the Hessian invertible.

We compute an approximation of the Hessian matrix using finite difference method, in which case k-th row vector $\mathbf{h}_k$ of the Hessian matrix $\mathbf{H}$ is given by

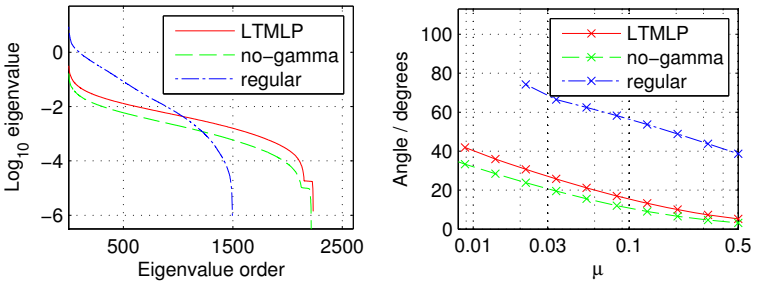
$$\mathbf{h}_k = \frac{\partial(\nabla E(\boldsymbol{\theta}))}{\partial\theta_k} \approx \frac{\nabla E(\boldsymbol{\theta} + \delta\boldsymbol{\phi}_k) - \nabla E(\boldsymbol{\theta} - \delta\boldsymbol{\phi}_k)}{2\delta}, \tag{12}$$

where $\boldsymbol{\phi}_k = (0, 0, \ldots, 1, \ldots, 0)$ is a vector of zeros and 1 at the k-th position, and the error function $E(\boldsymbol{\theta}) = -\sum_t \log p(\mathbf{y}_t \mid \mathbf{x}_t, \boldsymbol{\theta})$. The resulting Hessian might still contain some very small or even negative eigenvalues which cause its inversion to blow up. Therefore we do not use the Hessian directly, but include a regularization term similarly as in the Levenberg-Marquardt algorithm, resulting in a second-order update direction

$$\Delta\boldsymbol{\theta} = (\mathbf{H} + \mu\mathbf{I})^{-1}\nabla E(\boldsymbol{\theta}), \tag{13}$$

where $\mathbf{I}$ denotes the unit matrix. Basically, Equation (13) combines the steepest descent and the second-order update rule in such a way, that when μ gets small, the update direction approaches the Newton's method and vice versa.

Computing the Hessian is computationally demanding and therefore we have to limit the size of the network used in the experiment. We study the MNIST handwritten digit classification problem where the dimensionality of the input data has been reduced to



(a) Eigenvalues (b) Angles

Fig. 1. Comparison of (a) distributions of the eigenvalues of Hessians (2600×2600 matrix) and (b) angles compared to the second-order update directions using LTMLP and regular MLP. In (a), the eigenvalues are distributed most evenly when using LTMLP. (b) shows that gradients of the transformed networks point to the directions closer to the second-order update.

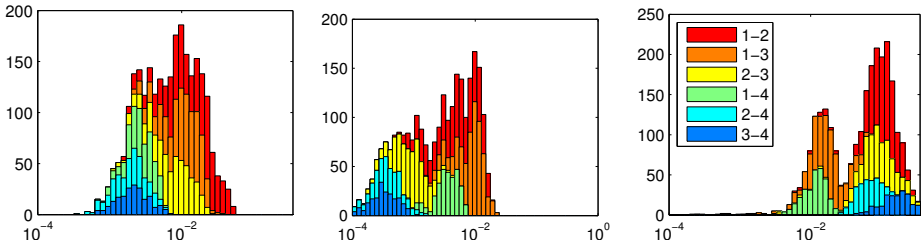
30 using PCA with a random rotation [9]. We use a network with two hidden layers with architecture 30–25–20–10. The network was trained using the standard gradient descent with weight decay regularization. Details of the training are given in [12].

In what follows, networks with all three transformations (*LTMLP*, linearly transformed multi-layer perceptron network), with two transformations (*no-gamma* where all γ_i are fixed to unity) and a network with no transformations (*regular*, where we fix $\alpha_i = 0$, $\beta_i = 0$, and $\gamma_i = 1$) were compared. The Hessian matrix was approximated according to Equation (12) 10 times in regular intervals during the training of networks. All figures are shown using the approximation after 4000 epochs of training, which roughly corresponds to the midpoint of learning. However, the results were parallel to the reported ones all along the training.

We studied the eigenvalues of the Hessian matrix (2600×2600) and the angles between the methods compared and second-order update direction. in the same training phase, after epoch number 4000. The distribution of eigenvalues in Figure 1a for the networks with transformations are more even compared to the regular MLP. Furthermore, there are fewer negative eigenvalues, which are not shown in the plot, in the transformed networks. In Figure 1b, the angles between the gradient and the second-order update direction are compared as a function of μ in Equation (13). The plots are cut when $\mathbf{H} + \mu\mathbf{I}$ ceases to be positive definite as μ decreases. Curiously, the update directions are closer to the second-order method, when γ is left out, suggesting that γs are not necessarily useful in this respect.

Figure 2 shows histograms of the diagonal elements of the Hessian after 4000 epochs of training. All the distributions are bimodal, but the distributions are closer to unimodal when transformations are used (subfigures (a) and (b))[1]. Furthermore, the variance of the diagonal elements in log-scale is smaller when using LTMLP, $\sigma_{\mathrm{a}}^2 = 0.90$, compared to the other two, $\sigma_{\mathrm{b}}^2 = 1.71$ and $\sigma_{\mathrm{c}}^2 = 1.43$. This suggests that when transformations are used, the second-order update rule in Equation (13) corrects different elements of the

[1] It can be also argued whether (a) is more unimodal compared to (b).



(a) LTMLP (b) no-gamma (c) regular

Fig. 2. Comparison of distributions of the diagonal elements of Hessians. Coloring according to legend in (c) shows which layers to corresponding weights connect (1 = input, 4 = output). Diagonal elements are most concentrated in LTMLP and more spread in the networks without γ (no-gamma, regular). Notice the logarithmic x-axis.

gradient vector more evenly compared to a regular back-propagation learning, implying that the gradient vector is closer to the second-order update direction when using all the transformations.

To conclude this section, there is no clear evidence in way or another whether the addition of γ benefits the back-propagation learning with only α and β. However, there are some differences between these two approaches. In any case, it seems clear that transforming the nonlinearities benefits the learning compared to the standard back-propagation learning.

4 Experiments: MNIST Classification

We use the proposed transformations for training MLP networks for MNIST classification task. Experiments are conducted without pretraining, weight-sharing, enhancements of the training set or any other known tricks to boost the performance. No weight decay is used and as only regularization we add Gaussian noise with $\sigma = 0.3$ to the training data. Networks with two and three hidden layers with architechtures 784–800–800–10 (solid lines) and 784–400–400–400–10 (dashed lines) are used. Details are given in [12].

Figure 3 shows the results as number of errors in classifying the test set of 10 000 samples. The results of the regular back-propagation without transformations, shown in blue, are well in line with previously published result for this task. When networks with same architecture are trained using the proposed transformations, the results are improved significantly. However, adding γ in addition to previously proposed α and β does not seem to affect results on this data set. The best results, 112 errors, is obtained by the smaller architecture without γ and for the three-layer architecture with γ the result is 114 errors. The learning seems to converge faster, especially in the three-layer case, with γ. The results are in line what was obtained in [9] where the networks were regularized more thoroughly. These results show that it is possible to obtain results comparable to dropout networks (see [4]) using only minimal regularization.

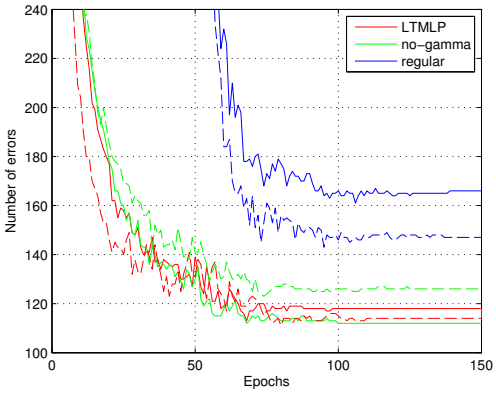


Fig. 3. The error rate on the MNIST test set for LTMLP training, LTMLP without γ and regular back-propagation. The solid lines show results for networks with two hidden layers of 800 neurons and the dashed lines for networks with three hidden layers of 400 neurons.

5 Discussion and Conclusions

We have shown that introducing linear transformation in nonlinearities significantly improves the back-propagation learning in (deep) MLP networks. In addition to two transformation proposed earlier in [9], we propose adding a third transformation in order to push the Fisher information matrix closer to unit matrix (apart from its scale). The hypothesis proposed in [9], that the transformations actually mimic a second-order update rule, was confirmed by experiments comparing the networks with transformations and regular MLP network to a second-order update method. However, in order to find out whether the third transformation, γ, we proposed in this paper, is really useful, more experiments ought to be conducted. It might be useful to design experiments where convergence is usually very slow, thus revealing possible differences between the methods. As hyperparameter selection and regularization are usually nuisance in practical use of neural networks, it would be interesting to see whether combining dropouts [4] and our transformations can provide a robust framework enabling training of robust neural networks in reasonable time.

The effect of the first two transformations is very similar to gradient factor centering [11, 10], but transforming the model instead of the gradient makes it easier to generalize to other contexts: When learning by by Markov chain Monte Carlo, variational Bayes, or by genetic algorithms, one would not compute the basic gradient at all. For instance, consider using the Metropolis algorithm on the weight matrices, and expecially matrices $\mathbf{A}$ and $\mathbf{B}$. Without transformations, the proposed jumps would affect the expected output $\mathbf{y}_t$ and the expected linear dependency $\partial \mathbf{y}_t / \partial \mathbf{x}_t$ in Eqs. (8)–(9), thus often leading to low acceptance probability and poor mixing. With the proposed transformations included, longer proposed jumps in $\mathbf{A}$ and $\mathbf{B}$ could be accepted, thus mixing the nonlinear part of the mapping faster. For further discussion, see [9], Section 6. The implications of the proposed transformations in these other contexts are left as future work.

References

[1] Amari, S.: Natural gradient works efficiently in learning. Neural Computation 10(2), 251–276 (1998)
[2] Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep big simple neural nets excel on handwritten digit recognition. CoRR, abs/1003.0358 (2010)
[3] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006)
[4] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580 (2012)
[5] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks (2012)
[6] Le Roux, N., Manzagol, P.A., Bengio, Y.: Topmoumoute online natural gradient algorithm. In: Advances in Neural Information Processing Systems 20, NIPS 2007 (2008)
[7] LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient backProp. In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, pp. 9–48. Springer, Heidelberg (1998)
[8] Martens, J.: Deep learning via Hessian-free optimization. In: Proceedings of the 27th International Conference on Machine Learning, ICML (2010)
[9] Raiko, T., Valpola, H., LeCun, Y.: Deep learning made easier by linear transformations in perceptrons. Journal of Machine Learning Research - Proceedings Track 22, 924–932 (2012)
[10] Schraudolph, N.N.: Accelerated gradient descent by factor-centering decomposition. Technical Report IDSIA-33-98, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (1998)
[11] Schraudolph, N.N.: Centering neural network gradient factors. In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, pp. 207–548. Springer, Heidelberg (1998)
[12] Vatanen, T., Raiko, T., Valpola, H., LeCun, Y.: Pushing stochastic gradient towards second-order methods – backpropagation learning with transformations in nonlinearities (pre-print, 2013), http://arxiv.org/abs/1301.3476