Deep Learning Made Easier by Linear Transformations in Perceptrons

Tapani Raiko
Aalto University

Harri Valpola
Aalto University

Yann LeCun
New York University

Abstract

We 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. This transformation aims at separating the problems of learning the linear and nonlinear parts of the whole input-output mapping, which has many benefits. We study the theoretical properties of the transformation by noting that they make the Fisher information matrix closer to a diagonal matrix, and thus standard gradient closer to the natural gradient. We experimentally confirm the usefulness of the transformations by noting that they make basic stochastic gradient learning competitive with state-of-the-art learning algorithms in speed, and that they seem also to help find solutions that generalize better. The experiments include both classification of small images and learning a low-dimensional representation for images by using a deep unsupervised auto-encoder network. The transformations were beneficial in all cases, with and without regularization and with networks from two to five hidden layers.

1 Introduction

Learning deep neural networks has become a popular topic since the invention of unsupervised pretraining [5]. Some later works have returned to traditional back-propagation learning and noticed that it can also provide impressive results given either a sophisticated learning algorithm [11] or simply enough computational power [3]. In this work we study back-propagation learning in deep networks with up to five hidden layers.

In learning multi-layer perceptron (MLP) networks by back-propagation, there are known transformations that speed up learning [10, 13, 14]. 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 [10]. Schraudolph [14, 13] proposed centering all factors in the gradient to have zero mean. This lead to a significant speed-up in learning when using shortcut connections. In this paper, we transform the nonlinearities in the hidden neurons. The effect is very similar to gradient factor centering, but transforming the model instead of the gradient makes it easier to generalize to other contexts such as variational Bayes. We explain the usefulness of these transformations by studying the Fisher information matrix.

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 [8] of the Fisher information matrix. If one has to approximate most of the matrix with zeros anyway, we should use transformations that move these elements as close to zero as possible.

2 Proposed Transformations

Let us study an MLP-network with a single hidden layer[1] and a shortcut mapping, that is, the output column vectors $\mathbf{y}_t$ for each sample t are modelled as a function of the input column vectors $\mathbf{x}_t$ with

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

[1] The assumption is done for notational simplicity only, the method is applied in the general deep case.

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

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 and β_i for each nonlinearity f_i. They are not learnt, but instead they will be set in a manner to help learn the other parameters. We define

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

where $\mathbf{b}_i$ is the ith row vector of matrix $\mathbf{B}$. An example f_i can be seen in Figure 1. We will ensure that

$$\sum_{t=1}^{T} f_i(\mathbf{b}_i\mathbf{x}_t) = 0 \tag{3}$$

$$\sum_{t=1}^{T} f_i'(\mathbf{b}_i\mathbf{x}_t) = 0 \tag{4}$$

by setting α_i and β_i to

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

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

as shown in the appendix.

The effect of the linear transformation can be compensated exactly by updating the shortcut mapping $\mathbf{C}$ by

$$\begin{aligned}\mathbf{C}_{\text{new}} = \mathbf{C}_{\text{old}} &- \mathbf{A}(\boldsymbol{\alpha}_{\text{new}} - \boldsymbol{\alpha}_{\text{old}})\mathbf{B} \\ &- \mathbf{A}(\boldsymbol{\beta}_{\text{new}} - \boldsymbol{\beta}_{\text{old}})\left[0\ \ 0 \ldots 1\right],\end{aligned} \tag{7}$$

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.

We also emphasize making the inputs x_k zero mean (and similar in scale) as a preprocessing step (see e.g. [10]).

Schraudolph [14, 13] 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 (3) corresponds to Schraudolph's *activity centering* and Equation (4) corresponds to *slope centering*.

3 Intuitive Justification

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 large models due to heavy computations with large matrices. The natural gradient is the basic gradient multiplied from the left by the inverse of the Fisher information matrix. Using basic gradient descent can thus be seen as using the natural gradient while approximating the Fisher information with a unit matrix. We will see how the proposed transformations moves the non-diagonal elements of the Fisher information matrix closer to zero, thus making the basic gradient closer to the natural gradient.

The Fisher information matrix contains elements

$$G_{ij} = \sum_t \left\langle \frac{\partial^2 \log p(\mathbf{y}_t \mid \mathbf{x}_t, \mathbf{A}, \mathbf{B}, \mathbf{C})}{\partial\theta_i\partial\theta_j} \right\rangle, \tag{8}$$

where $\langle\cdot\rangle$ is the expectation over the Gaussian distribution of noise $\boldsymbol{\epsilon}_t$ in Equation (1), and vector $\boldsymbol{\theta}$ contains all the elements of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. Note that $\mathbf{y}_t$ is a random variable and thus the Fisher information does not depend on the output data.

These elements are:

$$\frac{\partial}{\partial a_{ij}}\frac{\partial}{\partial a_{i'j'}}\log p = \begin{cases} 0 & i' \neq i \\ -\frac{1}{\sigma_i^2}\sum_t f_j(\mathbf{b}_j\mathbf{x}_t)f_{j'}(\mathbf{b}_{j'}\mathbf{x}_t) & i' = i, \end{cases} \tag{9}$$

where a_{ij} is the ijth element of matrix $\mathbf{A}$, f_j is the jth nonlinearity, and $\mathbf{b}_j$ is the jth row vector of matrix $\mathbf{B}$. Similarly

$$\frac{\partial}{\partial b_{jk}}\frac{\partial}{\partial b_{j'k'}}\log p = -\sum_i \frac{1}{\sigma_i^2}a_{ij}a_{ij'}\sum_t f_j'(\mathbf{b}_j\mathbf{x}_t)f_{j'}'(\mathbf{b}_{j'}\mathbf{x}_t)x_{kt}x_{k't} \tag{10}$$

and

$$\frac{\partial}{\partial c_{ik}}\frac{\partial}{\partial c_{i'k'}}\log p = \begin{cases} 0 & i' \neq i \\ -\frac{1}{\sigma_i^2}\sum_t x_{kt}x_{k't} & i' = i. \end{cases} \tag{11}$$

The cross terms are

$$\frac{\partial}{\partial a_{ij}}\frac{\partial}{\partial b_{j'k}}\log p = -\frac{1}{\sigma_i^2}a_{ij'}\sum_t f_j(\mathbf{b}_j\mathbf{x}_t)f_{j'}'(\mathbf{b}_{j'}\mathbf{x}_t)x_{kt} \tag{12}$$

$$\frac{\partial}{\partial c_{ik}}\frac{\partial}{\partial a_{i'j}}\log p = \begin{cases} 0 & i' \neq i \\ -\frac{1}{\sigma_i^2}\sum_t f_j(\mathbf{b}_j\mathbf{x}_t)x_{kt} & i' = i \end{cases} \tag{13}$$

$$\frac{\partial}{\partial c_{ik}}\frac{\partial}{\partial b_{jk'}}\log p = -\frac{1}{\sigma_i^2}a_{ij}\sum_t f_j'(\mathbf{b}_j\mathbf{x}_t)x_{kt}x_{k't}. \tag{14}$$

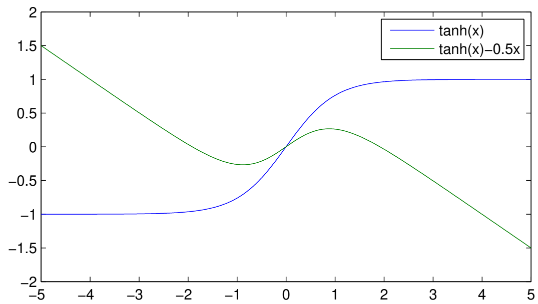

Figure 1: As a positive side effect, the nonlinearity in Equation (2) does not saturate at all for example with a typical $\alpha = -0.5$ and $\beta = 0$.

Now we can notice that Equations (9–14) contain factors such as $f_i(\cdot)$, $f_i'(\cdot)$, and x_{it}. We argue that by making the factors as close to zero as possible, we help in making nondiagonal elements of the Fisher information closer to zero. For instance, $E[f_i(\cdot)f_j(\cdot)] = E[f_i(\cdot)]E[f_j(\cdot)] + \mathrm{Cov}[f_i(\cdot), f_j(\cdot)]$, so assuming that the hidden units i and j are representing different things, that is, $f_i(\cdot)$ and $f_j(\cdot)$ are uncorrelated, the nondiagonal element of the Fisher information in Equation (9) becomes exactly zero by using the transformation. When the units are not completely uncorrelated, the element in question will be only approximately zero. The same argument applies to all other elements in Equations (10–14), some of them also highlighting the benefit of making the input data $\mathbf{x}_t$ zero-mean.

3.1 Positive Side Effect

Having a non-zero α_i has a positive side effect of reducing plateaus in learning. Typical nonlinearities like the tanh function saturate exponentially on positive and negative sides. When the derivative of an activation $f_i'(\cdot)$ is about zero for most of the data samples, the gradient propagated through it also becomes almost zero, and learning can proceed very slowly or even seem to stop completely. This may explain plateaus in typical learning curves, where the learning proceeds slowly at times. To alleviate the problem, Glorot and Bengio [4] suggested to use the soft-sign nonlinearity that saturates more slowly, but having a non-zero α_i provides a nonlinearity that does not saturate at all. The difference is illustrated in Figure 1. In practice, α_i tends to vary from -0.8 to -0.5 as will be seen in Figure 5.

4 Practical Issues

There are many practical issues when learning MLP networks and they are addressed below.

Learning

Back-propagation learning is basically a gradient ascent algorithm to maximize the log likelihood $\log p(\{\mathbf{y}_t\}_{t=1}^T \mid \{\mathbf{x}_t\}_{t=1}^T, \boldsymbol{\theta})$ of the parameter vector $\boldsymbol{\theta}$, where the actual back-propagation corresponds to using the chain rule of derivatives and dynamic programming to compute the gradient efficiently.

The gradient is

$$g_i = \frac{\partial \langle \log p(\mathbf{y}_t \mid \mathbf{x}_t, \boldsymbol{\theta}) \rangle}{\partial \theta_i}, \tag{15}$$

where $\langle \cdot \rangle$ is the expectation over the data set. The update is

$$\theta_i \leftarrow \theta_i + \gamma g_i, \tag{16}$$

where γ_i is a learning rate.

Online Learning

It is well known (see e.g. [2]) that looking at all the available data before each update is wasteful, because many samples possess redundant information. We will use a mini-batch learning algorithm, where each update is done based on the 1000 next samples from the randomly shuffled data set. To reduce the effect of noise due to such a small sample, a momentum term is used. The update direction is thus set to

$$g_i \leftarrow 0.1 \frac{\partial \langle \log p(\mathbf{y}_t \mid \mathbf{x}_t, \boldsymbol{\theta}) \rangle}{\partial \theta_i} + 0.9 g_i \tag{17}$$

for all learned parameters, including the shortcut mappings.

The transformations parameters α_i and β_i, however are updated after the initialization and after every 1000 iterations thereafter, using the whole data set in Equations (5–6). At the same time, the changes are compensated by updating the shortcut mappings according to Equation (7) and the momentum g_i for the gradient updates is reset to zero. Using the transformations only rarely lowers the computational overhead.

Discrete Outputs

In classification problems, the output y_t is discrete and one can use the soft-max model. The Equation (1) is replaced by

$$P(y_t = i \mid \mathbf{x}_t, \boldsymbol{\theta}) = \frac{\exp\left[\mathbf{A}_i \mathbf{f}\left(\mathbf{B}\mathbf{x}_t\right) + \mathbf{C}_i \mathbf{x}_t\right]}{\sum_j \exp\left[\mathbf{A}_j \mathbf{f}\left(\mathbf{B}\mathbf{x}_t\right) + \mathbf{C}_j \mathbf{x}_t\right]}, \tag{18}$$

where $\mathbf{A}_i$ is the ith row of matrix $\mathbf{A}$. Back-propagation is done as before.

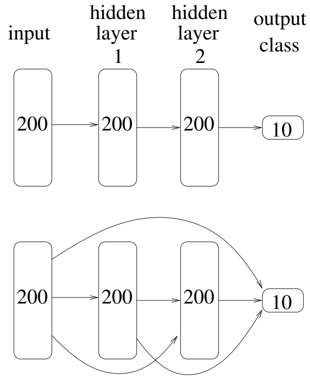

Figure 2: Top: Traditional structure for a feed-forward multi-layer perceptron network with full connections. Bottom: Network with shortcut connections included, also used in the proposed method with transformations.

Multiple Hidden Layers

The extension of the proposed approach to multiple hidden layers is straightforward. Equations (2–6) apply to all nonlinear units with $\mathbf{b}_i\mathbf{x}_t$ replaced by the appropriate input signal. The shortcut mappings need to be included for skipping any number of layers (see Figure 2 for an example). The number of weights of course increases quadratically with the number of layers, but this can be avoided by including a layer without transformations as shown in Figure 6.

Initialization

We use the initialization proposed by Glorot and Bengio [4] for weights between consecutive layers, that is, the weight is drawn from a uniform distribution between $\pm\sqrt{6}/\sqrt{n_j + n_{j+1}}$ where n_j is the number of neurons on the jth layer. This normalized initialization is based on the objectives of maintaining activation variances and back-propagated gradient's variance throughout the layers. With unnormalized initialization, the gradient tends to vanish or explode exponentially with the number of layers [4]. Biases are drawn from a uniform distribution between ± 0.5. Weights for all shortcut connections are initialized to zero.

Learning Rate

We use a hand-set learning rate γ_i for each problem. The base learning rate γ is halved for connection weights once for each layer that the connection skips. This is a heuristic to take into account that shortcut connections have a more direct influence to the final output of the network. Also we linearly decrease the learning rate to zero after half of the allocated computational time has been used.

Regularization

Overfitting is a crucial problem in a large network, and regularization is essential to make it perform well with new data. We use three different regularization methods. Firstly, the dimensionality of input data was decreased as a preprocessing step. We used principal component analysis (PCA) followed by a random rotation (see Figure 3). The motivation for the random rotation was to make each input approximately equally important. Secondly, we use weight decay (see e.g. [7]), or equivalently a Gaussian prior on the parameters $\boldsymbol{\theta}$. The final update direction becomes

$$g_i \leftarrow 0.1 \left[\frac{\partial \langle \log p(\mathbf{y}_t \mid \mathbf{x}_t, \boldsymbol{\theta}) \rangle}{\partial \theta_i} - \lambda \theta_i \right] + 0.9 g_i, \quad (19)$$

where λ is the weight decay parameter set by hand. Thirdly, we add randomly generated Gaussian noise to the original input data each time they are presented to the learner, inspired by denoising autoencoders [15] and also recently used in classification [12].

5 Experiments

We compare MLP learning with and without proposed transformations in three problems where we can also compare to other state-of-the-art learning algorithms. The two first experiments are image classification tasks, and the last one is an autoregressive model to find a low-dimensional representation of images. Even though all experiments use image data, we do not use or compare to any image-specific processing such as convolutional networks or elastic distortions. Our approach would work exactly the same even if the order of pixels was randomly permutated. All experiments were run on a desktop computer with Intel Core i7 2.93GHz and 8 gigabytes of memory.

5.1 MNIST Handwritten Digit Classification

The MNIST data set [9] consists of 28 by 28 pixel gray-scale images of handwritten digits with examples depicted in Figure 3. There are 60000 training samples and 10000 test samples. The mean activation of each pixel was subtracted from the data, and the dimensionality was dropped from $28 \times 28 = 784$ to 200 using PCA followed by a random rotation (see Figure 3). A classification network with layer sizes 200–200–200–10 was learned with a normal MLP model (original), one with shortcut mappings included (shortcuts), and the proposed model with shortcut mappings and transformations in the nonlinearities (transformations) (see Figure 2). We also ran a simple 200–10 network (*linear*) for comparison. Training and test errors were tracked during learning and their computation was not

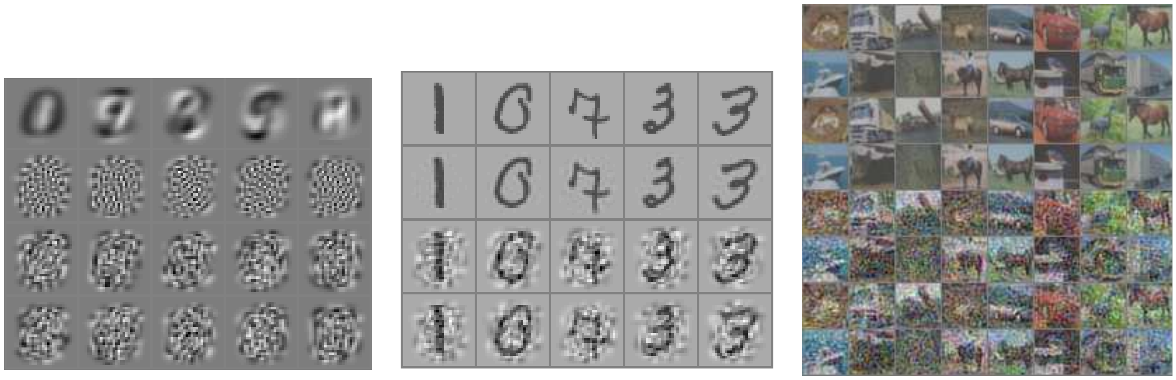

Figure 3: Left: Top row shows the PCA filters corresponding to the largest eigenvalues (1–5), second row to the smallest eigenvalues (196–200). The bottom rows show 10 filters after the random rotation in the principal subspace. Middle: Top row shows 5 examples from the MNIST handwritten digit data set. The second row shows reconstructions from the 200 component principal subspace. The third and forth row show reconstructions when including the added noise to the training data. Two instantiations of the noise is shown to remind that the noise is resampled for each epoch. Right: The corresponding images (16 examples) for the CIFAR-10 data set with 500 components.

		linear	original	shortcuts	transformations	literature
MNIST classification	training error	8.99	0.063	0.058	0.068	-
	test error	8.58	1.15	1.22	**1.10**	1.64
	learning rate	-	1.0	0.5	1.0	-
	# of iterations	30k	4717	3498	2674	-
CIFAR-10 classification	training error	58.07	23.21	22.46	4.56	-
	test error	59.09	44.42	44.99	**43.70**	48.47
	# of iterations	32k	12k	8k	8k	-
MNIST autoencoder	training error	8.11	2.37	2.11	1.94	1.75
	test error	7.85	2.76	2.61	**2.44**	2.55
	# of iterations	92k	49k	38k	37k	-

Table 1: Results of the proposed method (transformations) compared against other methods run with the same settings and against results in the literature [4, 6, 11]. The number of iterations in the allocated time is reported to compare the computational complexities.

included in learning time. Learning time was restricted to 15 minutes, weight decay parameter was $\lambda = 0.0001$ and the regularization noise was used with standard deviation 0.4 for each input (see Figure 3).

The results are shown in Table 1 and Figure 4. With a proper learning rate (1.0), the proposed method gets the test error below 1.5% in six minutes and to 1.1% in fifteen minutes. Thus, the transformations make a simple gradient descent algorithm competitive in speed with complex state-of-the-art learning methods such as TONGA [8], which reaches test error 1.7% in around 30 minutes. Deep networks learned with back-propagation have reached 1.64% error in [4]. Deep belief network [5] gives 1.20% error.

The experiments were run with an 8-core desktop computer and Matlab implementation. The learning time was measured with Matlab function cputime, which counts time spent by each core, so wall-clock learning times were smaller. For example one 900 cpu-second run took 233 seconds including computations of the training and test errors for result graphs. Table 1 reports the number of iterations reached in the allocated learning time.

One can note that the errors drop fast in the latter half of learning when the learning rate is decreased (See middle of Figure 4). This is mostly due to filtering out the noise caused by the stochastic gradient. It might seem that the comparison methods are even catching up at the end. However, when the experiment is repeated with a longer learning time (not shown here), the curves look qualitatively similar with comparison methods almost catching up at the end.

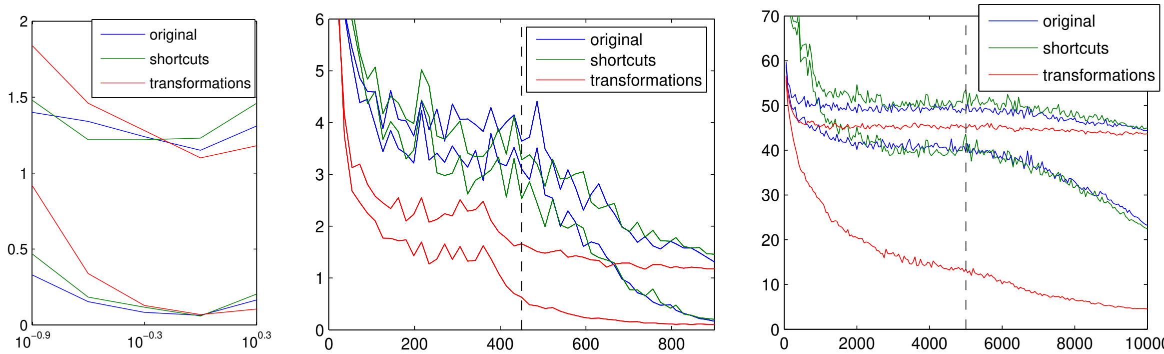

Figure 4: Left: MNIST problem. Classification error rate in percentage as a function of learning rate after 15 minutes of learning. Lower curves in each figure are the training error rates and higher curves are test error rates. The vertical dashed line shows the point at which the learning rate starts to be decreased. Middle: MNIST problem. Error rates against learning time for the best learning rates for each method. Right: CIFAR-10 problem. Error rates against learning time.

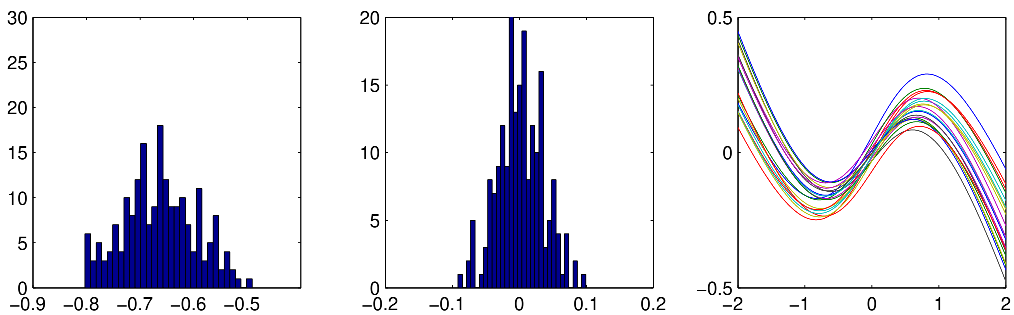

Figure 5: MNIST classification problem. Histograms of the transformation parameters α_i (left) and β_i (middle) for the first hidden layer after learning. Right: Twenty examples of corresponding nonlinearities $f_i(\cdot)$.

Let us also study some of the properties of the transformations. Figure 5 shows what the transformed nonlinearities look like in practice. How do they affect the Fisher information matrix? Equation (9) measures the covariance of the signals $f_i(\cdot)$. The ratio of mean square nondiagonal element of the covariance to mean square diagonal element drops from 0.051 to 0.007 in the first hidden layer and from 0.080 to 0.009 in the second hidden layer, when comparing models learned traditionally or with transformation. The transformations also decrease the norm of the gradient with respect to weights of adjacent layers. The decrease is about 2 to 3-fold in the initial phase of learning. This might also explain why the proposed model performed worse than the others with a too small learning rate (See left part of Figure 4). With a small norm of the gradient and a small learning rate, the optimization simply does not finish in the allocated time.

To study which regularization method was important, the runs were repeated with several variants. The table below shows test errors evaluated by including regularization methods one by one, using the learning rate $\gamma = 1.0$. The final run was repeated with ten times the learning time and $\gamma = 0.5$.

regularization	original	shortcuts	transform.
none	1.87	2.02	1.63
weight decay	1.85	1.77	1.65
PCA	1.62	1.59	1.56
rotation	1.63	1.60	1.48
input noise	1.15	1.23	1.10
long run	1.03	1.17	**1.02**

Adding noise to the inputs turned out to be the most important regularization method, followed by dimensionality reduction by PCA. Using the transformations improved all variants.

5.2 CIFAR-10 Classification

The CIFAR-10 data set [6] consists of 32 by 32 pixel color images classified to 10 different classes, with examples depicted in Figure 3. There are 50000 training samples and 10000 test samples. Each channel of each pixel was normalized to zero mean unit variance, and the dimensionality was dropped from $32 \times 32 \times 3 = 3072$ to 500 using PCA followed by a random rotation. A classification network with layer sizes 500–500–500–10 was used with the same structure and variants as in MNIST classification. Learning time was restricted to 10000 seconds, the base learning rate was set by hand to $\gamma = 0.3$, weight decay parameter was $\lambda = 0.001$ and the regularization noise was used with standard deviation 0.4 for each input (see Figure 3).

The results are shown in Table 1 and Figure 4. Earlier test errors with MLP networks include 48.47% in [6] and 52.92% in [4]. The test error of 44.42% obtained here with the original MLP is already much better and it is further improved to 43.70% by using the proposed transformations. It should be noted that even the best back-propagation results are far behind from results obtained with for instance unsupervised pretraining or convolutional networks.

Role of Regularization

The role of regularization was studied by rerunning the CIFAR-10 classification experiment without any regularization. The network size was thus 3072–500–500–10, and we dropped the learning rate to $\gamma = 0.03$ to better compare to the initialization. All three methods reached the training error 0.0%, but the test errors increased to 50.7%, 49.1%, and 46.8%. This overfitting was expected since the number of weights in the network is much larger than the number of labels in the training set, which makes the system underdetermined.

To further study the found solutions, we compute the angles between the 3072-dimensional incoming weight vectors of the neurons in the first hidden layer against the vectors that they were initialized to. The median angle over the 500 units was $39.6°$, $29.8°$, and $22.0°$ in the three models. Firstly, the angles are surprisingly small taking into account that high-dimensional vectors easily become rather orthogonal, which indicates that the found solution still retains much of the randomness of the initialization.[2] Secondly, shortcut weights seem to help against overfitting.

5.3 MNIST Autoencoder

The third problem uses the same MNIST handwritten digit data set [9], but in this case both the inputs and outputs of the network are the images. The network topology is 784–500–250–30–250–500–784 with tanh-nonlinearity at each layer except the bottleneck layer in the middle. The output is scaled from -1 to 1 to match the tanh nonlinearity. The goal in this problem is to find a good 30-dimensional representation from which the image can be reconstructed. Naturally the shortcut connections that skip the bottleneck are not used (see left part of Figure 6). The labels are not used in this experiment at all. Learning time was restricted to 60000 seconds, the base learning rate was $\gamma = 0.05$, weight decay parameter was $\lambda = 0.001$ and the regularization noise was used with standard deviation 0.1 for each input. To avoid early divergence in learning, the learning rate was increased from one hundredth to the full rate exponentially during the first one percent of learning time.

The performance is measured by the average sum of squared reconstruction errors on the test data when it is scaled back to the range from 0 to 1. The final reconstruction error is 2.44 and some reconstructions are visualized in the middle of Figure 6. As a comparison, a linear 784–30–784 autoencoder gives an error 7.85. State-of-the-art comparison results are presented by Martens [11]: Hessian-free optimization gives 2.55 with a larger network. The results in [11] were further improved to 2.28 by initializing the network with layerwise pretraining, which could be used here, too.

Typical applications of dimensionality reduction methods include visualization, data denoising, or missing value imputation, but we will show another simple demonstration by comparing them as preprocessing methods for the k nearest neighbor (kNN) classifier. By using the raw pixel data, kNN gives 2.95% test error. Using a linear network to reduce dimensionality to 30 as a preprocessing gives 2.39% error, while using the proposed model gives 1.95% error. It can be concluded that the bottleneck layer has found a representation that also better separates the clusters formed by the different classes, despite the fact that the labels were not used in learning.

6 Discussion

We proposed transformations to nonlinearities that make learning MLP networks much easier. The motivation is to make the nonlinear mapping as separate as possible from the linear mapping which is modelled using shortcut weights. A basic stochastic gradient optimization became faster than state-of-the-art learning algorithms. Those algorithms could also be tested with transformations for further improvements. The theory of the speed-up is based on making the standard gradient more similar to the natural gradient by having the nondiagonal terms of the Fisher information matrix closer to zero. As a side effect, the transformed nonlinearities might also help in avoiding plateaus [10]. The

[2] This also partly explains why initializing with unsupervised pretraining works so well in large networks.

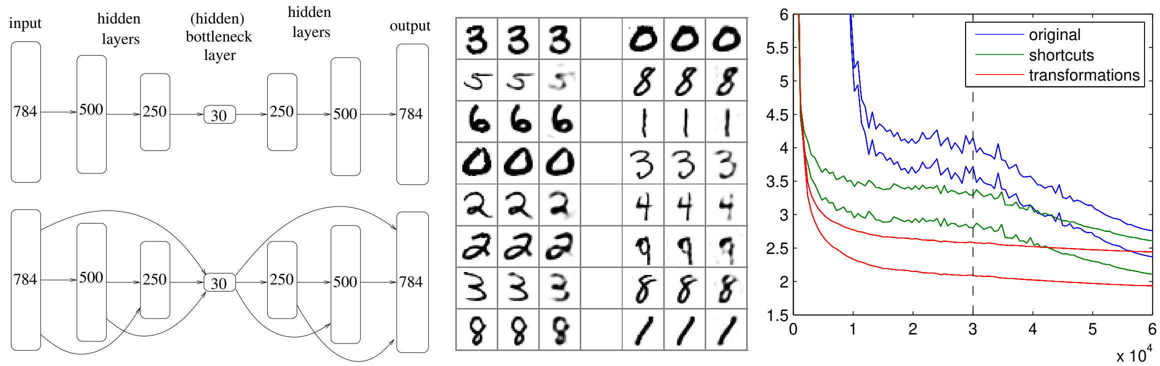

Figure 6: MNIST autoencoder. Left: Traditional autoencoder network above, shortcut connections included below. Note that the bottleneck layer does not have a nonlinearity or transformations. Middle: Each triplet shows an example digit from test data, its reconstruction with a deep autoencoder, and a reconstruction with a linear autoencoder as a comparison. Right: Error rate in average sum of squared reconstruction errors plotted against learning time in seconds. Higher curves are test errors, lower curves are training errors. The vertical dashed line shows the point at which the learning rate starts to be decreased.

experiments showed that these simple transformations helps learning deep structures at least up to 5 hidden layers using good-old back-propagation learning.

The transformations also seemed to help generalization when no regularization was used. We think that this is because the transformations help to separate the simpler and more complex parts of each mapping. Let us think of a network with just one hidden layer. The linear part of the problem (the shortcut connections **C**) do not suffer much from the overfitting and can be learned more reliably even without regularization. Overfitting the more complex parts **A** and **B** might not hurt the performance as much when they do not influence the linear part of the whole mapping. It might be possible to test this hypothesis in the future.

Another theoretical study that could be done in the future, is to measure the angle between the traditional gradient and the natural gradient with and without transformations. It would require a small network such that computing the natural gradient would be feasible.

The effect of the transformations could also be studied in other contexts. For variational Bayesian (VB) learning, the proposed transformations make both the signals in the network and the network weights less dependent a posteriori. Often they are assumed to be independent in the posterior approximation of VB anyway. Thus, the transformation makes the effect of the assumption smaller and the approximation more accurate. This should help in avoiding the problem of underfitting (or output weights of too many hidden neurons going to zero). One could also use MCMC methods for sampling network weights. The effectiveness of the sampling process depends heavily on how large jumps can be made in the parameter space. We can make longer jumps for the matrices **A** and **B** if we use the proposed transformations to ensure that even large changes in them do not affect the linear part of the input-output mapping. These new contexts would highlight the difference between transforming nonlinearities compared to transforming gradient factors [14, 13].

The term deep learning refers either to networks with many layers (as in this work) but sometimes it is used for unsupervised pretraining which allows for well-performing deeper networks in practice. The proposed transformations could also be applied to initializations based on unsupervised pretraining. We could also study a multi-task learning problem combining classification and auto-encoding. This simple alternative to layerwise pretraining might provide some useful insights about combining unsupervised and supervised learning.

A "poor man's variant" of the proposed framework would be to use shortcut connections and fixed nonlinearities such as $f(x) = \tanh(x) - 0.5x$. Dispite its simplicity, this variant might still provide most of the discussed benefits in practice.

Another future direction is to introduce a third transformation. While currently we aim at making the Fisher information matrix diagonal, we could also make it closer to the unit matrix. This could be done by using a multiplicative transformation in order to normalize the scale of the output and the slope of each nonlinearity.

References

[1] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] R. Battiti. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.

[3] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

[5] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[6] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[7] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4 (NIPS 1991)*, pages 950–957, 1992.

[8] N. Le Roux, P. A. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20 (NIPS*2007)*, 2008.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[10] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: tricks of the trade*. Springer-Verlag, 1998.

[11] J. Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[12] S. Rifai, X. Glorot, Y. Bengio, and P. Vincent. Adding noise to the input of a model trained with a regularized objective. Technical Report 1359, Université de Montréal, Montréal (QC), H3C 3J7, Canada, April 2011.

[13] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical Report IDSIA-33-98, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, 1998.

[14] N. N. Schraudolph. Centering neural network gradient factors. In Genevieve Orr and Klaus-Robert Mller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 548–548. Springer Berlin / Heidelberg, 1998.

[15] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML08)*, pages 1096–1103, 2008.

Derivations

Derivation of Equations (5–6):

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} f_i'(\mathbf{b}_i \mathbf{x}_t) = \sum_{t=1}^{T} \left[\tanh'(\mathbf{b}_i \mathbf{x}_t) + \alpha_i \mathbf{b}_i\right] \\
\Rightarrow \alpha_i &= -\frac{1}{T} \sum_{t=1}^{T} \tanh'(\mathbf{b}_i \mathbf{x}_t). \\
0 &= \sum_{t=1}^{T} f_i(\mathbf{b}_i \mathbf{x}_t) = \sum_{t=1}^{T} \left[\tanh(\mathbf{b}_i \mathbf{x}_t) + \alpha_i \mathbf{b}_i \mathbf{x}_t + \beta_i\right] \\
\Rightarrow \beta_i &= -\frac{1}{T} \sum_{t=1}^{T} \left[\tanh(\mathbf{b}_i \mathbf{x}_t) + \alpha_i \mathbf{b}_i \mathbf{x}_t\right].
\end{aligned}
$$