A Refinement Algorithm for Deep Learning via Error-Driven Propagation of Target Outputs

Vincenzo Laveglia[1,2] and Edmondo Trentin[2(✉)]

[1] DINFO, Università di Firenze, Via di S. Marta, 3, 50139 Florence, Italy
vincenzo.laveglia@unifi.it
[2] DIISM, Università di Siena, Via Roma, 56, 53100 Siena, Italy
trentin@dii.unisi.it

Abstract. Target propagation in deep neural networks aims at improving the learning process by determining target outputs for the hidden layers of the network. To date, this has been accomplished via gradient-descent or relying on autoassociative networks applied top-to-bottom in order to synthesize targets at any given layer from the targets available at the adjacent upper layer. This paper proposes a different, error-driven approach, where a regular feed-forward neural net is trained to estimate the relation between the targets at layer ℓ and those at layer $\ell - 1$ given the error observed at layer ℓ. The resulting algorithm is then combined with a pre-training phase based on backpropagation, realizing a proficuous "refinement" strategy. Results on the MNIST database validate the feasibility of the approach.

Keywords: Target propagation · Deep learning
Deep neural network · Refinement learning

1 Introduction

The impressive results attained nowadays in a number of AI applications of neural networks stem mostly from using deep architectures with proper deep learning techniques [10]. Looking under the hood, deep learning still heavily relies (explicitly or implicitly) on the traditional backpropagation (BP) algorithm [18]. While BP works outstandingly on networks having a limited number of hidden layers, several weaknesses of the algorithm emerge when dealing with significantly deep architectures. In particular, due to the non-linearity of the activation functions associated to the units in the hidden layers, the backpropagated gradients tend to vanish in the lower layers of the network, hence hindering the corresponding learning process [8]. Besides its numerical problems, BP is also known to lack any plausible biological interpretation [16].

To overcome these difficulties, researchers proposed improved learning strategies, such as pre-training of the lower layers via auto-encoders [1], the use of rectifier activation functions [9], and the dropout technique [21] to avoid neurons

L. Pancioni et al. (Eds.): ANNPR 2018, LNAI 11081, pp. 78–89, 2018.
https://doi.org/10.1007/978-3-319-99978-4_6

co-adaptation. Amongst these and other potential solutions to the aforementioned difficulties, target propagation has been arousing interest in the last few years [2,5,16], albeit it still remains an under-investigated research area. Originally proposed in [3,4] within the broader framework of learning the form of the activation functions, the idea underlying target propagation goes as follows. While in BP the delta values δ_i to be backpropagated are related to the partial derivatives of the global loss function w.r.t. the layer-specific parameters of the network, in target propagation the real target outputs (naturally defined at the output layer in regular supervised learning) are propagated downward through the network, from the topmost to the bottommost layers. In so doing, each layer gets explicit target output vectors that, in turn, define layer-specific loss functions that can be minimized locally (on a layer by layer basis) without any need to involve explicitly the partial derivatives of the overall loss function defined at the whole network level. Therefore, the learning process gets rid altogether of the troublesome numerical problems determined by repeatedly backpropagating partial derivatives from top to bottom.

To this end, [16] proposed an approach called difference target propagation (DTP) that relies on autoencoders. DTP is aimed at realizing a straight mapping $\hat{\mathbf{y}}_{\ell-1} = \phi(\hat{\mathbf{y}}_\ell)$ from the targets $\hat{\mathbf{y}}_\ell$ at layer ℓ to the expected[1] targets $\hat{\mathbf{y}}_{\ell-1}$ at layer $\ell - 1$. As shown by [16], the technique is effective (it improves over regular gradient-descent in the experiments carried out on the MNIST dataset), although the accuracy yielded by DTP does not compare favorably with the state-of-the-art methods (mostly based on convolutional networks). Moreover, DTP offers the advantages of being readily applied to stochastic and discrete neural nets. The approach is loosely related to the algorithm proposed by [12], where a layer-specific neural network is used to estimate the gradients of the global loss function w.r.t. the weights of the corresponding layer (instead of the target outputs).

Differently from DTP, the core of the present approach is that the backward mapping from layer ℓ to $\ell - 1$ shall be learnt by a regular feed-forward neural network as an explicit function $\varphi(.)$ of the actual error $\mathbf{e}_\ell$ observed at layer ℓ (namely, the signed difference between the target and actual outputs at ℓ), that is $\hat{\mathbf{y}}_{\ell-1} = \varphi(\hat{\mathbf{y}}_\ell, \mathbf{e}_\ell)$. In so doing, after training has been completed, the image of $\varphi(\hat{\mathbf{y}}_\ell, \mathbf{0})$ is an estimated "optimal" value of $\hat{\mathbf{y}}_{\ell-1}$ that is expected to result in a null error $\mathbf{e}_\ell = \mathbf{0}$ when propagated forward (i.e., from $\ell - 1$ to ℓ) through the original network. It is seen that learning $\varphi(.)$ requires that at least a significant fraction of the training samples result in small errors (such that $\mathbf{e}_\ell \simeq \mathbf{0}$). This is the reason why the proposed technique can hardly be expected to be a suitable replacement for the established learning algorithms altogether, but it rather results in an effective refinement method for improving significantly the models realized by pre-trained deep neural networks. The proposed approach is different

[1] The term "expected" is herein used according to its statistical notion, since such a $\phi(.)$ is not strictly a function, but it may be reduced to a proper function if we interpret the images in the codomain of $\phi(.)$ as the expected values of $\hat{\mathbf{y}}_{\ell-1}$ given $\hat{\mathbf{y}}_\ell$.

from that introduced in [3,4], as well, since the latter relies on gradient-descent (or, the pseudo-inverse method) and, above all, it does not involve $\mathbf{e}_\ell$.

The error-driven target propagation algorithm is introduced formally in Sect. 2. Section 2.1 presents the details for realizing target propagation via an inversion network used to learn $\varphi(.)$. Section 2.2 hands out the formal procedure for refining pre-trained deep networks relying on the proposed target propagation scheme. Experimental results obtained on the MNIST dataset are presented in Sect. 3, showing that the refinement strategy allows for accuracies that are at least in line with the established results yielded by regular (i.e., non-convolutional) deep networks, relying on much less complex models (i.e., using much fewer free parameters). Finally, preliminary conclusions are drawn in Sect. 4.

2 Error-Driven Target Propagation: Formalization of the Algorithms

Let us consider a deep neural network *dnet* having l layers. When *dnet* is fed with an input vector $\mathbf{x}$, the i-th layer of *dnet* (for $i = 1, \dots, l$, while $i = 0$ represents the input layer which is not counted) is characterized by a state $\mathbf{h}_i \in \mathbb{R}^{d_i}$, where d_i is the number of units in layer i, $\mathbf{h}_i = \sigma(W_i \mathbf{h}_{i-1} + \mathbf{b}_i)$, and $\mathbf{h}_0 = \mathbf{x}$ as usual. The quantity W_i represents the weights matrix associated to layer i, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\mathbf{b}_i \in \mathbb{R}^{d_i}$ denotes the corresponding bias vector, and $\sigma(.)$ represents the vector of the element-wise outcomes of the neuron-specific activation functions. The usual logistic sigmoid activation function is used in the present research. Consider a supervised training dataset $\mathcal{D} = \{(\mathbf{x}_j, \hat{\mathbf{y}}_j) | j = 1, \dots, k\}$. Given a generic input pattern $\mathbf{x}_j \in \mathbb{R}^n$ and the corresponding target output $\hat{\mathbf{y}}_j \in \mathbb{R}^m$ drawn from $\mathcal{D}$, the state $\mathbf{h}_0 \in \mathbb{R}^n$ of the input layer of *dnet* is then defined as $\mathbf{h}_0 = \mathbf{x}_j$, while the target state $\hat{\mathbf{h}}_l \in \mathbb{R}^m$ of the output layer is $\hat{\mathbf{h}}_l = \hat{\mathbf{y}}_j$. Relying on this notation, it is seen that the function $f_i(.)$ realized by the generic i-th layer in *dnet* can be written as

$$f_i(\mathbf{h}_{i-1}) = \sigma(W_i \mathbf{h}_{i-1} + \mathbf{b}_i)$$

Therefore, the mapping $F_i : \mathbb{R}^n \to \mathbb{R}^{d_i}$ realized by the i bottommost layers of *dnet* over current input $\mathbf{x}_j$ can be expressed as the composition of i layer-specific functions as follows:

$$F_i(\mathbf{x}_j) = f_i(f_{i-1} ... (f_1(\mathbf{x}_j)))$$

Eventually, the function realized by *dnet* (that is an l-layer network) is $F_l(\mathbf{x}_j)$. Bearing in mind the definition of $\mathcal{D}$, the goal of training *dnet* is having $F_l(\mathbf{x}_j) \simeq \hat{\mathbf{y}}_j$ for $j = 1, \dots, k$. This is achieved by minimizing a point-wise loss function measured at the output layer. In this paper such a loss is the usual squared error $\mathcal{L}(\mathbf{x}_j; \theta) = (F_l(\mathbf{x}_j) - \hat{\mathbf{y}}_j)^2$ where θ represents the overall set of the parameters of *dnet*. In the traditional supervised learning framework the targets are defined only at the output layer. Nevertheless, while no explicit "loss" functions are associated to the hidden layers, the backpropagation (BP) algorithm allows the update of the hidden layers weights by back-propagating the gradients of the top-level loss $\mathcal{L}(.)$. To the contrary, target propagation consists in propagating the topmost layer targets $\hat{\mathbf{y}}_j$ to lower layers, in order to obtain explicit target states for the hidden units of the network, as well. Eventually, standard gradient-descent with no BP is applied in order to learn the layer-specific parameters as a function of the corresponding targets. In this research, at the core of the target propagation algorithm there is another, subsidiary network called the *inversion net*. Its nature and its application to target propagation are handed out in the following section.

2.1 The Inversion Net

Let us assume that the target value $\hat{\mathbf{h}}_i$ is known for a certain layer i (e.g. for the output layer, in the first place). The inversion net is then expected to estimate the targets $\hat{\mathbf{h}}_{i-1}$ for the preceding layer, that is layer $i-1$. In this research the inversion net is a standard feed-forward neural network having a much smaller number of parameters than *dnet* has, e.g. having a single hidden layer. In principle, as in [16], the inversion net could be trained such that it learns to realize a function $g_i() : \mathbb{R}^{d_i} \rightarrow \mathbb{R}^{d_{i-1}}$ defined as

$$g_i(\hat{\mathbf{h}}_i) = \hat{\mathbf{h}}_{i-1}$$

where $\hat{\mathbf{h}}_{i-1}$ represents the estimated target at layer $i-1$. Let us assume that such inversion nets were trained properly to realize $g_i(.)$ for $i = l, \ldots, 1$. Then, layer-specific targets could be defined according to the following recursive procedure. First of all (basis of the recursion), if the layer i is the output layer, i.e. $i = l$, then $\hat{\mathbf{h}}_i = \hat{\mathbf{y}}$ and $g_l(\hat{\mathbf{y}}) = \hat{\mathbf{h}}_{l-1}$. Then (recursive step) the target outputs for the subsequent layers $(l-1, \ldots, 1)$ are obtained by applying $g_i(.)$ to the estimated targets available at the adjacent upper (i.e., i-th) layer.

The actual error-driven training procedure for the inversion net proposed herein modifies this basic framework in the following manner. Given the generic layer i for which we want to learn the inversion function $g_i(.)$, let us define a layer-specific dataset $\mathcal{D}_i = \{(\mathbf{x}'_{i,j}, \hat{\mathbf{y}}'_{i,j}) | j = 1, \ldots, k\}$ where, omitting the pattern-specific index j for notational convenience, the generic input pattern is $\mathbf{x}'_i = (\hat{\mathbf{h}}_i, \mathbf{e}_i)$ given by the concatenation of the target value at layer i (either known, if $i = l$, or pre-computed from the upper layers if $i < l$) and the corresponding layer-specific signed error $\mathbf{e}_i = \mathbf{h}_i - \hat{\mathbf{h}}_i$. Herein $\mathbf{h}_i$ is the actual state of layer i of *dnet* upon forward propagation of its input, such that $\mathbf{x}'_i \in \mathbb{R}^{2 \times d_i}$. In turn, $\hat{\mathbf{y}}'_i$ is defined to be the state of the $(i-1)$-th layer of *dnet*, namely $\hat{\mathbf{y}}'_i = \mathbf{h}_{i-1}$. Once the supervised dataset $\mathcal{D}_i$ has been built this way, the inversion net can be trained using standard BP with an early-stopping criterion. We say that this scheme is error-driven, meaning that the inversion net learns a target-estimation mapping which relies on the knowledge of the errors $\mathbf{e}_i$ stemming from the forward-propagation process in *dnet*.

Once training of the inversion net is completed, the proper target-propagation step (from layer i to layer $i-1$) can be accomplished as follows. The inversion network is fed with the vector $(\hat{\mathbf{h}}_i, \mathbf{e}_i)$ where we let $\mathbf{e}_i = \mathbf{0}$ in order to get $g_i(\hat{\mathbf{h}}_i) = \hat{\mathbf{h}}_{i-1} \simeq f_i^{-1}(\hat{\mathbf{h}}_i)$. In so doing, the inversion net generates layer-specific targets that, once propagated forward by *dnet*, are expected to result in a null error, as sought. The resulting training procedure is formalized in Algorithms 1 and 2 in the form of pseudo-code. The algorithms assume the availability of two procedures, namely: *feedForward*$(net, \mathbf{x})$, realizing the forward propagation of an input pattern $\mathbf{x}$ through a generic neural network *net*; and, *backpropagation*$(net, \mathcal{D})$ that implements the training of the network *net* via BP from the generic supervised training set $\mathcal{D}$.

In practice, in order to reduce the bias intrinsic to the training algorithm, target propagation is accomplished relying on a modified strategy, as in the difference target propagation scheme [16], accounting for the bias that the layer-specific inversion nets $g_i(.)$ are likely to introduce in estimating the corresponding target outputs $\hat{\mathbf{h}}_{i-1}$. To this end we let

$$\hat{\mathbf{h}}_{i-1} = \mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i, \mathbf{0}) - g_i(\mathbf{h}_i, \mathbf{0}) \tag{1}$$

The rationale behind this equation is the following. First of all, $g_i(.)$ can be applied to invert the actual state $\mathbf{h}_i$ of *dnet* instead of the target state $\hat{\mathbf{h}}_i$. Ideally, if the mapping realized by the inversion net were perfect, we would have $g_i(\mathbf{h}_i, \mathbf{0}) = \mathbf{h}_{i-1}$. To the contrary, since $g_i(.)$ is the noisy outcome of an empirical learning procedure, in practice $g_i(\mathbf{h}_i, \mathbf{0}) \neq \mathbf{h}_{i-1}$ holds, i.e. an offset is observed whose magnitude is given by $|g_i(\mathbf{h}_i, \mathbf{0}) - \mathbf{h}_{i-1}|$. Equation (1) exploits this offset as a bias corrector when applying $g_i(.)$ to the computation of $\hat{\mathbf{h}}_{i-1}$, as well. Note that whenever $g_i(\mathbf{h}_i, \mathbf{0}) = \mathbf{h}_{i-1}$ (unbiased inversion net) then the equation reduces to $\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i, \mathbf{0})$, as before. The details of the present bias-correction strategy are handed out in [16].

Algorithm 1. Training of the inversion net

Procedure `train_inv_net`$(invNet_i, dnet, \mathcal{D}, i, \hat{\mathbf{h}}_i)$
Input: initialized inversion net $invNet_i$ with $2 \times d_i$ input units and d_{i-1} output units, deep network *dnet*, training set $\mathcal{D} = \{(\mathbf{x}_j, \hat{\mathbf{y}}_j) | j = 1, \ldots, k\}$, layer i, targets $\hat{\mathbf{h}}_i$ at layer i
Output: The trained inversion net $invNet_i$ for layer i, capable of computing $\hat{\mathbf{h}}_{i-1}$ from $\hat{\mathbf{h}}_i$

```
1: $\mathcal{D}_i = \varnothing$
2: for $j = 1$ to $k$ do
3:     feedForward$(dnet, \mathbf{x}_j)$
4:     $\mathbf{e}_{i,j} \leftarrow \hat{\mathbf{h}}_{i,j} - \mathbf{h}_{i,j}$
5:     $\mathbf{x}'_{i,j} \leftarrow (\hat{\mathbf{h}}_{i,j}, \mathbf{e}_{i,j})$
6:     $\mathbf{y}'_{i,j} \leftarrow \mathbf{h}_{i-1,j}$
7:     $\mathcal{D}_i = \mathcal{D}_i \cup \{(\mathbf{x}'_{i,j}, \mathbf{y}'_{i,j})\}$
8: end for
9: $invNet_i$ = backpropagation$(invNet_i, \mathcal{D}_i)$
```

Algorithm 2. Target propagation

Procedure `tgt_prop`($invNet_i$, i, k, $\hat{\mathbf{h}}_{i,1}, \ldots, \hat{\mathbf{h}}_{i,k}$)
Input: The inversion net $invNet_i$, layer i, number of patterns k, targets to be propagated $\hat{\mathbf{h}}_{i,j}$ for $j = 1. \ldots, k$
Output: The propagated targets $\hat{\mathbf{h}}_{i-1,1}, \ldots, \hat{\mathbf{h}}_{i-1,k}$

```
1: for $j = 1$ to $k$ do
2:     $\mathbf{e}_{i,j} = \mathbf{0}$
3:     $\mathbf{x}'_{i,j} = (\hat{\mathbf{h}}_{i,j}, \mathbf{e}_{i,j})$
4:     $\hat{\mathbf{h}}_{i-1,j} = \texttt{feedForward}(invNet_i, \mathbf{x}'_{i,j})$
5: end for
```

Algorithm 3. Deep learning with refinement based on target propagation

Procedure `network_refinement`($dnet, \mathcal{D}$)
Input: deep network $dnet$, supervised training set $\mathcal{D} = \{(\mathbf{x}_j, \hat{\mathbf{y}}_j) | j = 1, \ldots, k\}$
Output: the refined network $dnet$

```
1: for $j = 1$ to $k$ do
2:     for $i = l$ to 1 do
3:         if $i = l$ then
4:             $\hat{\mathbf{h}}_{i,j} = \hat{\mathbf{y}}_j$
5:         end if
6:         $\mathbf{h}_{i,j} = F_i(\mathbf{x}_j)$
7:         $\mathbf{h}_{i-1,j} = F_{i-1}(\mathbf{x}_j)$
8:     end for
9: end for
10: for $i = l$ to 2 do
11:     Initialize_Network($invNet_i$)
12:     $invNet_i = \texttt{train\_inv\_net}(invNet_i, dnet, \mathcal{D}, i, \hat{\mathbf{h}}_i)$
13:     $\{\hat{\mathbf{h}}_{i-1,1}, \ldots, \hat{\mathbf{h}}_{i-1,k}\} = \texttt{tgt\_prop}(invNet_i, i, k, \hat{\mathbf{h}}_{i,1}, \ldots, \hat{\mathbf{h}}_{i,k})$
14: end for
15: for $j = 1$ to $k$ do
16:     $\mathbf{h}_{0,j} = \mathbf{x}_j$
17:     layer_backprop($\mathbf{h}_{0,j}, \hat{\mathbf{h}}_{1,j}$)
18:     for $i = 2$ to $l$ do
19:         $\mathbf{h}_{i-1,j} = F_{i-1}(\mathbf{x}_j)$
20:         layer_backprop($\mathbf{h}_{i-1,j}, \hat{\mathbf{h}}_{i,j}$)
21:     end for
22: end for
```

2.2 Refinement of Deep Learning via Target Propagation

The algorithms presented in the previous section form the basis for building a refinement technique for pre-trained deep networks. The overall approach goes as follows. In a first phase the deep network is trained via BP, as usual. In a second phase, targets are propagated downward through the layers, as in Algorithms 1 and 2, and the network is trained layer-wise accordingly. This phase is called "refinement". Algorithm 2 provides a detailed description of this refinement strategy in terms of pseudo-code. The algorithm invokes a routine *Initialize_Network*(.) used for initializing a generic feed-forward neural net with random parameters before the actual training takes place. Finally, the routine *layer_backprop*($\mathbf{h}_{i-1,j}, \hat{\mathbf{h}}_{i,j}$) realizes the adaptation of the weights between layers $i-1$ and i (for $i = 1, \ldots, l$) via online gradient-descent. This application of gradient-descent uses $\mathbf{h}_{i-1,j}$ as its input, and $\hat{\mathbf{h}}_{i,j}$ as the corresponding target output. It is seen that extensions of the procedure to batch gradient-descent and/or multi-epochs training are straightforward by working out the skeleton of pseudo-code offered by Algorithm 3.

3 Experiments

Experiments were conducted on the popular MNIST dataset[2] [14]. We used all the 70,000 MNIST patterns, representing pixel-based images of handwritten digits (10 classes overall) having a dimensionality equal to 784. A 10-fold cross-validation strategy was applied, where for each fold as much as 80% of the data were used for training, 10% for validation/model selection, and 10% for test. The most significant results on MNIST published so far, obtained with a variety of different approaches, are listed in [15]. Variants on the theme of convolutional neural nets are known to yield the highest accuracies to date [6,23], as expected given the visual nature of the dataset. Our aim here is to exploit MNIST as a significant and difficult learning task suitable to assess the effectiveness of the present approach, and to compare the proposed algorithms to established non-convolutional feed-forward networks and target propagation methods previously applied to MNIST [16,20].

The topology of each layer and the hyperparameters were selected via grid search. Gradient-based training of the main network *dnet* (the classifier) relied on the root mean square propagation (RMSProp) variant of BP [22], while for the inversion net and the layer-wise refinement of *dnet* upon target propagation (routine `layer_backprop(.)` in Algorithm 3) the Adam variant of stochastic BP [13] turned out to be best. Besides a 784-dimensional input layer with linear activation functions and a class-wise 10-dimensional output layer with softmax activations, *dnet* had 3 hidden layers having 140, 120, and 100 neurons, respectively. Logistic sigmoid activation functions were used in the hidden layers. Connection weights and bias values for the sigmoids were initialized at random from a uniform distribution over the range $(-0.5, 0.5)$. RMSProp was applied for a maximum of 10^4 steps with early stopping (based on the generalization error not improving over the last 2000 steps), using a mini-batch size of 128 and a learning rate set to 0.01. As for the inversion nets, the dimensionality of the input and output layers were fixed according to the topology of the specific, adjacent layers in *dnet* between which the output targets had to be propagated (the input layer of $InvNet$ had linear activation functions, while its output layer

[2] Available at http://yann.lecun.com/exdb/mnist/.

used logistic sigmoids), as explained in the previous section. A single hidden layer of 200 sigmoid units was used. The Adam optimizer was applied for a maximum of 1000 steps with early stopping, using a mini-batch size of 128 and a learning rate set to 0.001. Finally, the adaptation of the layer-specific weights in *dnet* upon propagation of the corresponding targets via the trained $InvNet$ (procedure `layer_backprop(.)` in Algorithm 3) relied on the Adam optimizer, as well, with mini-batch size of 32 and learning rate set to 0.001.

Table 1 presents the average accuracies (± the corresponding standard deviations) on the 10-fold crossvalidation for *dnet* trained with RMSProp, with the bare target propagation, and with the refinement algorithm, respectively, evaluated on the training and the test sets. It is seen that the target propagation scheme required a proper BP-based initialization in order to achieve significant accuracies. in fact, In terms of learning capabilities (evaluated on the training sets), target propagation applied to the pre-trained *dnet* according to the refinement strategy yielded a relative 32.75% average error rate reduction over RMSProp, along with a much more stable behavior (the standard deviation was reduced as much as 42%). The statistical significance of the improvement evaluated via Welch's t-test (in order to account for the different variances of the two populations) results in a confidence level that is $\geq$ 99.75%. In terms of generalization capabilities (evaluated on the test sets), when applying the refinement strategy a significant relative 8.20% error rate reduction over RMSProp was observed on average, preserving the same stability of the performance (in fact, the difference between the standard deviations yielded by the two approaches is neglectable, namely 0.002%). Welch's t-test assessed a statistical significance of the gap between the results yielded by the two algorithms which is even higher than before (due to the much smaller variance of the RMSProp results), that is a confidence level $\geq$ 99.9%.

Table 1. Accuracies on the MNIST 10-class classification task (avg. ± std. dev. on a 10-fold crossvalidation).

Algorithm	Training	Test
RMSProp	99.48 ± 0.13	98.12 ± 0.05
Target propagation	87.30 ± 0.29	86.64 ± 0.27
Refinement	99.65 ± 0.08	98.27 ± 0.06

Table 2 offers a comparison among MNIST classifiers based on non-convolutional feed-forward deep neural networks using no augmentation of the training set (see [7,17] for established results obtained using augmentation). The comparison involves the error rate as observed on the test set (average ± standard deviation on the 10-fold crossvalidation, whenever available) and the number of free (i.e., adaptive) parameters in the model, that is an index of the model complexity. The proposed technique (target propagation with refinement) is compared with the approach by [20], that is a 2-hidden layer network with

Table 2. Comparison between the proposed algorithm and the established approaches, in terms of error rate and number of adaptive parameters.

Algorithm	Test error	#Parameters
Refinement	1.73 ± 0.06	3.04×10^5
[16]	1,94	5.36×10^5
[20]	1,6	1.28×10^6

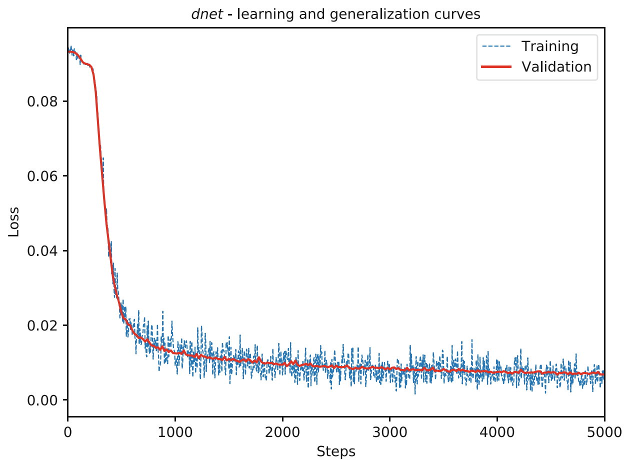

Fig. 1. Learning and generalization curves for *dnet*.

800 units per layer (resulting in a very complex machine), and by [16], that is a 7 hidden layer network having 240 neurons per layer. It is seen that the error rate achieved by the proposed refinement algorithm is in the middle between its competitors, but the complexity of the machine is dramatically smaller. A relative 11.02% error rate reduction is yielded by the present refinement approach over the difference target propagation algorithm, while a relative 7.25% reduction is still offered by [20] (credited by [11] of being the best performance yielded by a "regular" feed-forward net) over the present refinement procedure, at the expense of the number of adaptive parameters, which is one order of magnitude higher. Figure 1 presents the learning and generalization curves (mean squared error on training and validation sets, respectively) obtained running regular BP learning of *dnet* in one of the 10-folds of the present experiment. For graphical convenience, the plot is limited to the first 5000 steps (no evident changes in behavior were observed during the following steps). Note that the loss used to plot the learning curve was evaluated, from step to step, on the corresponding training mini-batch only, while the generalization curve was always evaluated on the whole validation set. This is the reason why the learning curve fluctuates locally, while the generalization curve is much smoother. The curves are compared with those corresponding to the refinement via target propagation,

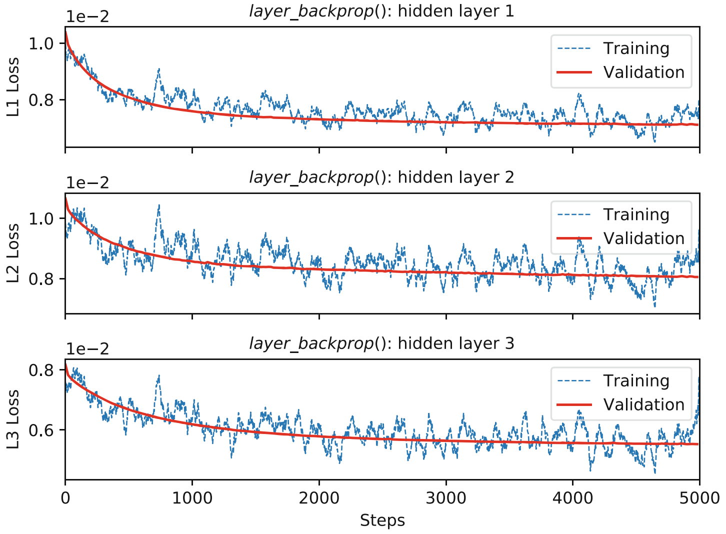

Fig. 2. Learning and generalization curves of the procedure `layer_backprop(.)` applied to the three hidden layers of *dnet*.

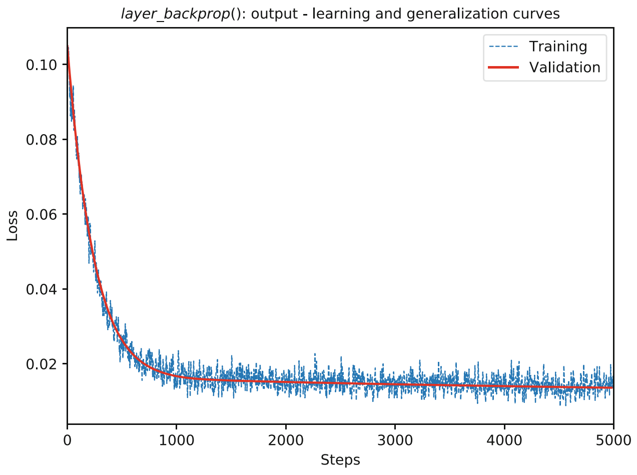

Fig. 3. Learning and generalization curves of the procedure `layer_backprop(.)` applied to the output layer of *dnet*.

namely Figs. 2 and 3. The former plots the learning and generalization curves of the layer-specific gradient-descent adaptation of the weights in the 1st, 2nd, and 3rd hidden layers of *dnet*, respectively, by means of the application of the procedure `layer_backprop(.)` to the target propagated via the inversion net. Similarly, Fig. 3 shows the curves for `layer_backprop(.)` applied to the weights in the topmost layer of *dnet*. Although eventually one is interested in solving the original learning problem, it is seen that the layer-specific sub-problems are actually difficult high-dimensional learning problems, which may just not admit any sound single-layered solution. This explains the observed difficulties met by gradient-descent in minimizing the corresponding layer-specific loss functions.

4 Conclusions

Target propagation emerges as a viable approach to learning and refinement of deep neural networks, tackling the vanishing-gradient issues stemming from application of plain BP to deep architectures. Albeit preliminary, the empirical evidence stresses that the proposed refinement strategy yields classification accuracies that are in line with the state-of-the-art algorithms for training feed-forward networks. The error rate reduction observed over the bare BP-based deep learning was shown to be statistically significant according to Welch's t-tests. The experiments presented in the paper revolved around a 5-layer architecture, yet our efforts are currently focusing on deeper networks. Consequently, also the application of inversion nets featuring more than one hidden layers is under investigation. The training set for the inversion net can be enriched, as well, by synthetically generating layer-specific input-output pairs obtained from the original ones with the addition of random noise, resulting in different examples of the signed errors $\mathbf{e}_i$ used to drive the learning of the target-propagation relationship.

References

1. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 4–7 December 2006, pp. 153–160 (2006)
2. Carreira-Perpiñán, M.Á., Wang, W.: Distributed optimization of deeply nested systems. In: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, 22–25 April 2014, pp. 10–19 (2014)
3. Castelli, I., Trentin, E.: Semi-unsupervised weighted maximum-likelihood estimation of joint densities for the co-training of adaptive activation functions. In: Schwenker and Trentin [19], pp. 62–71
4. Castelli, I., Trentin, E.: Supervised and unsupervised co-training of adaptive activation functions in neural nets. In: Schwenker and Trentin [19], pp. 52–61
5. Castelli, I., Trentin, E.: Combination of supervised and unsupervised learning for training the activation functions of neural networks. Pattern Recognit. Lett. **37**, 178–191 (2014)
6. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012), pp. 3642–3649. IEEE Computer Society (2012)
7. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural Comput. **22**(12), 3207–3220 (2010)

8. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010, pp. 249–256 (2010)
9. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, 11–13 April 2011, pp. 315–323 (2011)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
11. 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). http://arxiv.org/abs/1207.0580
12. Jaderberg, M., et al.: Decoupled neural interfaces using synthetic gradients. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017, pp. 1627–1635 (2017)
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014)
14. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
15. LeCun, Y., Cortes, C., Burges, C.: MNIST handwritten digit database (2018). http://yann.lecun.com/exdb/mnist/. Accessed 02 Feb 2018
16. Lee, D.-H., Zhang, S., Fischer, A., Bengio, Y.: Difference target propagation. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Soares, C., Gama, J., Jorge, A. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9284, pp. 498–515. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23528-8_31
17. Meier, U., Ciresan, D.C., Gambardella, L.M., Schmidhuber, J.: Better digit recognition with a committee of simple neural nets. In: Proceedings of the 2011 International Conference on Document Analysis and Recognition (ICDAR 2011), pp. 1250–1254. IEEE Computer Society, Washington, D.C. (2011)
18. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L., Group, P.R. (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, pp. 318–362. MIT Press, Cambridge (1986)
19. Schwenker, F., Trentin, E. (eds.): PSL 2011. LNCS (LNAI), vol. 7081. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28258-4
20. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: 7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3–6 August 2003, Edinburgh, Scotland, UK, pp. 958–962 (2003)
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
22. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. Technical report (2012)
23. Wan, L., Zeiler, M.D., Zhang, S., LeCun, Y., Fergus, R.: Regularization of neural networks using DropConnect. In: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16–21 June 2013, pp. 1058–1066 (2013)