Difference Target Propagation

Dong-Hyun Lee[1(✉)], Saizheng Zhang[1], Asja Fischer[1], and Yoshua Bengio[1,2]

[1] Université de Montréal, Montreal, QC, Canada
[2] CIFAR Senior Fellow, Montreal, Canada
donghyun.lee.dl@gmail.com

Abstract. Back-propagation has been the workhorse of recent successes of deep learning but it relies on infinitesimal effects (partial derivatives) in order to perform credit assignment. This could become a serious issue as one considers deeper and more non-linear functions, e.g., consider the extreme case of non-linearity where the relation between parameters and cost is actually discrete. Inspired by the biological implausibility of back-propagation, a few approaches have been proposed in the past that could play a similar credit assignment role. In this spirit, we explore a novel approach to credit assignment in deep networks that we call target propagation. The main idea is to compute targets rather than gradients, at each layer. Like gradients, they are propagated backwards. In a way that is related but different from previously proposed proxies for back-propagation which rely on a backwards network with symmetric weights, target propagation relies on auto-encoders at each layer. Unlike back-propagation, it can be applied even when units exchange stochastic bits rather than real numbers. We show that a linear correction for the imperfectness of the auto-encoders, called difference target propagation, is very effective to make target propagation actually work, leading to results comparable to back-propagation for deep networks with discrete and continuous units and denoising auto-encoders and achieving state of the art for stochastic networks.

1 Introduction

Recently, deep neural networks have achieved great success in hard AI tasks [2,12,14,19], mostly relying on back-propagation as the main way of performing credit assignment over the different sets of parameters associated with each layer of a deep net. Back-propagation exploits the chain rule of derivatives in order to convert a loss gradient on the activations over layer l (or time t, for recurrent nets) into a loss gradient on the activations over layer $l-1$ (respectively, time $t-1$). However, as we consider deeper networks– e.g., consider the recent best ImageNet competition entrants [20] with 19 or 22 layers – longer-term dependencies, or stronger non-linearities, the composition of many non-linear operations becomes more strongly non-linear. To make this concrete, consider the composition of many hyperbolic tangent units. In general, this means that derivatives obtained by back-propagation are becoming either very small (most of the time) or very large (in a few places). In the extreme (very deep computations), one would get discrete functions, whose derivatives are 0 almost everywhere, and

A. Appice et al. (Eds.): ECML PKDD 2015, Part I, LNAI 9284, pp. 498–515, 2015.
DOI: 10.1007/978-3-319-23528-8_31

infinite where the function changes discretely. Clearly, back-propagation would fail in that regime. In addition, from the point of view of low-energy hardware implementation, the ability to train deep networks whose units only communicate via bits would also be interesting.

This limitation of back-propagation to working with precise derivatives and smooth networks is the main machine learning motivation for this paper's exploration into an alternative principle for credit assignment in deep networks. Another motivation arises from the lack of biological plausibility of back-propagation, for the following reasons: (1) the back-propagation computation is purely linear, whereas biological neurons interleave linear and non-linear operations, (2) if the feedback paths were used to propagate credit assignment by back-propagation, they would need precise knowledge of the derivatives of the non-linearities at the operating point used in the corresponding feedforward computation, (3) similarly, these feedback paths would have to use exact symmetric weights (with the same connectivity, transposed) of the feedforward connections, (4) real neurons communicate by (possibly stochastic) binary values (spikes), (5) the computation would have to be precisely clocked to alternate between feedforward and back-propagation phases, and (6) it is not clear where the output targets would come from.

The main idea of target propagation is to associate with each feedforward unit's activation value a *target value* rather than a *loss gradient*. The target value is meant to be close to the activation value while being likely to have provided a smaller loss (if that value had been obtained in the feedforward phase). In the limit where the target is very close to the feedforward value, target propagation should behave like back-propagation. This link was nicely made in [16,17], which introduced the idea of target propagation and connected it to back-propagation via a Lagrange multipliers formulation (where the constraints require the output of one layer to equal the input of the next layer). A similar idea was recently proposed where the constraints are relaxed into penalties, yielding a different (iterative) way to optimize deep networks [9]. Once a good target is computed, a layer-local training criterion can be defined to update each layer separately, e.g., via the delta-rule (gradient descent update with respect to the cross-entropy loss).

By its nature, target propagation can in principle handle stronger (and even discrete) non-linearities, and it deals with biological plausibility issues (1), (2), (3) and (4) described above. Extensions of the precise scheme proposed here could handle (5) and (6) as well, but this is left for future work.

In this paper, we describe how the general idea of target propagation by using auto-encoders to assign targets to each layer (as introduced in an earlier technical report [4]) can be employed for supervised training of deep neural networks (section 2.1 and 2.2). We continue by introducing a linear correction for the imperfectness of the auto-encoders (2.3) leading to robust training in practice. Furthermore, we show how the same principles can be applied to replace back-propagation in the training of auto-encoders (section 2.4). In section 3 we provide several experimental results on rather deep neural networks as well as discrete and stochastic networks and auto-encoders. The results show that the

proposed form of target propagation is comparable to back-propagation with RMSprop [22] - a very popular setting to train deep networks nowadays- and achieves state of the art for training stochastic neural nets on MNIST.

2 Target Propagation

Although many variants of the general principle of target propagation can be devised, this paper focuses on a specific approach, which is based on the ideas presented in an earlier technical report [4] and is described in the following.

2.1 Formulating Targets

Let us consider an ordinary (supervised) deep network learning process, where the training data is drawn from an unknown data distribution $p(\mathbf{x}, \mathbf{y})$. The network structure is defined by

$$\mathbf{h}_i = f_i(\mathbf{h}_{i-1}) = s_i(W_i \mathbf{h}_{i-1}), \;\; i = 1, \ldots, M \tag{1}$$

where $\mathbf{h}_i$ is the state of the i-th hidden layer (where $\mathbf{h}_M$ corresponds to the output of the network and $\mathbf{h}_0 = \mathbf{x}$) and f_i is the i-th layer feed-forward mapping, defined by a non-linear activation function s_i (e.g. the hyperbolic tangents or the sigmoid function) and the weights W_i of the i-th layer. Here, for simplicity of notation, the bias term of the i-th layer is included in W_i. We refer to the subset of network parameters defining the mapping between the i-th and the j-th layer ($0 \leq i < j \leq M$) as $\theta_W^{i,j} = \{W_k, k = i+1, \ldots, j\}$. Using this notion, we can write $\mathbf{h}_j$ as a function of $\mathbf{h}_i$ depending on parameters $\theta_W^{i,j}$, that is we can write $\mathbf{h}_j = \mathbf{h}_j(\mathbf{h}_i; \theta_W^{i,j})$.

Given a sample $(\mathbf{x}, \mathbf{y})$, let $L(\mathbf{h}_M(\mathbf{x}; \theta_W^{0,M}), \mathbf{y})$ be an arbitrary global loss function measuring the appropriateness of the network output $\mathbf{h}_M(\mathbf{x}; \theta_W^{0,M})$ for the target $\mathbf{y}$, e.g. the MSE or cross-entropy for binomial random variables. Then, the training objective corresponds to adapting the network parameters $\theta_W^{0,M}$ so as to minimize the expected global loss $\mathbb{E}_p\{L(\mathbf{h}_M(\mathbf{x}; \theta_W^{0,M}), \mathbf{y})\}$ under the data distribution $p(\mathbf{x}, \mathbf{y})$. For $i = 1, \ldots, M-1$ we can write

$$L(\mathbf{h}_M(\mathbf{x}; \theta_W^{0,M}), \mathbf{y}) = L(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i}); \theta_W^{i,M}), \mathbf{y}) \tag{2}$$

to emphasize the dependency of the loss on the state of the i-th layer.

Training a network with back-propagation corresponds to propagating error signals through the network to calculate the derivatives of the global loss with respect to the parameters of each layer. Thus, the error signals indicate how the parameters of the network should be updated to decrease the expected loss. However, in very deep networks with strong non-linearities, error propagation could become useless in lower layers due to exploding or vanishing gradients, as explained above.

To avoid this problem, the basic idea of target propagation is to assign to each $\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})$ a nearby value $\hat{\mathbf{h}}_i$ which (hopefully) leads to a lower global loss, that is which has the objective to fulfill

$$L(\mathbf{h}_M(\hat{\mathbf{h}}_i; \theta_W^{i,M}), \mathbf{y}) < L(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i}); \theta_W^{i,M}), \mathbf{y}) \ . \tag{3}$$

Such a $\hat{\mathbf{h}}_i$ is called a *target* for the i-th layer.

Given a target $\hat{\mathbf{h}}_i$ we now would like to change the network parameters to make $\mathbf{h}_i$ move a small step towards $\hat{\mathbf{h}}_i$, since – if the path leading from $\mathbf{h}_i$ to $\hat{\mathbf{h}}_i$ is smooth enough – we would expect to yield a decrease of the global loss. To obtain an update direction for W_i based on $\hat{\mathbf{h}}_i$ we can define a layer-local target loss L_i, for example by using the MSE

$$L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i) = ||\hat{\mathbf{h}}_i - \mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})||_2^2 \ . \tag{4}$$

Then, W_i can be updated locally within its layer via stochastic gradient descent, where $\hat{\mathbf{h}}_i$ is considered as a *constant* with respect to W_i. That is

$$W_i^{(t+1)} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i)}{\partial W_i} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i)}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})}{\partial W_i} \ , \tag{5}$$

where η_{f_i} is a layer-specific learning rate.

Note, that in this context, derivatives can be used without difficulty, because they correspond to computations performed inside a single layer. Whereas, the problems with the severe non-linearities observed for back-propagation arise when the chain rule is applied through many layers. This motivates target propagation methods to serve as alternative credit assignment in the context of a composition of many non-linearities.

However, it is not directly clear how to compute a target that guarantees a decrease of the global loss (that is how to compute a $\hat{\mathbf{h}}_i$ for which equation (3) holds) or that at least leads to a decrease of the local loss L_i of the next layer, that is

$$L_{i+1}(\hat{\mathbf{h}}_{i+1}, f_{i+1}(\hat{\mathbf{h}}_i)) < L_{i+1}(\hat{\mathbf{h}}_{i+1}, f_{i+1}(\mathbf{h}_i)) \ . \tag{6}$$

Proposing and validating answers to this question is the subject of the rest of this paper.

2.2 How to Assign a Proper Target to Each Layer

Clearly, in a supervised learning setting, the top layer target should be directly driven from the gradient of the global loss

$$\hat{\mathbf{h}}_M = \mathbf{h}_M - \hat{\eta} \frac{\partial L(\mathbf{h}_M, \mathbf{y})}{\partial \mathbf{h}_M} \ , \tag{7}$$

where $\hat{\eta}$ is usually a small step size. Note, that if we use the MSE as global loss and $\hat{\eta} = 0.5$ we get $\hat{\mathbf{h}}_M = \mathbf{y}$.

But how can we define targets for the intermediate layers? In the previous technical report [4], it was suggested to take advantage of an "approximate inverse". To formalize this idea, suppose that for each f_i we have a function g_i such that

$$f_i(g_i(\mathbf{h}_i)) \approx \mathbf{h}_i \quad \text{or} \quad g_i(f_i(\mathbf{h}_{i-1})) \approx \mathbf{h}_{i-1} \ . \tag{8}$$

Then, choosing

$$\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i) \tag{9}$$

would have the consequence that (under some smoothness assumptions on f and g) minimizing the distance between $\mathbf{h}_{i-1}$ and $\hat{\mathbf{h}}_{i-1}$ should also minimize the loss L_i of the i-th layer. This idea is illustrated in the left of Figure 1. Indeed, if the feed-back mappings were the perfect inverses of the feed-forward mappings ($g_i = f_i^{-1}$), one gets

$$L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1})) = L_i(\hat{\mathbf{h}}_i, f_i(g_i(\hat{\mathbf{h}}_i))) = L_i(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_i) = 0 \ . \tag{10}$$

But choosing g to be the perfect inverse of f may need heavy computation and instability, since there is no guarantee that f_i^{-1} applied to a target would yield a value that is in the domain of f_{i-1}. An alternative approach is to learn an approximate inverse g_i, making the f_i / g_i pair look like an *auto-encoder*. This suggests parametrizing g_i as follows:

$$g_i(\mathbf{h}_i) = \bar{s}_i(V_i \mathbf{h}_i), \quad i = 1, ..., M \tag{11}$$

where $\bar{s}_i$ is a non-linearity associated with the decoder and V_i the matrix of feed-back weights of the i-th layer. With such a parametrization, it is unlikely that the auto-encoder will achieve zero reconstruction error. The decoder could be trained via an additional auto-encoder-like loss at each layer

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1})) - \mathbf{h}_{i-1}||_2^2 \ . \tag{12}$$

Changing $\mathbf{V}_i$ based on this loss, makes g closer to f_i^{-1}. By doing so, it also makes $f_i(\hat{\mathbf{h}}_{i-1}) = f_i(g_i(\hat{\mathbf{h}}_i))$ closer to $\hat{\mathbf{h}}_i$, and is thus also contributing to the decrease of $L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1}))$. But we do not want to estimate an inverse mapping only for the concrete values we see in training but for a region around the these values to facilitate the computation of $g_i(\hat{\mathbf{h}}_i)$ for $\hat{\mathbf{h}}_i$ which have never been seen before. For this reason, the loss is modified by noise injection

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ , \tag{13}$$

which makes f_i and g_i approximate inverses not just at $\mathbf{h}_{i-1}$ but also in its *neighborhood*.

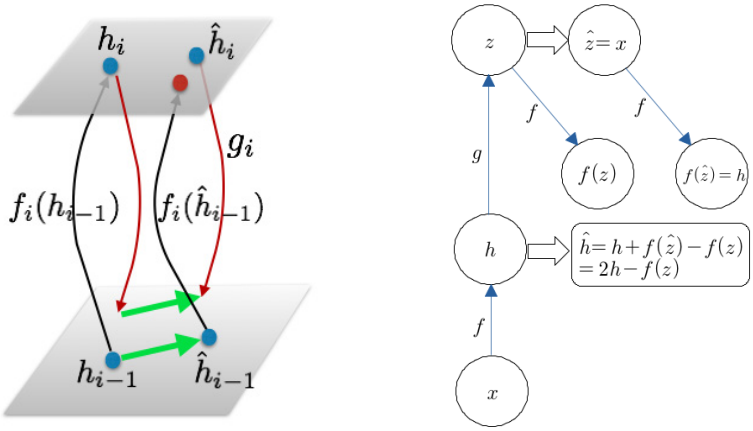

Fig. 1. (left) How to compute a target in the lower layer via difference target propagation. $f_i(\hat{\mathbf{h}}_{i-1})$ should be closer to $\hat{\mathbf{h}}_i$ than $f_i(\mathbf{h}_{i-1})$. (right) Diagram of the back-propagation-free auto-encoder via difference target propagation.

As mentioned above, a required property of target propagation is that the layer-wise parameter updates, each improving a layer-wise loss, also lead to an improvement of the global loss. The following theorem shows that, for the case that g_i is a perfect inverse of f_i and f_i having a certain structure, the update direction of target propagation does not deviate more then 90 degrees from the gradient direction (estimated by back-propagation), which always leads to a decrease of the global loss.

Theorem 1.[1] *Assume that* $g_i = f_i^{-1}, i = 1, ..., M$, *and* f_i *satisfies* $\mathbf{h}_i = f_i(\mathbf{h}_{i-1}) = W_i s_i(\mathbf{h}_{i-1})$[2] *where* s_i *can be any differentiable monotonically increasing element-wise function. Let* δW_i^{tp} *and* δW_i^{bp} *be the target propagation update and the back-propagation update in* i*-th layer, respectively. If* $\hat{\eta}$ *in Equation* (7) *is sufficiently small, then the angle* α *between* δW_i^{tp} *and* δW_i^{bp} *is bounded by*

$$0 < \frac{1 + \Delta_1(\hat{\eta})}{\frac{\lambda_{max}}{\lambda_{min}} + \Delta_2(\hat{\eta})} \leq cos(\alpha) \leq 1 \tag{14}$$

Here λ_{max} *and* λ_{min} *are the largest and smallest singular values of* $(J_{f_M} \dots J_{f_{i+1}})^T$, *where* J_{f_k} *is the Jacobian matrix of* f_k *and* $\Delta_1(\hat{\eta})$ *and* $\Delta_2(\hat{\eta})$ *are close to 0 if* $\hat{\eta}$ *is sufficiently small.*

2.3 Difference Target Propagation

From our experience, the imperfection of the inverse function leads to severe optimization problems when assigning targets based on equation (9). This brought

[1] The proof can be found in the appendix.
[2] This is another way to obtain a non-linear deep network structure.

Algorithm 1. Training deep neural networks via difference target propagation

Compute unit values for all layers:
for $i = 1$ to M **do**
  $\mathbf{h}_i \leftarrow f_i(\mathbf{h}_{i-1})$
end for
Making the first target: $\hat{\mathbf{h}}_{M-1} \leftarrow \mathbf{h}_{M-1} - \hat{\eta}\frac{\partial L}{\partial \mathbf{h}_{M-1}}$, ($L$ is the global loss)
Compute targets for lower layers:
for $i = M-1$ to 2 **do**
  $\hat{\mathbf{h}}_{i-1} \leftarrow \mathbf{h}_{i-1} - g_i(\mathbf{h}_i) + g_i(\hat{\mathbf{h}}_i)$
end for
Training feedback (inverse) mapping:
for $i = M-1$ to 2 **do**
  Update parameters for g_i using SGD with following a layer-local loss L_i^{inv}
  $L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma)$
end for
Training feedforward mapping:
for $i = 1$ to M **do**
  Update parameters for f_i using SGD with following a layer-local loss L_i
  $L_i = ||f_i(\mathbf{h}_{i-1}) - \hat{\mathbf{h}}_i||_2^2$ if $i < M$, $L_i = L$ (the global loss) if $i = M$.
end for

us to propose the following linearly corrected formula for target propagation which we refer to as "*difference target propagation*"

$$\hat{\mathbf{h}}_{i-1} = \mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i) \ . \tag{15}$$

Note, that if g_i is the inverse of f_i, difference target propagation becomes equivalent to vanilla target propagation as defined in equation (9). The resulting complete training procedure for optimization by difference target propagation is given in Algorithm 1.

In the following, we explain why this linear corrected formula stabilizes the optimization process. In order to achieve stable optimization by target propagation, $\mathbf{h}_{i-1}$ should approach $\hat{\mathbf{h}}_{i-1}$ as $\mathbf{h}_i$ approaches $\hat{\mathbf{h}}_i$. Otherwise, the parameters in lower layers continue to be updated even when an optimum of the global loss is reached already by the upper layers, which then could lead the global loss to increase again. Thus, the condition

$$\mathbf{h}_i = \hat{\mathbf{h}}_i \ \Rightarrow \ \mathbf{h}_{i-1} = \hat{\mathbf{h}}_{i-1} \tag{16}$$

greatly improves the stability of the optimization. This holds for vanilla target propagation if $g_i = f_i^{-1}$, because

$$\mathbf{h}_{i-1} = f_i^{-1}(\mathbf{h}_i) = g_i(\hat{\mathbf{h}}_i) = \hat{\mathbf{h}}_{i-1} \ . \tag{17}$$

Although the condition is not guaranteed to hold for vanilla target propagation if $g_i \neq f_i^{-1}$, for difference target propagation it holds by construction, since

$$\hat{\mathbf{h}}_{i-1} - \mathbf{h}_{i-1} = g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i) \ . \tag{18}$$

Furthermore, under weak conditions on f and g and if the difference between $\mathbf{h}_i$ and $\hat{\mathbf{h}}_i$ is small, we can show for difference target propagation that if the input of the i-th layer becomes $\hat{\mathbf{h}}_{i-1}$ (i.e. the $i-1$-th layer reaches its target) the output of the i-th layer also gets closer to $\hat{\mathbf{h}}_i$. This means that the requirement on targets specified by equation (6) is met for difference target propagation, as shown in the following theorem

Theorem 2.[3] *Let the target for layer $i-1$ be given by Equation (15), i.e. $\hat{\mathbf{h}}_{i-1} = \mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i)$. If $\hat{\mathbf{h}}_i - \mathbf{h}_i$ is sufficiently small, f_i and g_i are differentiable, and the corresponding Jacobian matrices J_{f_i} and J_{g_i} satisfy that the largest eigenvalue of $(I - J_{f_i}J_{g_i})^T(I - J_{f_i}J_{g_i})$ is less than 1, then we have*

$$||\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1})||_2^2 < ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2 \ . \tag{19}$$

The third condition in the above theorem is easily satisfied in practice, because g_i is learned to be the inverse of f_i and makes $g_i \circ f_i$ close to the identity mapping, so that $(I - J_{f_i}J_{g_i})$ becomes close to the zero matrix which means that the largest eigenvalue of $(I - J_{f_i}J_{g_i})^T(I - J_{f_i}J_{g_i})$ is also close to 0.

2.4 Training an Auto-Encoder with Difference Target Propagation

Auto-encoders are interesting for learning representations and serve as building blocks for deep neural networks [10]. In addition, as we have seen, training auto-encoders is part of the target propagation approach presented here, where they model the feedback paths used to propagate the targets.

In the following, we show how a regularized auto-encoder can be trained using difference target propagation instead of back-propagation. Like in the work on denoising auto-encoders [23] and generative stochastic networks [6], we consider the denoising auto-encoder as a stochastic network with noise injected in input and hidden units, trained to minimize a reconstruction loss. This is, the hidden units are given by the encoder as

$$\mathbf{h} = f(\mathbf{x}) = sig(W\mathbf{x} + \mathbf{b}) \ , \tag{20}$$

where sig is the element-wise sigmoid function, W the weight matrix and $\mathbf{b}$ the bias vector of the input units. The reconstruction is given by the decoder

$$\mathbf{z} = g(\mathbf{h}) = sig(W^T(\mathbf{h} + \epsilon) + \mathbf{c}), \ \ \epsilon \sim N(0, \sigma) \ , \tag{21}$$

[3] The proof can be found in the appendix.

with $\mathbf{c}$ being the bias vector of the hidden units. And the reconstruction loss is

$$L = ||\mathbf{z} - \mathbf{x}||_2^2 + ||f(\mathbf{x} + \epsilon) - \mathbf{h}||_2^2, \quad \epsilon \sim N(0, \sigma) \ , \tag{22}$$

where a regularization term can be added to obtain a contractive mapping. In order to train this network without back-propagation (that is, without using the chain rule), we can use difference target propagation as follows (see Figure 1 (right) for an illustration): at first, the target of $\mathbf{z}$ is just $\mathbf{x}$, so we can train the reconstruction mapping g based on the loss $L_g = ||g(\mathbf{h}) - \mathbf{x}||_2^2$ in which $\mathbf{h}$ is considered as a constant. Then, we compute the target $\hat{\mathbf{h}}$ of the hidden units following difference target propagation where we make use of the fact that f is an approximate inverse of g. That is,

$$\hat{\mathbf{h}} = \mathbf{h} + f(\hat{\mathbf{z}}) - f(\mathbf{z}) = 2\mathbf{h} - f(\mathbf{z}) \ , \tag{23}$$

where the last equality follows from $f(\hat{\mathbf{z}}) = f(\mathbf{x}) = \mathbf{h}$. As a target loss for the hidden layer, we can use $L_f = ||f(\mathbf{x} + \epsilon) - \hat{\mathbf{h}}||_2^2$, where $\hat{\mathbf{h}}$ is considered as a constant and which can be also augmented by a regularization term to yield a contractive mapping.

3 Experiments

In a set of experiments we investigated target propagation for training deep feedforward deterministic neural networks, networks with discrete transmissions between units, stochastic neural networks, and auto-encoders.

For training supervised neural networks, we chose the target of the top hidden layer (number $M-1$) such that it also depends directly on the global loss instead of an inverse mapping. That is, we set $\hat{\mathbf{h}}_{M-1} = \mathbf{h}_{M-1} - \tilde{\eta}\frac{\partial L(\mathbf{h}_M, \mathbf{y})}{\partial \mathbf{h}_{M-1}}$, where L is the global loss (here the multiclass cross entropy). This may be helpful when the number of units in the output layer is much smaller than the number of units in the top hidden layer, which would make the inverse mapping difficult to learn, but future work should validate that.

For discrete stochastic networks in which some form of noise (here Gaussian) is injected, we used a decaying noise level for learning the inverse mapping, in order to stabilize learning, i.e. the standard deviation of the Gaussian is set to $\sigma(e) = \sigma_0/(1 + e/e_0)$ where σ_0 is the initial value, e is the epoch number and e_0 is the half-life of this decay. This seems to help to fine-tune the feedback weights at the end of training.

In all experiments, the weights were initialized with orthogonal random matrices and the bias parameters were initially set to zero. All experiments were repeated 10 times with different random initializations. We put the code of these experiments online (https://github.com/donghyunlee/dtp).

3.1 Deterministic Feedforward Deep Networks

As a primary objective, we investigated training of ordinary deep supervised networks with continuous and deterministic units on the MNIST dataset. We used a held-out validation set of 10000 samples for choosing hyper-parameters. We trained networks with 7 hidden layers each consisting of 240 units (using the hyperbolic tangent as activation function) with difference target propagation and back-propagation.

Training was based on RMSprop [22] where hyper-parameters for the best validation error were found using random search [7]. RMSprop is an adaptive learning rate algorithm known to lead to good results for back-propagation. Furthermore, it is suitable for updating the parameters of each layer based on the layer-wise targets obtained by target propagation. Our experiments suggested that when using a hand-selected learning rate per layer rather than the automatically set one (by RMSprop), the selected learning rates were different for each layer, which is why we decided to use an adaptive method like RMSprop.

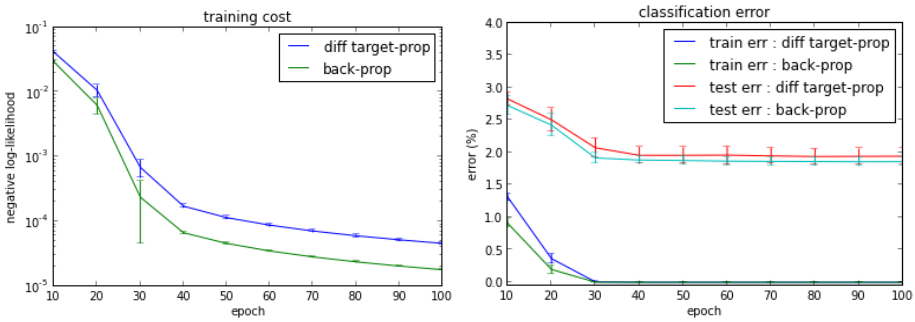

Fig. 2. Mean training cost (left) and train/test classification error (right) with target propagation and back-propagation using continuous deep networks (tanh) on MNIST. Error bars indicate the standard deviation over 10 independent runs with the same optimized hyper-parameters and different initial weights.

The results are shown in Figure 2. We obtained a test error of 1.94% with target propagation and 1.86% with back propagation. The final negative log-likelihood on the training set was 4.584×10^{-5} with target propagation and 1.797×10^{-5} with back propagation. We also trained the same network with rectifier linear units and got a test error of 3.15% whereas 1.62% was obtained with back-propagation. It is well known that this nonlinearity is advantageous for back-propagation [11], while it seemed to be less appropriate for this implementation of target propagation.

In a second experiment we investigated training on CIFAR-10. The experimental setting was the same as for MNIST (using the hyperbolic tangent as activation function) except that the network architecture was 3072-1000-1000-1000-10. We did not use any preprocessing, except for scaling the input values to lay in [0,1], and we tuned the hyper-parameters of RMSprop using a held-out validation set of 1000 samples. We obtained mean test accuracies of 50.71% and 53.72% for target propagation and back-propagation, respectively. It was reported in [15], that a network with 1 hidden layer of 1000 units achieved 49.78% accuracy with back-propagation, and increasing the number of units to 10000 led to 51.53% accuracy. As the current state-of-the-art performance on the permutation invariant CIFAR-10 recognition task, [13] reported 64.1% but when using PCA without whitening as preprocessing and zero-biased auto-encoders for unsupervised pre-training.

3.2 Networks with Discretized Transmission Between Units

To explore target propagation for an extremely non-linear neural network, we investigated training of discrete networks on the MNIST dataset. The network architecture was 784-500-500-10, where only the 1st hidden layer was discretized. Inspired by biological considerations and the objective of reducing the communication cost between neurons, instead of just using the step activation function, we used ordinary neural net layers but with signals being discretized when transported between the first and second layer. The network structure is depicted in the right plot of Figure 3 and the activations of the hidden layers are given by

$$\mathbf{h}_1 = f_1(\mathbf{x}) = \tanh(W_1\mathbf{x}) \quad \text{and} \quad \mathbf{h}_2 = f_2(\mathbf{h}_1) = \tanh(W_2 sign(\mathbf{h}_1)) \tag{24}$$

where $sign(x) = 1$ if $x > 0$, and $sign(x) = 0$ if $x \leq 0$. The network output is given by

$$p(\mathbf{y}|\mathbf{x}) = f_3(\mathbf{h}_2) = softmax(W_3\mathbf{h}_2) \ . \tag{25}$$

The inverse mapping of the second layer and the associated loss are given by

$$g_2(\mathbf{h}_2) = \tanh(V_2 sign(\mathbf{h}_2)) \ , \tag{26}$$

$$L_2^{inv} = ||g_2(f_2(\mathbf{h}_1 + \epsilon)) - (\mathbf{h}_1 + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ . \tag{27}$$

If the feed-forward mapping is discrete, back-propagated gradients become 0 and useless when they cross the discretization step. So we compare target propagation to two baselines. As a first baseline, we train the network with back-propagation and the *straight-through estimator* [5], which is biased but was found to work well, and simply ignores the derivative of the step function (which is 0 or infinite) in the back-propagation phase. As a second baseline, we train only the upper layers by back-propagation, while not changing the weight W_1 which are affected by the discretization, i.e., the lower layers do not learn.

The results on the training and test sets are shown in Figure 3. The training error for the first baseline (straight-through estimator) does not converge to zero (which can be explained by the biased gradient) but generalization performance is fairly good. The second baseline (fixed lower layer) surprisingly reached zero training error, but did not perform well on the test set. This can be explained by the fact that it cannot learn any meaningful representation at the first layer. Target propagation however did not suffer from this drawback and can be used to train discrete networks directly (training signals can pass the discrete region successfully). Though the training convergence was slower, the training error did approach zero. In addition, difference target propagation also achieved good results on the test set.

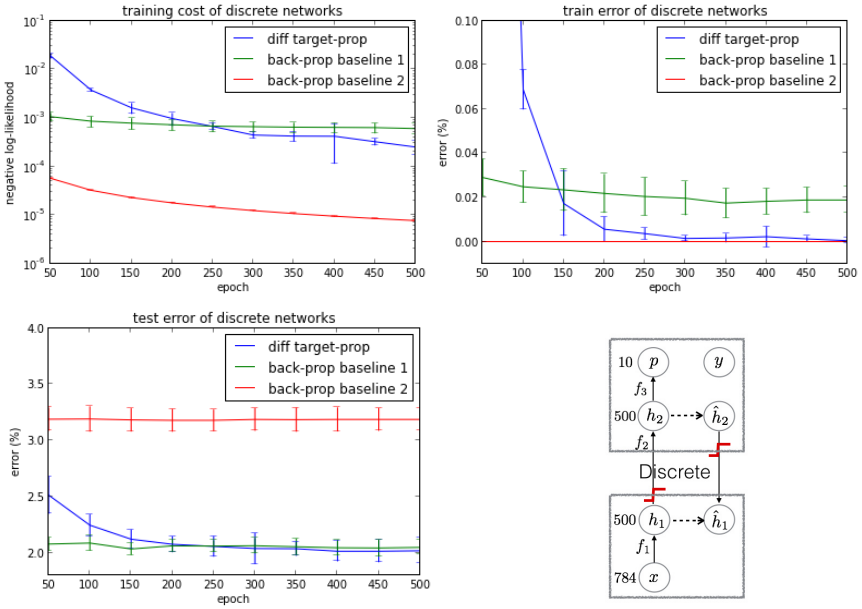

Fig. 3. Mean training cost (top left), mean training error (top right) and mean test error (bottom left) while training discrete networks with difference target propagation and the two baseline versions of back-propagation. Error bars indicate standard deviations over 10 independent runs with the same hyper-parameters and different initial weights. Diagram of the discrete network (bottom right). The output of $\mathbf{h}_1$ is discretized because signals must be communicated from $\mathbf{h}_1$ to $\mathbf{h}_2$ through a long cable, so binary representations are preferred in order to conserve energy. With target propagation, training signals are also discretized through this cable (since feedback paths are computed by bona-fide neurons).

3.3 Stochastic Networks

Another interesting model class which vanilla back-propagation cannot deal with are stochastic networks with discrete units. Recently, stochastic networks have attracted attention [3,5,21] because they are able to learn a multi-modal conditional distribution $P(Y|X)$, which is important for structured output predictions. Training networks of stochastic binary units is also biologically motivated, since they resemble networks of spiking neurons. Here, we investigate whether one can train networks of stochastic binary units on MNIST for classification using target propagation. Following [18], the network architecture was 784-200-200-10 and the hidden units were stochastic binary units with the probability of turning on given by a sigmoid activation

$$\mathbf{h}_i^p = P(\mathbf{H}_i = 1|\mathbf{h}_{i-1}) = sig(W_i\mathbf{h}_{i-1}), \quad \mathbf{h}_i \sim P(\mathbf{H}_i|\mathbf{h}_{i-1}) \ , \tag{28}$$

that is, $\mathbf{h}_i$ is one with probability $\mathbf{h}_i^p$.

As a baseline, we considered training based on the *straight-through* biased gradient estimator [5] in which the derivative through the discrete sampling step is ignored (this method showed the best performance in [18].) That is

$$\delta\mathbf{h}_{i-1}^p = \delta\mathbf{h}_i^p \frac{\partial\mathbf{h}_i^p}{\partial\mathbf{h}_{i-1}^p} \approx sig'(W_i\mathbf{h}_{i-1})W_i^T\delta\mathbf{h}_i^p \ . \tag{29}$$

With difference target propagation the stochastic network can be trained directly, setting the targets to

$$\hat{\mathbf{h}}_2^p = \mathbf{h}_2^p - \eta\frac{\partial L}{\partial\mathbf{h}_2} \qquad \text{and} \qquad \hat{\mathbf{h}}_1^p = \mathbf{h}_1^p + g_2(\hat{\mathbf{h}}_2^p) - g_2(\mathbf{h}_2^p) \tag{30}$$

where $g_i(\mathbf{h}_i^p) = \tanh(V_i\mathbf{h}_i^p)$ is trained by the loss

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ , \tag{31}$$

and layer-local target losses are defined as $L_i = ||\hat{\mathbf{h}}_i^p - \mathbf{h}_i^p||_2^2$.

For evaluation, we averaged the output probabilities for a given input over 100 samples, and classified the example accordingly, following [18]. Results are given in Table 1. We obtained a test error of 1.71% using the baseline method and 1.54% using target propagation, which is – to our knowledge – the best result for stochastic nets on MNIST reported so far. This suggests that target propagation is highly promising for training networks of binary stochastic units.

Table 1. Mean test Error on MNIST for stochastoc networks. The first row shows the results of our experiments averaged over 10 trials. The second row shows the results reported in [18]. M corresponds to the number of samples used for computing output probabilities. We used M=1 during training and M=100 for the test set.

Method	Test Error(%)
Difference Target-Propagation, M=1	1.54%
Straight-through gradient estimator [5] + backprop, M=1 as reported in Raiko et al. [18]	1.71%
as reported in Tang and Salakhutdinov [21], M=20	3.99%
as reported in Raiko et al. [18], M=20	1.63%

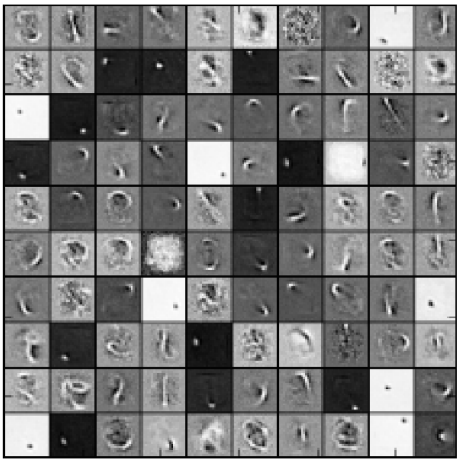

Fig. 4. Filters learned by the back-propagation-free auto-encoder. Each filter corresponds to the hidden weights of one of 100 randomly chosen hidden units. We obtain stroke filters, similar to those usually obtained by regularized auto-encoders.

3.4 Auto-Encoder

We trained a denoising auto-encoder with 1000 hidden units with difference target propagation as described in Section 2.4 on MNIST. As shown in Figure 4 stroke-like filters can be obtained by target propagation. After supervised fine-tuning (using back-propagation), we got a test error of 1.35%. Thus, by training an auto-encoder with target propagation one can learn a good initial representation, which is as good as the one obtained by regularized auto-encoders trained by back-propagation on the reconstruction error.

4 Conclusion

We introduced a novel optimization method for neural networks, called target propagation, which was designed to overcome drawbacks of back-propagation and is biologically more plausible. Target propagation replaces training signals based on partial derivatives by targets which are propagated based on an auto-encoding feedback loop. Difference target propagation is a linear correction for this imperfect inverse mapping which is effective to make target propagation actually work. Our experiments show that target propagation performs comparable to back-propagation on ordinary deep networks and denoising auto-encoders. Moreover, target propagation can be directly used on networks with discretized transmission between units and reaches state of the art performance for stochastic neural networks on MNIST.

Acknowledgments. We would like to thank Junyoung Chung for providing RMSprop code, Caglar Gulcehre and Antoine Biard for general discussion and feedback, Jyri Kivinen for discussion of backprop-free auto-encoder, Mathias Berglund for explanation of his stochastic networks. We thank the developers of Theano [1,8], a Python library which allowed us to easily develop a fast and optimized code for GPU. We are also grateful for funding from NSERC, the Canada Research Chairs, Compute Canada, and CIFAR.

References

1. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N., Bengio, Y.: Theano: new features and speed improvements. In: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop (2012)
2. Bengio, Y.: Learning deep architectures for AI. Now Publishers (2009)
3. Bengio, Y.: Estimating or propagating gradients through stochastic neurons. Tech. Rep. Universite de Montreal (2013). arXiv:1305.2982
4. Bengio, Y.: How auto-encoders could provide credit assignment in deep networks via target propagation. Tech. rep. (2014). arXiv:1407.7906
5. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation (2013). arXiv:1308.3432
6. Bengio, Y., Thibodeau-Laufer, E., Yosinski, J.: Deep generative stochastic networks trainable by backprop. In: ICML 2014 (2014)
7. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Machine Learning Res. **13**, 281–305 (2012)
8. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy), oral Presentation, June 2010
9. Carreira-Perpinan, M., Wang, W.: Distributed optimization of deeply nested systems. In: AISTATS 2014, JMLR W&CP, vol. 33, pp. 10–19 (2014)
10. Erhan, D., Courville, A., Bengio, Y., Vincent, P.: Why does unsupervised pre-training help deep learning? In: JMLR W&CP: Proc. AISTATS 2010, vol. 9, pp. 201–208 (2010)

11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), April 2011
12. Hinton, G., Deng, L., Dahl, G.E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine **29**(6), 82–97 (2012)
13. Konda, K., Memisevic, R., Krueger, D.: Zero-bias autoencoders and the benefits of co-adapting features. Under review on International Conference on Learning Representations (2015)
14. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: NIPS 2012 (2012)
15. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, University of Toronto (2009)
16. LeCun, Y.: Learning processes in an asymmetric threshold network. In: Fogelman-Soulié, F., Bienenstock, E., Weisbuch, G. (eds.) Disordered Systems and Biological Organization, pp. 233–240. Springer-Verlag, Les Houches (1986)
17. LeCun, Y.: Modèles connexionistes de l'apprentissage. Ph.D. thesis, Université de Paris VI (1987)
18. Raiko, T., Berglund, M., Alain, G., Dinh, L.: Techniques for learning binary stochastic feedforward neural networks. In: NIPS Deep Learning Workshop 2014 (2014)
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Tech. rep. (2014). arXiv:1409.3215
20. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. Tech. rep. (2014). arXiv:1409.4842
21. Tang, Y., Salakhutdinov, R.: A new learning algorithm for stochastic feedforward neural nets. In: ICML 2013 Workshop on Challenges in Representation Learning (2013)
22. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning **4** (2012)
23. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Machine Learning Res. **11** (2010)

Appendix

A Proof of Theorem 1

Proof. Given a training example $(\mathbf{x}, \mathbf{y})$ the back-propagation update is given by

$$\delta W_i^{bp} = -\frac{\partial L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})}{\partial W_i} = -J_{f_{i+1}}^T \dots J_{f_M}^T \frac{\partial L}{\partial \mathbf{h}_M} (s_i(\mathbf{h}_{i-1}))^T \ ,$$

where $J_{f_k} = \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} = W_i \cdot S_i'(\mathbf{h}_{k-1}), k = i+1, \dots, M$. Here $S_i'(\mathbf{h}_{k-1})$ is a diagonal matrix with each diagonal element being element-wise derivatives and J_{f_k} is

the Jacobian of $f_k(\mathbf{h}_{k-1})$. In target propagation the target for $\mathbf{h}_M$ is given by $\hat{\mathbf{h}}_M = \mathbf{h}_M - \hat{\eta}\frac{\partial L}{\partial \mathbf{h}_M}$. If all $\mathbf{h}_k$'s are allocated in smooth areas and $\hat{\eta}$ is sufficiently small, we can apply a Taylor expansion to get

$$\hat{\mathbf{h}}_i = g_{i+1}(\dots g_M(\hat{\mathbf{h}}_M)\dots) = g_{i+1}(\dots g_M(\mathbf{h}_M)\dots) - \hat{\eta} J_{g_{i+1}} \dots J_{g_M} \frac{\partial L}{\partial \mathbf{h}_M} + \mathbf{o}(\hat{\eta}) \ ,$$

where $\mathbf{o}(\hat{\eta})$ is the remainder satisfying $\lim_{\hat{\eta}\to 0} \mathbf{o}(\hat{\eta})/\hat{\eta} = \mathbf{0}$. Now, for δW_i^{tp} we have

$$\begin{aligned}
\delta W_i^{tp} &= -\frac{\partial ||\mathbf{h}_i(\mathbf{h}_{i-1}; W_i) - \hat{\mathbf{h}}_i||_2^2}{\partial W_i} \\
&= -(\mathbf{h}_i - (\mathbf{h}_i - \hat{\eta} J_{f_{i+1}}^{-1} \dots J_{f_M}^{-1} \frac{\partial L}{\partial \mathbf{h}_M} + \mathbf{o}(\hat{\eta})))(s_i(\mathbf{h}_{i-1}))^T \\
&= -\hat{\eta} J_{f_{i+1}}^{-1} \dots J_{f_M}^{-1} \frac{\partial L}{\partial \mathbf{h}_M} (s_i(\mathbf{h}_{i-1}))^T + \mathbf{o}(\hat{\eta})(s_i(\mathbf{h}_{i-1}))^T \ .
\end{aligned}$$

We write $\frac{\partial L}{\partial \mathbf{h}_M}$ as $\boldsymbol{l}$, $s_i(\mathbf{h}_{i-1})$ as $\boldsymbol{v}$ and $J_{f_M} \dots J_{f_{i+1}}$ as J for short. Then the inner production of vector forms of δW_i^{bp} and δW_i^{tp} is

$$\begin{aligned}
&\langle vec(\delta W_i^{bp}), vec(\delta W_i^{tp}) \rangle = tr((J^T \boldsymbol{l}\boldsymbol{v}^T)^T (\hat{\eta} J^{-1} \boldsymbol{l}\boldsymbol{v}^T + \mathbf{o}(\hat{\eta})\boldsymbol{v}^T)) \\
&= \hat{\eta} tr(\boldsymbol{v}\boldsymbol{l}^T J J^{-1} \boldsymbol{l}\boldsymbol{v}^T) - tr(\boldsymbol{v}\boldsymbol{l}^T J \mathbf{o}(\hat{\eta})\boldsymbol{v}^T) = \hat{\eta} ||\boldsymbol{v}||_2^2 ||\boldsymbol{l}||_2^2 - \langle J^T \boldsymbol{l}, \mathbf{o}(\hat{\eta}) \rangle ||\boldsymbol{v}||_2^2 \ .
\end{aligned}$$

For $||vec(\delta W_i^{bp})||_2$ and $||vec(\delta W_i^{tp})||_2$ we have

$$||vec(\delta W_i^{bp})||_2 = \sqrt{tr((-J^T \boldsymbol{l}\boldsymbol{v}^T)^T (-J^T \boldsymbol{l}\boldsymbol{v}^T))} = ||\boldsymbol{v}||_2 ||J^T \boldsymbol{l}||_2 \le ||\boldsymbol{v}||_2 ||J^T||_2 ||\boldsymbol{l}||_2$$

and similarly

$$||vec(\delta W_i^{tp})||_2 \le \hat{\eta} ||\boldsymbol{v}||_2 ||J^{-1}||_2 ||\boldsymbol{l}||_2 + ||\mathbf{o}(\hat{\eta})||_2 ||\boldsymbol{v}||_2 \ ,$$

where $||J^T||_2$ and $||J^{-1}||_2$ are matrix Euclidean norms, i.e. the largest singular value of $(J_{f_M} \dots J_{f_{i+1}})^T$, λ_{max}, and the largest singular value of $(J_{f_M} \dots J_{f_{i+1}})^{-1}$, $\frac{1}{\lambda_{min}}$ (λ_{min} is the smallest singular value of $(J_{f_M} \dots J_{f_{i+1}})^T$, because f_k is invertable, so all the smallest singular values of Jacobians are larger than 0). Finally, if $\hat{\eta}$ is sufficiently small, the angle α between $vec(\delta W_i^{bp})$ and $vec(\delta W_i^{tp})$ satisfies:

$$\begin{aligned}
cos(\alpha) &= \frac{\langle vec(\delta W_i^{bp}), vec(\delta W_i^{tp}) \rangle}{||vec(\delta W_i^{bp})||_2 \cdot ||vec(\delta W_i^{tp})||_2} \\
&\ge \frac{\hat{\eta} ||\boldsymbol{v}||_2^2 ||\boldsymbol{l}||_2^2 - \langle J^T \boldsymbol{l}, \mathbf{o}(\hat{\eta}) \rangle ||\boldsymbol{v}||_2^2}{(||\boldsymbol{v}||_2 \lambda_{max} ||\boldsymbol{l}||_2)(\hat{\eta} ||\boldsymbol{v}||_2 (\frac{1}{\lambda_{min}}) ||\boldsymbol{l}||_2 + ||\mathbf{o}(\hat{\eta})||_2 ||\boldsymbol{v}||_2)} \\
&= \frac{1 + \frac{-\langle J^T \boldsymbol{l}, \mathbf{o}(\hat{\eta}) \rangle}{\hat{\eta} ||\boldsymbol{l}||_2^2}}{\frac{\lambda_{max}}{\lambda_{min}} + \frac{\lambda_{max} ||\mathbf{o}(\hat{\eta})||_2}{\hat{\eta} ||\boldsymbol{l}||_2}} = \frac{1 + \Delta_1(\hat{\eta})}{\frac{\lambda_{max}}{\lambda_{min}} + \Delta_2(\hat{\eta})}
\end{aligned}$$

where the last expression is positive if $\hat{\eta}$ is sufficiently small and $cos(\alpha) \le 1$ is trivial.

B Proof of Theorem 2

Proof. Let $\mathbf{e} = \hat{\mathbf{h}}_i - \mathbf{h}_i$. Applying Taylor's theorem twice, we get

$$\begin{aligned}\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1}) &= \hat{\mathbf{h}}_i - f_i(\mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i)) = \hat{\mathbf{h}}_i - f_i(\mathbf{h}_{i-1} + J_{g_i}\mathbf{e} + \mathbf{o}(||\mathbf{e}||_2)) \\ &= \hat{\mathbf{h}}_i - f_i(\mathbf{h}_{i-1}) - J_{f_i}(J_{g_i}\mathbf{e} + \mathbf{o}(||\mathbf{e}||_2)) - \mathbf{o}(||J_{g_i}\mathbf{e} + \mathbf{o}(||\mathbf{e}||_2)||_2) \\ &= \hat{\mathbf{h}}_i - \mathbf{h}_i - J_{f_i}J_{g_i}\mathbf{e} - \mathbf{o}(||\mathbf{e}||_2) = (I - J_{f_i}J_{g_i})\mathbf{e} - \mathbf{o}(||\mathbf{e}||_2)\end{aligned}$$

where the vector $\mathbf{o}(||\mathbf{e}||_2)$ represents the remainder satisfying $\lim_{\mathbf{e}\to\mathbf{0}} \mathbf{o}(||\mathbf{e}||_2)/||\mathbf{e}||_2 = \mathbf{0}$. Then for $||\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1})||_2^2$ we have

$$\begin{aligned}||\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1})||_2^2 &= ((I - J_{f_i}J_{g_i})\mathbf{e} - \mathbf{o}(||\mathbf{e}||_2))^T((I - J_{f_i}J_{g_i})\mathbf{e} - \mathbf{o}(||\mathbf{e}||_2)) \\ &= \mathbf{e}^T(I - J_{f_i}J_{g_i})^T(I - J_{f_i}J_{g_i})\mathbf{e} - \mathbf{o}(||\mathbf{e}||_2)^T(I - J_{f_i}J_{g_i})\mathbf{e} \\ &\quad -\mathbf{e}^T(I - J_{f_i}J_{g_i})^T\mathbf{o}(||\mathbf{e}||_2) + \mathbf{o}(||\mathbf{e}||_2)^T\mathbf{o}(||\mathbf{e}||_2)) \\ &= \mathbf{e}^T(I - J_{f_i}J_{g_i})^T(I - J_{f_i}J_{g_i})\mathbf{e} + o(||\mathbf{e}||_2^2) \\ &\leq \lambda||\mathbf{e}||_2^2 + |o(||\mathbf{e}||_2^2)| \qquad \text{(A-1)}\end{aligned}$$

where $o(||\mathbf{e}||_2^2)$ is the scalar value resulting from all terms depending on $\mathbf{o}(||\mathbf{e}||_2)$ and λ is the largest eigenvalue of $(I - J_{f_i}J_{g_i})^T(I - J_{f_i}J_{g_i})$. If $\mathbf{e}$ is sufficiently small to guarantee $|o(||\mathbf{e}||_2^2)| < (1 - \lambda)||\mathbf{e}||_2^2$, then the left of Equation (A-1) is less than $||\mathbf{e}||_2^2$ which is just $||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2$.