TARGET PROPAGATION

Dong-Hyun Lee[1], **Saizheng Zhang**[1], **Antoine Biard**[12], **& Yoshua Bengio**[13]
[1]Université de Montréal, [2]Ecole polytechnique, [3] CIFAR Senior Fellow

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 as backprop. 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 is very effective to make target propagation actually work, along with adaptive learning rates.

1 INTRODUCTION

Recently, deep neural networks have achieved great success in hard AI tasks (Bengio, 2009; Hinton et al., 2012; Krizhevsky et al., 2012; Sutskever et al., 2014), 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 (Szegedy et al., 2014) with 19 or 22 layers – longer-term dependencies, or stronger non-linearities, the composition of many non-linear operations becomes more non-linear. To make this concrete, consider the composition of many hyperbolic tangent units. In general, this means that derivatives obtained by backprop 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 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 backprop 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 backprop, 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 (LeCun, 1986; 1987), 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 (Carreira-Perpinan and Wang, 2014). 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, but this is left for future work.

In this paper, we provide several experimental results on rather deep neural networks as well as discrete and stochastic networks. The results show that the proposed form of target propagation is comparable to back-propagation with RMSprop (Tieleman and Hinton, 2012) - a very popular setting to train deep networks nowadays.

2 Proposed Target Propagation Implementation

Although many variants of the general principle of target propagation can be devised, this paper focuses on a specific approach, described below, which fixes a problem in the formulation introduced in an earlier technical report (Bengio, 2014).

2.1 Formulating Targets

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

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

where $\mathbf{h}_i$ is the i th hidden layer, $\mathbf{h}_M$ is the output of network, $\mathbf{h}_0$ is the input $\mathbf{x}$, s_i is the non-linearity (e.g. $tanh$ or $sigmoid$) and W_i corresponds to the weights for layer i, f_i is the i-th layer feed-forward mapping. For simplicity (but an abuse) of notation, the bias term of each layer is included in W_i. We define $\theta_W^{i,j}$ as the subset of network parameters $\theta_W^{i,j} = \{W_k, k = i+1, \ldots, j\}$. By this notion, each $\mathbf{h}_j$ is a function of $\mathbf{h}_i$ where $\mathbf{h}_j = \mathbf{h}_j(\mathbf{h}_i; \theta_W^{i,j})$ for $0 \leq i < j \leq M$. We define the global loss function for one sample $(\mathbf{x}, \mathbf{y})$ as $L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})$, where

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M}) &= loss(\mathbf{h}_M(\mathbf{x}; \theta_W^{0,M}), \mathbf{y}) \\ &= loss(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i}); \theta_W^{i,M}), \mathbf{y}), i = 1, \ldots, M-1 \end{aligned} \tag{2}$$

Here $loss(\cdot)$ can be any kind of loss measure function (e.g. MSE, Binomial cross-entropy). Then the expected loss function over the whole data distribution $p(\mathbf{x}, \mathbf{y})$ is written

$$\mathcal{L} = \mathop{\mathbb{E}}_{p}\{L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})\}. \tag{3}$$

Training a network with back-propagation corresponds to propagating error signals through the network, signals which indicate how the unit activations or parameters of the network could be updated to decrease the expected loss $\mathcal{L}$. In very deep networks with strong non-linearities, error propagation could become useless in lower layers due to the difficulties associated with strong non-linearities, e.g. exploding or vanishing gradients, as explained above. Given a data sample $(\mathbf{x}, \mathbf{y})$ and the corresponding activations of the hidden layers $\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})$, a possible solution to avoid these issues could be to assign a nearby value $\hat{\mathbf{h}}_i$ for each $\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})$ that could lead to a lower global loss. For a sample $(\mathbf{x}, \mathbf{y})$, we name such value $\hat{\mathbf{h}}_i$ a *target*, with the objective that

$$loss(\mathbf{h}_M(\hat{\mathbf{h}}_i; \theta_W^{i,M}), \mathbf{y}) < loss(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x}; \theta_W^{0,i}); \theta_W^{i,M}), \mathbf{y}) \tag{4}$$

In local layer i, we hope to train the network to make $\mathbf{h}_i$ move towards $\hat{\mathbf{h}}_i$. As $\mathbf{h}_i$ approaches $\hat{\mathbf{h}}_i$, if the path leading from $\mathbf{h}_i$ to $\hat{\mathbf{h}}_i$ is smooth enough, we expect that the global loss $L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})$ would then decrease. To update the W_i, instead of using the error signals propagated from global loss $L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})$ with back-propagation, we define a layer-local target loss L_i. For example, using a MSE loss gives :

$$L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i) = ||\hat{\mathbf{h}}_i - \mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})||_2^2. \tag{5}$$

In such a case, W_i is updated locally within its layer via stochastic gradient descent, where $\hat{\mathbf{h}}_i$ is considered as a *constant* with respect to W_i

$$W_i^{(t+1)} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i)}{\partial W_i} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{\mathbf{h}}_i, \mathbf{h}_i)}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i(\mathbf{x}; \theta_W^{0,i})}{\partial W_i}. \tag{6}$$

In this context, derivatives can be used within a local layer because they typically correspond to computation performed inside each neuron. The severe non-linearity that may originate from the chain rule arises mostly when it 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. What a target propagation method requires is a way to compute the target $\hat{\mathbf{h}}_{i-1}$ from the higher-level target $\hat{\mathbf{h}}_i$ and from $\mathbf{h}_i$, such that it is likely to respect the constraint defined by Eq.4 and at least satisfies weaker assumptions, like for example :

$$L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1})) < L_i(\hat{\mathbf{h}}_i, f_i(\mathbf{h}_{i-1})) \tag{7}$$

2.2 How to assign a proper target to each layer

The problem of credit assignment is the following: how should each unit change its output so as to increase the likelihood of reducing the global loss?

With the back-propagation algorithm, we compute the gradient of the loss with respect to the output of each layer, and we can interpret that gradient as an error signal. That error signal is propagated recursively from the top layer to the bottom layer using the chain rule.

$$\delta \mathbf{h}_{i-1} = \frac{\partial L}{\partial \mathbf{h}_{i-1}} = \frac{\partial L}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \delta \mathbf{h}_i \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \tag{8}$$

In the target-prop setting, the signal that gives the direction for the update is the difference $\hat{\mathbf{h}} - \mathbf{h}$. So we can rewrite the first and the last terms of the previous equation and we get :

$$\hat{\mathbf{h}}_{i-1} - \mathbf{h}_{i-1} = (\hat{\mathbf{h}}_i - \mathbf{h}_i) \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = -\frac{1}{2} \frac{\partial ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2}{\partial \mathbf{h}_{i-1}} \tag{9}$$

Still in the target-prop framework, the parameter update at a specific layer is obtain by a stochastic gradient descent (sgd) step to minimize the layer wise cost and can be written :

$$W_i^{(t+1)} = W_i^{(t)} - \eta \frac{\partial ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2}{\partial W_i} \tag{10}$$

With back-propagation to compute the gradients at each layer, we can consider that the target of a lower layer is computed from the target of an upper layer as if gradient descent had been applied (non-parametrically) to the layer's activations, such that $L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1})) < L_i(\hat{\mathbf{h}}_i, f_i(\mathbf{h}_{i-1}))$. This could be called "target propagation through optimization" and reminiscent of (Carreira-Perpinan and Wang, 2014).

However, in order to avoid the chain of derivatives through many layers, another option, introduced in (Bengio, 2014), is to take advantage of an "approximate inverse". For example, suppose that we have a function g_i such that

$$f_i(g_i(\hat{\mathbf{h}}_i)) \approx \hat{\mathbf{h}}_i, \tag{11}$$

then choosing $\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i)$ would have the consequence that the level i loss L_i (to make the output match the target at level i) would be minimized. This is the vanilla target propagation introduced in (Bengio, 2014):

$$\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i) \tag{12}$$

Note that g_i does not need to invert f_i everywhere, only in the vicinity of the targets. If the feedback mappings were the perfect inverses of the feed-forward mappings ($g_i = f_i^{-1}$), we would get directly

$$L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1})) = L_i(\hat{\mathbf{h}}_i, f_i(g_i(\hat{\mathbf{h}}_i))) = L_i(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_i) = 0. \tag{13}$$

This would be ideal for target propagation. In fact, we have the following proposition for the case of a perfect inverse:

Proposition 1. *Assume that g_i is a perfect inverse of f_i, where $g_i = f_i^{-1}, i = 1, ..., M-1$ and f_i satisfies: 1. f_i is a linear mapping or, 2. $\mathbf{h}_i = f_i(\mathbf{h}_{i-1}) = W_i s_i(\mathbf{h}_{i-1})$, which is another way to obtain a non-linear deep network structure (here s_i can be any differentiable monotonically increasing element-wise function). Consider one update for both target propagation and back-propagation, with the target propagation update (with perfect inverse) in ith layer being δW_i^{tp}, and the back-propagation update being δW_i^{bp}. Then the angle α_i between δW_i^{tp} and δW_i^{bp} is bounded by*

$$0 \leq \alpha_i \leq cos^{-1}(\frac{\lambda_{min}}{\lambda_{max}}) \tag{14}$$

Here λ_{max} and λ_{min} are the largest and smallest singular values of $(J_{f_{M-1}} \dots J_{f_{i+1}})^T$, where J_{f_k} is the Jacobian matrix of f_k.

See proof in Appendix A[1]. Proposition 1 says that if f_i has the assumed structures, the descent direction of target propagation with perfect inverse at least partly matches with the gradient descent direction, which makes the global loss always decrease. But a perfect inverse may be impractical for computational reasons and unstable (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}). So here we prefer 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:

$$\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i) = s_i(V_i \mathbf{h}_i), \quad i = 0, ..., M \tag{15}$$

where s_i is a non-linearity associated with the decoder and V_i the matrix of feedback weights for layer i. 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:

$$L_i^{inv} = ||f_i(g_i(\hat{\mathbf{h}}_i)) - \hat{\mathbf{h}}_i||_2^2 \tag{16}$$

This makes $f_i(\hat{\mathbf{h}}_{i-1})$ closer to $\hat{\mathbf{h}}_i$, thus making $L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1}))$ closer to zero. But we should get inverse mapping around the targets. This could help to compute targets which have never been seen before. For this, we can modify inverse loss using noise injection.

$$L_i^{inv} = ||f_i(g_i(\hat{\mathbf{h}}_i + \epsilon)) - (\hat{\mathbf{h}}_i + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \tag{17}$$

However, the imperfection of the inverse yields severe optimization problems which has brought us to propose the following linearly corrected formula for the target propagation:

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

We call this variant "*difference target propagation*" and we found in the experiments described below that it can significantly reduce the optimization problems associated with Eq. 12. Note that if g_i was an inverse of f_i, then difference target propagation would be equivalent to the vanilla target propagation of Eq. 12. For the "difference target propagation", we have following proposition:

Proposition 2. *During the $t+1$ th update in difference target propagation, we use $L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)})$ to update $V_i^{(t+1)}$ and we define $\bar{L}_i^{inv}(V_i, W_i^{(t)})$ as the expected local auto-encoder-like loss function over all possible $\hat{\mathbf{h}}_i^{(t)} + \epsilon$ with $W_i^{(t)}$ fixed,*

$$\bar{L}_i^{inv}(V_i, W_i^{(t)}) = \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)})\} \tag{19}$$

[1] In the arXiv version of this paper.

If 1.$\bar{L}_i^{inv}(V_i, W_i^{(t)})$ has only one minimum with optimal $V_i^*(W_i^{(t)})$; 2. proper learning rates for V_i and W_i are given; 3. All the Jacobian and Hessian like matrices are bounded during learning; 4. $\nabla_{V_i}\bar{L}_i^{inv}(V_i, W_i^{(t)})$ always points towards optimal $V_i^*(W_i^{(t)})$; 5. $\mathbb{E}\{V_i^*(W_i^{(t+1)}) - V_i^*(W_i^{(t)}) \mid W_i^{(t)}\} = 0$. Then $V_i^{(t)} - V_i^*(W_i^{(t)})$ will almost surely converge to 0 at t th update when t goes to infinity. Condition 1, 2, 4 follow the settings of stochastic gradient descent convergence similar to (Bottou, 1998).

See proof in Appendix[2]. Proposition 2 says that in difference target propagation, g_i can learn a good approximation of f_i's inverse, which will quickly minimize the auto-encoder-like error of each layer.

The top layer does not have a layer above it and it has its own loss function which is also the global loss function. In our experiments we chose to set the first target of the target-prop chain such that $L(\hat{\mathbf{h}}_{M-1}) < L(\mathbf{h}_{M-1})$. This can be achieved for classification loss as follows:

$$\hat{\mathbf{h}}_{M-1} = \mathbf{h}_{M-1} - \eta_0 \frac{\partial L}{\partial \mathbf{h}_{M-1}} \tag{20}$$

where η_0 is a "target-prop" learning rate for making the first target – i.e. one of the hyper-parameters. Making the first target at layer $M-1$ with the specific output and loss function instead of the output layer can reduce algorithm's dependence on specific type of output and loss function. So we can apply consistent formulation to compute target in lower layers. And then, once we have a method to assign proper targets to each layer, we only have to optimize layer-local target losses to decrease global loss function.

2.3 The advantage of difference target propagation

In order to make optimization stable in target propagation, $\mathbf{h}_{i-1}$ should approach to $\hat{\mathbf{h}}_{i-1}$ as $\mathbf{h}_i$ approaches to $\hat{\mathbf{h}}_i$. If not, even though optimization is finished in upper layers, the weights in lower layers would continue to be updated. As a result, the target losses in upper layers as well as the global loss can increase even after we reach the optimum situation. So we found the following condition to greatly improve the stability of the optimization.

$$\mathbf{h}_i = \hat{\mathbf{h}}_i \;\rightarrow\; \mathbf{h}_{i-1} = \hat{\mathbf{h}}_{i-1} \tag{21}$$

If we have the perfect inverse $g_i = f_i^{-1}$, it holds with vanilla target propagation because

$$\mathbf{h}_{i-1} = f_i^{-1}(\mathbf{h}_i) = g_i(\hat{\mathbf{h}}_i) = \hat{\mathbf{h}}_{i-1}. \tag{22}$$

Although it is not guaranteed with an imperfect inverse mapping $g_i \neq f_i^{-1}$ in vanilla target propagation, with difference target propagation, it naturally holds by construction.

$$\hat{\mathbf{h}}_{i-1} - \mathbf{h}_{i-1} = g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i) \tag{23}$$

More precisely, we can show that the when the input of a layer become the target of lower layer computed by difference target propagation, the output of the layer moves toward the side of its target

$$f_i(\hat{\mathbf{h}}_{i-1}) = f_i(\mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i)) \sim \mathbf{h}_i + f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i)(\hat{\mathbf{h}}_i - \mathbf{h}_i) \tag{24}$$

$$(\hat{\mathbf{h}}_i - \mathbf{h}_i)^T(f_i(\hat{\mathbf{h}}_{i-1}) - \mathbf{h}_i) \sim (\hat{\mathbf{h}}_i - \mathbf{h}_i)^T f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i)(\hat{\mathbf{h}}_i - \mathbf{h}_i)) > 0 \tag{25}$$

if $\hat{\mathbf{h}}_i \sim \mathbf{h}_i$ and $f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i) = (f_i(g_i(\mathbf{h}_i)))'$ is positive definite. It is far more flexible condition than the perfect inverseness. Even when g_i is a random mapping, this condition can be satisfied. Actually, if f_i and g_i are linear mappings and g_i has a random matrix, difference target propagation is equivalent to feedback alignment (Lillicrap et al., 2014) which works well on many datasets. As a target framework, we also can show that the output of the layer get closer to its target

$$||\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1})||_2^2 < ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2 \tag{26}$$

if $\hat{\mathbf{h}}_i \sim \mathbf{h}_i$ and the maximum eigenvalue of $(I - f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i))^T(I - f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i))$ is less than 1 because $\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1}) \sim [I - f_i'(\mathbf{h}_{i-1})g_i'(\mathbf{h}_i)](\hat{\mathbf{h}}_i - \mathbf{h}_i)$. Moreover, as g_i approaches to f_i^{-1}, this approaches to vanilla target propagation formula in (Bengio, 2014).

$$g_i(\mathbf{h}_i) \sim \mathbf{h}_{i-1} \;\rightarrow\; \hat{\mathbf{h}}_{i-1} = \mathbf{h}_{i-1} - g_i(\mathbf{h}_i) + g_i(\hat{\mathbf{h}}_i) \sim g_i(\hat{\mathbf{h}}_i) \tag{27}$$

[2]In the arXiv version of this paper.

3 EXPERIMENTS

3.1 VERY DEEP NETWORKS

As a primary objective, we investigated whether one can train ordinary deep networks on the MNIST dataset. The network has 7 hidden layers and the number of hidden units is 240. The activation function is the hyperbolic tangent (tanh). we use RMSprop as a adaptive learning rate algorithm because we do not have a global loss to optimize. Instead, we have the local layer-wise target losses that might need their learning rates to be on different scales (this is actually what we find when we do hyper-parameter optimization over the separate learning rates for each layer). To get this result, we chose the optimal hyper-parameters for the best training cost using random search. And the weights are initialized with orthogonal random matrices.

To improve optimization results, layers are updated one at a time from the bottom layer to the top layer, thus avoiding issues with the current input of each layer being invalid if we update all layers at once.

As a baseline, back-propagation with RMSprop is used. The same weight initialization and adaptive learning rate and hyper-parameter searching method are used as with target-prop. We report our results in figure 1. We got test error 1.92% in target propagation, 1.88% in back propagation. And we got negative log-likelihood 3.38×10^{-6} in target propagation, 1.81×10^{-5} in back propagation. These results are averaged over 5 trials using chosen hyper-parameters.

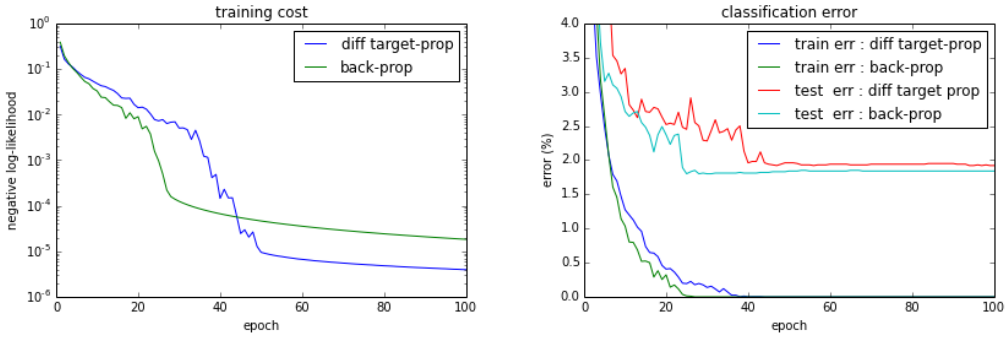

Figure 1: Training cost (left) and train/test classification error (right) with target-prop and backprop. Target propagation can converge to lower values of cost with the similar generalization performance to backprop.

3.2 NETWORKS WITH DISCRETIZED TRANSMISSION BETWEEN UNITS

As an example of extremely non-linear networks, we investigated whether one can train even discrete networks on the MNIST dataset. The network architecture is 784-500-500-10 and only the 1st hidden layer is discretized. Instead of just using the step activation function, we have normal neural layers with tanh, and signals are discretized when transporting between layer 1 and layer 2, based on biological considerations and the objective of reducing the communication cost between neurons.

$$\mathbf{h}_1 = f_1(\mathbf{x}) = tanh(W_1\mathbf{x}) \tag{28}$$

$$\mathbf{h}_2 = f_2(\mathbf{h}_1) = tanh(W_2 sign(\mathbf{h}_1)) \tag{29}$$

$$p(\mathbf{y}|\mathbf{x}) = f_3(\mathbf{h}_2) = softmax(W_3\mathbf{h}_2) \tag{30}$$

where $sign(x) = 1$ if $x > 0$, 0 if $x \leq 0$. We also use feedback mapping with inverse loss. But in this case, we cannot optimize full auto-encoding loss because it is not differentiable. Instead, we can use only reconstruction loss given the input and the output of feed-forward mapping.

$$g_2(\mathbf{h}_2) = tanh(V_2 sign(\mathbf{h}_2)) \tag{31}$$

$$L_2^{inv} = ||g_2(f_2(\mathbf{h}_1 + \epsilon)) - (\mathbf{h}_1 + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \tag{32}$$

If only feed-forward mapping is discrete, we can train the network using back-propagation with biased gradient estimator as if we train continuous networks with tanh. However, if training signals also should be discrete, it is very hard to train using back-propagation. So we compare our result to two backprop baselines. One baseline is to train the discrete networks directly so we cannot train W_1 using backprop. It still can make training error be zero but we cannot learn any meaningful representation on h_1, so test error is poor in Figure 3 (left). Another baseline is to train continuous-activation networks with tanh and to test with the discrete networks (that is, indirect training). Though the estimated gradient is biased so training error does not converge to zero, generalization performance is fairly good, as seen in Figure 2 (right), 3 (left).

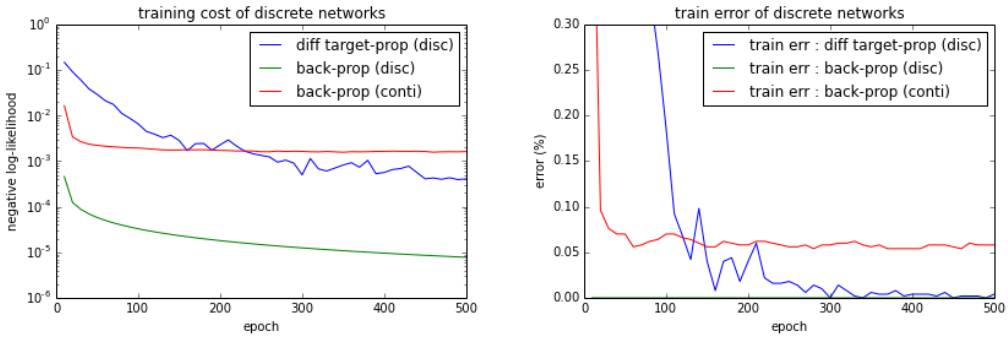

Figure 2: Training cost (left) and train error (right) while training discrete networks. (backprop disc) Because training signals cannot go across a discretization step, layer 1 cannot be trained by back-prop. Though training cost is very low, it overfits, and test error is high. (backprop conti) An option is to use a biased gradient estimator when we train the network as if it were continuous, and test on the discretized version of the network. It is an indirect training, not overcoming the discreteness during training. Training error cannot approach zero due to the biased estimator. (diff target-prop) Target propagation can train discrete networks directly, so training error actually approaches zero. Moreover, test error is comparable to (backprop conti). It clearly suggests that using target-prop, training signals can go across a discretization step successfully.

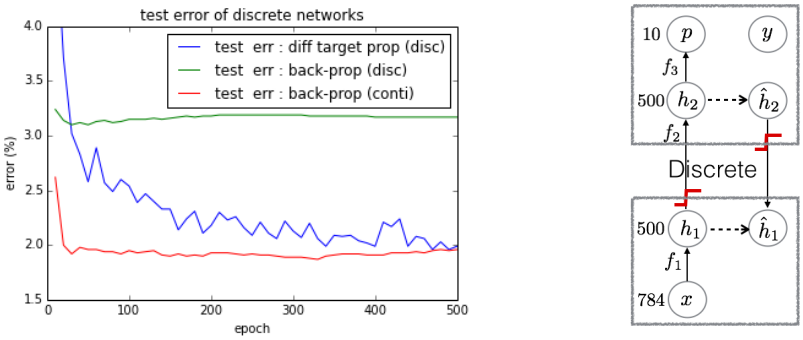

Figure 3: Test error (left) and diagram of the discrete networks (right). The output of $\mathbf{h}_1$ is discretized because signals must be communicated from $\mathbf{h}_1$ to $\mathbf{h}_2$ through a long cable, so binary representations are preferred in order to conserve energy. Training signals are also discretized through this cable (since feedback paths are computed by bona-fide neurons), so it is very difficult to train the network directly. The test error of diff target-prop is comparable to (backprop conti) even though both feed-forward signals and training signals are discretized.

However, with target propagation, because we can learn an inverse mapping with a discrete layer and we do not use derivatives through layers, we can successfully train discrete networks directly. Though training convergence is slower, training error approaches zero, unlike the biased gradient estimator with backprop and continuous networks. The remarkable thing is that test error is comparable to biased gradient estimator with backprop and continuous networks. We can train W_1 properly, that is, training signals can go across the discrete region successfully. Of course, as shown on the figure, the generalization performance is much better than the vanilla backprop baseline.

3.3 STOCHASTIC NETWORKS

Another interesting learning problem which backprop cannot deal with well is stochastic networks with discrete units. Recently such networks have attracted attention (Bengio, 2013; Tang and Salakhutdinov, 2013; Bengio et al., 2013) because a stochastic network can learn a multi-modal conditional distribution $P(Y|X)$, which is important for structured output predictions. Training networks of stochastic binary units is also motivated from biology, i.e., 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 Raiko et al. (2014), the network architecture is 784-200-200-10 and the hidden units are stochastic binary units with the probability of turning on given by a sigmoid activation.

$$\mathbf{h}_i^p = \sigma(W_i \mathbf{h}_{i-1}), \quad \mathbf{h}_i = sample(\mathbf{h}_i^p) \tag{33}$$

where $sample(p)$ is a binary random variable which is 1 with probability p.

As a baseline, we consider a biased gradient estimator in which we do back-propagation as if it were just continuous sigmoid networks. This baseline showed the best performance in Raiko et al. (2014).

$$\delta \mathbf{h}_{i-1}^p = \delta \mathbf{h}_i^p \frac{\partial \mathbf{h}_i^p}{\partial \mathbf{h}_{i-1}^p} \sim \sigma'(W_i \mathbf{h}_{i-1}) W_i^T \delta \mathbf{h}_i^p \tag{34}$$

In target propagation, we can train this network directly.

$$\hat{\mathbf{h}}_2^p = \mathbf{h}_2^p - \eta \frac{\partial L}{\partial \mathbf{h}_2}, \quad \hat{\mathbf{h}}_1^p = \mathbf{h}_1^p + g_2(\hat{\mathbf{h}}_2^p) - g_2(\mathbf{h}_2^p) \tag{35}$$

$$g_i(\mathbf{h}_i^p) = tanh(V_i \mathbf{h}_i^p), \quad L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \tag{36}$$

Using layer-local target losses $L_i = ||\hat{\mathbf{h}}_i^p - \mathbf{h}_i^p||_2^2$, we can update all the weights.

We obtained a test error of 1.51% using target propagation and 1.71% using the baseline method. In the evalution, we averaged the output probabilities of an example over 100 noise samples, and then classify the example accordingly, following Raiko et al. (2014) This suggests that target propagation can directly deal with networks of binary stochastic units.

Method	Test Error(%)
Difference Target-Propagation, M=1	1.51%
Biased gradient estimator like backprop (followed by Raiko and Berglund, 2014, M=1)	1.71%
Tang and Salakhutdinov, 2013, M=20	3.99%
Raiko and Berglund, 2014, M=20	1.63%

Table 1: Test Error on MNIST with stochastoc networks. The first row shows the results in our experiments. These are averaged results over 5 trials using the same hyper-parameter combination which is chosen for the best valid error. The second row shows the results from (Raiko et al., 2014). In our experiment, we used RMS-prop and maximum epochs is 1000 different from (Raiko et al., 2014). M is the number of samples when computing output probability. we use M=100 at test time.

3.4 BACKPROP-FREE AUTO-ENCODER

Auto-encoders are interesting building blocks for learning representations, especially deep ones (Erhan et al., 2010). In addition, as we have seen, training an auto-encoder is also part of what is

required for target propagation according to the approach presented here, in order to train the feedback paths that propagate the targets. We show here how a regularized auto-encoder can be trained using difference target propagation, without backprop.

Like in the work on denoising auto-encoders (Vincent et al., 2010) and Generative Stochastic Networks (Bengio et al., 2014), we consider the denoising auto-encoder like a stochastic network with noise injected in input and hidden units, trained to minimize a reconstruction loss.

$$\mathbf{h} = f(\mathbf{x}) = sigm(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{37}$$

$$\mathbf{z} = g(\mathbf{h}) = sigm(\mathbf{W}^T(\mathbf{h} + \epsilon) + \mathbf{c}), \quad \epsilon \sim N(0, \sigma) \tag{38}$$

$$L = ||\mathbf{z} - \mathbf{x}||_2^2 + ||f(\mathbf{x} + \epsilon) - \mathbf{h}||_2^2, \quad \epsilon \sim N(0, \sigma) \tag{39}$$

where we also use regularization to obtain contractive mappings. In order to train this network without backprop (that is, chain rule), we can use difference target propagation. At first, the target of $\mathbf{z}$ is just $\mathbf{x}$, so we can train reconstruction mapping g with $L_g = ||g(\mathbf{h}) - \mathbf{x}||_2^2$ in which $\mathbf{h}$ is considered as a constant. And then, we compute the target $\hat{\mathbf{h}}$ of hidden units following difference target propagation.

$$\hat{\mathbf{h}} = \mathbf{h} + f(\hat{\mathbf{z}}) - f(\mathbf{z}) = 2\mathbf{h} - f(\mathbf{z}) \tag{40}$$

where f is used as a inverse mapping of g without additional functions, and $f(\hat{\mathbf{z}}) = f(\mathbf{x}) = \mathbf{h}$. As a target loss for the hidden layer, we can use $L_f = ||f(\mathbf{x} + \epsilon) - \hat{\mathbf{h}}||_2^2$ in which regularization for contractive mapping is also incorporated and $\hat{\mathbf{h}}$ is considered as a constant. Using layer-local target losses L_f and L_g, we train on MNIST a denoising auto-encoder whose architecture is 784-1000-784. Stroke-like filters can be obtained (See Figure 4) and after supervised fine-tuning (using backprop), we get 1.35% test error. That is, our auto-encoder can train a good initial representation as good as the one obtained by regularized auto-encoders trained by backprop on the reconstruction error.

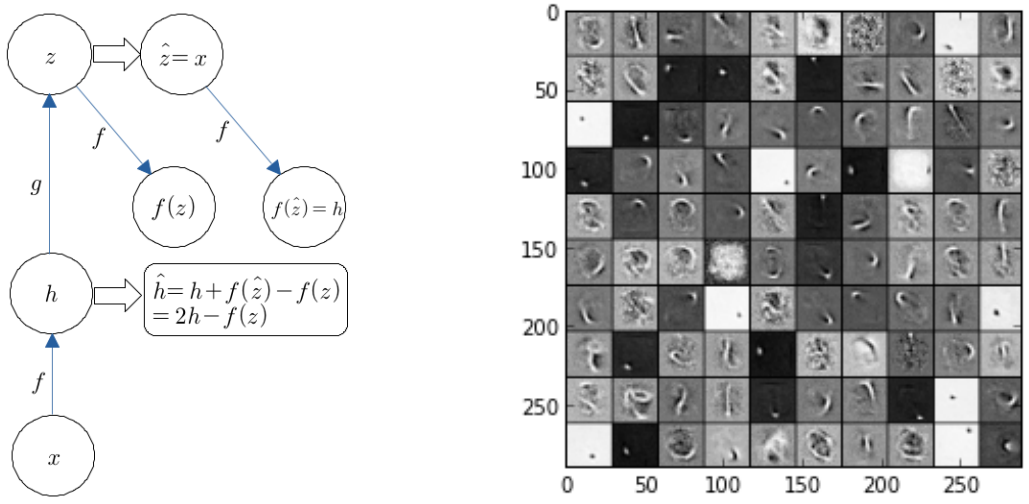

Figure 4: Diagram of the evaluated backprop-free auto-encoder (left) and its trained filters, i.e., layer 1 weights (right). Even though we train the networks using only layer-local target losses instead of a global loss (reconstruction error), we obtain stroke filters, similar to those usually obtained by regularized auto-encoders. Moreover, we can pre-train good hidden representations for initialization a classifier, which achieved a test error of 1.35% (after fine-tuning the whole network). The target $\hat{\mathbf{h}}$ of the hidden layer is naturally derived from difference target propagation.

ACKNOWLEDGMENTS

We would like to thank Junyoung Chung for providing RMSprop code, Caglar Gulcehre 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 (Bergstra et al., 2010; Bastien et al., 2012), a Python library which allowed us to easily develop a fast and optimized code for GPU. We also thank the developers of Pylearn2 (Goodfellow et al., 2013), a Python library built on the top of Theano which allowed us to easily interface the data sets with our Theano code. We are also grateful for funding from NSERC, the Canada Research Chairs, Compute Canada, and CIFAR.

REFERENCES

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers.

Bengio, Y. (2013). Estimating or propagating gradients through stochastic neurons. Technical Report arXiv:1305.2982, Universite de Montreal.

Bengio, Y. (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. Technical report, arXiv preprint arXiv:1407.7906.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *ICML'2014*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bottou, L. (1998). Online algorithms and stochastic approximations. In Saad, D., editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK. revised, oct 2012.

Carreira-Perpinan, M. and Wang, W. (2014). Distributed optimization of deeply nested systems. In *AISTATS'2014, JMLR W&CP*, volume 33, pages 10–19.

Erhan, D., Courville, A., Bengio, Y., and Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *JMLR W&CP: Proc. AISTATS'2010*, volume 9, pages 201–208.

Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

Hinton, G., Deng, L., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS'2012*.

LeCun, Y. (1986). Learning processes in an asymmetric threshold network. In Fogelman-Soulié, F., Bienenstock, E., and Weisbuch, G., editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer-Verlag, Les Houches, France.

LeCun, Y. (1987). *Modèles connexionistes de l'apprentissage*. PhD thesis, Université de Paris VI.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support learningin deep neural networks. Technical report, arXiv preprint arXiv:1411.0247.

Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2014). Techniques for learning binary stochastic feedforward neural networks. *NIPS Deep Learning Workshop 2014*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. Technical report, arXiv preprint arXiv:1409.3215.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. Technical report, arXiv preprint arXiv:1409.4842.

Tang, Y. and Salakhutdinov, R. (2013). A new learning algorithm for stochastic feedforward neural nets. ICML'2013 Workshop on Challenges in Representation Learning.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Res.*, 11.

APPENDIX

A PROOF OF PROPOSITION 1

Proof. During one update, the training sample is $(\mathbf{x}, \mathbf{y})$, with assumed f_i we have:

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

Here s_i is identity element-wise function if f_i is linear mapping. According to the loss function in section 2.1, the back-propagation update δW_i^{bp} is then

$$\begin{aligned}
\delta W_i^{bp} &= -\eta_{bp} \frac{\partial L(\mathbf{x}, \mathbf{y}; \theta_W^{0,M})}{\partial W_i} \\
&= -\eta_{bp} (\frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i})^T \dots (\frac{\partial \mathbf{h}_{M-1}}{\partial \mathbf{h}_{M-2}})^T \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}} (s_i(\mathbf{h}_{i-1}))^T \\
&= -\eta_{bp} J_{f_{i+1}}^T \dots J_{f_{M-1}}^T \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}} (s_i(\mathbf{h}_{i-1}))^T
\end{aligned} \tag{A-2}$$

where

$$J_{f_k} = \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} = W_i \cdot S_i'(\mathbf{h}_{k-1}), k = i+1, \dots, M-1 \tag{A-3}$$

$S_i'(\mathbf{h}_{k-1})$ is a diagonal matrix with each diagonal element being corresponding element-wise derivatives and J_{f_k} is the Jocobian matirx of $f_k(\mathbf{h}_{k-1})$ with respect to $\mathbf{h}_{i-1}$.

Target propagation update is more complicated. For layer $M-1$, as mentioned in the end of section 2.2, target $\hat{\mathbf{h}}_{M-1}$ of $\mathbf{h}_{M-1}$ is assigned by

$$\hat{\mathbf{h}}_{M-1} = \mathbf{h}_{M-1} - \eta_0 \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}} \tag{A-4}$$

If all $\mathbf{h}_k$s are allocated in smooth areas and η_0 is sufficiently small, then the target $\hat{\mathbf{h}}_i$ in layer i is achieved by perfect inverse $g_k = f_k^{-1}, k = i+1, \dots, M-1$ that

$$\begin{aligned}
\hat{\mathbf{h}}_i &= g_{i+1}(\dots g_{M-1}(\hat{\mathbf{h}}_{M-1}) \dots) \\
&= g_{i+1}(\dots g_{M-1}(\mathbf{h}_{M-1}) \dots) - \eta_0 J_{g_{i+1}} \dots J_{g_{M-1}} \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}} + o(\eta_0^2) \\
&\simeq \mathbf{h}_i - \eta_0 J_{f_{i+1}}^{-1} \dots J_{f_{M-1}}^{-1} \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}}
\end{aligned} \tag{A-5}$$

Now for target propagation update δW_i^{tp} we have

$$\begin{aligned}
\delta W_i^{bp} &= -\eta_{tp} \frac{\partial ||\mathbf{h}_i(\mathbf{h}_{i-1}; W_i) - \hat{\mathbf{h}}_i||_2^2}{\partial W_i} \\
&= -\eta_{tp} (\mathbf{h}_i - (\mathbf{h}_i - \eta_0 J_{f_{i+1}}^{-1} \dots J_{f_{M-1}}^{-1} \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}})) (s_i(\mathbf{h}_{i-1}))^T \\
&= -\tilde{\eta}_{tp} J_{f_{i+1}}^{-1} \dots J_{f_{M-1}}^{-1} \frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}} (s_i(\mathbf{h}_{i-1}))^T
\end{aligned} \tag{A-6}$$

here $\tilde{\eta}_{tp} = \eta_{tp} \cdot \eta_0$ and we write $\frac{\partial loss(\mathbf{h}_M, y)}{\partial \mathbf{h}_{M-1}}$ as $\boldsymbol{l}$ and $s_i(\mathbf{h}_{i-1})$ as $\boldsymbol{v}$ for short. Since δW_i^{bp} and δW_i^{tp} are in matrix form, the inner production of their vector forms $vec(\delta W_i^{bp})$ and $vec(\delta W_i^{tp})$ is

$$\begin{aligned}
\langle vec(\delta W_i^{bp}), vec(\delta W_i^{tp}) \rangle &= tr((-\eta_{bp} J_{f_{i+1}}^T \dots J_{f_{M-1}}^T \boldsymbol{l}\boldsymbol{v}^T)^T (-\tilde{\eta}_{tp} J_{f_{i+1}}^{-1} \dots J_{f_{M-1}}^{-1} \boldsymbol{l}\boldsymbol{v}^T)) \\
&= \eta_{bp} \tilde{\eta}_{tp} tr(\boldsymbol{v}\boldsymbol{l}^T J_{f_{M-1}} \dots J_{f_{i+1}} J_{f_{i+1}}^{-1} \dots J_{f_{M-1}}^{-1} \boldsymbol{l}\boldsymbol{v}^T) \\
&= \eta_{bp} \tilde{\eta}_{tp} tr(\boldsymbol{v}\boldsymbol{l}^T \boldsymbol{l}\boldsymbol{v}^T) \\
&= \eta_{bp} \tilde{\eta}_{tp} ||\boldsymbol{v}||_2^2 \cdot ||\boldsymbol{l}||_2^2
\end{aligned} \tag{A-7}$$

Also for $||vec(\delta W_i^{bp})||_2^2$ and $||vec(\delta W_i^{tp})||_2^2$ we have

$$\begin{aligned} ||vec(\delta W_i^{bp})||_2^2 &= tr((-\eta_{bp} J_{f_{i+1}}^T \dots J_{f_{M-1}}^T \boldsymbol{l}\boldsymbol{v}^T)^T(-\eta_{bp} J_{f_{i+1}}^T \dots J_{f_{M-1}}^T \boldsymbol{l}\boldsymbol{v}^T)) \\ &= \eta_{bp}^2 tr(\boldsymbol{v}((J_{f_{M-1}} \dots J_{f_{i+1}})^T \boldsymbol{l})^T((J_{f_{M-1}} \dots J_{f_{i+1}})^T \boldsymbol{l})\boldsymbol{v}^T) \\ &= \eta_{bp}^2 ||\boldsymbol{v}||_2^2 \cdot ||(J_{f_{M-1}} \dots J_{f_{i+1}})^T \boldsymbol{l}||_2^2 \\ &\leq \eta_{bp}^2 ||\boldsymbol{v}||_2^2 \cdot ||(J_{f_{M-1}} \dots J_{f_{i+1}})^T||_2^2 \cdot ||\boldsymbol{l}||_2^2 \end{aligned} \tag{A-8}$$

and similarly,

$$||vec(\delta W_i^{tp})||_2^2 \leq \tilde{\eta}_{tp}^2 ||\boldsymbol{v}||_2^2 \cdot ||(J_{f_{M-1}} \dots J_{f_{i+1}})^{-1}||_2^2 \cdot ||\boldsymbol{l}||_2^2 \tag{A-9}$$

here $||(J_{f_{M-1}} \dots J_{f_{i+1}})^T||_2$ and $||(J_{f_{M-1}} \dots J_{f_{i+1}})^{-1}||_2$ are Euclidian norms, i.e. the largest singular value of $(J_{f_{M-1}} \dots J_{f_{i+1}})^T$, λ_{max}, and the largest singular value of $(J_{f_{M-1}} \dots J_{f_{i+1}})^{-1}$, $\frac{1}{\lambda_{min}}$ (λ_{min} is the smallest singular value of $(J_{f_{M-1}} \dots J_{f_{i+1}})^T$, because f_k is invertable, so all the smallest singular values of Jacobians are larger than 0). Finally, the angle α_i between $vec(\delta W_i^{bp})$ and $vec(\delta W_i^{tp})$ satisfies:

$$\begin{aligned} cos(\alpha_i) &= \frac{\langle vec(\delta W_i^{bp}), vec(\delta W_i^{tp})\rangle}{||vec(\delta W_i^{bp})||_2 \cdot ||vec(\delta W_i^{tp})||_2} \\ &\geq \frac{\eta_{bp}\tilde{\eta}_{tp}||\boldsymbol{v}||_2^2 \cdot ||\boldsymbol{l}||_2^2}{\sqrt{\eta_{bp}^2||\boldsymbol{v}||_2^2 \cdot \lambda_{max}^2 \cdot ||\boldsymbol{l}||_2^2}\sqrt{\tilde{\eta}_{bp}^2||\boldsymbol{v}||_2^2 \cdot (\frac{1}{\lambda_{min}^2}) \cdot ||\boldsymbol{l}||_2^2}} \\ &= \frac{\lambda_{min}}{\lambda_{max}} \end{aligned} \tag{A-10}$$

we have $0 \leq \alpha_i \leq cos^{-1}(\frac{\lambda_{min}}{\lambda_{max}})$, where $\alpha_i \geq 0$ is trivial. □

B PROOF OF PROPOSITION 2

Proof. Let us first give detail explanation for condition 2, 3, 4. For condition 2, proper learning rates η_v and η_w satisfy

$$\sum_{t=1}^{\infty} \eta_v^{(t)}(\eta_w^{(t)}) = +\infty, \sum_{t=1}^{\infty} (\eta_v^{(t)})^2((\eta_w^{(t)})^2) < +\infty \tag{A-11}$$

Note that the the beginning learning rate $\eta_v^{(1)}(\eta_w^{(1)})$ can be assigned as $\frac{1}{n_0}$ to be sufficiently small to satisfy locally smooth condition if needed. Condition 3 basically says that the norm of first order terms like $\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i+\epsilon; V_i, W_i)$ and $\nabla_{W_i} L_i(W_i)$ which are special cases of Jacobians, and eigenvalues of second order terms like $\frac{\partial^2 L_i^{inv}(\hat{\mathbf{h}}_i+\epsilon;V_i,W_i)}{\partial V_i^2}$, $\frac{\partial^2 L_i^{inv}(\hat{\mathbf{h}}_i+\epsilon;V_i,W_i)}{\partial V_i \partial W_i}$ are bounded. Codition 4 is equavilent to the following

$$\forall \varepsilon > 0, \inf_{||V_i - V_i^*(W_i^{(t)})||_2 > \varepsilon} (V_i - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i, W_i^{(t)}) > 0 \tag{A-12}$$

above condition basically says that oppoite of the gradient $-\nabla_{V_i} \bar{L}_i^{inv}(V_i, W_i^{(t)})$ always at least partly points towards its minimum with optimal $V_i^*(W_i^{(t)})$.

Note that in the following proof, all V_is and W_is are in *vector form*. $V_i^{(t)}$ and $W_i^{(t)}$ follow their update rules like

$$V_i^{(t+1)} = V_i^{(t)} - \eta_v^{(t)} \nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)}) \tag{A-13}$$

$$W_i^{(t+1)} = W_i^{(t)} - \eta_w^{(t)} \delta W_i^{(t)} \tag{A-14}$$

$\delta W_i^{(t)}$ respects to $\nabla_{W_i} L_i(W_i)$ in difference target propagation. We define γ_t that

$$\gamma_t = ||V_i^{(t)} - V_i^*(W_i^{(t)})||_2^2 \tag{A-15}$$

The γ_t measures how far the current $V_i^{(t)}$ is from the optimum. During the learning process, the randomness is only introduced by every update's sample $(\mathbf{x}, \mathbf{y})$ and the ϵ used in updating V_i. We care about whether γ_t converges and we check the following conditional expcation

$$\begin{aligned}
& \mathbb{E}\{\gamma_{t+1} - \gamma_t \mid V_i^{(t)}, W_i^{(t)}\} \\
= \; & \mathbb{E}\{||V_i^{(t)} - \eta_v^{(t)} \nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)}) - V_i^*(W_i^{(t+1)})||_2^2 \\
& -||V_i^{(t)} - V_i^*(W_i^{(t)})||_2^2 \mid V_i^{(t)}, W_i^{(t)}\} \\
= \; & -2\eta_v^{(t)}(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)})\} \\
& +(\eta_v^{(t)})^2 \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{||\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)})||_2^2\} \\
& +2(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \mathbb{E}\{V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)}) \mid W_i^{(t)}\} \qquad \text{(A-16)} \\
& -2\eta_v^{(t)} \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{(\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)}))^T (V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)}) \mid W_i^{(t)}\} \\
& +\mathbb{E}\{||V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)})||_2^2 \mid W_i^{(t)}\} \\
= \; & -2\eta_v^{(t)}(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i^{(t)}, W_i^{(t)}) \qquad \text{(A-17)} \\
& +(\eta_v^{(t)})^2 \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{||\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)})||_2^2\} \qquad \text{(A-18)} \\
& -2\eta_v^{(t)} \mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{(\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)}))^T (V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)}) \mid W_i^{(t)}\} \qquad \text{(A-19)} \\
& +\mathbb{E}\{||V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)})||_2^2 \mid W_i^{(t)}\} \qquad \text{(A-20)}
\end{aligned}$$

We can see that term (A-16) is cancelled by condition 5. In term (A-18), because the norm of $\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i + \epsilon; V_i, W_i)$ is bounded by some non-negative constant α_v, we have

$$\mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i, \epsilon} \{||\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i + \epsilon; V_i, W_i^{(t)})||_2^2\} \leq \alpha_v^2 \tag{A-21}$$

Let us check term (A-19) and term (A-20) where both of them have the term

$$\Delta^* = V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)}) \tag{A-22}$$

Because $V_i^*(W_i^{(t)})$ is the optimum minimizing $\bar{L}_i^{inv}(V_i, W_i^{(t)})$, we have

$$\begin{aligned} \mathbf{0} &= \nabla_{V_i} \bar{L}_i^{inv}(V_i^*(W_i^{(t)}), W_i^{(t)}) \\ &= \nabla_{V_i} \bar{L}_i^{inv}(V_i^*(W_i^{(t+1)}), W_i^{(t+1)}) \\ &= \nabla_{V_i} \bar{L}_i^{inv}(V_i^*(W_i^{(t)}) - \Delta^*, W_i^{(t)} - \eta_w^{(t)} \delta W_i^{(t)}) \end{aligned} \tag{A-23}$$

If the learning rate for V_i and W_i is small enough with local smoothness satisfied, we can transform Eq.A-23 like

$$\mathbf{0} = -\frac{\partial^2 \bar{L}_i^{inv}(V_i, W_i)}{\partial V_i^2} \Delta^* - \eta_w^{(t)} \frac{\partial^2 \bar{L}_i^{inv}(V_i, W_i)}{\partial V_i \partial W_i} \delta W_i^{(t)} + \mathbf{o}(||\Delta^*||_2^2) + \mathbf{o}(||\eta_w^{(t)} \delta W_i^{(t)}||_2^2) \tag{A-24}$$

Here $||\mathbf{o}(||\Delta^*||_2^2)||_2 \leq \varepsilon_\Delta ||\Delta^*||_2$ and $||\mathbf{o}(||\eta_w^{(t)} \delta W_i^{(t)}||_2^2)||_2 \leq \varepsilon_w ||\eta_w^{(t)} \delta W_i^{(t)}||_2$ for local smoothness. With the fact that all the second order terms are bounded, based on Eq.A-24 we have

$$||\Delta^*||_2 \leq \alpha_\Delta \eta_w^{(t)} ||\delta W_i^{(t)}||_2 \tag{A-25}$$

Here α_Δ is some non-negative constant. Further more, because the first order term of W_i like $\nabla_{W_i} L_i(W_i)$ is also bounded, we have

$$||\delta W_i^{(t)}||_2 \leq \alpha_w \tag{A-26}$$

Here α_w is some non-negative constant. Now for term (A-19) we have

$$|\mathop{\mathbb{E}}_{\hat{\mathbf{h}}_i^{(t)}, \epsilon} \{(\nabla_{V_i} L_i^{inv}(\hat{\mathbf{h}}_i^{(t)} + \epsilon; V_i, W_i^{(t)}))^T (V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)}) \mid W_i^{(t)}\}| \leq \alpha_\Delta \alpha_v \alpha_w \eta_w^{(t)} \eta_v^{(t)} \tag{A-27}$$

and the absolute value of the entire term (A-19) is then bounded by

$$2\alpha_\Delta \alpha_v \alpha_w \eta_w^{(t)} \eta_v^{(t)} \tag{A-28}$$

Based on *Cauthy-Schwarz inequality*, $\eta_w^{(t)} \eta_v^{(t)}$ satisfies

$$(\sum_{t=1}^{N} \eta_w^{(t)} \eta_v^{(t)})^2 \leq (\sum_{t=1}^{N} (\eta_w^{(t)})^2)(\sum_{t=1}^{N} (\eta_v^{(t)})^2) \tag{A-29}$$

From the learning rates condition, we know that

$$\lim_{N \to \infty} \sum_{t=1}^{N} (\eta_w^{(t)})^2 ((\eta_v^{(t)})^2) < +\infty \tag{A-30}$$

Because $\eta_w^{(t)}$ and $\eta_v^{(t)}$ are positive, then we can easily have

$$\lim_{N \to \infty} \sum_{t=1}^{N} \eta_w^{(t)} \eta_v^{(t)} < +\infty \tag{A-31}$$

For term (A-20) we have

$$\mathbb{E}\{||V_i^*(W_i^{(t)}) - V_i^*(W_i^{(t+1)})||_2^2 \mid W_i^{(t)}\} \leq \alpha_\Delta^2 \alpha_w^2 (\eta_w^{(t)})^2 \tag{A-32}$$

Finally, $\mathbb{E}\{\gamma_{t+1} - \gamma_t \mid V_i^{(t)}, W_i^{(t)}\}$ satisfies

$$\begin{aligned} & \mathbb{E}\{\gamma_{t+1} - \gamma_t \mid V_i^{(t)}, W_i^{(t)}\} \\ \leq \; & -2\eta_v^{(t)}(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i^{(t)}, W_i^{(t)}) \\ & +(\eta_v^{(t)})^2(\alpha_v)^2 + 2\alpha_\Delta \alpha_v \alpha_w \eta_w^{(t)} \eta_v^{(t)} + \alpha_\Delta^2 \alpha_w^2 (\eta_w^{(t)})^2 \\ \leq \; & (\eta_v^{(t)})^2(\alpha_v)^2 + 2\alpha_\Delta \alpha_v \alpha_w \eta_w^{(t)} \eta_v^{(t)} + \alpha_\Delta^2 \alpha_w^2 (\eta_w^{(t)})^2 \end{aligned} \tag{A-33}$$

From Eq.A-11 and Eq.A-31, we know that the right side of Eq.A-33 is the summand of a convergent infinite sum. Since process $\{\gamma_t\}$ always larger than 0, and Eq.A-33 gives the upper bound of positive expected variation of γ_t, from *Quasi-martingale convergence theorem* in section 4.4 of (Bottou, 1998), we have that γ_t converges amlost surely. The almost surely convergence of γ_t and Eq.A-33 imply that

$$\sum_{t=1}^{\infty} \eta_v^{(t)}(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i^{(t)}, W_i^{(t)}) \leq +\infty, \quad a.s. \tag{A-34}$$

Here *a.s.* means *almost surely*. With the learning rate $\eta_v^{(t)}$ satisfies

$$\sum_{t=1}^{\infty} \eta_v^{(t)} = +\infty \tag{A-35}$$

and $(V_i^{(t)} - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i^{(t)}, W_i^{(t)})$ is always positive because of condition 4, we have

$$\lim_{t \to \infty} (V_i^{(t)} - V_i^*(W_i^{(t)}))^T \nabla_{V_i} \bar{L}_i^{inv}(V_i^{(t)}, W_i^{(t)}) = 0, \quad a.s. \tag{A-36}$$

Assume that γ_t converges to some positive constant rather than 0. It implies that when t is large enough, $\gamma_t = ||V_i^{(t)} - V_i^*(W_i^{(t)})||_2^2 > \varepsilon > 0$. This is incompatible with condition 3 and Eq.A-36. Therefore γ_t converges to 0 almost surely and we have

$$\lim_{t \to \infty} V_i^{(t)} - V_i^*(W_i^{(t)}) = 0, \quad a.s. \tag{A-37}$$

□