Université de Montréal

# Difference Target Propagation

**par Dong-Hyun Lee**

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Juillet, 2018

# Résumé

L'algorithme de rétropropagation a été le cheval de bataille du succès récent de l'apprentissage profond, mais elle s'appuie sur des effets infinitésimaux (dérivées partielles) afin d'effectuer l'attribution de crédit. Cela pourrait devenir un problème sérieux si l'on considère des fonctions plus profondes et plus non linéaires, avec à l'extrême la non-linéarité où la relation entre les paramètres et le coût est réellement discrète.

Inspirée par la présumée invraisemblance biologique de la rétropropagation, cette thèse propose une nouvelle approche, *Target Propagation*. L'idée principale est de calculer des cibles plutôt que des gradients à chaque couche, en faisant en sorte que chaque paire de couches successive forme un auto-encodeur.

Nous montrons qu'une correction linéaire, appelée *Difference Target Propagation*, est très efficace, conduisant à des résultats comparables à la rétropropagation pour les réseaux profonds avec des unités discrètes et continues et des auto- encodeurs et atteignant l'état de l'art pour les réseaux stochastiques.

**Mots clés:** réseaux de neurones, apprentissage automatique, apprentissage supervisé, optimisation, règle d'apprentissage, règle d'apprentissage biologiquement plausible, rétropropagation, Target Propagation, Difference Target Propagation

# Summary

*Backpropagation* 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 *Backpropagation*, this thesis proposes a novel approach, *Target Propagation*. The main idea is to compute targets rather than gradients, at each layer in which feedforward and feedback networks form Auto-Encoders.

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 *Backpropagation* for deep networks with discrete and continuous units, Denoising Auto-Encoders and achieving state of the art for stochastic networks.

In Chapters 1, we introduce several classical learning rules in Deep Neural Networks, including *Backpropagation* and more biological plausible learning rules.

In Chapters 2 and 3, we introduce a novel approach, *Target Propagation*, more biological plausible learning rule than *Backpropagation*. In addition, we show that *Target Propagation* is comparable to *Backpropagation* in Deep Neural Networks.

**Keywords:** neural networks, machine learning, deep learning, representation learning, optimization, biological plausibility, learning rule, backpropagation, target propagation

# Table des matières

# Table des figures

# Liste des tableaux

# List of Abbreviations

| | |
|---|---|
| AE | Auto-Encoder |
| ANN | Artificial Neural Network |
| CHL | Contrastive Hebbian Learning |
| GeneRec | Generalized Recirculation |
| CE | Cross Entropy |
| DAE | Denoising Auto-Encoder |
| GD | Gradient Descent |
| I.I.D | Independent and Identically Distributed |
| MLP | Multi-Layer Perceptron |
| MSE | Mean Squired Error |
| NLL | Negative Log-Likelihood |
| RBM | Restricted Boltzmann Machine |
| ReLU | Rectified Linear Unit |
| SGD | Stochastic Gradient Descent |

# Acknowledgments

First of all, I would like to thank my mother, Min-Ok Lee, who always support me in studying Machine Learning. And I would like to express my deepest gratitude to my supervisor Prof. Yoshua Bengio, who spent a lot of time to supervise me in this ambitious research field. It is only possible with his deep insight and his plenty of experience to try this kind of ground-breaking research.

I would like to extend my thanks to my co-authors Saizheng Zhang for his support and rigorous knowledge of mathematics, and Asja Fischer for her encouragement, support, guidance and research attitute.

I must also acknowledge great help and support from my friends and colleagues, in alphabetical order: Anirudh Goyal, Caglar Gulchehre, Chiheb Trabelsi, David Scott Krueger, Devansh Arpit, Frédéric Bastien, Jose Sotelo, Jörg Bornschein, Junyoung Chung, Kyle Kastner, Li Yao, Mohammad Pezeshki, Myriam Côté, Pascal Lamblin, Samira Shabanian, Sina Honari, Soroush Mehri, Taesup Kim, Tomas Mesnard, Ying Zhang and Zhouhan Lin.

Finally, the work reported in this thesis would not have been possible without the financial support from: Samsung, NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

# 1 Learning Rules in Deep Neural Networks

## 1.1 Artificial Neural Networks

A lot of attempts have been made to build human-level intelligent machines for half a century. One of the attempts is to mimic a biological neuron and its networks in the human brain. But the biological details of a real neuron is fairly complicated (Hodgkin and Huxley, 1952). A simplified computational model called *artificial neuron* (also called *Perceptron* by (Rosenblatt, 1958)) was proposed by (McCulloch and Pitts, 1943). An Artificial Neural Network (ANN) is composed of artificial neurons and their weighted connections. Though ANNs are not so similar to biological Neural Networks in the brain, they have been leading recent successes of Deep Learning and Artificial Intelligence.

### 1.1.1 An artificial neuron

An artificial neuron is a simple computational unit to map from several inputs to an output. Consider an input vector $\mathbf{x} = (x_1, x_2, ..., x_d)^T$ (pre-synaptic neuron activities), a weight vector $\mathbf{w} = (w_1, w_2, ..., w_d)^T$ and a scalar bias term $b$. An artificial neuron calculates a single output $h$ (post-synaptic neuron activity) as follows,

$$h = s\left(\sum_{i=1}^{d} w_i x_i + b\right) = s\left(\mathbf{w}^T\mathbf{x} + b\right), \tag{1.1}$$

where $s(.)$ is an *activation function* (also called *transfer function*). We can use several types of activation functions, **Identity** $s(x) = x$, **Hyperbolic tangent**

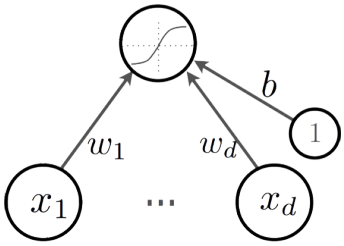


**Figure 1.1** – A graphical illustration of an artificial neuron. The input is the vector $\mathbf{x} = (x_1, x_2, ..., x_d)^T$ to which a weight vector $\mathbf{w} = (w_1, w_2, ..., w_d)^T$ and a bias term $b$ is assigned. (Figure adapted from Hugo Larochelle's slides)

$s(x) = tanh(x)$, **Sigmoid** $s(x) = 1/(1 + e^{-x})$, **Step function**

$$s(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{1.2}$$

and **Rectified Linear Unit (ReLU)** $s(x) = x^+ = max(0, x)$, which is frequently used nowadays. **Softmax** is a rather special activation function which can deal with a categorical value.

$$s(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{1.3}$$

where $s(\mathbf{x})_i$ means the probability to belong to $i$th class.

### 1.1.2 Deep Neural Networks

A layer of Neural Networks is composed of artificial neurons. Consider an input vector $\mathbf{x} \in \mathbb{R}^{d_0}$ (where $d_0$ is the dimension of the input vector), an output vector $\mathbf{h} \in \mathbb{R}^{d_1}$ of neurons (where $d_1$ is the number of neurons), a weight matrix $\mathbf{W} = (\mathbf{w}_1^T, \mathbf{w}_2^T, ..., \mathbf{w}_{d_1}^T)^T \in \mathbb{R}^{d_0 \times d_1}$ and a bias vector $\mathbf{b} = (b_1, b_2, ..., b_{d_1})^T$. We can express the layer as follows,

$$\mathbf{h} = s(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{1.4}$$

where $s(.)$ is an element-wise activation function. If we have pair data $\{(\mathbf{x}^{(m)}, \mathbf{h}^{(m)}).\}_{m=1}^N$, we can train the network layer to approximate the true function $\mathbf{h}(\mathbf{x})$ that gene-

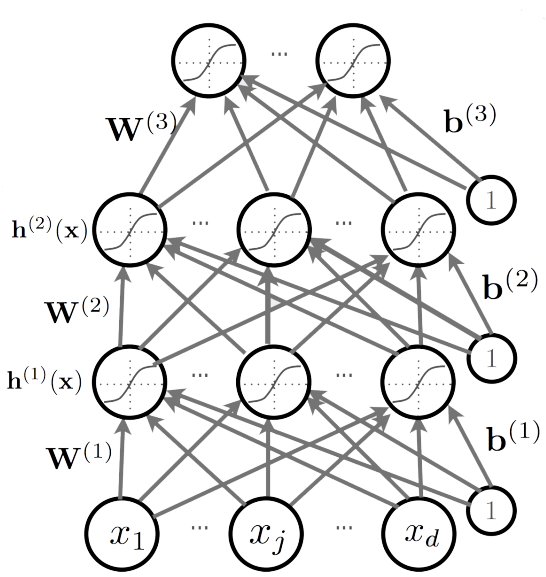


**Figure 1.2** – A Three layer neural network. The matrix $\mathbf{W}^k$ connects the $(k-1)^{th}$ layer to the $k^{th}$ layer and therefore $\mathbf{W}^k \in \mathbb{R}^{d_k \times d_{k-1}}$ and $\mathbf{b}^k \in \mathbb{R}^{d_k}$. After each linear transformation (weight multiplication and bias addition), an activation function is applied. (Figure adapted from Hugo Larochelle's slides)

rates the data properly. But it usually hasn't enough capacity to approximate real data.

Artificial Neural Networks are usually organized in a multi-layer fashion (also called *Multi-Layer Perceptron*). As shown in figure 1.2, each layer takes the output vector of the lower network as an input. From the input in the bottom layer to the output in the top layer, more complicated computation can be done as follows.

$$\mathbf{h}^{(k)} = s^{(k)}(\mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}), \quad k = 1, ..., L \tag{1.5}$$

where $k$ is a layer index, $L$ is the number of layers, $\mathbf{h}^{(k)}$ is a vector of hidden units when $k = 1, ..., L-1$. We don't have values in the data to match the hidden units. $\mathbf{h}^{(0)} = \mathbf{x}$ is the input of the network and $\mathbf{h}^{(L)} = \mathbf{y}$ is the output of the network. If networks have more than 2 hidden layers, we call it *Deep Neural Networks*.

In fact, this simplest kind of neural networks is called *Feed-Forward Networks* because connections between the units do not form a cycle, so the information are carried in only forward direction without any feedback. There are also neural networks that have cyclic or feedback connections : *Recurrent Neural Networks*, *Hopfield Networks*, *Boltzmann Machines* and so on. And sometimes neural networks

may have special layers like *Convolution Layer*, *Pooling Layer* and so on.

In spite of its mathematical simplicity, ANNs are known to be able to learn very complicated functions in real world applications like *Image Classification*, *Speech Recognition* and so on.

## 1.2 Learning Rules for Single-Layer Networks

Single-Layer Networks are rather easier to train than Multi-Layer Networks because we don't need to decide how to update weights in hidden layers. We will review several learning rules only applicable to Single-Layer Networks.

### 1.2.1 Hebbian Learning Rule

The neuro-psychologist Donald Hebb postulated a rule for synaptic plasticity, that is, how neurons learn in the brain in (Hebb, 1949) :

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells such that A's efficiency as one of the cells firing B, is increased."

In short, *"Neurons wire together if they fire together"* (Lowel and Singer, 1992). Basically, it can perform association tasks. If a stimulus and its response are given to two neurons, they would develop strong connections, and subsequent activation by the stimulus could generate the associated response.

Consider activations of two neurons $x_i$, $x_j$ and a weight of their connection $w_{ij}$. The Hebb's rule can be expressed by

$$\Delta w_{ij} = \eta x_i x_j, \tag{1.6}$$

where $\eta$ is the learning rate. Hebb's original suggestion concerned only increases in synaptic strength, but it has been generalized to include decreases due to stability of the learning process. So the general forms of Hebbian Rule state that weights change in proportion to the correlation or covariance of the activities of connected neurons (Dayan and Abbott, 2001).

One of important features in Hebbian Learning is *Local Learning*, which means that the update rules should depend only on variables that are available locally, such as the pre- and post-synaptic neuronal activities (Baldi and Sadowski, 2016). It is a very recommended one for physical neural systems, for examples, the real brain or hardware implementation of ANNs.

Moreover, Hebbian Learning can lead to *Ocular Dominance* and *Orientation Selectivity* in the visual cortex of the brain (Dayan and Abbott, 2001).

### 1.2.2 The Delta Rule

One drawback of Hebbian Learning is that synaptic modification does not depend on the actual performance of the network. More direct method to improve the performance is gradient-based optimization of the error function (Dayan and Abbott, 2001). Consider a neuron $a_i = \sum_j w_{ij} x_j$ and $h_i = s(a_i)$ and an squared error function $E = \sum_i 0.5(y_i - h_i)^2$ where $y_i$ is a target.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta (y_i - h_i) s'(a_i) x_j \tag{1.7}$$

If we use a linear neuron with squared error or a sigmoid neuron with cross entropy error, the delta rule is simplified as follows,

$$\Delta w_{ij} = \eta (y_i - h_i) x_j \tag{1.8}$$

It is similar to *Perceptron Learning Rule* (Dayan and Abbott, 2001), but the derivation is different in that the Perceptron uses non-differentiable step activation function, so cannot adopt gradient-based learning.

### 1.2.3 Competitive Learning Rule

*Competitive Learning* is a kind of Unsupervised Learning, in which the output units compete for the right to respond to a given subset of inputs (Rumelhart, 1986). Only single or a group of neurons is active at a time, that is usually maximally activated one or has minimal (Euclidean) distance to a prototype. If only one neuron is permitted to be activated, it is also called *Winner-Take-All* Mechanism. To determine which neuron is the winner, we need lateral connections between

neurons within a layer in the brain.

As long as the winner neuron (index $i$) is taken, the simplest synaptic update rule is following.

$$\Delta w_{ij} = \eta_i(x_j - w_{ij}) \tag{1.9}$$

Basically, this kind of rules can perform Clustering. Models and algorithms based on competitive learning include *Vector Quantization* (Burton et al., 1983) and *Self-Organizing Maps* (Kohonen, 1982) .

## 1.3 Learning Rules for Multi-Layer Networks

Multi-Layer Networks are difficult to train because it is highly non-linear and we don't know how to update the weights in a layer without knowing the information of other layers.

### 1.3.1 Credit Assignment Problems and Biological Plausibility of Learning Rule

The credit assignment problem concerns determining how the success of a system's overall performance is due to the various contributions of the system's components (Minsky, 1963).

In Multi-Layer Neural Networks, we have to determine how to update the weights in each layer to minimize the global cost. It is easy for the top layer related with the cost directly, but not possible for the lower layers without the information of the upper layers.

So we need to know how a weight in a layer relates with the global cost using the information of upper layers. But it is not easy with Biological Plausible *Local Learning Rule*s in that it only depends on pre- and post-synaptic activities.

### 1.3.2 Backpropagation Algorithm

Backpropagation Algorithm (Rumelhart et al., 1986) is purely gradient-based optimization for weights of a layer in Multi-Layer Networks. It uses the chain rule

to compute the gradients iteratively from the top layer to that layer. In this way, it can solves *Credit Assignment Problems* via chain rule of gradient calculation. Therefore, all parts of the networks must be differentiable.

Backpropagation Algorithm consists of two steps. At first, *The Forward Step*, we compute neuron activations from the bottom layer to the top layer and the prediction error given the inputs. Then, *The Backward Step*, we calculate the gradients of the loss function w.r.t the weights from the top layer to the bottom layer through error propagation.

Consider the loss function $L$ and hidden activations $\mathbf{h}^{k-1}$, $\mathbf{a}^k = \mathbf{W}^k\mathbf{h}^{k-1} + \mathbf{b}^k$, $\mathbf{h}^k = s(\mathbf{a}^k)$ in $k$th and $(k-1)$th layer. From the chain rule of differentiation,

$$\frac{\partial L}{\partial \mathbf{h}^{k-1}} = \frac{\partial L}{\partial \mathbf{h}^k}\frac{\partial \mathbf{h}^k}{\partial \mathbf{h}^{k-1}} = \frac{\partial L}{\partial \mathbf{h}^k} \odot s'(\mathbf{a}^k)(\mathbf{W}^k)^T \tag{1.10}$$

Consider $\frac{\partial L}{\partial \mathbf{h}^k}$ as propagated error $\mathbf{e}^k$ in $k$th layer. we re-formulate 1.10 as follows.

$$\mathbf{e}^{k-1} = \mathbf{e}^k \odot s'(\mathbf{a}^k)(\mathbf{W}^k)^T \tag{1.11}$$

And then, the update rule for weights is following.

$$\Delta\mathbf{W}^k \propto -\frac{\partial L}{\partial \mathbf{h}^k}\frac{\partial \mathbf{h}^k}{\partial \mathbf{W}^k} = -\mathbf{e}^k \odot s'(\mathbf{a}^k)(\mathbf{h}^{k-1})^T \tag{1.12}$$

Backpropagation has been questioned not to be biologically plausible (Crick, 1989). Though there are feedback system in the brain, It is not certain they carry the error signal. Moreover, it is not plausible that the brain can calculate the exact analytic gradients including the derivative of activation function. The most serious problem is the *Weight Transport Problem* (Lillicrap et al., 2016). When the error is propagated across a layer, we need the transpose of the weight matrix in *the Forward Step*. But it is not plausible that the separate feedback networks can get the exact information of weights in feedforward networks at the same time.

### 1.3.3 Contrastive Hebbian Learning

*Contrastive Hebbian Learning* (CHL) is supervised learning for multi-layer networks. It was originally formulated for the Boltzmann Machine and was extended later to deterministic networks (Movellan, 1991; Xie and Seung, 2003).

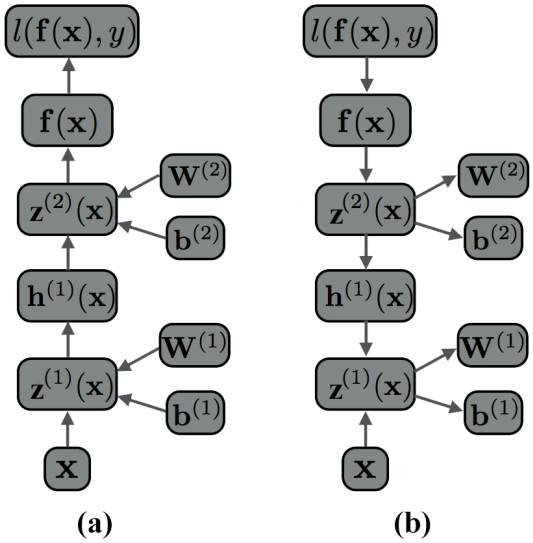


**Figure 1.3** – (a) The forward path and (b) the backward path in backpropagation algorithm on a two-layer neural network. Note that in the backward path, in order to be able to go through each module, it must be differentiable. (Figure adapted from Hugo Larochelle's slides.)

It is quite different from Backpropagation in that it is a Hebbian-type algorithm based on the correlation of pre- and postsynaptic activities, not using error propagation. In addition, CHL is implemented in networks with feedback.

CHL updates the synaptic weights based on steady states of bidirectional networks in two different phases : positive phase with the output clamped to the desired values, negative phase with the output not clamped. Consider the neuron activities in the $k$th and $(k-1)$th layer, $\mathbf{h}_+^k$, $\mathbf{h}_+^{k-1}$ in positive phase, $\mathbf{h}_-^k$, $\mathbf{h}_-^{k-1}$ in negative phase. The weights update rule in CHL is

$$\Delta \mathbf{W}^k \propto \mathbf{h}_+^k (\mathbf{h}_+^{k-1})^T - \mathbf{h}_-^k (\mathbf{h}_-^{k-1})^T \tag{1.13}$$

It also requires symmetric weights in feedback networks, so also suffers from *Weight Transport Problem*.

### 1.3.4 Generalized Recirculation

*Recirculation* provided more biological plausible learning rule than Backpropagation (Hinton and McClelland, 1988), but they applied it to train Auto-Encoder, not general networks, so O'Reilly (1996) proposed *Generalized Recirculation* (Ge-

neRec). It is similar to CHL in that it is also based on steady states of neurons in the positive and the negative phase. But the update rule is slightly different.

$$\Delta \mathbf{W}^k \propto (\mathbf{h}_+^k - \mathbf{h}_-^k)(\mathbf{h}_-^{k-1})^T \tag{1.14}$$

Two important ideas allow us to implement learning in a more biological plausible manner. First, bidirectional connections convey the information of neuron activities itself, not infinitesimal error to be propagated. Second, by using the difference of steady states of neural activities, we implicitly compute the derivative of activation function without the exact analytic formula (O'Reilly and Munakata, 2000; O'Reilly, 1996; Hinton and McClelland, 1988). In addition, it is the example of *Local Learning Rule*s in that this rule only depends on locally available activity signals.

# 2 Prologue to the Article

**Difference Target Propagation**. Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio, in Machine Learning and Knowledge Discovery in Databases, pages 498-515, Springer International Publishing (ECML/PKDD), 2015

*Personal Contribution.* The main idea of the working formula was proposed by Dong-Hyun Lee (me), based on biological implausibility of *Backpropagation* and the idea of target propagation proposed by Yoshua Bengio. Saizheng Zhang contributed to rigorous mathematical analyses (Theorems and Proofs) of *Target Propagation*. Asja Fischer contributed to mathematics, consistency of concepts and re-writing some parts of my first writing. I did all experiments in this work, made the main formula, wrote at the first item and reasoning behind the framework, supervised by Yoshua Bengio.

# 3 Difference Target Propagation

## 3.1 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., 2015) 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 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 (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) 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 (Bengio, 2014)) can be employed for supervised training of deep neural networks (section 3.2.1 and 3.2.2). We continue by introducing a linear correction for the imperfectness of the auto-encoders (3.2.3) 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 3.2.4). In section 3.3 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 (Tieleman and Hinton, 2012) - a very popular setting to train deep networks nowadays- and achieves state of the art for training stochastic neural nets on MNIST.

## 3.2 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 (Bengio, 2014) and is described in the following.

### 3.2.1 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

$$\mathbf{h}_i = f_i(\mathbf{h}_{i-1}) = s_i(W_i \mathbf{h}_{i-1}), \;\; i = 1, \ldots, M \tag{3.1}$$

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

$$L(\mathbf{h}_M(\mathbf{x};\theta_W^{0,M}),\mathbf{y}) = L(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x};\theta_W^{0,i});\theta_W^{i,M}),\mathbf{y}) \tag{3.2}$$

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 problems, 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

$$L(\mathbf{h}_M(\hat{\mathbf{h}}_i;\theta_W^{i,M}),\mathbf{y}) < L(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x};\theta_W^{0,i});\theta_W^{i,M}),\mathbf{y}) \ . \tag{3.3}$$

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

$$L_i(\hat{\mathbf{h}}_i,\mathbf{h}_i) = ||\hat{\mathbf{h}}_i - \mathbf{h}_i(\mathbf{x};\theta_W^{0,i})||_2^2 \ . \tag{3.4}$$

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

$$W_i^{(t+1)} = W_i^{(t)} - \eta_{f_i}\frac{\partial L_i(\hat{\mathbf{h}}_i,\mathbf{h}_i)}{\partial W_i} = W_i^{(t)} - \eta_{f_i}\frac{\partial L_i(\hat{\mathbf{h}}_i,\mathbf{h}_i)}{\partial \mathbf{h}_i}\frac{\partial \mathbf{h}_i(\mathbf{x};\theta_W^{0,i})}{\partial W_i} \ , \tag{3.5}$$

where $\eta_{f_i}$ 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 (3.3) holds) or that at least leads to a decrease of the local loss $L_{i+1}$ of the next layer, that is

$$L_i(\hat{\mathbf{h}}_{i+1}, f_i(\hat{\mathbf{h}}_i)) < L_i(\hat{\mathbf{h}}_{i+1}, f_i(\mathbf{h}_i)) \ . \tag{3.6}$$

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

### 3.2.2 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

$$\hat{\mathbf{h}}_M = \mathbf{h}_M - \hat{\eta}\frac{\partial L(\mathbf{h}_M, \mathbf{y})}{\partial \mathbf{h}_M} \ , \tag{3.7}$$

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 (Bengio, 2014), 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

$$f_i(g_i(\mathbf{h}_i)) \approx \mathbf{h}_i \quad \text{or} \quad g_i(f_i(\mathbf{h}_{i-1})) \approx \mathbf{h}_{i-1} \ . \tag{3.8}$$

Then, choosing

$$\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i) \tag{3.9}$$

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 3.1. Indeed, if the feedback mappings were the perfect inverses of the feed-forward mappings ($g_i = f_i^{-1}$), one gets

$$L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1})) = L_i(\hat{\mathbf{h}}_i, f_i(g_i(\hat{\mathbf{h}}_i))) = L_i(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_i) = 0 \ . \tag{3.10}$$

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:

$$g_i(\mathbf{h}_i) = \bar{s}_i(\mathbf{V}_i\mathbf{h}_i), \quad i = 1, ..., M \tag{3.11}$$

where $\bar{s}_i$ is a non-linearity associated with the decoder and $\mathbf{V}_i$ the matrix of feedback 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

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1})) - \mathbf{h}_{i-1}||_2^2 \ . \tag{3.12}$$

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

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ , \tag{3.13}$$

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

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* $\mathbf{h}_i = f_i(\mathbf{h}_{i-1}) =$

1. See the proof in the Appendix.

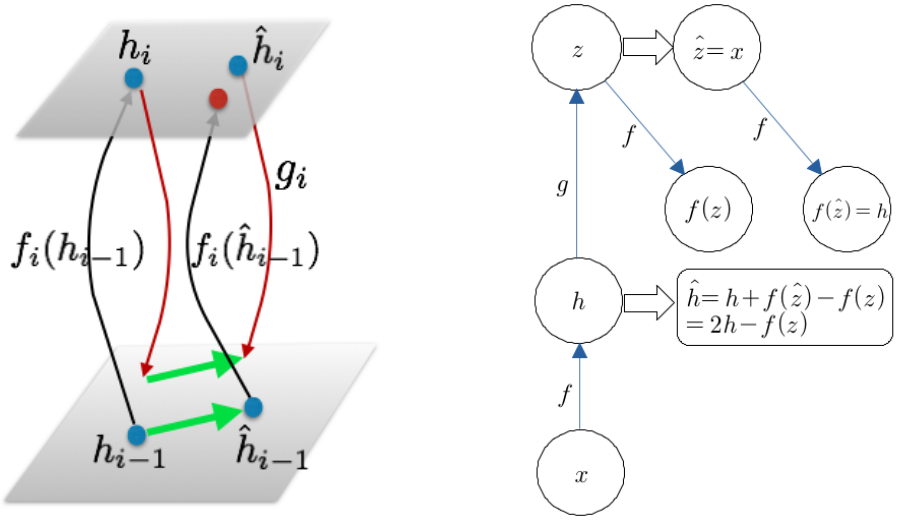


**Figure 3.1** – (left) How to compute a target in the lower layer via difference target propagation. $f_i(\hat{\mathbf{h}}_{i-1})$ should be closer to $\hat{\mathbf{h}}_i$ than $f_i(\mathbf{h}_{i-1})$. (right) Diagram of the back-propagation-free auto-encoder via difference target propagation.

$W_i s_i(\mathbf{h}_{i-1})$ [2] *where* $s_i$ *can be any differentiable monotonically increasing element-wise function. Let* $\delta W_i^{tp}$ *and* $\delta W_i^{bp}$ *be the target propagation update and the back-propagation update in* $i$*-th layer, respectively. If* $\hat{\eta}$ *in Equation* (3.7) *is sufficiently small, then the angle* $\alpha$ *between* $\delta W_i^{tp}$ *and* $\delta W_i^{bp}$ *is bounded by*

$$0 < \frac{1 + \Delta_1(\hat{\eta})}{\frac{\lambda_{max}}{\lambda_{min}} + \Delta_2(\hat{\eta})} \leq cos(\alpha) \leq 1 \tag{3.14}$$

*Here* $\lambda_{max}$ *and* $\lambda_{min}$ *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.*

### 3.2.3 Difference target propagation

From our experience, the imperfection of the inverse function leads to severe optimization problems when assigning targets based on equation (3.9). This brought us to propose the following linearly corrected formula for target propagation which we refer to as "*difference target propagation*"

$$\hat{\mathbf{h}}_{i-1} = \mathbf{h}_{i-1} + g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i) \ . \tag{3.15}$$

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**

Note, that if $g_i$ is the inverse of $f_i$, difference target propagation becomes equivalent to vanilla target propagation as defined in equation (3.9). 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

$$\mathbf{h}_i = \hat{\mathbf{h}}_i \;\Rightarrow\; \mathbf{h}_{i-1} = \hat{\mathbf{h}}_{i-1} \tag{3.16}$$

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

$$\mathbf{h}_{i-1} = f_i^{-1}(\mathbf{h}_i) = g_i(\hat{\mathbf{h}}_i) = \hat{\mathbf{h}}_{i-1} \ . \tag{3.17}$$

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

$$\hat{\mathbf{h}}_{i-1} - \mathbf{h}_{i-1} = g_i(\hat{\mathbf{h}}_i) - g_i(\mathbf{h}_i) \ . \tag{3.18}$$

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 (3.6) 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 (3.15), 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*

$$||\hat{\mathbf{h}}_i - f_i(\hat{\mathbf{h}}_{i-1})||_2^2 < ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2 \ . \tag{3.19}$$

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.

### 3.2.4 Training an auto-encoder with difference target propagation

Auto-encoders are interesting for learning representations and serve as building blocks for deep neural networks (Erhan et al., 2010). 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 (Vincent et al., 2010) and generative stochastic networks (Ben-

3. See the proof in Appendix.

gio 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. This is, the hidden units are given by the encoder as

$$\mathbf{h} = f(\mathbf{x}) = sig(W\mathbf{x} + \mathbf{b}) \ , \tag{3.20}$$

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

$$\mathbf{z} = g(\mathbf{h}) = sig(W^T(\mathbf{h} + \epsilon) + \mathbf{c}), \ \ \epsilon \sim N(0, \sigma) \ , \tag{3.21}$$

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

$$L = ||\mathbf{z} - \mathbf{x}||_2^2 + ||f(\mathbf{x} + \epsilon) - \mathbf{h}||_2^2, \ \ \epsilon \sim N(0, \sigma) \ , \tag{3.22}$$

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 3.1 (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,

$$\hat{\mathbf{h}} = \mathbf{h} + f(\hat{\mathbf{z}}) - f(\mathbf{z}) = 2\mathbf{h} - f(\mathbf{z}) \ , \tag{3.23}$$

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.

## 3.3 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 $\sigma_0$ 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).

### 3.3.1 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 (Tieleman and Hinton, 2012) where hyper-parameters for the best validation error were found using random search (Bergstra and Bengio, 2012). 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 lear-

ning rates were different for each layer, which is why we decided to use an adaptive method like RMSprop.

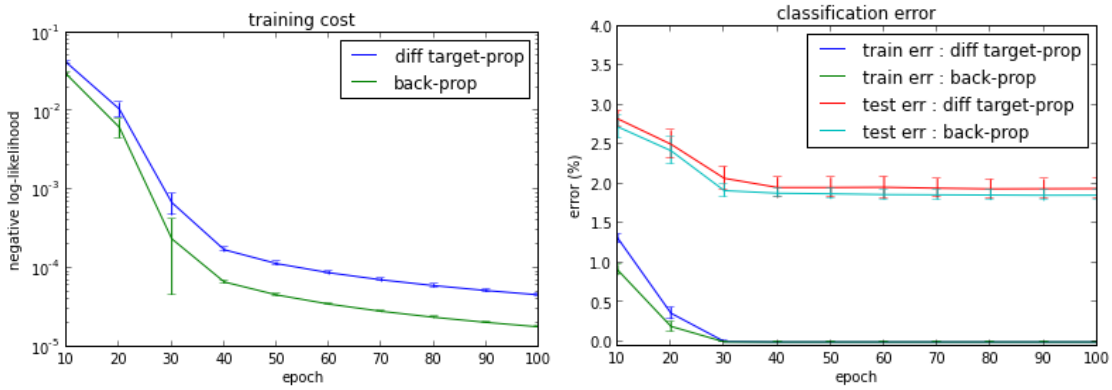


**Figure 3.2** – Mean training cost (left) and train/test classification error (right) with target propagation and back-propagation using continuous deep networks (tanh) on MNIST. Error bars indicate the standard deviation.

The results are shown in Figure 3.2. 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 $4.584 \times 10^{-5}$ with target propagation and $1.797 \times 10^{-5}$ 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, while it seemed to be less appropriate for 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 Krizhevsky and Hinton (2009), 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, Konda et al. (2014) reported 64.1% but when using PCA without whitening as preprocessing and zero-biased auto-encoders for unsupervised pre-training.

### 3.3.2 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 3.3 and the activations of the hidden layers are given by

$$\mathbf{h}_1 = f_1(\mathbf{x}) = \tanh(W_1\mathbf{x}) \quad \text{and} \quad \mathbf{h}_2 = f_2(\mathbf{h}_1) = \tanh(W_2 sign(\mathbf{h}_1)) \tag{3.24}$$

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

$$p(\mathbf{y}|\mathbf{x}) = f_3(\mathbf{h}_2) = softmax(W_3\mathbf{h}_2) \ . \tag{3.25}$$

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

$$g_2(\mathbf{h}_2) = \tanh(V_2 sign(\mathbf{h}_2)) \ , \tag{3.26}$$

$$L_2^{inv} = ||g_2(f_2(\mathbf{h}_1 + \epsilon)) - (\mathbf{h}_1 + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ . \tag{3.27}$$

If 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* (Bengio et al., 2013), 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 3.3. 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.

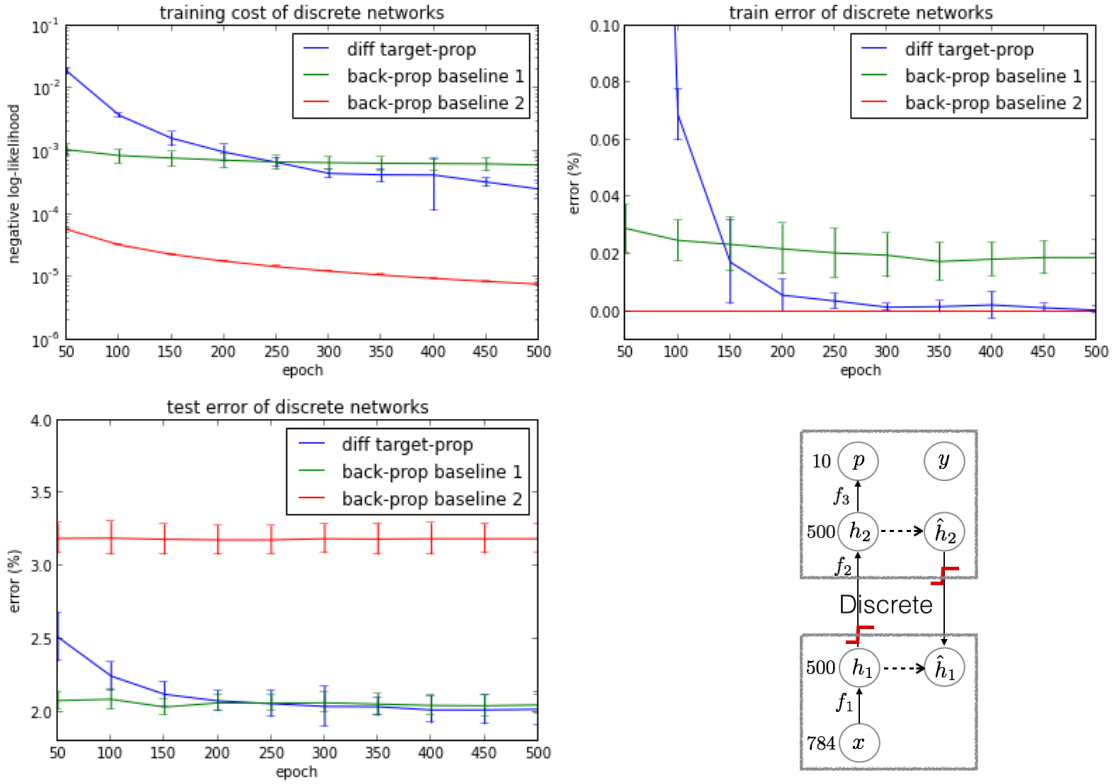


**Figure 3.3** – Mean training cost (top left), mean training error (top right) and mean test error (bottom left) while training discrete networks with difference target propagation and the two baseline versions of back-propagation. Error bars indicate standard deviations over the 10 runs. Diagram of 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).

### 3.3.3 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 (Bengio, 2013; Tang and Salakhutdinov, 2013; Bengio

et al., 2013) 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 Raiko et al. (2014), 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:

$$\mathbf{h}_i^p = P(\mathbf{H}_i = 1|\mathbf{h}_{i-1}) = \sigma(W_i\mathbf{h}_{i-1}), \quad \mathbf{h}_i \sim P(\mathbf{H}_i|\mathbf{h}_{i-1}) \ , \tag{3.28}$$

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 (Bengio et al., 2013) in which the derivative through the discrete sampling step is ignored (this method showed the best performance in Raiko et al. (2014).) That is

$$\delta\mathbf{h}_{i-1}^p = \delta\mathbf{h}_i^p \frac{\partial\mathbf{h}_i^p}{\partial\mathbf{h}_{i-1}^p} \approx \sigma'(W_i\mathbf{h}_{i-1})W_i^T\delta\mathbf{h}_i^p \ . \tag{3.29}$$

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

$$\hat{\mathbf{h}}_2^p = \mathbf{h}_2^p - \eta\frac{\partial L}{\partial\mathbf{h}_2} \qquad \text{and} \qquad \hat{\mathbf{h}}_1^p = \mathbf{h}_1^p + g_2(\hat{\mathbf{h}}_2^p) - g_2(\mathbf{h}_2^p) \tag{3.30}$$

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

$$L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma) \ , \tag{3.31}$$

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 Raiko et al. (2014). Results are given in Table 3.1. We obtained a test error of 1.71% using the baseline method and 1.54% using target propagation. This suggests that target propagation is highly promising for training networks of binary stochastic units.

| Method | Test Error(%) |
|---|---|
| Difference Target-Propagation, M=1 | 1.54% |
| Straight-through gradient estimator (Bengio et al., 2013) + backprop, M=1 as reported in Raiko et al. (2014) | 1.71% |
| as reported in Tang and Salakhutdinov (2013), M=20 | 3.99% |
| as reported in Raiko et al. (2014), M=20 | 1.63% |

**Table 3.1** – 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 (Raiko et al., 2014). 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.

### 3.3.4 Auto-encoder

We trained a denoising auto-encoder with 1000 hidden units with difference target propagation as described in Section 3.2.4 on MNIST. As shown in Figure 3.4 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.

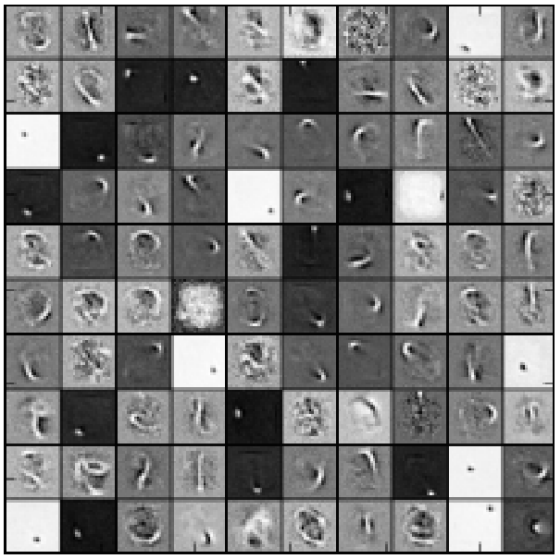

**Figure 3.4** – Filters learned by the back-propagation-free auto-encoder. Each filter corresponds to the hidden weights of one of 100 randomly chosen hidden units. We obtain stroke filters, similar to those usually obtained by regularized auto-encoders.

# 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.

Future works should be aimed at the following directions. First, we will investigate whether target propagation can be realized in the biological brain or not. Second, we will try to implement target propagation in hardware. And finally, we will apply this method to recurrent network or reinforcement learning in which backpropagation cannot do well.

# A Theorems and Proofs

## A.1 Proof of Theorem 1

*Démonstration.* 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 $\delta W_i^{tp}$ 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 $\delta W_i^{bp}$ and $\delta W_i^{tp}$ 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

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

and similarly

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

where $||J^T||_2$ and $||J^{-1}||_2$ are matrix Euclidean norms, i.e. the largest singular value of $(J_{f_M} \dots J_{f_{i+1}})^T$, $\lambda_{max}$, and the largest singular value of $(J_{f_M} \dots J_{f_{i+1}})^{-1}$, $\frac{1}{\lambda_{min}}$ ($\lambda_{min}$ 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 $\alpha$ between $vec(\delta W_i^{bp})$ and $vec(\delta W_i^{tp})$ satisfies:

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

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

## A.2 Proof of Theorem 2

*Démonstration.* 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 $\lambda$ 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 $||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2$.

□

# Bibliographie


Baldi, P. and P. Sadowski (2016). A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks 83*, 51–74.

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. *arXiv preprint arXiv:1407.7906*.

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

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

Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *J. Machine Learning Res. 13*, 281–305.

Burton, D., J. Shore, and J. Buck (1983). A generalization of isolated word recognition using vector quantization. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'83.*, Volume 8, pp. 1021–1024. IEEE.

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

Crick, F. (1989). The recent excitement about neural networks. *Nature 337*(6203), 129–132.

Dayan, P. and L. F. Abbott (2001). *Theoretical neuroscience*, Volume 806. Cambridge, MA: MIT Press.

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

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Psychology Press.

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

Hinton, G. E. and J. L. McClelland (1988). Learning representations by recirculation. In *Neural information processing systems*, pp. 358–366.

Hodgkin, A. L. and A. F. Huxley (1952). Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *The Journal of physiology 116*(4), 449–472.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics 43*(1), 59–69.

Konda, K., R. Memisevic, and D. Krueger (2014). Zero-bias autoencoders and the benefits of co-adapting features. *arXiv preprint arXiv:1402.3337*.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

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

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

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

Lillicrap, T. P., D. Cownden, D. B. Tweed, and C. J. Akerman (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications 7*, 13276.

Lowel, S. and W. Singer (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science 255*(5041), 209–212.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*(4), 115–133.

Minsky, M. L. (1963). Steps toward artificial intelligence. *Computers and thought*, 406–450.

Movellan, J. R. (1991). Contrastive hebbian learning in the continuous hopfield model. In *Connectionist Models*, pp. 10–17. Elsevier.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation 8*(5), 895–938.

O'Reilly, R. C. and Y. Munakata (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT press.

Raiko, T., M. Berglund, G. Alain, and L. Dinh (2014). Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65*(6), 386.

Rumelhart, D. E. (1986). Parallel distributed processing. *Bradford Books*, 1–2.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *nature 323*(6088), 533.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. (2015). Going deeper with convolutions. Cvpr.

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

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

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

Xie, X. and H. S. Seung (2003). Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation 15*(2), 441–454.