COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS
AND INFORMATICS

ANALYSIS OF THE GENERALIZED RECIRCULATION-BASED LEARNING ALGORITHMS IN BIDIRECTIONAL NEURAL NETWORKS

MASTER THESIS

Bratislava 2014 **Bc. Peter CSIBA**

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Peter Csiba
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Analysis of the generalized recirculation-based learning algorithms in bidirectional neural networks

Aim:
1. Study the literature on general recirculation-based learning algorithms in neural networks and write the state-of-the art of the topic.
2. Implement the GeneRec and BAL algorithms and test their properties on selected data sets, using computer simulations and visualization techniques.
3. Consider suitable modifications of the algorithms aimed at improving the network performance.

Literature:
O'Reilly, R.C. (1996). Biologically plausible error-driven learning using local activation differences: The Generalized Recirculation algorithm. Neural Computation, 8, 895-938.
Farkaš I., Rebrová K. (2013). Bidirectional activation-based neural network learning algorithm. In Proceedings of the International Conference on Artificial Neural Networks (ICANN), Springer. 154-161.

Annotation: The advantage of the GeneRec algorithm resides in its biological plausibility, as opposed to the well-known error backpropagation, because it only allows propagation of neuron activations. Recently, a completely bidirectional model BAL has been proposed, inspired by GeneRec, but has not been analyzed in detail (Farkaš and Rebrová, 2013).

Keywords: supervised learning, neural network, heteroassociative mapping

Supervisor:	doc. Ing. Igor Farkaš, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. PhDr. Ján Rybár, PhD.
Assigned:	10.12.2012
Approved:	10.12.2012

prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

Student

Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Csiba
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Analysis of the generalized recirculation-based learning algorithms in bidirectional neural networks
Analýza algoritmov učenia na báze zovšeobecnenej recirkulácie v obojsmerných neurónových sieťach

Cieľ: 1. Preštudujte literatúru o algoritmoch učenia založených na zovšeobecnenej recirkulácii v neurónových sieťach a urobte prehľad aktuálneho stave výskumu.
2. Implementujte učiace algoritmy GeneRec BAL a otestujte ich vlastnosti na vybratých dátových množinách, pomocou počítačových simulácií a vizualizačných techník.
3. Preskúmajte modifikácie algoritmov s cieľom vylepšiť správanie siete.

Literatúra: O'Reilly, R.C. (1996). Biologically plausible error-driven learning using local activation differences: The Generalized Recirculation algorithm. Neural Computation, 8, 895-938.
Farkaš I., Rebrová K. (2013). Bidirectional activation-based neural network learning algorithm. In Proceedings of the International Conference on Artificial Neural Networks (ICANN), Springer. 154-161.

Anotácia: Algoritmus GeneRec je biologicky prijateľný, na rozdiel od známeho algoritmu spätného šírenia chyby, pretože dovoľuje šírenie len aktivácií. Na jeho základe bol nedávno navrhnutý podobný model BAL úplne obojsmernej siete (Farkaš a Rebrová, 2013), ktorý nebol ešte dostatočne preskúmaný.

Kľúčové slová: učenie s učiteľom, neurónová sieť, heteroasociatívne zobrazenie

Vedúci: doc. Ing. Igor Farkaš, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 10.12.2012

Dátum schválenia: 10.12.2012 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgements

The completion of this thesis could not have been possible without my supervisor Igor Farkaš who encouraged and helped me out through the whole process. Especially, I want to thank him for investing his precious time in our weekly sessions which navigated me what to do next and made me work regularly.

I also want to thank my parents Viktor and Jarmila for their lifelong investment in providing me with the best education they could, while often sacrifing their own comfort.

Abstract

In our work, we used computational simulations to analyse supervised artificial neural networks based on the Generalized recirculation algorithm (GeneRec) by O'Reilly (1996) and the Bidirectional Activation-based Learning algorithm (BAL) by Farkaš and Rebrová (2013). The main idea of both algorithms is to update weights based on the difference between forward and backward propagation of neuron activations rather than based on error backpropagation (BP) between layers, which is considered biologically implausible. However, both algorithms struggle to learn low dimensional mappings which could be easily learned by BP. The aim of this work is to fill this gap.

Several modifications of BAL are proposed and after systematic analysis a Two learning rates (TLR) version is introduced. TLR uses different learning rates for different weight matrices. The simulations prove increase in success rate and show smooth relation between success and learning rates. For the networks with highest success rate the two learning rates can be in ratio 10^6. Further the idea of TLR is applied to GeneRec. Finally, additional experiments for momentum, weight initialization, hidden activations and dynamic learning rate are analysed.

We believe that using the idea of TLR could lead to performance increase in other artificial neural network models as well, and even multi-layered networks. Intuitively, an increase in success rate could be achieved by generalizing the idea of TLR to additional parameters, such as momentum or weight initialization. Further experiments are outlined.

Keywords: supervised learning, artificial neural network, heteroassociative mapping, dynamic learning rate, activation based learning.

Abstrakt

Táto práca pomocou výpočtových simulácií analyzuje umelé neurónové siete (UNS), ktoré sú založené na Generalized recirculation algorithm (GeneRec) (O'Reilly, 1996) a Bidirectional Activation-based Learning algorithm (BAL) (Farkaš and Rebrová, 2013). Od štandardných sietí, akými sú napríklad siete spätne šíriace chybu (BP), sa líšia tým, že zmena váh je založená na rozdiely dopredných a spätných aktivácií. Takéto siete sa považujú za prirodzené pre ich obojsmernosť a preto, lebo šíria iba aktiváciu a nie chybu. Je známe, že tieto siete majú problémy s naučením sa aj jednoduchých úloh, ktoré sa BP vie naučiť. Cieľom práce je preto zvýšenie úspešnosti BALu.

Analyzujeme viacero modifikácií BALu. Na základe pozorovaní navrhujeme model Two learning rates (TLR), ktorý využíva rozdielne rýchlosti učenia pre rôzne matice. Pomocou simulácií potvrdíme, že TLR značne zvyšuje úspešnosť BALu vo viacerých úlohách. Navyše, pozorujeme jasné závislosti medzi rýchlosťami učenia a úspešnosťou siete. Zaujímavosťou je, že pre najlepšie siete môže byť podiel medzi dvoma rýchlosťami učenia až 10^6. Myšlienku TLR aplikujeme aj na GeneRec. Navyše, skúšame viacero štandardných modifikácií UNS, ako sú napríklad moment, dávkové učenie, dynamická rýchlosť učenia alebo inicializácia váh.

Veríme, že aplikácia myšlienky TLR má potenciál zvýšiť úspešnosť aj iných modelov UNS. Myšlienka sa dá zovšeobecniť aj na iné parametre, ako sú napríklad moment alebo inicializácia váh.

Kľúčové slová: učenie s učiteľom, neurónová sieť, heteroasociatívne zobrazenie, dynamická rýchlosť učenia, učenie na základe aktivácií

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The field of artificial neural networks (ANN) gets lots of attention nowadays. ANNs are interesting for both psychologists and computer scientists. For psychologists they provide a simulation environment of the human brain which could be used to prove their hypotheses. For computer scientists ANNs are a general model which could be used to solve a broad range of practical problems.

We choose to analyse the Bidirectional Activation-based Learning algorithm (BAL) recently introduced by Farkaš and Rebrová (2013). The main reasons why we have chosen BAL over standard models are its simplicity, *bidirectionality* and *biological plausibility*. The term biological plausibility is stated by six principles by Hinton and McClelland (1988). One of these principles is the *bidirectional activation propagation* is achieved in BAL by using backward representations for learning the forward representations and vice versa.

Although BAL performs well on high dimensional tasks, it has problems to learn low dimensional tasks with 100% reliability, while BP is able to learn them. Therefore, our primary goal was to find reasons for this performance gap and use them to derive a modification of BAL which will perform comparably to BP. In our work, we were able to follow this process and find reasons which were then used for deriving the Two learning rate model (TLR). TLR uses different learning rates for different weight matrices and shows some counter intuitive behaviour. It performs comparable to BP if it is initialized correctly.

Our work starts with an overview of ANNs in Chapter 1 which consists of necessary preliminaries and related models. In Chapter 2 we describe our simulations, experiments and datasets used through our work. We finish our work with Chapter 3 which contains simulation results aimed at TLR, which proved to be the most successful modification of BAL. Finally we conclude our results and outline future experiments in Section 3.3.3.

1 Overview

1.1 Preliminaries

In this section, we will describe the basics of artificial neural networks. We will also introduce the notation used in this work. Note that the definitions and notations vary through the literature and therefore we use the one which the author is familiar with. For the reader who is comfortable with this topic we recommend to continue to Section 1.2.

1.1.1 Perceptron

The theory of artificial neural networks started with the model of *Perceptron* introduced by McCulloch and Pitts (1943). It is a simple model which transforms a vector of inputs s to an output value y. The notation used is depicted in Figure 1: x is the *input vector* where always $x_0 = 1$, w_{0k} is the *weight* vector, Σ is the *summing* junction, η_k is the *net input*, ϕ is the *activation function*, θ_k is the *treshold*, y_k is the *output* and b_k is the *bias*.

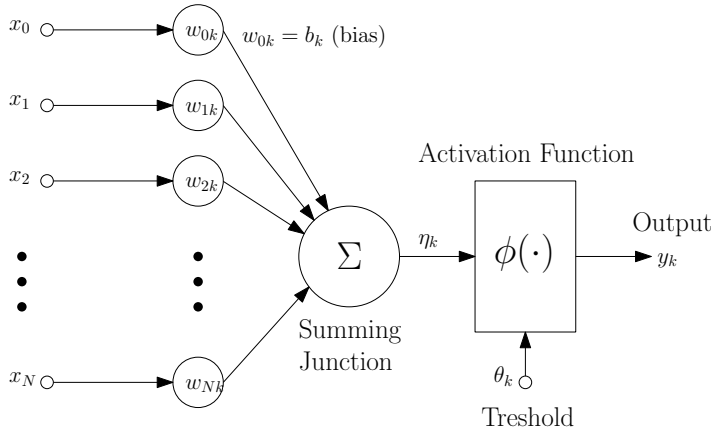

Figure 1: Perceptron transforming *inputs* $[x_0, x_1, \ldots, x_N]$ to *output* y_k.

We can write the whole transformation of the input vectors to the output activation y_k:

$$y_k = \begin{cases} 1 & \text{if } \phi(\sum_{i=0}^{N} x_i w_{ik}) > \theta_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Equation (1) describes a simple *binary treshold perceptron*. One could observe that the binary perceptron divides the vector space $\mathbb{R}^N$ by a $(n-1)$–dimensional hyperplane,

where the bias is the absolute term (Rosenblatt, 1958). This leads to the fact that for one perceptron is impossible to classify non–*linearly separable* vectors. Now we see the importance of bias which is the absolute term in the equation of the hyperplane.

Learning. The goal of a perceptron is to *learn* the mapping given by the set $T = \{(X_j, t_j)\}$ of pairs, where X_j is the input vector $(x_{j0}, x_{j1}, \ldots, x_{jN})$ and t_j is the corresponding target. It could be formalized as minimizing the error function:

$$E = \sum_{k=1}^{N} \frac{1}{2}(t_k - y_k)^2. \tag{2}$$

A straightforward method for the network to minimize the error function (2) is by simply updating weights according to the partial derivates of the error function:

$$\frac{\partial E}{\partial w_{ik}} = (t_k - y_k)\phi'(\eta_k)x_i = (t_k - y_k)y_k(1 - y_k)x_i, \tag{3}$$

which gives us the *update rule* going opposite the gradient:

$$\Delta w_{ik} = \lambda(t_k - y_k)y_k(1 - y_k)x_i, \tag{4}$$

where λ is the *learning rate*. Using the learning rule (4) we can design *training* algorithm shown in Algorithm 1.1. It applies the *weight update rule* (4) in loop for each sample in T. One main loop is called *epoch*.

Algorithm 1.1 Perceptron training pseudocode.

```
for epoch = 1 to Epoch_max do
    for all (X_j, t_j) in T do
        y_j ← [φ(Σ_{i=0}^{N} x_i w_ik) > θ_k]
        for i = 0 to N do
            w_ij ← w_ij + λ(t_k − y_k)y_k(1 − y_k)x_i
        end for
    end for
end for
```

Continuous perceptron. Till now, the Perceptron could give only discrete outputs. We put additional constraints for the activation function $\phi : \mathbb{R} \mapsto (0, 1)$ that

ϕ is differentiable, monotonously increasing and satisfying two asymptotic conditions $t(-\infty) = 0$ and $t(\infty) = 1$. Usually, the activation function is realized by the logistic function $1/(1 + \exp -\eta)$. To allow real numbered results from the range $(0, 1)$, we drop the treshold function and simply output $\phi(\eta_k)$.

1.1.2 Multi-layer feedworward networks

We will define *multi-layer feedforward networks* as in Haykin (1994). First, we define a *layered* neural network where neurons are organised to form layers. In the simplest version we have an *input layer* of source nodes and an *output layer* which is formed by continuous perceptrons (1.1.1). In other words this is a *feedforward* or *acyclic* type of network as the *activation*, i.e. outputs of the neurons are computed from the input to the output layer and never *backwards*.

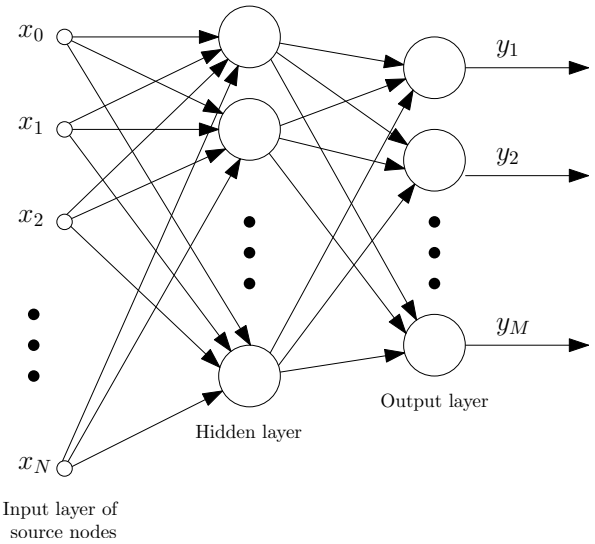

Figure 2: Fully connected feedforward *multi-layer* network with one *hidden* layer.

Multi-layer neural network has one or more *hidden layers* in addition to the input and ouput layer as shown in Figure 2. The source nodes supply the activation pattern, i.e. input vector, which is applied to next layer of neurons, i.e. the hidden layer. The output signal of the hidden layer is then used as the input for the output layer. As shown by Cybenko (1989) the three layer network is an universal approximator of continuous functions on compact subsets of $\mathbb{R}^n$.

There exists several methods for training multi-layer networks. First, we will describe the most common backpropagation in Section (1.1.5) and then methods related

to our work such as CHL (1.2.1), GeneRec (1.2.3) and BAL (1.3).

1.1.3 Recurrent networks

In *recurrent* neural networks also cycles of connections are allowed. In other words, the output of a particular unit could affect its input. Therefore, the activations in general could not be computed only by one forward pass. This introduces real valued dynamic systems for computing the activations. We can observe that it holds that $\partial\eta/\partial t = 0$ for the activations of neurons in the fixed point state. There are several approaches solving these dynamic systems and deriving the learning rule (Pineda, 1987; Pearlmutter, 1989; Williams and Zipser, 1989; Elman, 1990; Haykin, 1994).

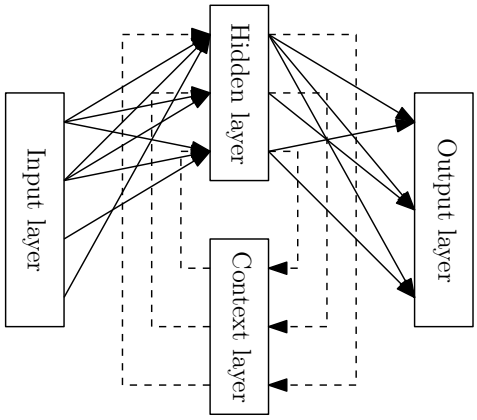

Figure 3: Simple recurrent network proposed by Elman (1990). Taken from Haykin (1994).

An *iterative method* is used by Movellan (1990) to compute activations. In the first step the input neurons have activations equal to the input vector and the other neurons have activations equal to zero. In the next steps activations from the last step are used to compute activation in the current step as shown in equation (5):

$$\eta_i(t+1) = \phi\left(\sum_j w_{ji}\eta_i(t)\right) \tag{5}$$

This rule is iterated while the activations are not settled. For particullar symmetric networks it can be proved that activations will converge (O'Reilly, 1996). For more general networks a dynamic system based on rule (5) could be introduced. The the fixed point solution is the settled activation. We experimented with the iterative method for a two way version of GeneRec in Section (2.3.2).

1.1.4 Hopfield networks

Hopfield (1984) introduced a network with arbitrary connections defined only by one weight matrix W. Some of the units are chosen as the *input units* which have stable activations for a given input pattern. We can treat a Hopfield network as a recurrent neural network. A Hopfield network comes with a continuous energy function for which usually function (6) is chosen:

$$E = -\frac{1}{2}\sum_i \sum_j a_i w_{ij} a_j, \tag{6}$$

where a_i is the activation of the i-th unit. The aim of the network is to settle the activations so that E settles in a global minima. Activation for the i-th unit is computed based on the following differential equation (Hopfield, 1984):

$$\frac{\partial a_i}{\partial t} = \alpha(-a_i + f_i(\eta_i)), \tag{7}$$

where $a^T = [a_1, \ldots, a_n]$ is the activation vector, f_i is bounded, monotically increasing, differentiable activation function. Hopfield (1984) proved for equation (7) that if the weights are symmetric, i.e. $w_{ij} = w_{ji}$, the activations will settle in the minimal error state defined in equation (7). This learning rule is typically used in *interactive activation networks* studied by Grossberg (1978) and McClelland and Rumelhart (1981).

1.1.5 Backpropagation

Backpropagation is a multi-layer feedforward network (1.1.2) which differs from our definition of multi-layer networks (1.1.2) only with its learning rule. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain (Rumelhart et al., 1986). A criticism of backpropagation is that it is neurally implausible (and hard to implement in hardware) because it requires all the connections to be used backward and it requires the units to use different input–output functions for the forward and backward passes (Hinton and McClelland, 1988).

Layer	Net Input	Activation
Input (s)	–	s_i = stimulus input
Hidden (h)	$\eta_j = \sum_i w_{ij} s_i$	$h_j = \sigma(\eta_j)$
Output (o)	$\eta_k = \sum_j w_{jk} h_j$	$o_k = \sigma(\eta_k)$

Table 1: Activation values in backpropagation.

Using the Perceptron error function (2) we can compute $\partial E/\partial y_j$ from output to input layer. For the hidden–to–output wieght it will look like:

$$\frac{\partial E}{\partial w_{ij}} = -\sum_k (t_k - y_k) w_{jk} \sigma'(\eta_j) s_i, \tag{8}$$

where t_k is the target value, o_k is the output value, σ is the nonlinear function, η_j is the net input and s_i is the stimulus input (O'Reilly, 1996).

1.2 Related models

In this section, we briefly mention models directly related to our work. Mainly it is the Bidirectional Activation-based Learning algorithm (1.3) by Farkaš and Rebrová (2013) and the Generalized recirculation (1.2.3) by O'Reilly (1996). The other two models CHL (1.2.1) and BAL-Recirc (2.3.2) are inspiration for the two former ones. Understanding the latter helps understanding the former. We compare these models to our specialized versions in Table 6.

1.2.1 Contrastive Hebbian learning

The main idea of *Contrastive Hebbian Learning* developed by Movellan (1990) is to have two activation phases in an aribtrary Hopfield network (Hopfield, 1984) as described in Section 1.1.4. In the first phase, called *minus phase* and denoted "−", only the input vector is *clamped*, i.e. activations of the clamped units as equal to the clamped values. In the second phase, called *plus phase* and denoted "+", both the input and target are clambed to the underlying network. The learning is based on the difference of these two activations. For an idea how it works see Table 2. Note that CHL makes no assumptions about the structure of the underlying network and therefore, it has no layers in general.

As mentioned previously CHL is based on Hopfield networks. Therefore, it has an energy function J which is based on the Helmholtz free energy function F (Hinton, 1989):

$$F = -\frac{1}{2}\sum_i \sum_j a_i w_{ij} a_j + \sum_i \int_{f(0)}^{a_i} f_i^{-1}(a)da \tag{9}$$

where $-\frac{1}{2}\sum_i \sum_j a_i w_{ij} a_j$ is the Hopfield energy function (6). Then the *contrastive* error function J is defined as:

$$J = \hat{F^+} - \hat{F^-} \tag{10}$$

where $\hat{F^+}$ and $\hat{F^-}$ respectively are the values of the energy functions at equilibrium states for the plus and the minus phases.

Based on the contrastive energy function (10) a learning rule is derived by Movellan (1990):

$$\Delta w_{ij} = \hat{a_i}^+ \hat{a_j}^+ - \hat{a_i}^- \hat{a_j}^- \tag{11}$$

where $\hat{a_i}$ and $\hat{a_j}$ denote the equilibrium state activations of the i–th and j–th unit. It could be shown that the learning rule (11) decreases the energy function (10) (Movellan, 1990). Moreover it could be shown that the CHL learning rule is equivalent to backpropagation learning rule in terms of computability while it is biologically more plausible as it uses only activation for computing the error gradient (O'Reilly, 1996; Xie and Seung, 2003).

1.2.2 Recirculation algorithm

The *Recirculation algorithm* designed by Hinton and McClelland (1988) is an unsupervised neural letwork for learning encoder tasks. Motivation for such a model comes from interesting hidden representations of backpropagation (1.1.5) which could be used as an encoder. It has only two layers denoted *visible layer* and *hidden layer* as shown in Figure 4. The aim of the network is to remember on the hidden layer the patterns presented to the visible layer. This could be used for compression if the hidden layer has fewer units than the visible layer. It also could be used as a content-addressable memory, when if novel patterns are presented to the network then it could show the blend of the most similar stored patterns.

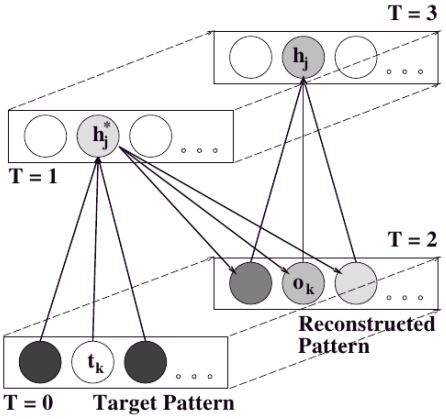

Figure 4: The recirculation algorithm by Hinton and McClelland (1988). Taken from (O'Reilly, 1996).

As depicted in Figure 4 the activation is propagated in four steps $T \in \{0, 1, 2, 3\}$. At the first phase, denoted by $T = 0$ only the input vector t is clamped on the visible layer, at $T = 1$ a forward pass h^* is computed from visible to hidden, at $T = 2$ a *reconstructed* pattern o_k as a function of hidden state h^* is computed and finally at $T = 3$ a hidden state h is computed from o_k.

For the reconstruction to work *symmetric* weights are used. The learning rule is common for both visible and hidden layers and it's based only on the difference of activations:

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= -(\eta_j^* - \eta_j)\phi'(\eta_j)t_i, \\ &\approx -(h_j^* - h_j)t_i.\end{aligned}$$

And similary we get $h_i(t_j - o_j)$ for the hidde to visible weights. The approximation step is possible because $\phi'(\eta_j)$ has *usually* the same sign as $(\eta_j^* - \eta_j)$ (Hinton and McClelland, 1988; O'Reilly, 1996). The approximation more precise if the difference of activations is smaller and therefore, the rule (12) could be used to make o similar to target pattern t.

$$o_k = \alpha t_k + (1 - \alpha)f(\eta_k). \tag{12}$$

1.2.3 Generalized recirculation

Introduction. The *Generalized recirculation algorithm*, or *GeneRec*, was introduced by O'Reilly (1996). It is a supervised learning algorithm which in comparison with

backpropagation (1.1.5) is argued to be a more biologically plausible model as error is computed locally as a difference between activations (O'Reilly, 1998, 2001; da Silva and Rosa, 2011; Schneider and Rosa, 2009). It extends the recirculation algorithm (1.2.2) by having a hidden layer of units and uses "+" and "−" phases as CHL (1.2.1) for weight update. This allows GeneRec to learn arbitrary mappings and not only conntent-based memories as the recirculation algorithm. For the error computation a backward weight matrix from output layer to hidden layer is used and the learning rule is derived from the CHL learning rule (11). It could be proven that GeneRec, as Backpropagation, could learn arbitrary input–output mappings (O'Reilly, 1996).

Activation. The main difference between CHL and GeneRec is that GeneRec has layers and it is based more on recurrent neural networks than on the Hopfield networks. Therefore, as shown in Table 2, we can compute the activations sequentially. We can see the inspiration from the recirculation algorithm (1.2.2) and a correspondence between T in recirculation and phases in GeneRec. In particular $s^- \approx T = 0$, $h^- \approx T = 1$, $o^- \approx T = 2$ and h^+ corresponds to $T = 3$. The activation flow is depicted in Figure 5.

Layer	Phase	Net Input	Activation
Input (s)	−	-	s_i = stimulus input
Hidden (h)	−	$\eta_j^- = \sum_i w_{ij}^{IH} s_i + \sum_k w_{kj}^{OH} o_k^-$	$h_j^- = \sigma(\eta_j^-)$
	+	$\eta_j^+ = \sum_i w_{ij}^{IH} s_i + \sum_k w_{kj}^{OH} o_k^+$	$h_j^+ = \sigma(\eta_j^+)$
Output (o)	−	$\eta_k^- = \sum_j w_{jk}^{HO} h_j$	$o_k^- = \sigma(\eta_k^-)$
	+	-	o_k^+ = target output

Table 2: Equilibrium network variables in GeneRec (O'Reilly, 1996).

In case of the *plus* phase only the hidden activations are necessary to compute and that could be achieved by computing $\phi(\eta_i)$. In case of the *minus* phase, where only inputs are clamped it is necessary to find an *equilibrium* activation state for which the equations (2) hold. There are several approaches as dicussed in recurrent networks (1.1.3). In our implementation we choose the *iterative method* with the following

rules for computing activations a_i:

$$a_i(t+1) = \begin{cases} s_i & \text{if } i \in \text{input} \\ \phi(\sum_j w_{ji} a_j(t)) & \text{otherwise} \end{cases}$$
$$a_i(0) = \begin{cases} s_i & \text{if } i \in \text{input} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where $a_i(t)$ is the activation of i–th unit in discrete time t. The rules (13) are iterated while $|a_i(t+1)a_i(t)| > \epsilon$ for some unit i, where $\epsilon \in \mathbb{R}^+$. For the successful networks it was enough to have 3 to 33 iterations. On the contrary, especially for BAL-recirc (2.3.2), the process was not able to converge. In such case we took the average of last two activations, which reduced the ratio of diverging networks. But still we encountered *fluctuation* (2.4.2) with arbitrary size. This method is further discussed in Orrú et al. (2008).

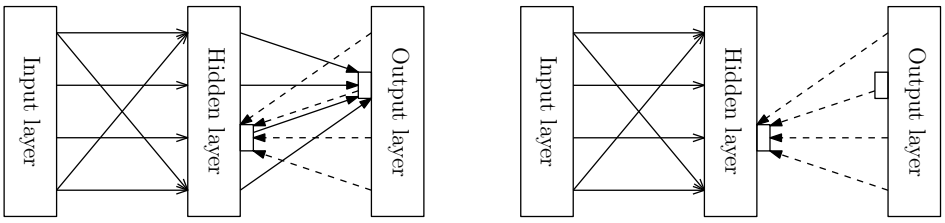

Figure 5: Depicting the minus (left) and plus (right) phases of GeneRec defined in Table 2. Taken from Orrú et al. (2008).

Learning rule. GeneRec uses three weight matrices W^{IH}, W^{HO} and W^{OH} for the input–hidden, hidden–output and output–hidden weights. It also has the "$-$" and "$+$" phases as CHL with same meaning, i.e. in the *minus* phase only the input vector is clamped and in the *plus* phase both input and target vectors are clamped as seen in Table 2. Generec uses the non symmetric version of the CHL rule for all three weight matrices:

$$\Delta w_{ij} = \lambda a_i^- (a_j^+ - a_j^-), \tag{14}$$

where a_i^- denotes the presynaptic and a_j^- denotes the postsynaptic unit activation in minus phase, a_i^+ is the postsynaptic activation from plus phase and λ denotes the learning rate. For example, when updating W^{HO} then $a_i^- = h_i^-$, $a_j^- = o_j^-$ and $a_j^+ = t_k$.

Modifications. It is important to note that O'Reilly (1996) proved that GeneRec converges if the learning rule (14) is a valid approximation to the error derivate and the weights are symmetric, i.e. $W^{HO} = (W^{OH})^T$. O'Reilly (1996) based on CHL and the midpoint method for gradient computation proposed two more learning rules for GeneRec:

$$\frac{1}{\lambda}\Delta w_{ij} = \frac{1}{2}(a_i^- + a_i^+)(a_j^+ - a_j^-) \tag{15}$$

$$\frac{1}{\lambda}\Delta w_{ij} = (a_j^+ a_i^- - a_j^- a_i^+) - 2a_j^- a_i^- \tag{16}$$

where (15) is called the *midpoint learning rule* and (16) is called the *symmetric learning rule* which aims to preserve the weight symmetry. By combining rules (15) and (16) we get the *CHL* learning rule (11). Thus we see that GeneRec is closely related to CHL.

$$\frac{1}{\lambda}\Delta w_{ij} = (a_i^+ a_j^+) - (a_i^- a_j^-) \tag{17}$$

1.3 Bidirectional activation-based learning algorithm

Design of Bidirectional Activation-based Learning algorithm (BAL) by Farkaš and Rebrová (2013) is motivated by the biological plausibility of GeneRec. BAL inherits the learning rule (14) of GeneRec and also the two phases. But unlike GeneRec, BAL aims to learn bidirectional mapping between inputs and outputs and for this purpose it uses four weights W^{IH}, W^{HO}, W^{OH} and W^{HI}. The design of BAL is symmetric as shown in Table 3 and thus we avoid calling inputs, outpus, minus phase or plus phase. We rather choose *forward* and *backward* which could be interchanged. This brings us different notation where a^{F} denotes forward activations, a^{B} backward activations, x is the first activation layer, i.e. *front layer*, y is the third activation layer, i.e. *back layer*, F means *forward pass* and B means *backward pass*. Layers x and y are *visible* and layer y is hidden. Note that all non- timulus units have learnable biases and their weights are updated in a same way as regular weights.

Layer	Phase	Net Input	Activation
x	F	-	x_i^{F} = forward stimulus
h	F	$\eta_j^{\mathrm{F}} = \sum_i w_{ij}^{IH} x_i^F$	$h_j^{\mathrm{F}} = \sigma(\eta_j^{\mathrm{F}})$
y	F	$\eta_k^{\mathrm{F}} = \sum_j w_{jk}^{HO} h_j^F$	$y_k^{\mathrm{F}} = \sigma(\eta_k^{\mathrm{F}})$
y	B	-	y_k^{B} = backward stimulus
h	B	$\eta_j^{\mathrm{B}} = \sum_k w_{kj}^{OH} y_k^{\mathrm{B}}$	$h_j^{\mathrm{B}} = \sigma(\eta_j^{\mathrm{B}})$
x	B	$\eta_i^{\mathrm{B}} = \sum_j w_{ji}^{HI} h_j^{\mathrm{B}}$	$x_i^{\mathrm{B}} = \sigma(\eta_i^{\mathrm{B}})$

Table 3: Activation phases and states in BAL (Farkaš and Rebrová, 2013).

In the first phase, called *forward pass*, the *forward stimulus* is clamped and forward activations are computed. In the same way, in the second phase, called *backward pass*, the *backward stimulus* is clamped and backward activations are computed. We can imagine the backward pass as a reconstruction of the target pattern for the forward pass. For the *forward* learning rule the *difference* between the forward pass and the backward pass is used as shown in equation (18).

$$\Delta w_{ij}^{\mathrm{F}} = \lambda \; a_i^{\mathrm{F}} (a_j^{\mathrm{B}} - a_j^{\mathrm{F}}). \tag{18}$$

The *backward* learning rule (19) is same as the forward learning rule (18). We will reference them together as BAL *learning rule*.

$$\Delta w_{ij}^{\mathrm{B}} = \lambda \; a_i^{\mathrm{B}} (a_j^{\mathrm{F}} - a_j^{\mathrm{B}}). \tag{19}$$

Note that we can treat the differences $(a_j^{\mathrm{B}} - a_j^{\mathrm{F}})$ and $(a_j^{\mathrm{F}} - a_j^{\mathrm{B}})$ as *error terms* which push the forward and backward activation to settle. Both forward (18) and backward (19) learning rules are same as the basic GeneRec learning rule (14). We experimented with different learning rules (1.2.3).

2 Simulations

Introduction

In this section, we describe all simulations which we performed in our work. We start with definition of our evaluation metrics in Section 2.1 and te we give a short description of datasets used for our simulations in Section 2.2. The main part are the descriptions of used modified versions of BAL in Section 2.3. Finally, we end with additional experiments used to prove or disprove our hypotheses in Section 2.4.

2.1 Evaluation methods

Following Farkaš and Rebrová (2013), we measured two key properties of a BAL-like network. The more important being *success rate* and second being *convergence time*.

Success rate. Before comparing *given* outputs on both visible layers the activations y^F and x^B are classified by a treshold:

$$g_k = \begin{cases} 1 & \text{if } x_k^B > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

For an input I from the sample set $\mathbb{S}$ we denote vector *given* by propagation of neuron activations G^I and the *target* vector we denote T^I. We distinguish two main success measures:

- *Bit success (bitSucc)* defined as $bitSucc = avg_{I \in \mathbb{S}} \sum_i^{|T^I|} |T_i^I - G_i^I|$ and
- *Pattern success (patSucc)* defined as

$$patSucc = avg_{I \in \mathbb{S}} \begin{cases} 1 & \text{if } T^I = G^I \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Convergence time. We denote the number of epochs before the stop of the training algorithm as *convergence time*. There are several possibilities when to stop the learning algorithm. Usually, training could be stopped for two reasons. The network could either reach the *stopping criteria* or the maximum epoch is reached. Given by nature of used datasets we trained the neural networks while $patSucc^F \neq 1$. In case of the

digits (2.2.3) dataset we decided to stop the training if $patSucc^F$ was not increased for 3 epochs. Note that we are motivated to decrease the convergence time as it makes the training process faster.

2.2 Datasets

For analysing our versions of BAL (1.3) we have chosen three datasets on which we tested and compared the performance of our models.

2.2.1 4-2-4 Encoder

The *4-2-4 econder* task is the simplest dataset we have been working with. It consists of four four–dimensional samples $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$ and $(0, 0, 0, 1)$ where each sample is mapped to itself. We use two units on the hidden layer what gives us the 4-2-4 architecture. The 4-2-4 encoder task is a well–known problem and used previously for testing GeneRec (O'Reilly, 1996) and BAL (Farkaš and Rebrová, 2013). In the case of BAL only 60–65% $patSucc^F$ was achieved what leaves window for improvement. We chose this dataset as it is convenient for testing novel approaches as the learning progress of the network could be checked by hand and eye.

2.2.2 Complex binary vector associations

The *complex binary vector associations (CBVA)* task was used in Farkaš and Rebrová (2013) and its motivated by the sensory–motor mappings between distributed patterns. The task is to associate between sixteen 16–dimensional vectors which all had 3 active units. There are always 4 distinct overlapping input patterns associated with exactly one output pattern. As there are several possibilities of inputs for each output, then in the *backward* way it is impossible to achieve perfect $bitSucc^B$ or $patSucc^B$. Therefore, we would expect from the network to give a *blend* of the four input patterns corresponding to one output.

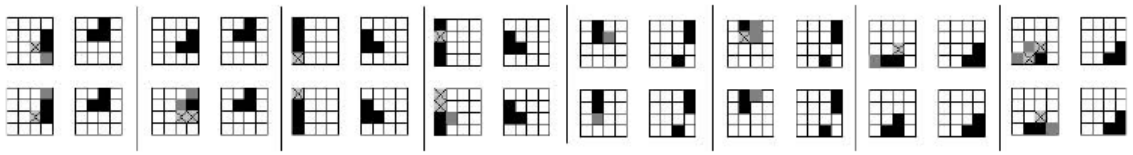

Figure 6: BAL performance on *CBVA*. Black color stands for target–estimate match, gray for target only and gray with a cross for false–positive estimate (Farkaš and Rebrová, 2013).

2.2.3 Handwritten digits

The well–known MNIST dataset of *handwritten digits* (LeCun et al., 1998b) first analysed by LeCun et al. (1998a) consists of 42,000 samples of 28×28 grayscale images mapped to one digit. We have chosen this dataset for three reasons. First, it is big and complex enough to test the practicallity of our models. Second, performance of many models is known on this dataset and therefore, we can easily compare performance of our models to these models. And third, we can easily visualize the backward *blend* representations and intuitively confirm if our models perform as expected. These visualisations could be found in Section 3.3.3.

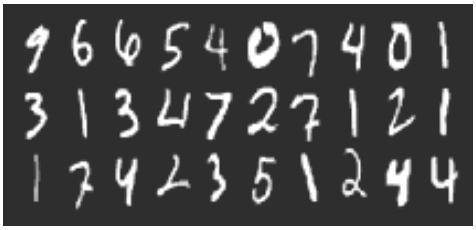

Figure 7: Samples from the *digits* dataset.

2.3 New models

Introduction

In this section, we describe models we developed based on BAL (1.3). We start with description of *TLR* (2.3.1) as the most important model of our work and continue with description of *BAL-recirc* which is a combinations both of BAL and GeneRec (1.2.3). Both models are inspired by our experiments further described in (2.4).

2.3.1 Two learning rates

We proposed the *Two learning rates (TLR)* model as a solution for the *hidden activation settling* (2.4.3) based on the dynamic learning rate model (2.4.4). As the name suggests this model uses two learning rates. The learning rate λ_v, i.e. *lambda visible*, for weights W^{HI} and W^{HO} and learning rate λ_h, i.e. *lambda hidden*, for weights W^{IH} and W^{OH}. Both λ_v and λ_h are constant during whole learning phase. Note that the names are derived from the layer names on which the error term $(a_j^+ - a_j^-)$ is computed.

Our simulations show that setting $\lambda_h \ll \lambda_v$ could lead to significantly better performance in comparison to the standard BAL model (3.1.1). Our intuition explains it as follows: because $\lambda_h \ll 1$ thus W^{IH} and W^{OH} are updated only little and also activations h^{F} and h^{B} change only a little and $|h^{\mathrm{F}} - h^{\mathrm{B}}|$ converges to zero slower. Thus error terms $(y_j^{\mathrm{B}} - y_j^{\mathrm{F}})$ and $(x_j^{\mathrm{F}} - x_j^{\mathrm{B}})$ from the BAL learning rule 18 for W^{HI} and W^{HO} impact the weight change longer with **non** *constant hidden activations* (2.4.3). The importance of hidden activations was confirmed by the candidate selection experiment (2.4.2).

We can also explain TLR in terms of biological plausibility. Results of TLR suggests $\lambda_h \ll \lambda_v$. That means the input–hidden mapping is changed only a little and mostly hidden–output mapping is trained. Let us put this in context of interpreting images from eyes in human brains. That would mean that human representations, i.e. hiddens, of eye images, i.e. inputs, are changed only little while the symbolic interpretations, i.e. outputs, could change rapidly.

Related work. Most of the previous work regarding different learning rates is based on *Dynamic learning rate (DLR)* model introduced by Jacobs (1988). Aim of DLR is to compute *best* learning rate in terms of successful convergence and avoidance of local minima (Behera et al., 2006). There are several approaches how to achieve this. Most of them have individual learning rates for each weight in the network which could change in time. Some approaches precompute learning rates (Weir, 1991) while others *adapt learning rates dynamically* through the training process (Yu and Chen, 1997; Magoulas et al., 1999; Yu and Liu, 2002).

The most relevant information for TLR we found was the *tip* given by LeCun et al. (2012) that "Beyond choosing a single global learning rate, it is clear that picking a

different learning rate λ_{ij} for each *weight* w_{ij} can improve convergence. Weights in lower layers should typically be larger than in the higher layers." Unfortunatelly, we were not able to find other simulations which backup this tip. Moreover, LeCun et al. (2012) focuses on individual *weights*, and not *matrices* as TLR. Therefore, we conclude that, to our best knowledge, the TLR model is unique in terms of number of learning rates.

2.3.2 Recirculation BAL

The aim of *Recirculation BAL* is to combine the ideas of BAL (1.3) and iterative activation from GeneRec (1.2.3). In other words, instead of computing the forward pass using only W^{IH} and W^{HO} we add a recirculation step between matrices W^{HO} and W^{OH}, and similary for the backward pass and W^{HI} and W^{IH}. We tried two approaches of such a combination. The first one is *Bidirectional Iterative Activation (BIA)* which is a straightforward implementation of the idea. Activation computation of BIA is shown in Table 4.

Layer	Phase	Net Input	Activation
$\mathbf{x}$	F	-	$x_i^{\mathrm{F}} = \text{stimulus}$
$\mathbf{h}$	F	$\eta_j^{\mathrm{F}} = \sum_i w_{ij}^{IH} x_i^F + \sum_k w_{kj}^{OH} y_k^F$	$h_j^{\mathrm{F}} = \sigma(\eta_j^{\mathrm{F}})$
$\mathbf{y}$	F	$\eta_k^{\mathrm{F}} = \sum_j w_{jk}^{HO} h_j^F$	$y_k^{\mathrm{F}} = \sigma(\eta_k^{\mathrm{F}})$
$\mathbf{y}$	B	-	$y_k^{\mathrm{B}} = \text{stimulus}$
$\mathbf{h}$	B	$\eta_j^{\mathrm{B}} = \sum_k w_{kj}^{OH} y_k^{\mathrm{B}} + \sum_i w_{ij}^{IH} x_i^{\mathrm{B}}$	$h_j^{\mathrm{B}} = \sigma(\eta_j^{\mathrm{B}})$
$\mathbf{x}$	B	$\eta_i^{\mathrm{B}} = \sum_j w_{ji}^{HI} h_j^{\mathrm{B}}$	$x_i^{\mathrm{B}} = \sigma(\eta_i^{\mathrm{B}})$

Table 4: Activation for BIA (2.3.2). The only difference with BAL (3) are the recurrent terms $\sum_k w_{kj}^{OH} y_k^F$ and $\sum_i w_{ij}^{IH} x_i^{\mathrm{B}}$.

The second one is *Bidirectional GeneRec (BiGeneRec)* which has three phases. The first F^- phase is same as the *minus* phase of GeneRec and the third C^+ phase is same as the *plus* phase of GeneRec. The second B^- phase is same as the F^- phase but from back to front. In other words we can treat F^- and B^- phases as *forward* and *backward* minus phase of GeneRec and the C^+ phase as the plus phase of GeneRec. As in the

F^- phase, only the forward weights W^{IH} and W^{HO} are updated and in the B^- phase only the backward weights W^{OH} and W^{HI} are updated. We can treat weight updates of BiGeneRec as two independent GeneRec update steps.

Layer	Phase	Net Input	Activation
Hidden (h)	C^+	$\eta_j^+ = \sum_i w_{ij}^{IH} x_i^{\mathrm{F}} + \sum_k w_{kj}^{OH} y_k^{\mathrm{B}}$	$h_j^+ = \sigma(\eta_j^+)$

Table 5: Difference between BiGeneRec (2.3.2) and BIA (4) is the additional C^+ phase corresponding to the plus phase of GeneRec (1.2.3).

For both BIA and BiGeneRec we experimented with both asymmetric and symmetric versions. For the asymmetric version we experienced problems with *fluctuation*. This is briefly discussed in Section 1.2.3 and in Section 2.4.2.

2.4 Experiments

In this section, we describe modifications which could be applied to any model of artificial neural networks. Some of them, such as momentum (2.4.1) and batch learning mode (2.4.4) are well–known approaches. Other, such as dynamic learning rates (2.4.4) or weight initialization classification (2.4.4) were chosen specially to reinforce important features of BAL (1.3).

2.4.1 Momentum

Momentum was introduced by Jacobs (1988) as an special case of dynamic learning rate (2.4.4). It is an extension to any learning rule for any artificial neural network by adding a *momentum* term:

$$\Delta w_{ij}(t) = \text{learning rule} + \mu \Delta w_{ij}(t-1),$$

where μ is a real–valued parameter.

It is argued that momentum could overcome settling in local minima by leveraging the second derivate (Phansalkar and Sastry, 1994). It is also "believed that momentum could render the learning procedure more stable and accelarate convergence" but "momentum setting is as practice shows problem dependend" (Riedmiller and Braun,

1993). As learning rate has an adaptive dynamic version (2.4.4), momentum also has an adaptive dynamic version (Miniani and Williams, 1990).

2.4.2 Candidate selection

The *candidate selection* model was used to test and confirm if some particular network *features* (2.4.2) have an impact to the overall network performance. The only difference between standard BAL (1.3) and candidate selection is that before the training phase, N networks are randomly generated from which a *best candidate network* is selected. For selecting the best candidate we use *feature function* defined as $F : \text{network} \mapsto \mathbb{R}$.

Algorithm 2.1 Candidate selection pseudocode.

```
best_candidate ← (∞, null)
for i = 1 to N do
    gn ← generate_network()
    candidate ← (F(gn), gn)
    if candidate < best_candidate then
        best_candidate ← candidate
    end if
end for
```

Features. We denote X_I, H_I and Y_I as the front, hidden and back activation *vector*s for input I (and the corresponding target). For the rest of the notation please consult Section 3. We measured the following *features*:

- $dist_H$ (real) – the average distance between all hidden activations of the inputs, i.e. $avg_{I \neq J} \left(dist(H_I^+, H_J^+) \right)$.

- $dist_H^{FB}$ (real) – the average distance between corresponding forward and backward *hidden* activations, i.e. $avg_I \left(dist(H_I^-, H_I^+) \right)$.

- $dist_V^{FB}$ (real)– the average distance between corresponding forward and backward *visible* activations, i.e. $avg_I \left(dist(Y_I^-, X_I^+) \right)$. Note that this feature is only relevant for auto–associative tasks such as 4-2-4 encoder (2.2.1).

- *matrix_weight* (real) – average weight of the network, i.e. average value of all weight matrices W^{IH}, W^{HO}, W^{OH} and W^{HI}. Note that each matrix value has the same impact to *matrix_weight*.

- *in_triangle* (bool) – check if hidden activations of inputs H_I^+ form a convex polygon, i.e. if the hidden activation points are all *lineary separable* (1.1.1). Therefore $in\_triangle = 0$ is a necessary condition for perfect success rate. Consult Figure 10 for examples of convex and non–convex hidden activations. Note that *in_triangle* was implemented only for hidden size equal to two, i.e. for the 4-2-4 encoder task.

- $fluctuation$ (real) – the maximal difference between activations in the last two iterations when using the *iterative method* for activation computation (13). With other words, let $a_i(t)$ be the activation of unit i in iteration t and T be number of iterations. Then $fluctuation$ is $max_i|a_i(T-1) - a_i(T)|$. So if $fluctuation \approx 0$ then the iterative method was successful and all activations settled.

Also all *parameters* of TLR were included such as λ_v, λ_h (2.3.1), momentum μ (2.4.1) and weight distribution σ (2.4.4).

Linear regression. To get the most important features we trained a feature function on a *feature dataset* consisting of individual features and with *label* equal to $bitErr^F = 1 - bitSucc^F$. The dataset was created by generating standard BAL networks, measuring feature values before the training phase and adding the success rate label after the training phase. On this feature dataset we trained a simple linear regression model shown in equation (22).

$$\begin{aligned} bitErr^F = &-0.328 \times dist_H + 0.140 \times dist_H^{FB} - 0.100 \times dist_V^{FB} \\ &+ 0.019 \times matrix\_sim - 0.127 \times \sigma + 0.000 \times matrix\_weight + 3.610 \end{aligned} \quad (22)$$

From equation (22) we observe that the feature which contributed the most to $patSucc^F$ is $dist_H$. This was used as a inspiration for the TLR model (2.3.1). Furthermore, we started using the feature function (23) for all candidate selection simulations. It simply choses the network with greatest $dist_H$. For the 4-2-4 encoder task we also add

in_triangle as primary feature to ensure the convexity of initial hidden activations.

$$F(network) = -dist_H(network) \tag{23}$$

2.4.3 Hidden activations

We observed that *hidden activations* in BAL (1.3) tend to settle fast as shown in Figure 13, i.e. the weight changes become close to zero because $|H^F - H^B| \approx 0$. Therefore, the network is de facto reduced to a two–layer network between the constant hidden activations and the target values. Thus for the cases when the hidden activations are not *linearly separable* (1.1.1), it is impossible for w_{HI} and w_{HO} to learn targets. This behaviour is demonstrated by the *in_triangle* measure (2.4.2).

A more formal explaination why the hidden activations tend to settle fast could be given by the GeneRec learning rule (14):

$$\Delta w_{ij} = a_i(b_j - a_j), \tag{24}$$

which for the W^{IH} and W^{OH} yields:

$$\Delta w_{ij}^{IH} = x_i^F(h_j^B - h_j^F)$$
$$\Delta w_{ij}^{OH} = y_i^B(h_j^F - h_j^B).$$

We see that both terms $(h_j^B - h_j^F)$ and $(h_j^F - h_j^B)$ push W^{IH} and W^{OH} to settling $h_j^B = h_j^F$. This experiment was one of the reasons we started to experiment with dynamic learning rate (2.4.4) which lead to the TLR model (2.3.1).

2.4.4 Other experiments

In this section, we describe additional models, model modifications and experiments we proposed and used during our work. We further introduce notation used in the results section.

Weight initialization classification. As candidate selection suggested in Section 2.4.2, weight initialization could be crucial for the success rate of BAL. To further analyse this hypothesis we propose the following experiment. First we generate n networks N_i with random weights W_i, train them and label them $s_i = patSucc^F$. This way we

get a dataset $D = (W_i, s_i)$ for which we can fit a model M and use it for prediction of s_i. Then we analyse M and try to propose hypothesis for successful weights. Finally, we would modify the weight initialization algorithm that it will support the more successful networks. Note that this experiment was not implemented but we recommend it for future work.

In our work, we used a weight initialization algorithm inspired by O'Reilly (1996). For each weight w_{ij} we select randomly a value from the normal probability distribution:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{22}$$

where weight initialization constant σ is one of the network parameters and $\mu = 0$ is the expected value of the normal distribution. We showed that σ influences the success rate (22). It is recommended by O'Reilly (1996) to set $\sigma = \frac{1}{\sqrt{N+1}}$ where N is the number of units on the input layer including bias.

Dynamic learning rate. The idea of *dynamic learning rate (DLR)* was introduced by Jacobs (1988). DLR uses separate *learning rate*s for each weight w_{ij} which could change in time t and are denoted $\lambda_{ij}(t)$. There are several possibilities how to set $\lambda_{ij}(t)$ which are briefly described in Section 2.3.1. We further tried to develop our own DLR model which would depend on error of the previous epoch. In one of our trials, we set the learning rate to be smaller with smaller error to make BAL settle hidden activations later. Although we were not able to increase the success rate, we admit there is space for further improvement (3.3.3). This modification was an inspiration for TLR (2.3.1).

Batch mode. Normally, before each epoch the training samples are shuffled. But in *batch mode* instead of updating weights after each training sample, the weight changes are accumulated for the whole epoch. With other words, all weight changes are summed for each sample in the training set and then weights are updated in *batch*. One can observe, that after the weights are initialized, the learning algorithm becomes deterministic. Therefore, this approach could be used to confirm or disprove the importance of weight initialization. We executed several simulations on BAL with *batch* weight update, but it had no significant impact.

Dropout. Based on the work of Hinton et al. (2012), we implemented the *dropout* method of learning. The main idea is that in each epoch, we randomly choose half of the hidden layer neurons, which will be ignored for this epoch. With other words, in each epoch a random subset of hidden neurons is chosen to be active, while the other hidden neurons are ignored. The motivation is to prevent co–adaptation of the hidden layer neurons (Hinton et al., 2012). We were not able to train any successful BAL (1.3) network using dropout on the 4-2-4 encoder (2.2.1) and CBVA (2.2.2) tasks. Therefore we soon dropped the idea, but we admit, that setting other probability p for dropout or applying it on higher dimensional tasks could have a positive impact on success rate.

Noise. Motivated by the *chaotic* behaviour of nature itself we tried adding *random noise* to each weight update. We hoped, that the possible noise could prevent settling of hidden activations to fast (2.4.3). Our simulations of BAL and BAL-recirc using random noise showed no significant increase in performance.

Multi-layer GeneRec. We implemented a multi-layer version of GeneRec (1.2.3). The recirculation step in the minus phase was extended to $2L - 3$ steps, where L is the number of layers. First the propagation of neuron activations goes $L - 1$ times forward and next it goes $L - 2$ times backward. Then the recirculation step (1.2.3) between layers 2, 3, ..., L is executed. Our implementation of multi-layer GeneRec using the 784–300–50–10 architecture achieved 43.22% success rate on the handwritten digits recognition task (2.2.3).

Symmetric BAL. Inspired by the necessary condition for convergence of GeneRec stated by O'Reilly (1996) we introduced *Symmetric BAL (SymBAL)*. SymBAL is a modification of BAL with symmetric weights $W^{IH} = (W^{HI})^T$ and $W^{HO} = (W^{OH})^T$. We found no significant improvement when using this approach with

3 Results

In this section, we present our most important *results* and an in depth analysis of TLR (2.3.1). We compare TLR to BAL (1.3), GeneRec (1.2.3) and other models. This section is organised by the datasets (2.2). We start with the *4-2-4 encoder* task in Section 3.1 which received most of our attention. Then we follow with the *CBVA* task in Section 3.2 and finish with our biggest dataset *digits* in Section 3.3.

3.1 4-2-4 Encoder

In this section, we analyse performance of TLR (2.3.1) for a broad range of parameters λ_h and λ_v. The network architecture is only 4-2-4 (2.2.1) what allows us to run plethora of simulations. There were two kinds of simulations. First are *two dimensional maps (TDM)*, where λ_v is plotted on the x axis, λ_h on the y axis and color is used for the z axis. Second are *timelines* which plot success rate to epoch for the best configuration found by TDM. To create TDM we ran 500 networks for each pair (λ_v, λ_h) and for timelines we ran 10000 networks. After the simulations ended, the average for each configuration was plotted. The networks were trained while $patSucc^F \neq 1$ or $epoch < Epoch_{\max}$. The $Epoch_{\max}$ was set to 100,000 in TDM and 1,000,000 in timelines. Note that most of the plots are in *logarithmic* scale.

3.1.1 Comparison

In Table 6 we can see the comparison of the most important models which we analysed on the *4-2-4 encoder* task. We achieved an improvement of BAL $patSucc^F$ from 62.7% to 93.1% by using two different learning rates (2.3.1). This result was further improved to 99.86% by selecting networks with candidate selection (2.4.2). This confirmed that hidden distance and convexity of hidden representations are important attributes of BAL.

On the other hand, many of the analysed models compared less to BAL. We tried the alternative GeneRec learning rules (1.2.3) on BAL, calling these models *BAL GeneRec Learning Rules (BAL GLR)*, but there was no instance which was able to achieve $patSucc^F > 0$. Also, success rate of *BAL-recirc* (2.3.2) was lower than success rate

of BAL. We experimented with *momentum* in Section 3.1.4 or BAL with symmetric weights, but both without significant improvement in success rate.

Algorithm (section)	λ_h	λ_v	$patSucc^F$	Epochs
BP (1.1.5)	2.4	2.4	100%	60
GR (1.2.3)	0.6	0.6	90%	418
GR Sym (16)	1.4	1.4	56%	88
GR Mid (15)	2.4	2.4	92%	60
CHL (1.2.1)	1.2	1.2	56%	77
BAL (1.3)	0.9	0.9	62.7%	5136.11
BAL TLR (2.3.1)	0.0002	500	93.12%	5845.01
BAL TLR Can (2.4.2)	0.0002	500	99.86%	150.417
BAL Recirc (2.3.2)	0.0001	1.0	36%	1221.6
BAL GLR (1.2.3)	any	any	0%	N/A

Table 6: Comparison of different models on the *4-2-4 encoder* task. Results for BP, GR, GR Sym, GR Mid and CHL are taken from O'Reilly (1996).

Note that, if we want to compare execution time based on *epochs* in Table 6, then we must be aware of that GeneRec and BAL-recirc epochs take longer than epochs of other models. This is because the recirculation step (1.2.3), for which about 3 to 33 iterations are needed for activation to settle (1.2.3). Thus the 418 epochs of GeneRec are comparable to the 5845 epochs of TLR in terms of execution time.

3.1.2 Two learning rates

In Figure 8 we compare success rate for range of λ_v and λ_h. It is interesting that the subspace with best achieving networks is around the half line $[(10, 0.001), (10^9, 0.001)]$. That means the performance mainly depends on λ_h while a constraint on λ_v is added. Also see the plot for epochs, where a *ridge* occurred around line $[(0.01, 0.0002), (10^9, 0.0002)]$. Unfortunatelly, we can only guess what is the reason behind this ridge. Maybe it is related to $Epoch_{\max}$ and the fact that we are calculating epochs only from successful networks. Therefore, successful networks having $\lambda_h < 10^{-6}$ need to converge using λ_v, because otherwise they would fail to converge because of $\lambda_h \cdot Epoch_{\max} < 1$.

Note that the success space is robust and therefore it is possible to find it by stochastic methods such as Monte Carlo as an additional parameters is introduced. This approach is needed as finding the optimal values of λ_v and λ_h could take long using the trivial exhaustive search.

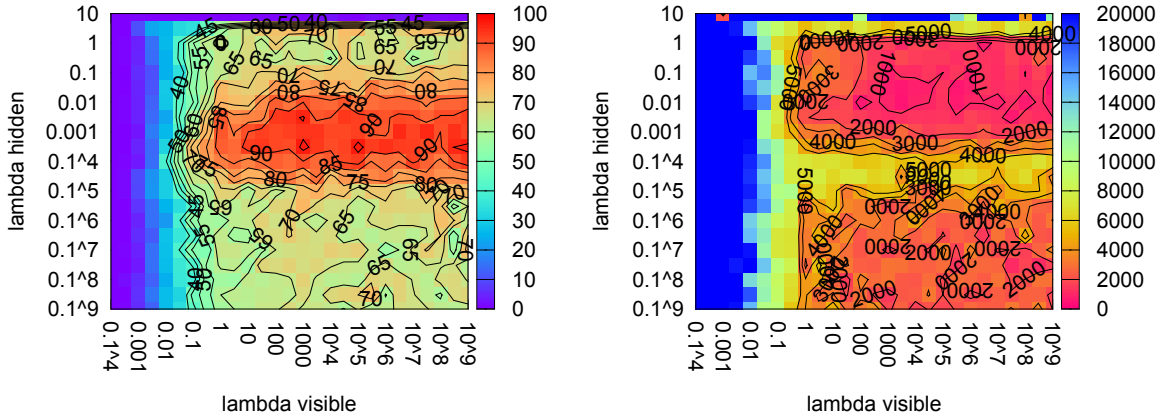

Figure 8: TLR success rate and convergence time needed for successful networks on the *4-2-4 encoder* task with $\sigma = 2.3$ and $\mu = 0.0$. Best network achieved 96.5$ with $\lambda_h = 0.0003$ and $\lambda_v = 1000.0$.

Note the inconsistency between Table 6, where 93.12% success rate was stated for TLR, and in Figure 8 where it was 96.5%. This is explained by the *law of big numbers*. In the first case the average performance of 10000 networks were used, thus the result is likely to mirror the reality. In the second case only 200 networks were used for a particular (λ_v, λ_h) pair. As there were about 50 candidates for best success rate, it was likely that some of them will achieved better than average.

In Figure 9 the success timeline for TLR with best λ_h and λ_v is analysed. We see that the success rate increases even after 10^5 epochs and our intuition tells us that it will continue even after 10^6 epochs. Another observation is that $patSucc^B$ first follows $patSucc^F$ for about 100 epochs, but then it stagnates at rate ≈ 0.8. We find this hard to explain as both the architecture and data are symmetric.

3.1.3 Hidden activations

In Figure 10 and in Figure 11 we show forward hidden activations. Each color represents the forward hidden representation of one of the four inputs in the *4-2-4 encoder* task. As the hidden layer size is 2 then the hidden activation can be mapped to the two

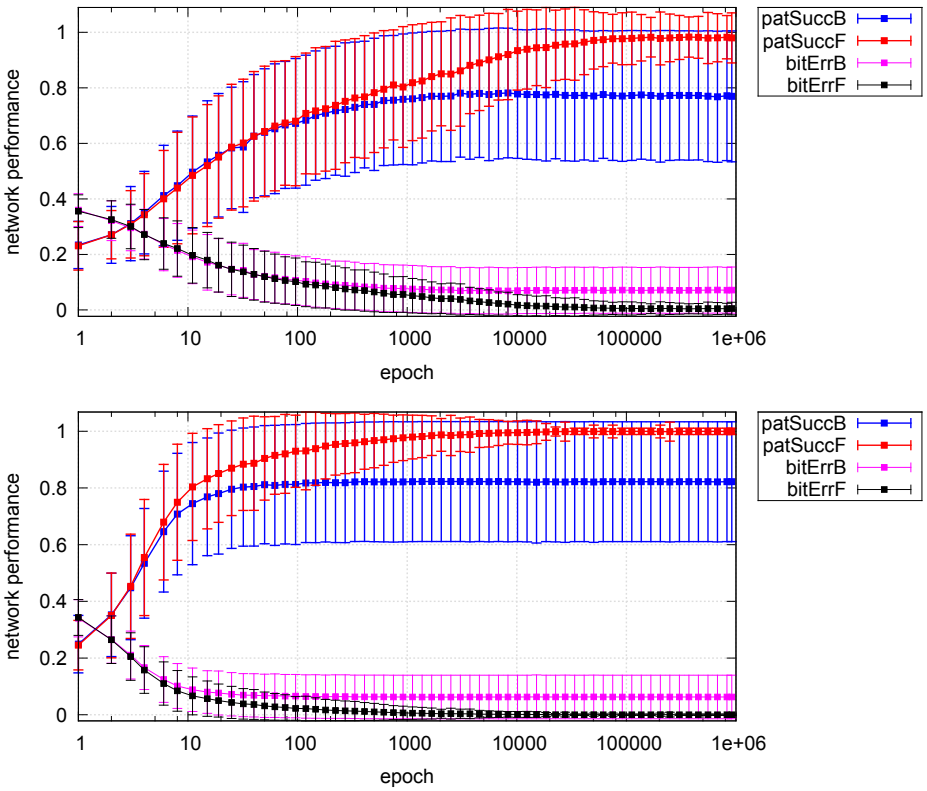

Figure 9: TLR success rate timeline for the *4-2-4 encoder* task with $\lambda_h = 0.0002$ and $\lambda_v = 500$. The top plot without candidate selection and the bottom plot with candidates selection.

dimensional space. The plotted activations start at *epoch* = 0 where the starts are depicted with black squares and continue as outlined by the the lines.

The main difference between TLR and BAL seems to be the speed of activation change. For BAL, as shown in Figure 10, we have a step of size 0.6, which corresponds to four, one for each input, weight updates. Another observation is that after some initial steps BAL tends to stop the activation change. This could be contributed to settling $|H^F - H^B| \approx 0$ as discussed in Section (2.4.3).

Another source of error could be *non–convex* hidden activation initializations. In the beginning, the weight matrices are initialized by random and that leads to random hidden activations. And if the hidden activations are also non–convex in the end then it is impossible to perfectly classify on the hidden–to–visible layer due the linear separability theorem discussed in Section 1.1.1. Therefore if the network had non–convex hidden activations in the beginning, then it must *escape* the non–convex state.

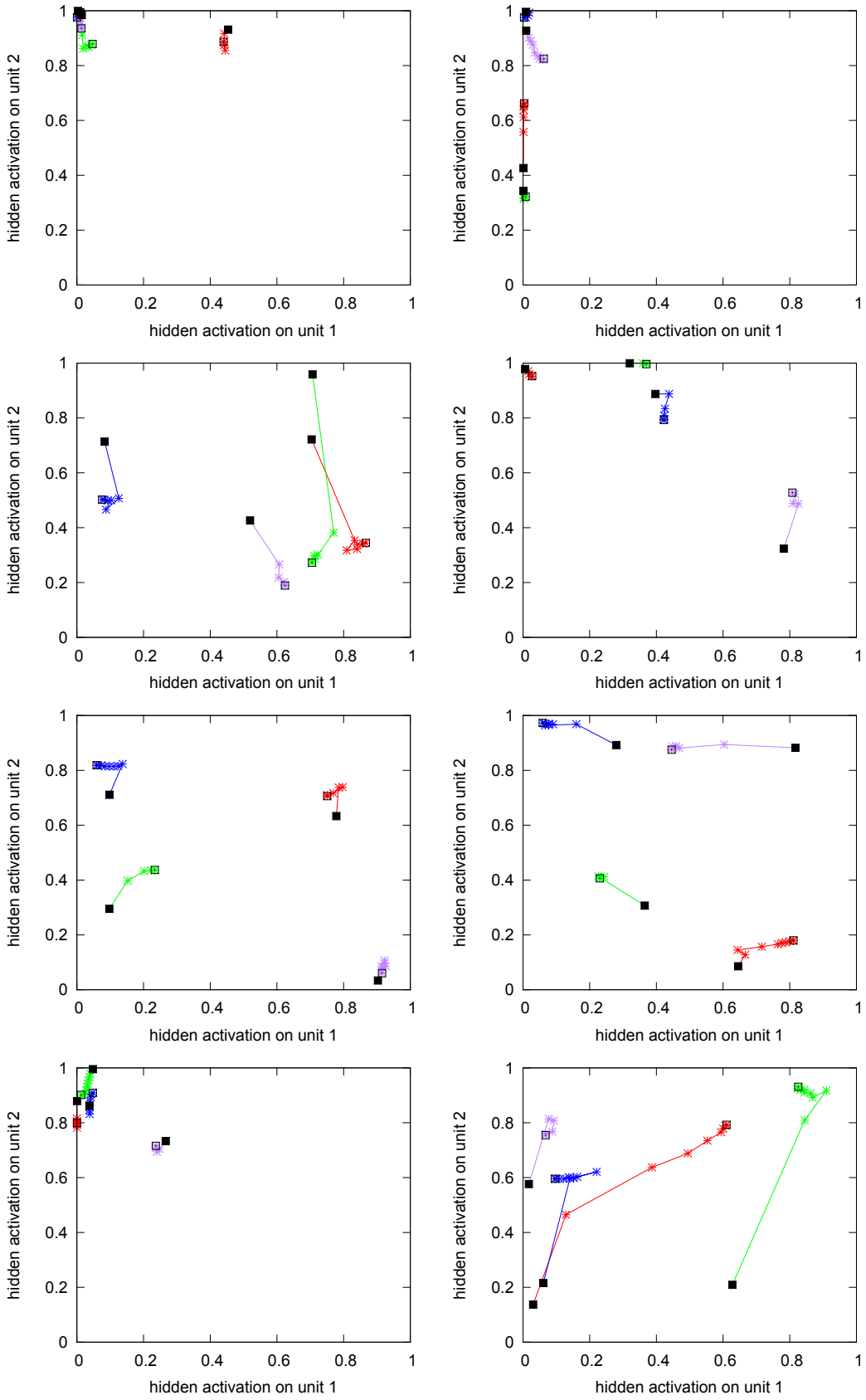

Figure 10: *BAL* hidden activations on the *4-2-4 encoder* task. The top 2×2 are **un**successful networks and the bottom 2×2 successful ones. Only the first ≈ 100 epochs had change in activation.

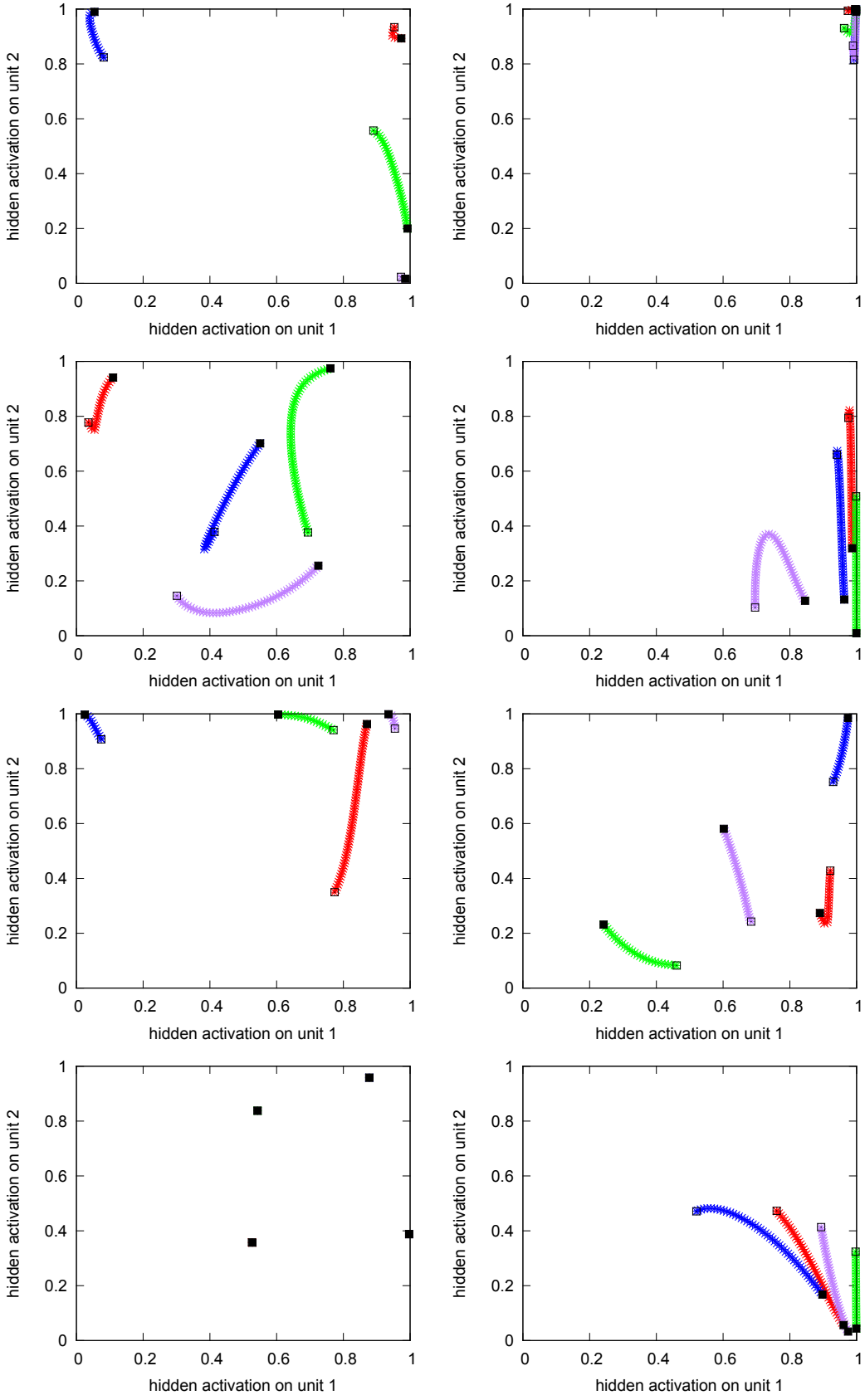

Figure 11: *TLR* hidden activations on the *4-2-4 encoder* task. The top 2×2 are **un**successful networks and the bottom 2×2 successful ones. Only the first ≈ 10000 epochs had change in activation.

3.1.4 Momentum

Adding momentum (2.4.1) to the basic TLR simulations (3.1.2) had no significant effect on network performance as shown in Table 7 where for each momentum we take average from all simulations. Only a little improvement in convergence rate was achieved. We ran simulations for range of λ_v and λ_h values as shown in Figure 12.

momentum	avg(success)
0.001	0.4419
0.003	0.4428
0.01	0.4440
0.03	0.4464
0.1	0.4468
0.3	0.4493

Table 7: Comparison of different momentums for TLR on the *4-2-4 encoder* task.

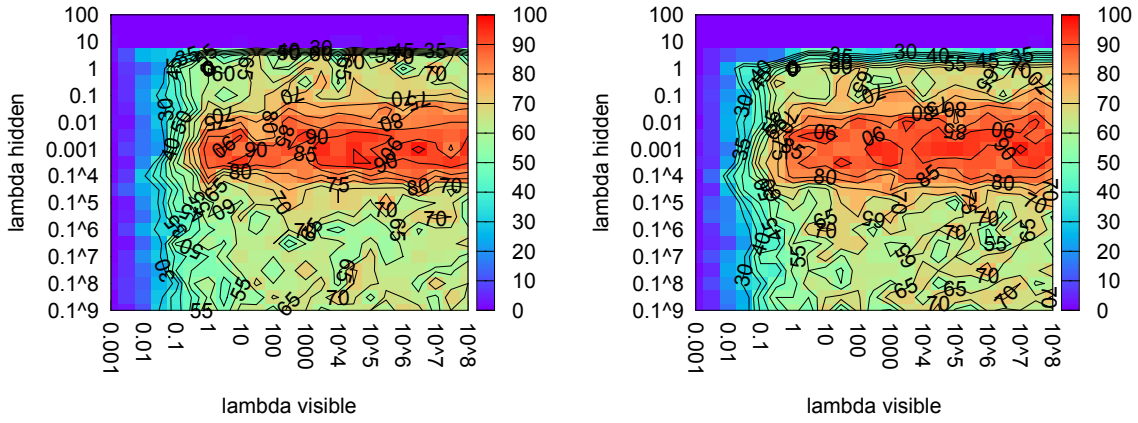

Figure 12: Comparison of momentums $\mu = 0.01$ (left) and $\mu = 0.3$ (right) for TLR on the *4-2-4 encoder* task.

3.1.5 Features

In this section, we analyse the timelines for features $dist_H^{FB}$, $dist_V^{FB}$, $dist_H$ and $matrix\_weight$ introduced in Section 2.4.2. We compare the values for BAL, TLR and TLR with candidates selection (TLR-can).

The importance of the distance between forward and hidden activations $dist_H^{FB}$ is

shown in Figure 13. We observe that $dist_H^{FB}$ settles fast for BAL. This means the network stops learning as the difference $(h_j^B - h_j^F) \approx 0$ renders the weight update in BAL update rule to zero. On the other hand, we see that $dist_H^{FB}$ of TLR is not affected in time. This could be contributed to lower magnitude of updates W^{IH} and W^{OH}. TLR-can is stationary as it converges in 150 epochs.

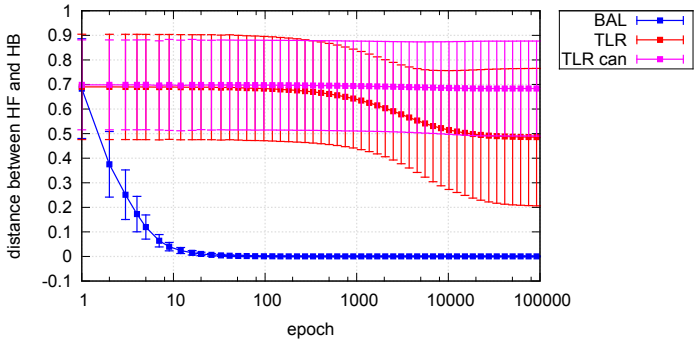

Figure 13: Comparison of $dist_H^{FB}$ (2.4.2) timelines for the 4-2-4 encoder task.

Difference between forward and backward outputs $dist_V^{FB}$ is shown in Figure 14. We see that BAL decreases the difference monotonously and converges to zero. That means the mappings learned by forward and backward weights are same. But, this is not true for TLR. We observe a *rebound* around epoch 4. This could be contributed to $\lambda_h \ll \lambda_v$ as it could greatly increase $|W_{ij}^{HI}|$ and $|W_{ij}^{HO}|$ and therefore activations on output layers change rapidly. After the rebound a convergence phase start which could be explained by settling of $|y^F - y^B| \approx 0$. This could explain the difference between y^F and x^B discussed in Section 3.1.2.

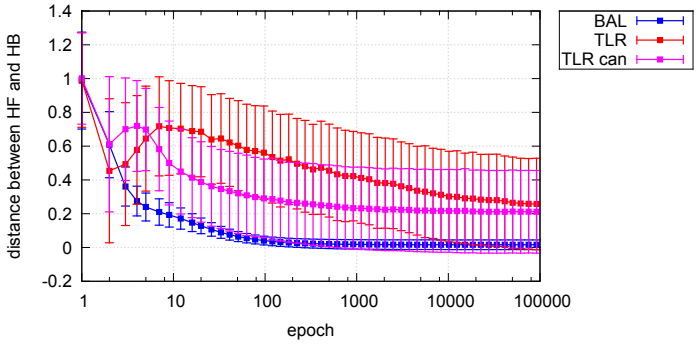

Figure 14: Comparison of $dist_V^{FB}$ (2.4.2) timelines for the 4-2-4 encoder task.

Candidate selection showed that $dist_H$ is the primary feature contributing to net-

works success rate (22). In Figure 15 we see that candide selection indeed picks networks with greater $dist_H$. We can observe that $dist_H$ stagnates through the training phase. This reinforces the importance of proper weight initialization.

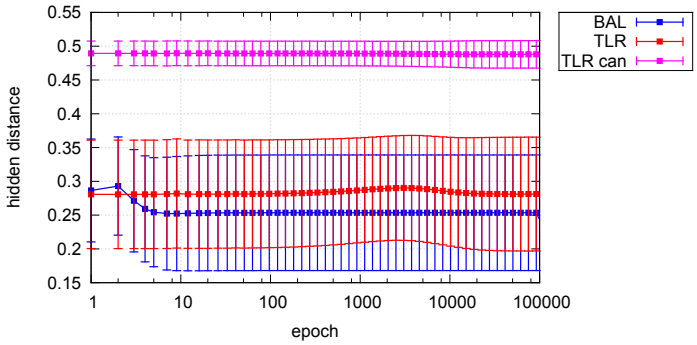

Figure 15: Comparison of $dist_H$ (2.4.2) timelines for the 4-2-4 encoder task.

Analysis of the average weight of matrices $matrix\_weight$ in Figure 16 shows that setting $1 \ll \lambda_v$ leads to greater $matrix\_weight$. We can observe that $matrix\_weight$ is not bounded for TLR and therefore a convenient stopping criteria should be picked.

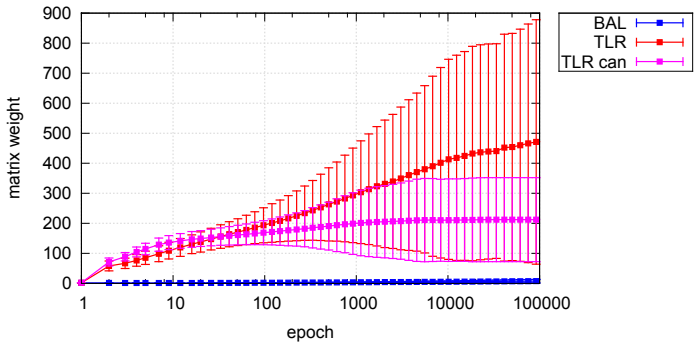

Figure 16: Comparison of $matrix\_weight$ (2.4.2) timelines for the 4-2-4 encoder task.

3.1.6 Other

Recirculation BAL. In Figure 17 we see that *BAL-recirc* (2.3.2) achieved lower success rate than BAL on the *4-2-4 encoder* task. In comparison with TLR we see that there is a global maxima at point $\lambda_h = 0.0001$ and $\lambda_v = 1.0$. We can therefore conclude that the space of successfull parameters λ_h and λ_v is bounded. Similar results were achieved for GeneRec as shown in Figure 18.

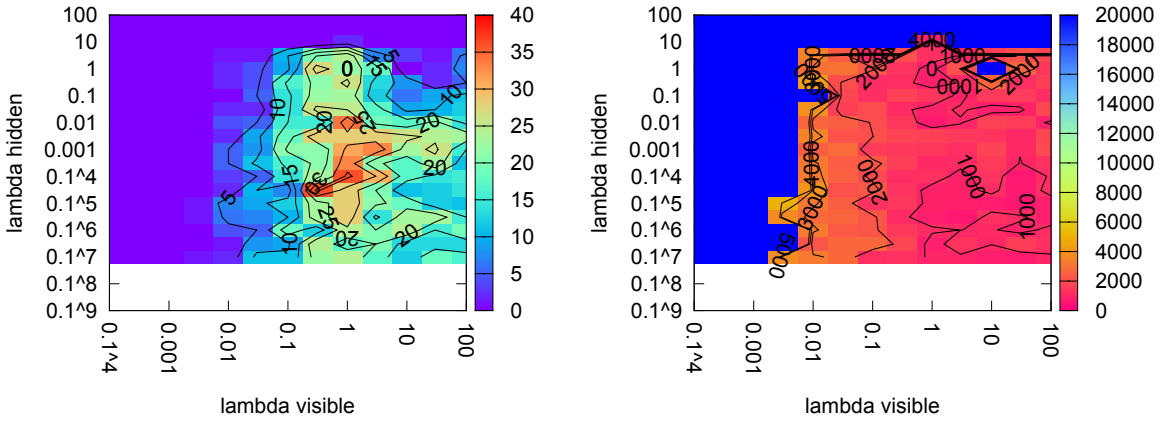

Figure 17: BAL-recirc (2.3.2) on the *4-2-4 encoder*. Best success rate 36% achieved with $\lambda_h = 0.0001$ and $\lambda_v = 1.0$.

GeneRec. As with generalizing of BAL to TLR we tried also generalizing GeneRec using the two learning rates approach. The results in Figure 18 show no increase in success rate in comparison with setting $\lambda_v = \lambda_h$, i.e. using the original GeneRec. We can observe that the results are similar with the results of BAL-recirc shown in Figure 17. Finally, as with BAL-recirc, we can conclude that the success space is bounded.

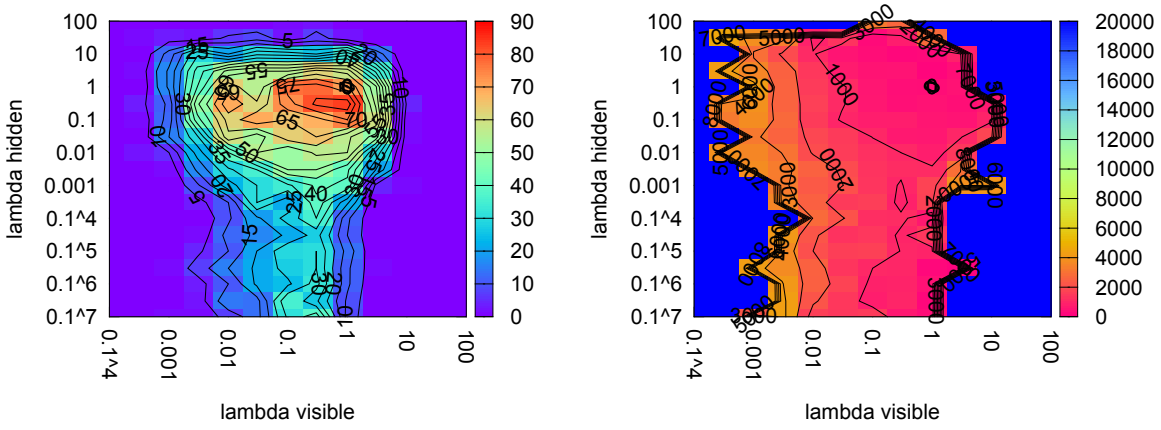

Figure 18: GeneRec (1.2.3) success rate and convergence time on the *4-2-4 encoder* task with $\sigma = 2.3$ and $\mu = 0.0$. Best result 83% achieved with $\lambda_h = 0.3$ and $\lambda_v = 1.0$.

3.1.7 Conclusion

The results in this chapter suggest a hypothesis why TLR outperforms BAL on the *4-2-4 encoder task*. The reason is that the hidden activations settle before W^{HO} and

W^{HI} adapt to them. This is explained by the fact that forward and backward hidden activations become same to fast (3.1.3, 2.4.3) and moreover, weight initialization could help this (9). The first issue is solved by setting $\lambda_h \ll \lambda_h$ what adds epochs to the training phase. The second issue is solved by candidate selection which prevents initializing hidden activations close to each other.

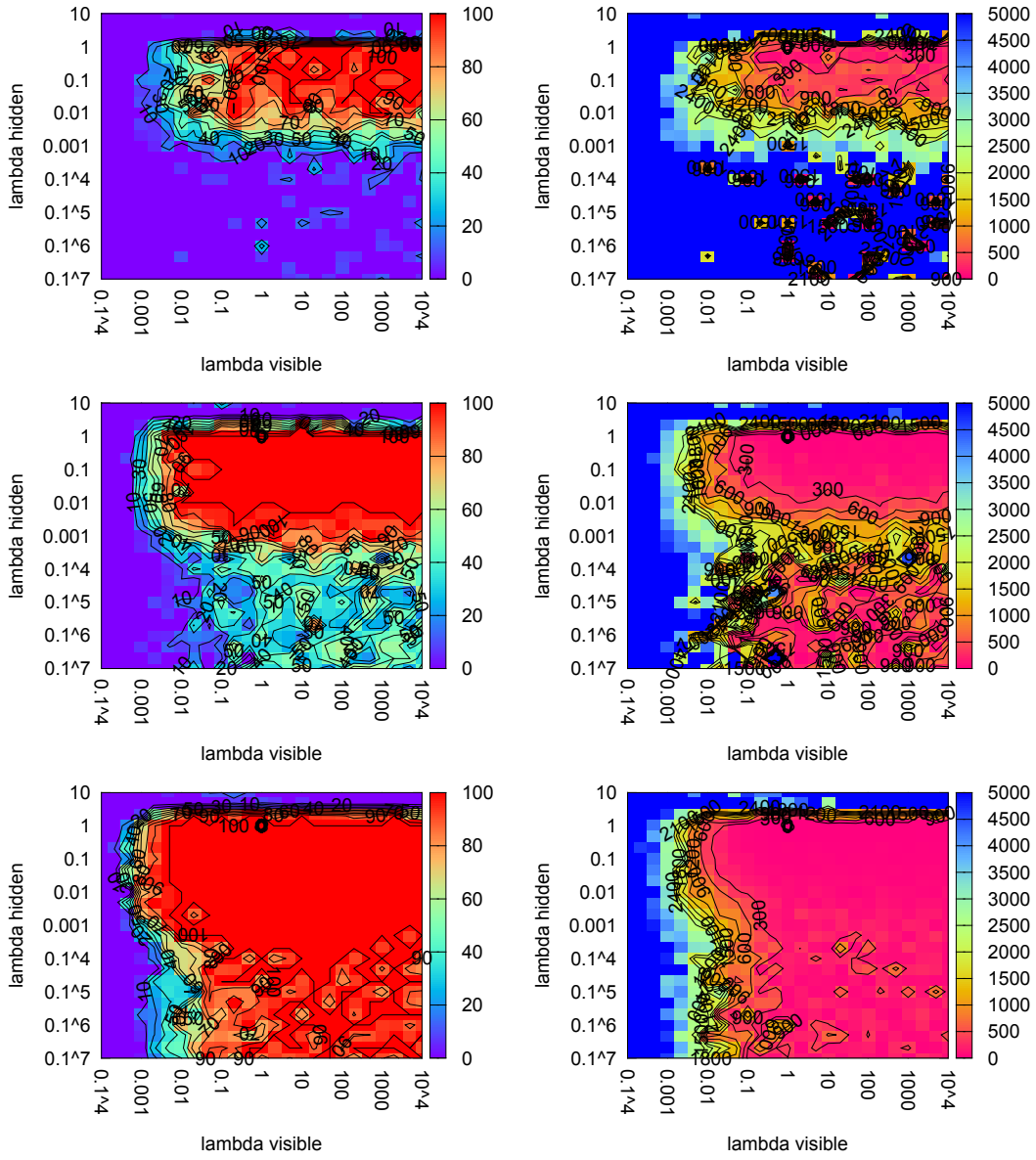

Figure 19: TLR performance on the *CBVA* task with hidden sizes 3 on top, 5 in middle and 8 at bottom.

3.2 Complex binary vector associations

In this section, we analyse TLR (2.3.1) and GeneRec (1.2.3) performance on the *CBVA* task (2.2.2). The methodology is similar to the case of the 4-2-4 encoder task described in Section 3.1. The main difference is that we tried several architectures of type 16-n-16 for $n \in \{3, 4, \cdots, 10\}$. Second difference is the setting of $Epoch_{\max}$ to 20,000 for TLR and 5,000 for GeneRec because the bigger network size.

3.2.1 Two learning rates

Comparison of success rates for different hidden sizes and (λ_v, λ_h) pairs is shown in Figure 19. We see that TLR was able to learn the CBVA task only with 3 hidden neurons. The successfull values of (λ_v, λ_h) lie on the half line $[(1, 0.1), (10^4, 0.1)]$. This behaviour of TLR is similar to the behaviour of TLR on the *4-2-4 encoder* task (17). Again, the performance mostly depends on λ_h and it is bounded by a value of λ_v. We observe that the perfect success space smoothly expands from top to down as increasing the hidden size. This is not true for GeneRec as shown in Figure 21.

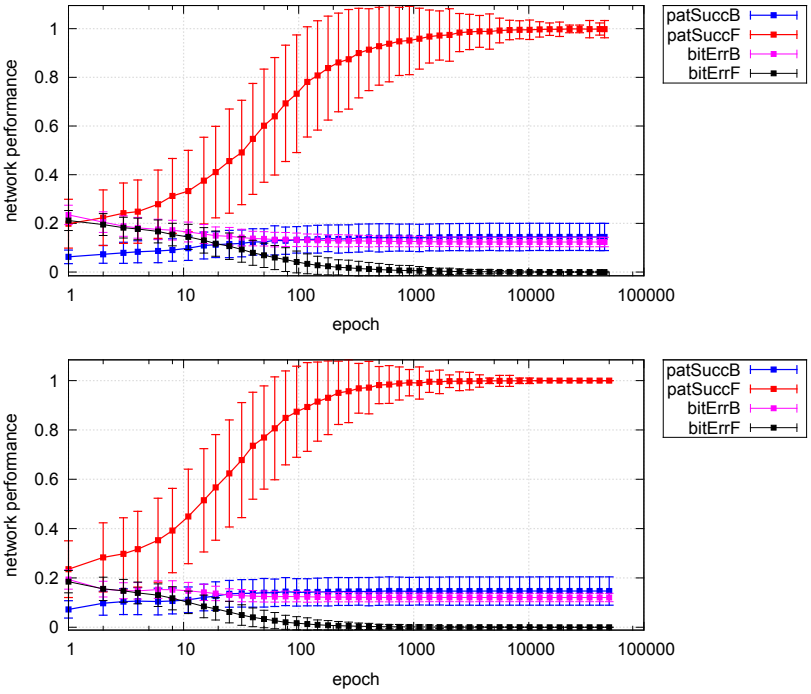

Figure 20: TLR success rate timeline for the *CBVA* task with $\lambda_h = 0.1$ and $\lambda_v = 100$. Without candidate selection on top and with candidate selection at bottom.

Next we analysed the performance of TLR for best λ_h and λ_v in Figure 9. We can observe that TLR is increasing its success rate in time steadily. Moreover, we see that candidate selection (2.4.2) has positive impact on the network performance. Note that the CBVA task is non bijective, i.e. one output has several inputs, therefore it is impossible to achieve 100% $patSucc^B$ or $bitSucc^B$.

3.2.2 Comparison

Although CBVA is a higher dimensional task than the 4-2-4 encoder task, the networks were able to achieve perfect success rate and comparable convergence time as shown in Table 8. The interesting fact is that having two learning rates has no advantage over setting $\lambda_h = \lambda_v$. This disproves our intuition that a more general model should achieve better.

Algorithm (section)	λ_h	λ_v	$patSucc^F$	Epochs
GR (1.2.3)	0.1	0.1	100	84
GR TLR	0.03	1.0	100	89
BAL (1.3)	0.5	0.5	100	54
BAL TLR (2.3.1)	1.0	5.0	100	64

Table 8: Comparison of TLR and GeneRec on *CBVA* using 3 hidden neurons.

3.2.3 GeneRec

We used the two learning rate approach also for GeneRec. In figure 19 we compare success rate for different values of (λ_v, λ_h) and different hidden sizes. We can observe a similar behaviour for GeneRec with the 16-3-16 architecture as for the 4-2-4 encoder task (18). That is the success space is bounded and the importance of both learning rates is same. Furthermore, as the hidden size becomes bigger, the success rate expands downwards to lower λ_h values as with TLR in Figure 19.

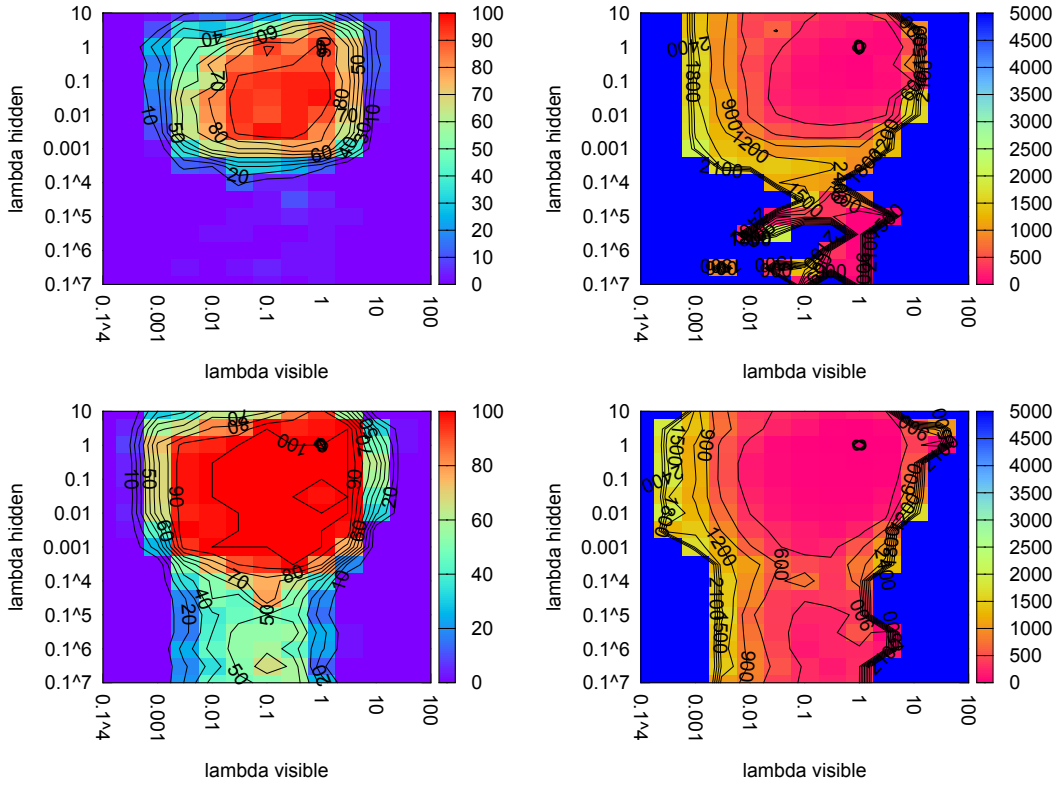

Figure 21: *GeneRec* success rate and convergence time on the *CBVA* task with hidden sizes 3 on top and 5 at bottom.

3.3 Handwritten digits

In this section, we wanted to test TLR on a higher dimensional graphical task and compare it to other known models. For this we chose the *handwritten digits* dataset (2.2.3). As in the previous simulations, we trained the networks for range of λ_v and λ_h values to find the best parameters. Then we analysed the network with thes best parameters individually.

Before the training we splitted the dataset to *train* set with 38,000 samples and *test* set with 4,000 samples. Then we trained the networks on the train set and evaluted them on the test set. The $Epoch_{\max}$ value was set to 20 and the training was stopped if $patSucc^F$ was not increased for 3 successive epochs. The network architecture 784–300–10 was chosen as results for BP with such architectures exists. Note that for the final classification we chose the unit with the maximal activation.

3.3.1 Two learning rates

We confirmed that TLR could learn high dimensional tasks as shown in Figure 22. The properties of the plot are similar to the 4-2-4 encoder case. It both holds that $\lambda_h \ll \lambda_v$ and the success space is smooth. The main difference is that the magnitude of values of both λ_h and λ_v is smaller. This notion indroduces a hypothesis that for *higher* dimensional tasks, which usually also have more samples, *lower* values of learning rates should be chosen.

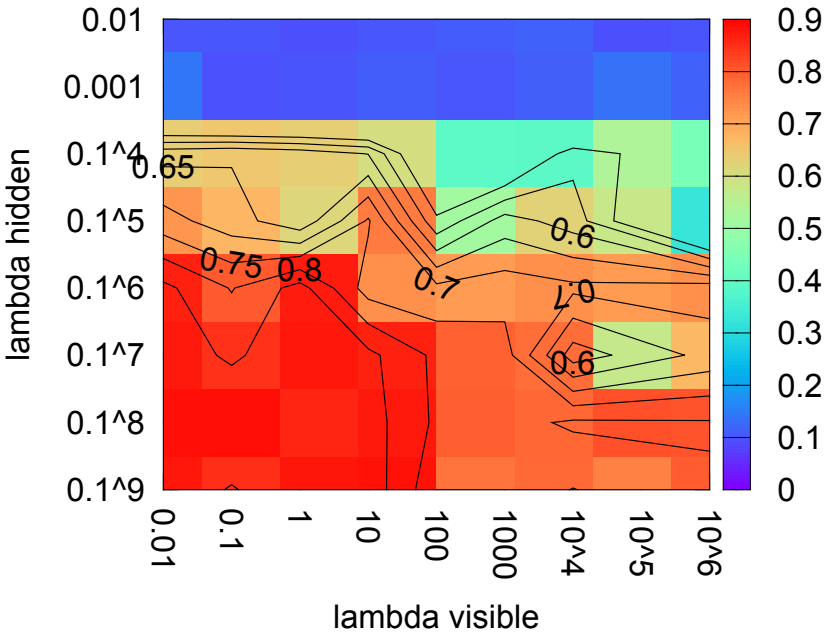

Figure 22: TLR performance on the *digits* task for $\sigma = 1/\sqrt{784+1} \approx 0.036$ and $\mu = 0.01$. Best $patSucc^F = 88.47\%$ with $\lambda_v = 0.1$ and $\lambda_h = 10^{-8}$.

3.3.2 Comparison

For comparison of TLR on the *digits* task we chose neural network models with similar architecture from LeCun et al. (1998b). In Table 9 we see that TLR is able to learn higher dimensional tasks, but still has a performance gap to fill. Note that it performs comparably as *linear classifier*, i.e. a two layer neural network.

Algorithm (section)	λ_h	λ_v	$patSucc^F$	Epochs
Linear classifier	–	–	88	–
BP 784–300–10 (1.1.5)	–	–	95.3	–
TLR 784–300–10 (1.1.5)	10^{-8}	0.1	88.47	20
GeneRec 784–300–50–10 (2.4.4)	0.03	0.03	43.22	50

Table 9: Comparison of different models on the *digits* task. Data from LeCun et al. (1998a) and LeCun et al. (1998b).

3.3.3 Backward representations

The goal of *backward representations* for *heteroassociative* tasks, i.e. tasks which have multiple inputs for the same output, is to *depict* the backward activation of an particular output. As the mapping is not bijective, we expect the backward image to be a blend of observed inputs.

As we can see in Figure 23, TLR gives us readable backward activations. This intuitively proves that the model is capable of bidirectional training. The shapes could be best seen for digits "0", "1" and "8". By using some imagination we can also see the other digits.

Figure 23: Backward representations for the most successful TLR instance on the *digits* task.

Conclusion

In our work, we proposed and analysed the Two learning rates (TLR) model, a modification of BAL, which increased the success rate from 62.7% to 93.1% on the 4-2-4 encoder task. We observed that BAL converges rapidly to the state, when the backward and forward activations converge to the same values. This has inspired our primary hypothesis to explain why BAL had problems learning the 4-2-4 encoder task. Our hypothesis was further confirmed by candidate selection, which selected the network with more distant hidden activations. This increased the success rate from 93.1% to 99.84% and reduced the number of epochs needed for convergence from 5845 to 150. Then we applied TLR on the handwritten digit recognition task using the architecture 784–300–10. Although TLR still has a performance gap compared to backpropagation, it achieved far better success rate than original BAL.

We experimented with many different modifications of BAL, notably BAL-recirc, other GeneRec learning rules, dropout and multi-layer GeneRec, but these did not prove to be useful. Standard modifications, such as batch training mode or adding momentum, showed no tendency in increasing the success rate of TLR. We admit that there is a space for improvement on these approaches.

Our work opened several ways to continue the analysis of BAL. For instance, it would be possible to predict success rate from the initial weights, as discussed in Section 2.4.4. Another option is to use four learning rates, one for each matrix, or dynamic learning rates for each connection as outlined in Section 2.4.4. Furthermore, we recommend analysing backward activations which showed counterintuitive behaviour in Figure 9. Finally, TLR introduced one more parameter to the network setup and therefore a method for finding best pair learning rates would be usefull.

Bibliography

Behera, L., Kumar, S., and Patnaik, A. (2006). On adaptive learning rate that guarantees convergence in feedforward networks. *Transactions on Neural Networks, IEEE*, 17(5):1116–1125.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, Springer*, 45(1):41–51.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems. Springer*, 2(4):303–314.

da Silva, A. B. and Rosa, J. L. G. (2011). Advances on criteria for biological plausibility in artificial neural networks: think of learning processes. In *IJCNN. International Joint Conference on Neural Networks*, pages 1394–1401. IEEE.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, Wiley Online Library*, 14(2):179–211.

Farkaš, I. and Rebrová, K. (2013). Bidirectional activation-based neural network learning algorithm. In *Artificial Neural Networks and Machine Learning (ICANN)*, pages 154–161. Springer.

Grossberg, S. (1978). A theory of visual coding, memory, and development. *Formal Theories of Visual Perception, Wiley*, pages 7–26.

Haykin, S. (1994). *Neural Networks: a Comprehensive Foundation.* Prentice Hall.

Hinton, G. (1989). Deterministic boltzmann learning performs steepest descent in weight-space. *Neural Computation, MIT Press*, 1(1):143–150.

Hinton, G. and McClelland, J. (1988). Learning representations by recirculation. In *Neural Information Processing Systems*, pages 358–366. American Institute of Physics.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks, Elsevier*, 1(4):295–307.

Kirkpatrick, S., Vecchi, M., et al. (1983). Optimization by simmulated annealing. *Science*, 220(4598):671–680.

LeCun, Y. A., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer.

LeCun, Y. A., Corinna, C., and Christopher, J. C. B. (1998b). The MNIST Database of Handwritten Digits. `http://yann.lecun.com/exdb/mnist/`. [Online; accessed 28-April-2014].

Magoulas, G. D., Vrahatis, M. N., and Androulakis, G. S. (1999). Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Computation, MIT Press*, 11(7):1769–1796.

McClelland, J. L. and Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: I. an account of basic findings. *Psychological Review, American Psychological Association*, 88(5):375.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology, Springer*, 5(4):115–133.

Miniani, A. and Williams, R. D. (1990). Acceleration of back-propagation through learning rate and momentum adaptation. In *Proceedings of International Joint Conference on Neural Networks*, volume 1, pages 676–679.

Movellan, J. (1990). Contrastive hebbian learning in the continuous hopfield model. In *Proceedings of the Connectionist Models Summer School*, pages 10–17.

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

O'Reilly, R. C. (1998). Six principles for biologically based computational models of cortical cognition. *Trends in Cognitive Sciences, Elsevier*, 2(11):455–462.

O'Reilly, R. C. (2001). Generalization in interactive networks: The benefits of inhibitory competition and hebbian learning. *Neural Computation, MIT Press*, 13(6):1199–1241.

Orrú, T., Rosa, J. L. G., and Andrade Netto, M. (2008). Sabio: A biologically plausible connectionist approach to automatic text summarization. *Applied Artificial Intelligence, Taylor & Francis*, 22(9):896–920.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation, MIT Press*, 1(2):263–269.

Phansalkar, V. and Sastry, P. (1994). Analysis of the back-propagation algorithm with momentum. *Transactions on Neural Networks, IEEE*, 5(3):505–506.

Pineda, F. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters, APS*, 59(19):2229–2232.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *International Conference Neural Networks.*, pages 586–591. IEEE.

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

Rumelhart, D., Hintont, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Schneider, M. O. and Rosa, J. L. G. (2009). Application and development of biologically plausible neural networks in a multiagent artificial life system. *Neural Computing and Applications, Springer*, 18(1):65–75.

Weir, M. K. (1991). A method for self-determination of adaptive learning rates in back propagation. *Neural Networks, Elsevier*, 4(3):371–379.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation, MIT Press*, 1(2):270–280.

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

Yu, C.-C. and Liu, B.-D. (2002). A backpropagation algorithm with adaptive learning rate and momentum coefficient. In *IJCNN. Proceedings of the International Joint Conference on Neural Networks.*, volume 2, pages 1218–1223. IEEE.

Yu, X.-H. and Chen, G.-A. (1997). Efficient backpropagation learning using optimal learning rate and momentum. *Neural Networks, Elsevier*, 10(3):517–527.