

Generalization by symbolic abstraction in cascaded recurrent networks

Mikael Bodén^a

^a*School of Information Technology and Electrical Engineering,
University of Queensland, QLD 4072, Australia.
mikael@itee.uq.edu.au.*

Abstract

Generalization performance in recurrent neural networks is enhanced by cascading several networks. By discretizing abstractions induced in one network, other networks can operate on a coarse symbolic level with increased performance on sparse and structural prediction tasks. The level of systematicity exhibited by the cascade of recurrent networks is assessed on the basis of three language domains.

Key words: Recurrent neural network, language, generalization, systematicity.

1 Introduction

The combinatorial nature of language has spawned a search for models with an appropriate learning bias operating on the elements of language. Specifically, recurrent neural networks have delivered evidence that models with realistic generalization properties can be adapted from simple language corpora [7,4,18]. Single-step prediction learning is a learning strategy that has proven particularly useful for producing generalizations according to contextual and functional similarities of untagged language data. Given a sequence of words presented to the network, the network essentially learns to output the probabilities of words occurring next but in a manner quite distinct from conventional n -gram modelling: by concurrently developing *abstractions* and *dynamics* for processing linguistic structure in its state space.

From a generalization point of view, one problem is that realistic language exposure is sparse. Human language users are found to exhibit extraordinary capacity to cope with learning situations which are statistically weak. As pointed out by Hadley [10], Phillips [15] and Marcus [12,13], generalization in humans sometimes goes beyond the regularities inferred with recurrent networks. As

an example, if you have heard “Smith feedled Jones” you would infer that it is grammatical to say “Smith feedled Belanger” even though you have never seen “Belanger” as the object in that context [13]. In error based learning the likelihood that Belanger would be predicted (in that position) by the network is decreased every time another word occurs instead. If Belanger never occurs – in that particular training context – the probability eventually goes down to zero [13] (cf. [19]). It has been argued that neural networks do not exhibit sufficient “systematicity” – an ability to deal with certain types of structural inferences as exemplified above [8,10].

Inducing means in recurrent networks for processing structurally complex languages (e.g. context-free and context-sensitive) is indeed possible but in many cases difficult and fragile [17,2,9]. Available demonstrations of learning structurally complex languages are limited in at least two senses.

- (1) Only a few terminal symbols are employed. It is unclear if large-scale languages can be used for induction.
- (2) The dynamics, which indicates that the network is actually going beyond regular language processing, is associated with individual terminals rather than word classes. It is unclear if the network will extrapolate, i.e. utilize the same principles for processing other related symbols possibly appearing at new levels of structural embedding.

The availability of grammatical and semantical abstractions, embodied as internal activation clusters (groups) in recurrent networks trained on language corpora, led us to investigate how discretization of such abstractions could improve learning the dynamics previously only detected for individual words. The idea is supported by observed performance improvements of classifying continuous features by discretization prior to machine induction [5].

In this paper we show that generalization performance in recurrent neural networks is enhanced by cascading several networks. By discretizing abstractions induced in one network, other networks can operate on a coarse symbolic level with increased performance on sparse and structural prediction tasks. The neural network architecture with associated learning mechanisms is described. The level of systematicity exhibited by the cascade of recurrent networks is assessed on three language domains. Considerable improvement is observed for two of the domains.

2 Structure and abstraction

Assume a learner faces examples presented sequentially and in random order from the simple context-free language $a^l b^l$ where $1 \leq l \leq 3$: $aabbabaaabbbbaabba\dots$

If we collect statistics for combinations of six consecutive letters (a 6-gram), a prediction task can be carried out to perfection. There would be 25 examples in such a table so to estimate the probabilities used for generating the strings would not take long.

A tabular approach like the 6-gram would not be inclined to process strings generated by $a^l b^l$ where $1 \leq l \leq 4$. With a window of six symbols, $aaaabbb$ would be matched with $aaabbb \mid a$. Even if l was allowed to take any value x while collecting statistics, and sufficient storage for storing $2x$ letter entries is assumed, the statistical learner would not generalize to process $a^{x+1}b^{x+1}$, or any l beyond x . Natural language contains many examples where embedding occurs. The inability to extrapolate beyond training data begs for a mechanism based on structure sensitivity and recursion (cf. [3]).

Consider “The mouse that the cat chases escapes”. Similar to $a^l b^l$, the sentence is center-embedded. Not counting the words “the” and “that” the sentence has two *nouns* (a) and two *verbs* (b). Natural language makes use of a large number of components and, consequently, a learner must abstract to cope. We extend $a^l b^l$ by having two different a s, and two different b s: $\{a_1, a_2\}^l \{b_1, b_2\}^l$ (a grammar which we refer to as a diversified center embedded language, DCEL). The statistically driven n -gram learner now needs to store $25 \cdot 2^6 = 1600$ entries in its table. More data is required to approximate the probabilities. If there are three variants of each a and b (3-DCEL), 18225 entries are required. If 25 variants of each a and b is used (25-DCEL), 6,103,515,625 entries are needed¹ and massive data sets are required to provide reasonable estimates of probabilities. Hidden Markov Models (HMMs) can improve on tabular representation. By attaching transition probabilities to states (either observed or hidden) the amount of memory required to represent the statistics is significantly reduced (assuming the right abstractions are found). The catch is that HMMs are still finite and are consequently not able to generalize beyond the level of embedding available in the training data.

Recurrent networks have been successful at learning small language fragments. As exemplified above there are two aspects to generalization of interest.

- (1) Abstraction. Inputs – in combination with contexts – that are associated with similar outputs are organized internally by learning processes into groups so that they impose similar influence on the network. The network is thus able to generalize from an item to other items within groups.
- (2) Dynamics for processing embedded structure. From a learning perspective there is a trade-off between available resources and the generality of mechanism. If state space is large and training set is small the network basically stores specific information about each sample. Otherwise, the

¹ “Don’t care” markers can reduce this number.

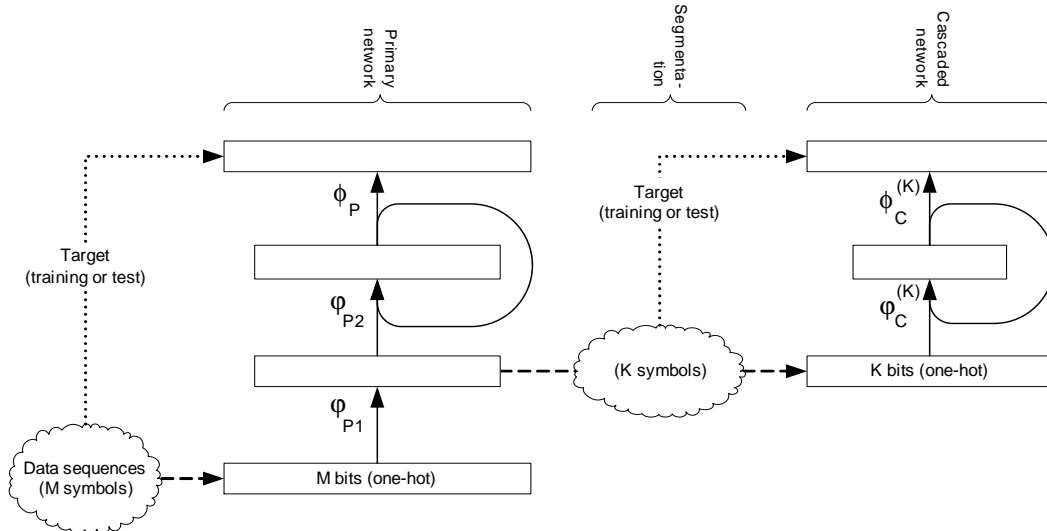


Fig. 1. The cascaded architecture has a minimum of three components: a primary network which is trained to predict the next input in sequences of M symbols at all times, a segmentation which produces one of K binary patterns from a hidden activation in the primary network, and a cascaded network which shadows the operation of the primary network but on basis of the patterns produced by the segmentation.

network is forced to seek other (more general) ways of representing the mechanism. A general mechanism may extrapolate beyond the limits of the training set. For example, in Long Short Term Memory (LSTM) networks and some Simple Recurrent Networks (SRNs [6]), linear counters are induced [9] for $a^l b^l$. For $a^l b^l$ some SRNs and Sequential Cascaded Networks (SCNs [16]) induce oscillating or spiralling dynamics [2,1].

3 Cascaded recurrent networks

The idea this paper presents is simple: the activation clusters found at a hidden layer of a prediction recurrent network can be segmented, discretized and be used as inputs and targets of another, cascaded, recurrent network, similarly trained to predict the next input. The basic architecture is shown in Figure 1.

Effectively, through continuous training, the cascaded network approximates a coarser probability distribution over a different set of discrete variables (e.g. “noun” and “verb” replace “mouse” and “escapes”). The new abstraction level is bounded by the cardinality of the segmentation mechanism. After processing at the abstract level, the variables can then be mapped back to its original patterns by simple association. At the expense of specificity, a higher degree of generality is supported by the system of networks.

Related work includes Nolfi and Tani [14] who used a cascaded setup to abstract from low-level events in a robotic environment. Schmidhuber and Prelinger [20] trained, on a variety of domains, an extra network to produce class representations (out of target data) which the main network is able to predict.

In the following, N is the number of symbols (or words) in the sequence that we use as a data set, M is the number of distinct symbols appearing in the sequence, and K is the number of distinct symbols that appear in the sequence after segmentation ($K \leq M$). At any level a symbol is encoded by a one-hot vector (a vector with one element set to 1 and the rest are 0).

The main recurrent network, called the *primary network*, has usually two hidden layers of which the last is recurrent and feeds back to itself with a discrete delay. Since inputs and outputs are encodings of symbols, M is also the number of inputs and outputs of the primary network.

The dynamic behavior of the primary network in response to an input vector $\vec{u}(n), n \in \{1, 2, \dots, N\}$ is described by a system of coupled equations.

$$\vec{x}_1(n) = \varphi_{P1}(\vec{u}(n), \vec{x}_1(n-1)) \quad (1)$$

$$\vec{y}(n) = \phi_P(\vec{x}_1(n)) \quad (2)$$

or

$$\vec{x}_1(n) = \varphi_{P1}(\vec{u}(n)) \quad (3)$$

$$\vec{x}_2(n) = \varphi_{P2}(\vec{x}_1(n), \vec{x}_2(n-1)) \quad (4)$$

$$\vec{y}(n) = \phi_P(\vec{x}_2(n)) \quad (5)$$

where φ_{P1} and φ_{P2} denote the activation functions of layer 1 and 2 of the primary network, ϕ_P denotes the activation function of the output layer. The complete flow of activation is illustrated in Figure 1.

If two hidden layers are used in the primary network (as in Figure 1) the first hidden pattern, $\vec{x}_1(n)$, is directly unaffected by the sequence context and only depends on the current input, $\vec{u}(n)$. $\vec{x}_1(n)$ is always subject to segmentation. The sequence context is only introduced at the last hidden layer. If only one hidden layer is used, however, the segmented patterns are blended by the input context.

The segmentation consists of identifying the closest prototype vector, \vec{w}_j , where $j \in \{1, 2, \dots, K\}$. There is one set of prototype vectors for a studied K (each randomly selected at the start of training). The prototype vectors are continuously adapted during training. The prototype vectors are optimally the means of K clusters. For each index, $h(\vec{x}_1) = j$, a bit vector is introduced.

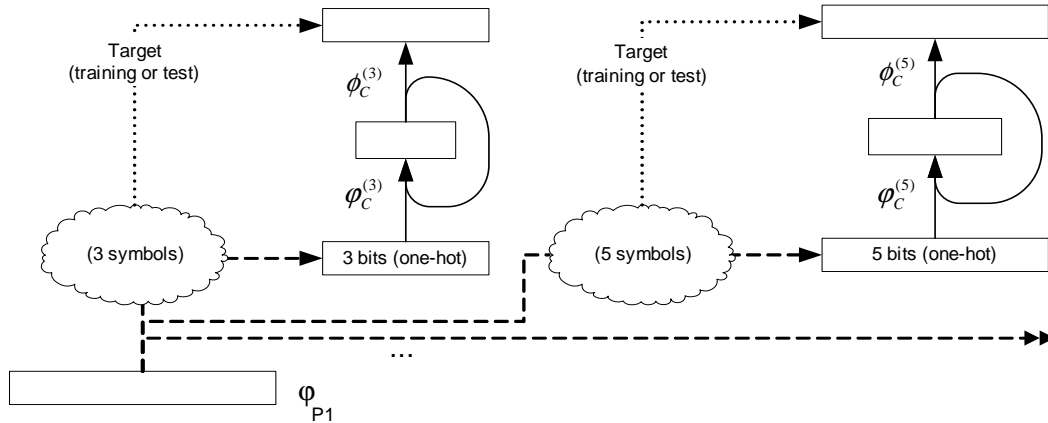


Fig. 2. Several cascaded networks (each with a separate segmentation) can be attached to the same primary network.

Each bit vector, $\vec{\Xi}$, has the j th element set to 1, the remaining $K - 1$ elements are set to 0.

$$h^{(K)}(\vec{\xi}) = \arg \min_j \| \vec{\xi} - \vec{w}_j^{(K)} \| \quad (6)$$

$$\Xi_i^{(K)}(n) = \begin{cases} 1 & \text{if } h^{(K)}(\vec{x}_1^{(K)}(n)) = i \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

The *cascaded network* is a recurrent network presented with the bit vectors determined by the segmentation. The cascaded network is tailored for a specific K value (indicated by the (K) superscript; see Figure 2 for an example where $K = 3$ and $K = 5$). We may study several different K values (levels of granularity) using the same primary network.

$$\vec{x}^{(K)}(n) = \varphi_C^{(K)}(\vec{\Xi}^{(K)}(n), \vec{x}^{(K)}(n-1)) \quad (8)$$

$$\vec{y}^{(K)}(n) = \phi_C^{(K)}(\vec{x}^{(K)}(n)) \quad (9)$$

where $\varphi_C^{(K)}$ and $\phi_C^{(K)}$ denote activation functions characterizing the layers in the cascaded network with K inputs and outputs. $\vec{y}^{(K)}(n)$ is the output vector of the cascaded network.

The output vector of the cascaded recurrent network can in many cases be evaluated separately (see Section 4.1). However, to compare the cascaded prediction directly with the prediction produced by the primary recurrent network a simple associative single-layered network, $\phi_A^{(K)}$, is trained concurrently with the cascaded network (see Sections 4.2 and 4.3).

$$\vec{\psi}^{(K)}(n) = \phi_A^{(K)}(\vec{y}^{(K)}(n)) \quad (10)$$

The target output of the associative network is the same as the target used for the primary network for each step n . The actual output of the primary network, $\vec{y}(n)$, can thus be compared with the actual output of the associative network, $\vec{\psi}^{(K)}(n)$, for each studied K , at each n .

The recurrent networks are trained using backpropagation through time to predict the next word in a sentence. There are three learning costs involved: C_P (prediction in the primary network), C_C (prediction in a cascaded network), and C_A (associating outputs from a cascaded network with the original bit patterns). All costs are minimized independently of one another.

In all but one simulation we interpret the outputs of the networks $\phi_P, \phi_C^{(K)}, \phi_A^{(K)}$ (for all tested K -values) as probabilities, use a log-likelihood cost function and the softmax output function. The costs are thus

$$C_P = \sum_n^N \sum_m^M u_m(n+1) \ln y_m(n) \quad (11)$$

$$C_C^{(K)} = \sum_n^N \sum_k^K \Xi_k^{(K)}(n+1) \ln y_k^{(K)}(n) \quad (12)$$

$$C_A^{(K)} = \sum_n^N \sum_m^M u_m(n+1) \ln \psi_m^{(K)}(n) \quad (13)$$

All other layers ($\varphi_{P1}, \varphi_{P2}, \varphi_C, \varphi_A$) are equipped with the logistic output function.

Training of all modules is concurrent. However, it was noted that a short delay before starting to minimize C_C and C_A is more efficient. Training inputs and targets of a cascaded network may change throughout training but do so more abruptly in the beginning of training.

The prototype vectors in the segmentation are adapted using K-means clustering. We tested competitive self-organizing maps with similar results. Hence, the specific technique by which prototype vectors are adapted should have little effect on the results obtained. Hypothetically, the segmentation can be implemented in various ways as long as clusters constitute the basis.

4 Simulations

Specific network architectures were designed and language data generated to address the question if the cascaded recurrent network (through its own learning task and its own independent dynamics) supports a wider range of generalizations – including abstraction and dynamics for structural extrapolation. For benchmarking, conventional recurrent networks (realized by the primary networks) with various number of hidden layers and various number of hidden units were studied.

All networks are trained to predict the next word in strings presented sequentially. The strings are generated according to grammars outlined below. Since each symbol in the sequence is represented as a one-hot code, input vectors are mutually orthogonal and share no similarities.

4.1 *An extremely sparse center embedded language*

In a center embedded language, the second half of each string is partly determined by the first half. Performance can thus be measured by the networks' ability to predict the next symbol (or group of symbols) at any time after the first half plus one symbol of the string has been presented.

A set of 300 strings from 25-DCEL with $1 \leq l \leq 10$ (which encompasses a total of $9.1 \cdot 10^{27}$ possible strings) were used for training and another 300 strings (non-overlapping but generated by the same grammar) were used for testing.² In addition, test strings which go beyond level 10 were used for assessing the networks ability to generalize in terms of structural properties.

Each network was trained for 100 epochs (rounds through the training set) and updated (and tested) after each completed string. Various learning rates were tested but significant differences were not observed. For the results reported we used a learning rate between 0.1 and 0.3. The error was backpropagated for 10 timesteps. Outputs need not be interpreted as probabilities and in preliminary trials the conventional summed squared error function performed better than the log-likelihood costs (Equations 11–13) and was selected for this simulation together with the logistic output function on all layers.

At each presentation the activation of the first hidden layer (\vec{x}_1 , preceding

² The distribution of the training set was skewed so that shorter strings were more frequent: $P(l = 1) = .2, P(l = 2) = .3, P(l = 3) = .2, P(l = 4) = .1, P(l = 5) = .06, P(l = 6) = .04, P(l = 7) = .03, P(l = 8) = .03, P(l = 9) = .02, P(l = 10) = .02$, where $2l$ is the length of the string.

Network	Mean #prediction errors		$Max(l)$
	$1 \leq l \leq 10$	$11 \leq l \leq 12$	
50/2R/50	16.2 (3.3)	6.0 (2.3)	–
50/2/2R/50	10.6 (1.3)	1.8 (0.4)	–
50/5/2R/50	10.0 (0.0)	1.8 (0.4)	–
50/5R/50	21.8 (3.7)	8.0 (1.0)	–
50/5/5R/50	11.4 (0.9)	2.4 (0.9)	–
Cascaded 2/2R/2	0.3 (0.5)	1.0 (1.4)	15.3

Table 1

The number of prediction errors made by network on the deterministic part of the string presented when full abstraction is assumed. Each configuration was rerun at least five times. The standard deviation is shown within parentheses. The final column shows the average maximum number of consecutively presented and correctly classified strings starting at $l = 1$ (all of the primary networks accumulated errors at levels below 10 and maximum level was not recorded). The networks are described using the number of units in each layer (input to output). R indicates that the layer is recurrent.

the recurrent layer) was segmented into two classes ($K = 2$, $\vec{\Xi}$ is set to either $\{0, 1\}$ or $\{1, 0\}$ depending on \vec{x}_1). In all networks, half-way through training, one discretized output was produced for a_1, a_2, \dots, a_{25} and another for b_1, b_2, \dots, b_{25} (see Figure 3). Thus, the primary network successfully abstracts the a -components and the b -components.

Training errors were not much lower than test errors. Hence, the network does not focus on specific strings or symbols. Instead, generalization seems to be based on groups of symbols as discovered through training.

Clusters as a basis for generalization is not sufficient to reliably address new levels of embedding. In Table 1 the numbers of prediction errors on test strings inside and outside the levels (of embedding) of the training set are presented. Primary networks are incapable of correctly predicting string endings even within the training set. The cascaded recurrent network, based on a segmentation into two classes ($K = 2$, optimal for full abstraction) and two recurrent nodes, is much more successful on extrapolating to test strings beyond the training set complexity. Due to the clear clustering effect in all the tested primary networks, the cascaded networks could have been trained on segmentations generated from any of them.

A key issue here is that the cascaded network is trained to produce outputs on the basis of inputs at a different granularity than in the original data set. This reduction in resolution turns out to have a marked effect on the ability to process deeper levels of embedding. Of 10 cascaded networks, seven

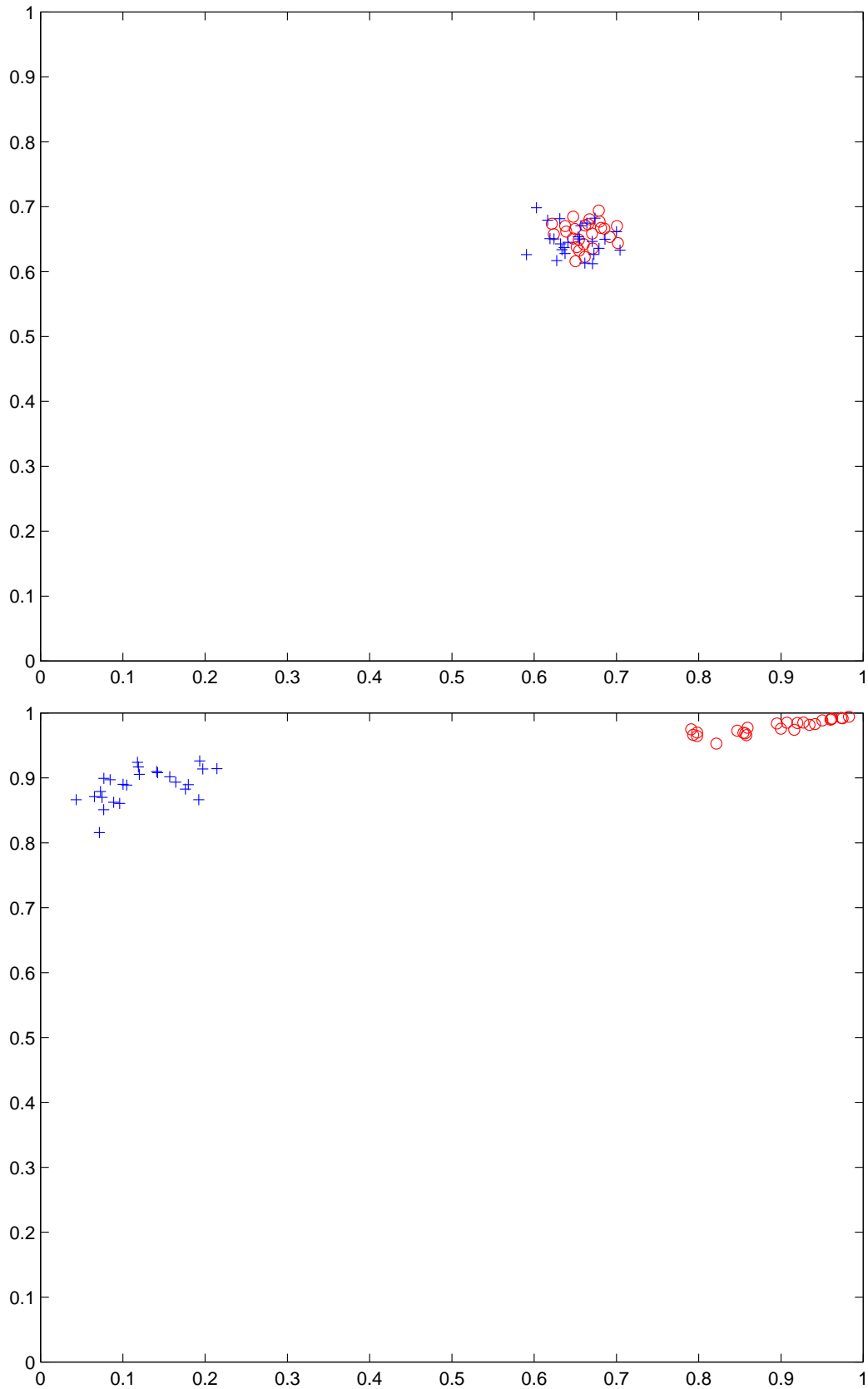


Fig. 3. The hidden activations at the first hidden layer of a recurrent network with two hidden layers (each with 2 units) after 20 epochs and 100 epochs of training. a_1, a_2, \dots, a_{25} are plotted as '+', b_1, b_2, \dots, b_{25} are plotted as 'o'.

managed to process strings to level 10 and beyond. The best network managed to correctly predict strings with $l \in 1, \dots, 27$. On average the cascaded network managed to process all strings up to $l = 15.3$ (see Table 1).

Qualitatively, the dynamics shows that the networks extrapolate. The two abstractions (the as and the bs) can be thought of as two autonomous dynamical systems (the input is fixed). For all of the networks that extrapolate, the first system oscillates towards an attractive point in a fractal fashion. When the second system starts (when the first b input is presented) the activation starts to diverge (by oscillation) from a repelling fixed point located in a separate part of the state space. The offset from the first fixed point determines the starting point for the second phase and indirectly determines how many oscillations that are required to reach the decision hyperplane which signals that the next input will be an a . The behavior of the systems can also be described by linearizing each dynamical system at the fixed points and calculating the corresponding eigenvalues and eigenvectors [17]. It turns out that all 7 successful networks share the following properties (illustrated as a hidden state trajectory in Figure 4).

- The largest absolute eigenvalues of the two systems are inversely proportional to one another. In practice one of them is usually around -0.7, the other around -1.4. The inverse proportionality ensures that the rate of contraction around the fixed point of the a system matches the rate of expansion around the fixed point of the b system [17].
- The fixed point of the a system lies on the axis given by the eigenvector corresponding to the smallest absolute eigenvalue of the b system (the direction in which the fixed point attracts) when projected through the second fixed point [17]. This configuration basically entails that the first thing that happens in the b system is a long-distance state shift along its eigenvector to a part in state space close to the b fixed point. The positioning of the eigenvector ensures that the final state of the a system (which identifies the a -count) correctly sets the starting point for the expansion phase.

4.2 A “gappy” context-free language

To further test the systematicity imposed by the cascaded system of networks another simulation was performed. A context-free grammar, here called L2, less homogenous than DCEL and resembling that of Elman [7], was used to train and test various architectures. The grammar (listed below as rewrite rules) mainly differs from Elman’s in that only verbs that require direct objects are allowed.

L2 produces sentences such as

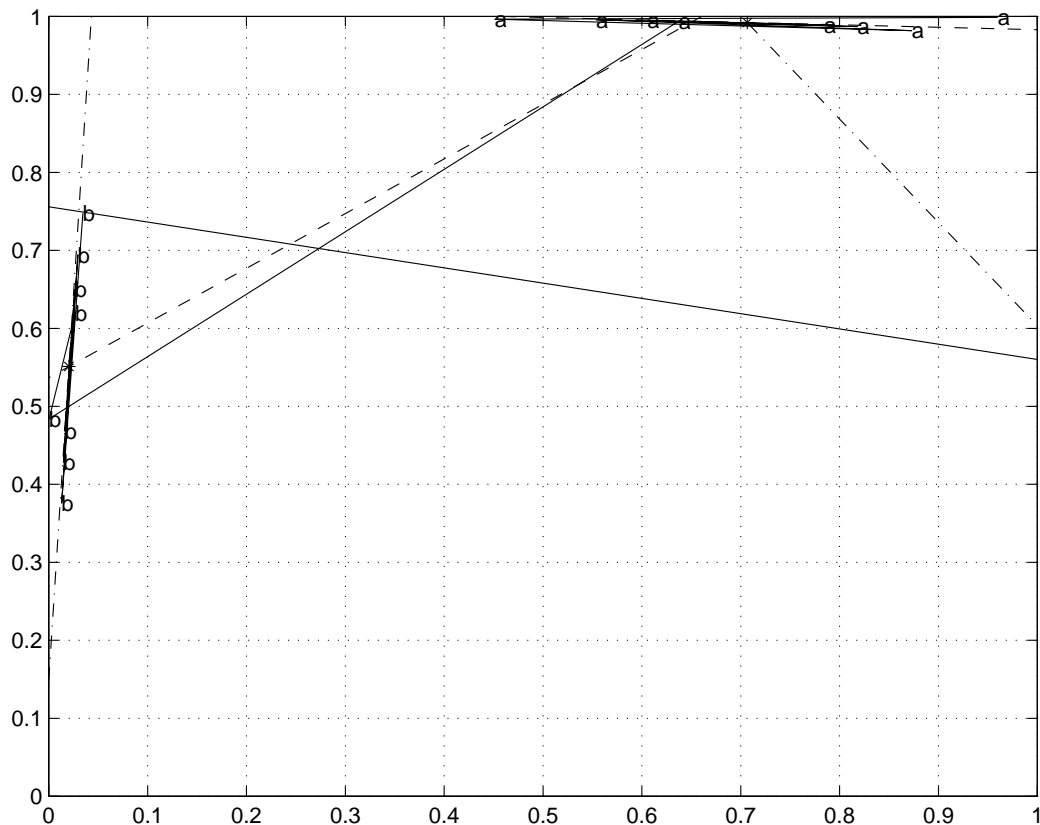


Fig. 4. A typical state trajectory of the cascaded network, $\vec{x}^{(2)}(1), \vec{x}^{(2)}(2), \dots, \vec{x}^{(2)}(16)$, when the string 'aaaaaaaabbbbbbb' is presented to the system of networks. The fixed points of the two systems are marked with '*'. The eigenvectors of the a- and b-system are plotted as dashed/dotted and dashed lines, respectively. Finally, the decision boundary implementing the prediction of 'a' is a solid line.

```

boy loves girl.
girls who Mary hates chase mice.
dog who cats who mouse hates chase loves John.

```

To achieve optimal prediction performance the model needs to find and exploit number agreement between nouns and verbs. Dependencies may stretch over considerable intervening material.

According to Hadley's definition of strong systematicity [10], the network should be able to generalize to a word in a novel position on a novel level of embedding. We ensured that the training data was non-exhaustive (gappy), the singular nouns "mouse" and "cat", and the plural nouns "dogs" and "boys" never appeared as subjects in the main clause. The test data, on the other hand, relaxed the constraint. Note that there is a separate item for punctuation. Whenever productions of the rewrite rules have an uneven probability of occurring, the specific probability is given within parenthesis.

$$\begin{aligned}
S &\rightarrow NP_{\text{sing}}, VP_{\text{sing}}, \cdot | NP_{\text{plur}}, VP_{\text{plur}}, \cdot \\
NP_{\text{sing}} &\rightarrow \text{PropN}(0.23) | N_{\text{sing}}(0.42) | N_{\text{sing}}, \text{RC}(0.35) \\
NP_{\text{plur}} &\rightarrow N_{\text{plur}}(0.65) | N_{\text{plur}}, \text{RC}(0.35) \\
\text{PropN} &\rightarrow \text{Mary} | \text{John} \\
N_{\text{sing}} &\rightarrow \text{boy} | \text{girl} | \text{cat} | \text{dog} | \text{mouse} \\
N_{\text{plur}} &\rightarrow \text{boys} | \text{girls} | \text{cats} | \text{dogs} | \text{mice} \\
VP_{\text{sing}} &\rightarrow V_{\text{sing}}, NP_{\text{sing}} | V_{\text{sing}}, NP_{\text{plur}} \\
VP_{\text{plur}} &\rightarrow V_{\text{plur}}, NP_{\text{sing}} | V_{\text{plur}}, NP_{\text{plur}} \\
V_{\text{sing}} &\rightarrow \text{chases} | \text{feeds} | \text{loves} | \text{hates} \\
V_{\text{plur}} &\rightarrow \text{chase} | \text{feed} | \text{love} | \text{hate} \\
\text{RC} &\rightarrow \text{who}, NP_{\text{sing}}, V_{\text{sing}} | \text{who}, NP_{\text{plur}}, V_{\text{plur}}
\end{aligned}$$

This time the simulation tested the performance of the cascaded architecture by transforming the output of the cascaded recurrent network using an associative network (Equation 10). $\vec{\psi}^{(K)}(n)$ can be compared directly with the output of the primary network, $\vec{y}(n)$, $n \in \{1, 2, \dots, N\}$. The outputs of the networks are interpreted as probabilities of the next word. The network outputs are compared with the probabilities that can be estimated by from the training and test sets (frequencies of words occurring in specific grammatical contexts).

Since the associative network is repeatedly trained to map a cluster member to its “original” bit vector, the network optimally produces a probability distribution over the cluster members. The probabilities reflect the frequencies of members appearing in the training set rather than in a particular context. The final probability distribution produced by the associative network should be the same for all cluster members. Somewhat circularly, clusters only develop in the primary network with those items that actually share a substantial proportion of functional and contextual features – from the *primary* network’s point of view.

Approximately 3000 sentences were generated from L2 (the constrained, gappy version) for training. Another 3000 sentences were used as a test (unconstrained version).

The performance of the cascaded architecture was measured by probing the output of the primary network \vec{y} versus the transformed outputs of cascaded networks, $\vec{\psi}^{(K)}$, for a whole range of K -values. The output activations were computed using the softmax function and the Kullback-Leibler divergence was used to measure the difference between the probability distribution produced by the network and the ideal probability distribution computed from the training set, $\vec{trn}(n)$ (see Rohde and Plaut, [18], for an extended discussion

Configuration Primary: #hid Casc	Divergence error								
	$\vec{trn}(n) \parallel$	$\vec{tst}(n) \parallel$							
	$\vec{y}(n)$	$\vec{y}(n)$	$\vec{\psi}^{(4)}(n)$	$\vec{\psi}^{(5)}(n)$	$\vec{\psi}^{(6)}(n)$	$\vec{\psi}^{(7)}(n)$	$\vec{\psi}^{(8)}(n)$	$\vec{\psi}^{(9)}(n)$	$\vec{\psi}^{(10)}(n)$
26/2/2R/26:2	0.67	0.68	0.74	0.71	0.68	0.69	0.67	0.67	0.64
26/5/5R/26:3	0.58	0.61	0.70	0.65	0.64	0.68	0.65	0.63	0.61
26/10/10R/26:3	0.56	0.62	0.70	0.65	0.66	0.68	0.65	0.63	0.61
26/3/3R/26:3	0.60	0.62	0.68	0.65	0.62	0.63	0.65	0.62	0.62
26/10/5R/26:3	0.58	0.61	0.70	0.66	0.65	0.68	0.65	0.64	0.61
26/5/5R/26:5	0.58	0.61	0.70	0.65	0.64	0.68	0.65	0.63	0.61
26/10/10R/26:5	0.56	0.62	0.68	0.62	0.60	0.62	0.63	0.61	0.60
26/10/10R/26:10	0.56	0.62	0.67	0.63	0.60	0.63	0.63	0.61	0.60
26/5/10R/26:5	0.56	0.62	0.68	0.63	0.60	0.63	0.64	0.63	0.60
26/5R/26:5	0.56	0.63	0.76	0.67	0.75	0.65	0.60	0.60	0.60

Table 2

The Kullback-Leibler divergence for L2 averaged from 3 repeats of each configuration (different initial weights for each run, measured at epoch 60). Lower value indicates better performance.

on the use of Kullback-Leibler divergence for language prediction tasks). Similarly, the generalization was measured by comparing network outputs and the probability distribution computed from the test set, $\vec{tst}(n)$.

In total the system of networks was trained for 60 epochs. To reduce computing time and ensure reasonably stable clusters, the segmentation was started after 10 complete epochs. Consequently, the cascaded networks were trained during 50 epochs. The learning rate of all networks was set to 0.01. Weight updates were batched over 10 words. The results are presented in Table 2.

A couple of things are worth noting from the simulation. The performance of the primary network on the test set is fairly good compared to most cascaded variants. The only cascaded network that outperforms the primary network is when $K = 10$.

Performance is poor when $K = 4$ due to the primary network’s tendency to make a clear distinction between singular and plural nouns. As a consequence the nouns typically end up in two groups rather than in one (where one group contains another word class as well).

When $K = 5$ there is room for two separate noun groups. However, there is not room for two separate verb classes to enable the cascaded network to account for number agreement. By observing the discrete states when $K = 5$ the following grammar is a typical outcome of predicting L2. In principle, this grammar generates the training and test data for the cascaded network operating on five groups.

$$\begin{aligned}
S &\rightarrow NP, VP, . \\
NP &\rightarrow N_{\text{sing}}|N_{\text{sing}}, RC|N_{\text{plur}}|N_{\text{plur}}, RC \\
VP &\rightarrow V, NP \\
RC &\rightarrow \text{who}, NP, V
\end{aligned}$$

The cascaded network sometimes learns to process sentences in L2 by making use of semi-fractal dynamics resembling that obtained for DCEL. In Figure 5 a cascaded network operating on five groups employs an attractive pair of points which is approached by oscillation while presenting a noun/who pair.³ When the first verb is presented the state is changed and the network starts to oscillate away from a repelling point. When the final verb is presented the state crosses a decision boundary that predicts a noun. The state which results from the final noun signals a prediction of punctuation. The prediction precision arising from this dynamical scheme is by no means perfect. However, the dynamics illustrates the qualitative nature of processing embedded structure. Moreover, the dynamical characteristics – analogous to the well known processes for predicting $a^l b^l$ – partly explains why small-scale dynamics is successful in handling novel levels of embedding. Another noun/who presentation means that the starting point for the repelling oscillation adjusts to fit another verb before signalling the final noun.

4.3 Hadley’s test for systematicity

Hadley et al. [11] investigate degrees of systematicity exhibited by a Hebbian-competitive network. Hadley’s network is not recurrent but utilizes an internal pre-wired buffering system, resembling that of a tapped delay-line feed forward network, to handle short sequential constraints. The performance of the network was compared with a recurrent network with a single self-recurrent layer [6]. The benchmark consisted of predicting language data generated from the following grammar, here called L3. Similar to our previous simulation a number of cascaded architectures were tested, this time on L3 data.

$$\begin{aligned}
S &\rightarrow NP, V, NP, . \\
NP &\rightarrow N(0.70)|N, RC(0.15)|N, PP(0.15) \\
N &\rightarrow \text{women}| \text{girls}| \text{birds}| \text{bats}| \text{men}| \text{boys}| \text{chairs}| \text{baseballs}| \text{dogs}| \text{tables}|
\end{aligned}$$

³ Even though singular and plural nouns are encoded differently in the hidden space of the primary network, as a result of the primary networks learning process, both types are treated similarly after a short period of training in the cascaded network. Since verbs (singular and plural forms) are represented by a single bit pattern by the segmentation, the cascaded prediction task does not have to account for number agreement between nouns and verbs.

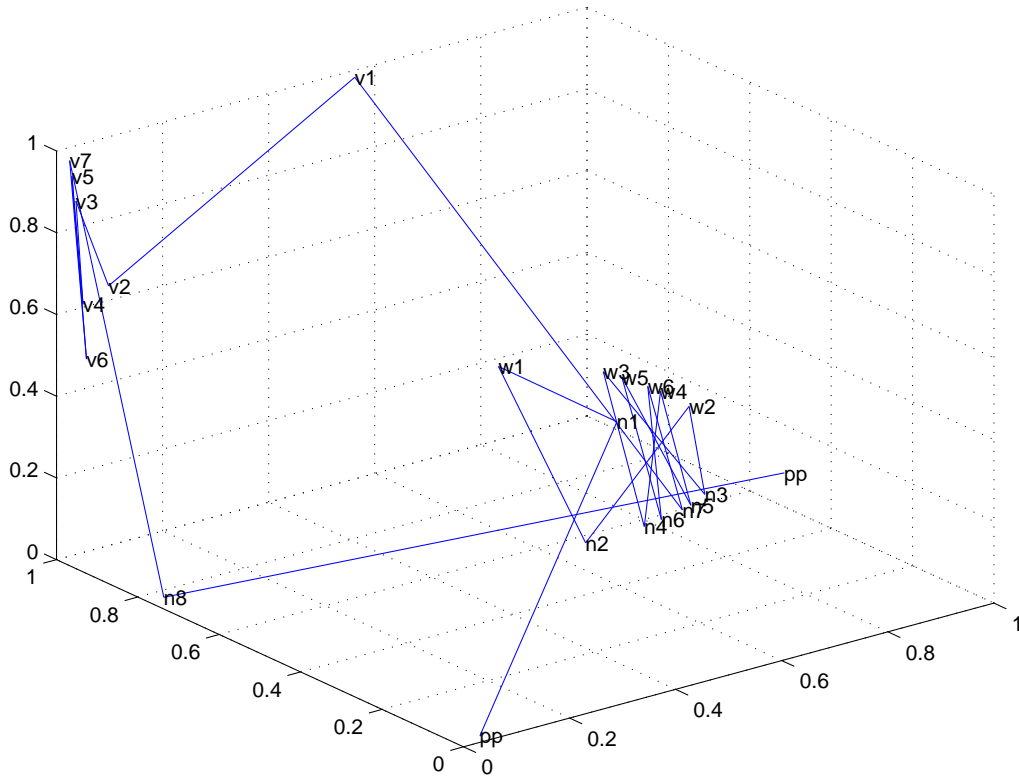


Fig. 5. The hidden activations of the fully trained cascaded network ($K=5$) when processing a sentence with the structure 'n1 who n2 who n3 who n4 who n5 who n6 who n7 v1 v2 v3 v4 v5 v6 v7 n8 pp' where 'n' corresponds to a noun and 'v' to a verb. 'pp' stands for punctuation and marks the end of a complete sentence.

```

cats|mice
V → chase|see|swing|love|avoid|follow|bump|hit|consume|dislike
RC → that, V, NP(0.40)|that, N, V(0.60)
PP → from, NP|with, NP

```

L3 is less complex than L2. For example, there is no number agreement and clauses are shallow (typically between zero and two levels of embedding).

In accordance with Hadley et al's simulation, a couple of constraints were added to the grammar. Four of the 12 nouns did not appear as subjects in the training set. A different set of four nouns did not appear as direct object during training. One of the remaining four nouns only appeared in conjunction with two specific verbs. These constraints were relaxed in the test set to see how well the network handled non-exhaustive exposure. Hadley's input vectors were manually set to reflect *semantic features* of the linguistic items. We do not follow this route, rather it is hypothesized that such similarities can be induced automatically through learning from the bit vectors. Hence,

our test is more challenging and not directly comparable to Hadley’s observations. Nevertheless, it should be noted that in Hadley et al’s simulations the best recurrent network generalized as well as their best Hebbian-competitive network.

In the simulation presented herein the frequencies of certain constructs appearing in the training and test sets are determined by probabilities attached to rewrite rules (see above). In the original simulation [11] specific ratios were provided for relative clauses and pre-positional phrases. Using the aforementioned probabilities attached to productions of rewrite rules we achieve comparable ratios but not identical.

Approximately 3000 sentences were generated from L3 (the constrained, gappy version) for training. Another 3000 sentences were used as a test (unconstrained version).

A general problem of neural networks is that they sometime overlearn, i.e. networks become too specific (weights grow large, become sensitive to specific data points, etc) after a lengthy period of training. Rohde and Plaut [19] observed that early in training, generalization on language data was more systematic. The systematicity gradually vanishes during continuous training. We observe similar tendencies in the L3 simulation. Only when the right abstractions are enforced the network avoids overlearning (see Figure 6).

The results below (see Table 3) contain the generalization performance at epoch 60 (epoch 50 for the cascaded networks). To illustrate the generalization ability during training we also include the best generalization result during the first 60 epochs. Hypothetically, by using validation data, training can be stopped at near-optimal points. The learning rate was 0.01 and the weights were updated every 10th word. Backpropagation through time unfolds the network for 3 timesteps. Again, a whole range of K -values were tested ($K \in 3, 4, 5, 6, 8, 10$).

Compared with the L2 simulation, from the L3 simulation it is clear that discretization of learned abstractions provides an overall advantage for the cascaded networks. The cascaded networks cope well with the increased number of gaps in the L3 training data. The divergence is typically smaller for the tested cascaded networks compared with the primary networks (see Table 3). According to the tendency that fewer segments (small K) cause smaller divergence, L3 seems to be handled best at a fairly coarse level of abstraction.

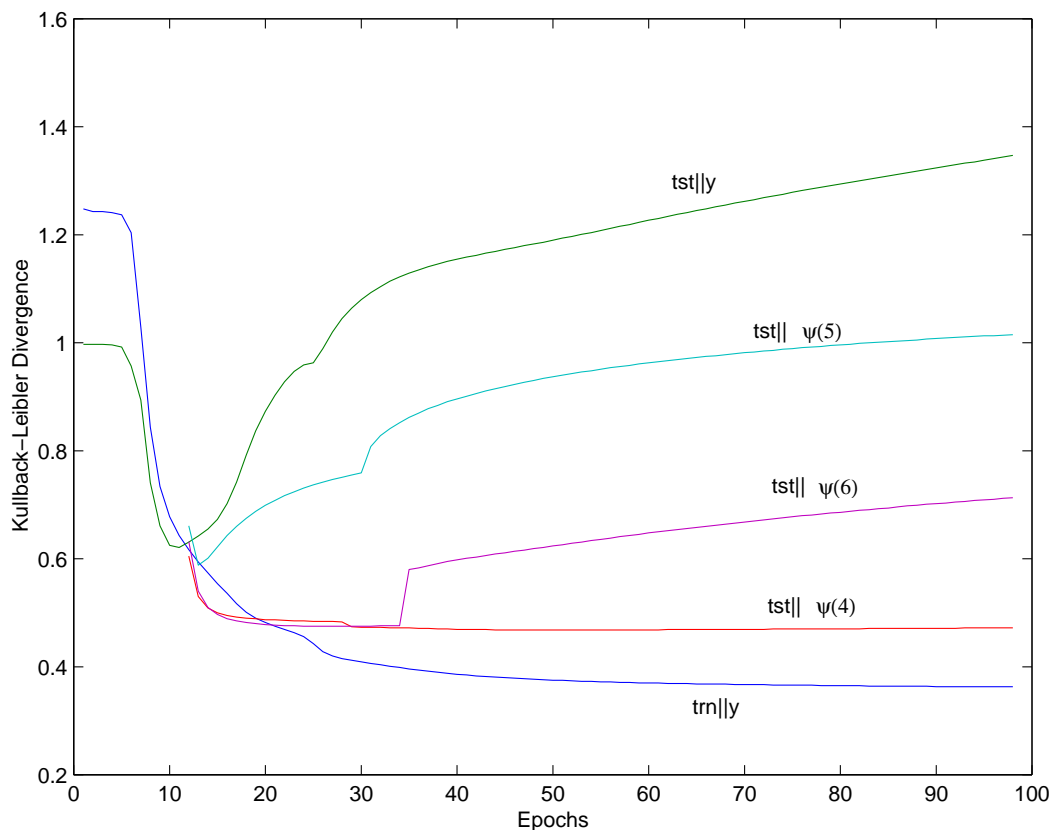


Fig. 6. A typical network’s divergence from the probabilities calculated from the training (trn) and test set (tst).

5 Discussion

The lowest training cost for a prediction task is attained if the network produces a probability distribution over all items reflecting the frequencies by which each item occurs in the specific context. Marcus has a point when he argues that a non-occurring item will not (in the limit) be predicted by a network that employs error-based learning. However, the network will internally encode similarities between items as they occur over the complete training set and not just as they appear in distinct contexts. By imposing severe processing bottlenecks, general similarities can carry over to more specific contexts and lead to some improvements as illustrated by Rohde and Plaut [19]. The approach proposed here relies on discretization as a means to bring up internal clustering effects to a “symbol surface.” By operating on the symbol surface, some subtleties are inevitably lost. However, less detail allows the cascaded network to focus on a more abstract level of processing. Firstly, fewer terminals mean simpler dynamics. Secondly, the frequencies of items appearing in general (i.e. in the training set) affect the probabilities produced by the cascaded networks for specific contexts. Two items appearing in the same group will be co-activated by the cascaded network in all contexts for which

Configuration	Divergence error								
	Primary:	$tr\vec{n}(n) \parallel$	$tst\vec{n}(n) \parallel$						
#hid Casc	$\vec{y}(n)$	$\vec{y}(n)$	$\vec{\psi}^{(3)}(n)$	$\vec{\psi}^{(4)}(n)$	$\vec{\psi}^{(5)}(n)$	$\vec{\psi}^{(6)}(n)$	$\vec{\psi}^{(8)}(n)$	$\vec{\psi}^{(10)}(n)$	
27/2/2R/27:2	0.48	0.87	0.54	0.79	0.76	0.91	0.91	1.00	
best	0.48	0.59	0.49	0.54	0.53	0.59	0.56	0.67	
27/3/3R/27:3	0.44	0.98	0.49	0.81	0.63	0.91	0.82	1.00	
best	0.44	0.60	0.48	0.58	0.51	0.58	0.57	0.66	
27/5/5R/27:2	0.37	1.26	0.53	0.63	0.81	0.74	0.82	0.98	
best	0.37	0.62	0.48	0.50	0.52	0.51	0.55	0.64	
27/5/5R/27:5	0.37	1.26	0.53	0.64	0.81	0.73	0.82	0.97	
best	0.37	0.62	0.48	0.51	0.53	0.52	0.59	0.64	
27/5/10R/27:3	0.36	1.37	0.66	0.95	0.68	0.79	0.85	0.94	
best	0.36	0.62	0.52	0.54	0.52	0.58	0.58	0.64	
27/10/10R/27:5	0.36	1.39	0.61	0.68	0.71	0.91	0.84	0.98	
best	0.36	0.63	0.56	0.54	0.58	0.63	0.63	0.69	
27/10/10R/27:10	0.36	1.39	0.61	0.68	0.71	0.90	0.84	0.98	
best	0.36	0.63	0.56	0.54	0.58	0.63	0.64	0.71	

Table 3

The Kullback-Leibler divergence for L3 averaged from 3 repeats of each configuration (different initial weights for each run). The divergence is measured after 60 rounds through the training set. The second row of each configuration provides the best/smallest divergence during training.

at least one of the items appears in. It is therefore likely that the cascaded network exhibits better generalization on symbolic and structural aspects of the prediction task, but not all. Specifically, we argue that the systematicity exhibited by the network is improved if the level of abstraction is carefully selected.

Speculatively, automatic and dynamic selection of the appropriate output can be realized through the use of a gate sensitive to the expertise offered by the primary and cascaded networks. Another resort is to impose evolutionary pressure on network architectures to reward higher degrees of systematicity.

6 Conclusion

In language processing there are at least two essential aspects to generalization: abstraction and structural extrapolation. Previous work has focused on one or the other.

We have shown that recurrent networks are well-suited to abstract symbols according to their similarities in a structurally homogeneous prediction task. However, the same networks are unable to reach the same level of performance on structural extrapolation as networks trained directly on abstract data. By feeding discretized state patterns to a second recurrent network we are able to associate the same dynamics previously observed only in small-scale experiments to a structurally similar but much larger data set.

Secondly, we have shown that the dynamics utilized by the network on very small and limited languages carry over to less homogeneous language tasks. We have also shown that overall generalization performance from sparse data can be improved by relying on a set of cascaded networks, each operating on a different level of abstraction. The output of a cascaded network can be projected back to the original input space.

Apart from scaling up to larger domains, the simulations indicate a resort for addressing the apparent systematicity of language in a completely unsupervised fashion. Hadley’s “strong systematicity” requires that a token should be appropriately processed in novel syntactic positions, at a novel level of embedding [10]. Admittedly, the tested domains are structurally homogeneous and have only a fraction of the complexity of natural language. However, the level of generalization that is achieved fulfills the requirements of strong systematicity. Hadley’s definition does not consider the structural complexity of the target grammar – a drawback which should be addressed since it may have an essential impact for learnability.

Acknowledgments

The author gratefully acknowledges the insightful comments made by two anonymous reviewers.

References

- [1] Mikael Bodén and Alan Blair. Learning the dynamics of embedded clauses. *Applied Intelligence: Special issue on natural language processing by neural*

networks, 19(1), 2003. In press.

- [2] Mikael Bodén and Janet Wiles. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, 12(3):197–210, 2000.
- [3] Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.
- [4] Morten Christiansen and Nick Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157–205, 1999.
- [5] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In Armand Prieditis and Stuart Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 194–202, San Francisco, CA, 1995. Morgan Kaufmann.
- [6] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- [7] Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.
- [8] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.
- [9] Felix A. Gers and Jürgen Schmidhuber. LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.
- [10] Robert F. Hadley. Systematicity in connectionist language learning. *Mind and Language*, 9(3):247–272, 1994.
- [11] Robert F. Hadley, Adam Rotaru-Varga, Dirk V. Arnold, and Vlad C. Cardei. Syntactic systematicity arising from semantic predictions in a hebbian-competitive network. *Connection Science*, 13(1):73–94, 2001.
- [12] G. F. Marcus, S. Vijayan, S. Bandi Rao, and P. M. Vishton. Rule learning by seven-month-old infants (1999), 283, 77–80. *Science*, 283:77–80, 1999.
- [13] Gary F. Marcus. Language acquisition in the absence of explicit negative evidence: can simple recurrent networks obviate the need for domain-specific learning devices? *Cognition*, 73:293–296, 1999.
- [14] Stefano Nolfi and Jun Tani. Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment. *Connection Science*, 11(2):125–148, 1999.
- [15] Steven Phillips. Constituent similarity and systematicity: the limits of first-order connectionism. *Connection Science*, 12(1):45–63, 2000.
- [16] Jordan B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227, 1991.

- [17] Paul Rodriguez, Janet Wiles, and Jeffrey L. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.
- [18] Douglas L. T. Rohde and David C. Plaut. Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72:67–109, 1999.
- [19] Douglas L. T. Rohde and David C. Plaut. Simple recurrent networks can distinguish non-occurring from ungrammatical sentences given appropriate task structure: reply to Marcus. *Cognition*, 73:297–300, 1999.
- [20] J. Schmidhuber and D. Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625–635, 1993.