

An FGREP Investigation into Phonotactics

Peter Kleiweg and John Nerbonne

Alfa-Informatica, Rijksuniversiteit Groningen

kleiweg@let.rug.nl, nerbonne@let.rug.nl

Abstract

We discuss experiments with neural networks being trained in a phonotactic processing task. A recurrent network not only learns to predict the next letter given a partial processed word, but also learns to represent the letters in a manner meaningful to the processing task. To this end, we use Miikkulainen's [1993] FGREP, augmented with an algorithm we call *dispersion*, to improve distinctness among the set of letter representations.

Our goal is to create a more realistic model of how humans might process natural language.

1 Related work

Cleeremans [1993] conducted an important series of experiments on sequence learning with neural networks. His Simple Recurrent Networks (SRNs) achieved perfect learning applied to the Reeber grammar, a formal language which is standardly used in machine learning. Tjong Kim Sang [1998] compared statistical, neural and symbolic approaches to computational models of language learning. He also studied sequence processing, but chose Dutch monosyllables as domain. Although he improved on Cleeremans' setup in several respects, his SRNs failed to learn Dutch phonotactics, and Tjong Kim Sang concluded that neural learning paradigms were not yet sophisticated enough to compete with others in this domain. Stoianov, Nerbonne & Bouma [1997] reported better performance, however.

We apply a neural network to phonotactic processing in the same way as Tjong Kim Sang [1998] and Stoianov et al. [1997] did. The network is presented with (monosyllabic) words, one letter at a time, and the network has to learn to predict the next letter. When training is completed, the network is tested, measuring how well it performs its task processing true words, compared to processing random strings.

Tjong Kim Sang [1998] and Stoianov et al. [1997] use local encodings, focusing on performance from a machine learning perspective. We use FGREP encoding, focusing on realistic modeling of language processing by humans.

2 FGREP

FGREP is short for *Forming Global Representations with Extended backPropagation* [Miikkulainen & Dyer 1991, Miikkulainen 1993]. It was introduced as a means of communication within a modular system of neural networks, and has been applied in several linguistic experiments.

FGREP is about data representation. Usually, a ‘flat’ neural network is trained to process a data set with fixed representations. With FGREP, the representations in the data set themselves are learned as well. This makes it possible to have neural networks learn to communicate, having the networks develop their language interface. The programmer doesn’t have to worry about what features have to be coded into the data set. FGREP codes for the features needed by the networks.

FGREP uses a global lexicon.¹ Initially, items are stored with randomized vector representations. Items are picked from this lexicon to serve both as input and target vectors to feedforward networks. Standard backpropagation is used to update the weights in the network, but it is extended back into the input vector, and this vector is updated as well. The updated vector is stored back into the lexicon.

3 Data sets

Our data set consists of monosyllabic words from the CELEX Lexical Database. Words were used in their normal spelling. As Tjong Kim Sang [1998] showed, this representation does not change the problem greatly, and it eases presentation. Words with one of the letters *q*, *x*, or *y* were removed (17 words). These three letters are so rare in Dutch that they could focus the attention in the FGREP representations too much, thereby blurring the details in the relations between the common letters. Also removed were foreign/loan words such as *bath*, *brunch* and *föhn* (232 words).

The remaining set of words was split randomly into a training set **T** of 3807 words, and a test set **P** of 424 words. In all experiments, the score distribution of both sets remained nearly identical, indicating the test set is a good representation of the training set.

Using frequency information from the CELEX Lexical Database, the training set was sorted, placing most frequently used words at the top of the list. During training, there was always a probability P that a word was selected from the top P^3 part of the list. Put in other words, when a word was selected for training, there was a 50% chance that it would be picked from the 12.5% ($= 0.5 \times 0.5 \times 0.5$) of words most frequent in the CELEX Database. For a set of 3807 words, this means that the word at the top of the list is used in training about 700 times as often as the word at the bottom of the list.

As ‘negative’ test data we created three sets. One set **R** consists of 800 random strings. It might be easy, even for a non-recurrent network, to distinguish this set from the training set, just because this random set contains ‘words’ with bigrams that are not in the training set. Therefore we created another set of ‘random words’, in which

1. To avoid confusion, we will reserve the term *lexicon* for the collection of FGREP vectors.

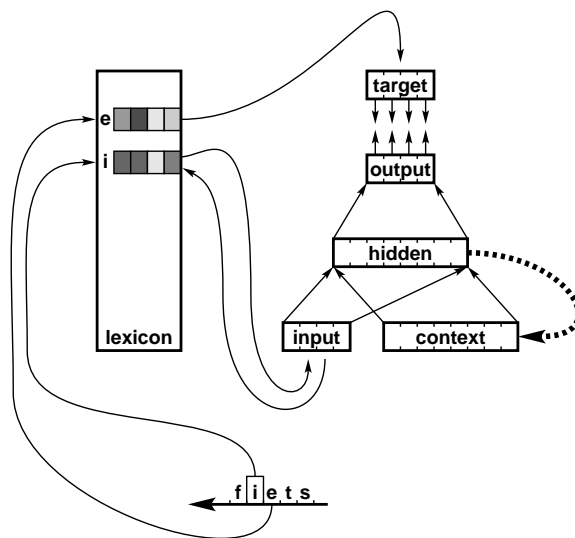


FIGURE 1: Layout of an FGREP network used in our experiments.

all words were made up of bigrams that are present in the training data. This set of 476 words was split into two sets. Set **B1** consists of 175 mono-syllabic words of which we thought they *might* exist in Dutch. Set **B0** consists of the remaining 292 words.

4 Network setup

Fig. 1 shows the layout of our network: we use FGREP in an Elman network [Elman 1990] capable of sequence processing. Before training, the lexicon is randomized with values close to the average output of the activation function (Eq. 1). Weights are randomized near zero.

Before a new word is presented to the network, the hidden units are reset to zero. Words are processed one letter at a time, using a copy of the hidden layer as additional input. The network has to predict the next letter, or the end-of-word symbol.

The activation function determines the output of a unit, given the sum of all inputs multiplied with the respective weights. Several activation functions were tested. The standard sigmoid did just fine (Eq. 1). An additional input unit, set to 1, served as a bias.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

Standard backpropagation [McClelland & Rumelhart 1988, Werbos 1995] was used to calculate error values and weight updates, but error values were calculated for the input layer as well. These error values, d_i , were used to update the input vector, using the update rule in Eq. 2 (η is the learning rate).

$$x_i = f(f^{-1}(x_i) + \eta d_i) \quad (2)$$

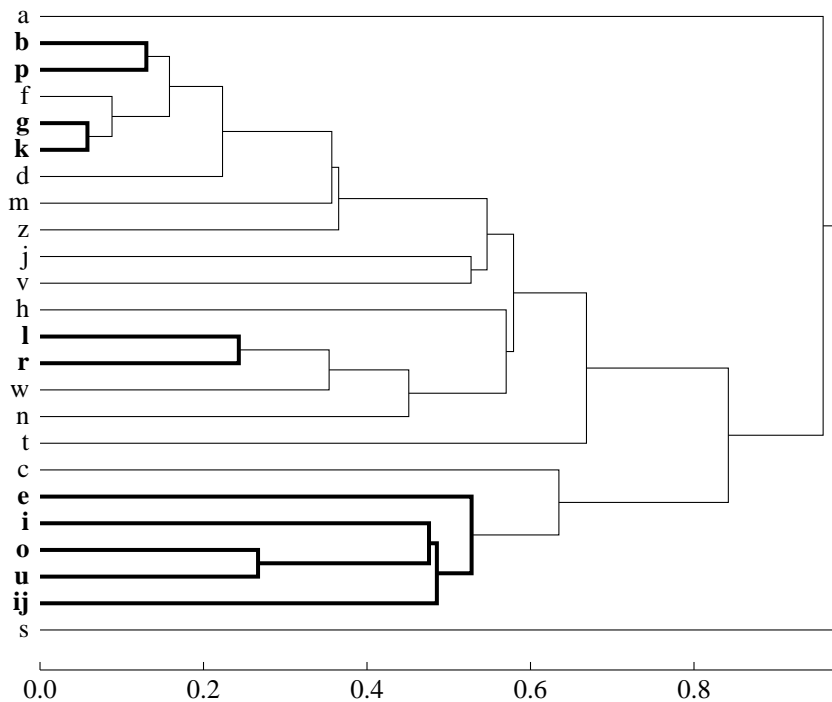


FIGURE 2: Clustering of FGREP data. Several interesting clusters are found, shown here in bold.

Using the inverse of the activation function $f()$, the network learns as if input comes from a preceding layer with a local encoding. Because $f()$ is a bitwise 1-to-1 mapping from the lexicon to the input, it is uniquely invertible.

5 First results

Several network sizes were tested, from 2 to 10 hidden units, and from 3 to 8 units for the FGREP vectors. The remaining part of this paper will discuss network setups with 6 hidden units, and an FGREP vector size of 7.

It was fairly easy to develop a ‘sensible’ set of vectors. Fig. 2 shows an example of a dendrogram, created with nearest neighbor clustering, using the Euclidean distance as a difference measure. (See Aldenderfer & Blashfield [1984] for an accessible introduction into data clustering.) Sensible clusters appear in the dendrogram, such as *b-p*, *g-k*, *l-r*, and one cluster of all but one vowel. Other clusters one might expect, such as perhaps *d-t*, are missing.

To see how well the network distinguishes valid from invalid words, we need a method to assign scores to words.

The method we use to assign a word score comes from Tjong Kim Sang & Nerbonne [1999]. The score of a letter c in the context of prefix s is symbolized by

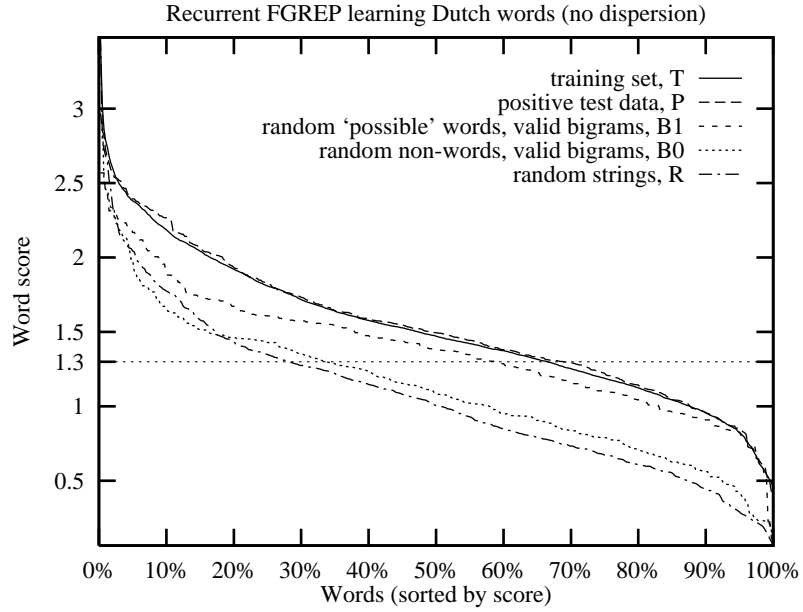


FIGURE 3: Poor network performance. A perfect network would show some word score level at which nearly all valid strings were accepted, and nearly no invalid ones.

$\|c|s\|$ and calculated according to Eq. 3.

$$\|c|s\| = 1 - \sqrt{\frac{1}{n} \sum_{i=1}^n (target_i(s) - output_i(s))^2} \quad (3)$$

The score of a word $\|w\|$, $w = c_1 \dots c_n$ is the product of the letter scores, Eq. 4.

$$\|w\| = \prod_i \|c_i|c_1 \dots c_{i-1}\| \quad (4)$$

We actually use a word score that is corrected using Eq. 5 and Eq. 6.

$$\text{new word score} = \text{old word score} * \text{factor}^{\text{length}} \quad (5)$$

$$\text{factor} = \frac{\text{average score word length 4}}{\text{average score word length 5}} \quad (6)$$

To get some idea of network performance, a score distribution of training and test sets were compared. Accepting strings as ‘valid words’ if they have a score of 1.3 or above, about 66% of the training set passes and 28% of the set of random strings, leaving a large margin of error. This network does very badly indeed when it comes to distinguishing valid from invalid words. Fig. 3 graphs performance at various score-levels for the five sets of data.

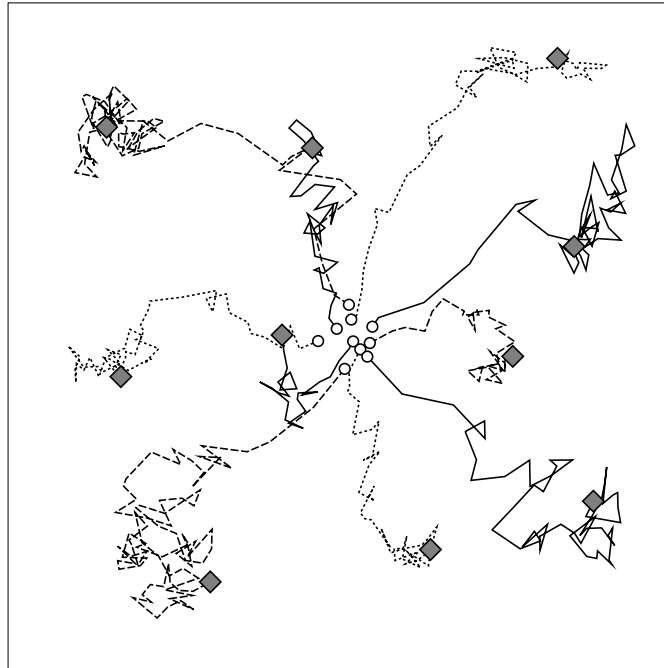


FIGURE 4: Dispersion. 10 vectors drift from their initial positions (open circles) to their final positions (filled diamonds). The vectors top and top-left have swapped places, but the other vectors have retained their relative order.

6 Dispersion

FGREP forces items with similar rôles to have similar representations. When these similarities are too close, it becomes hard for the network to distinguish between them. To overcome this problem, we introduce an algorithm for *dispersion*. The goal is to keep the vectors as different as possible, without destroying the order that is created by the FGREP algorithm. The algorithm is simple:

1. select a random point \mathbf{y} in vector space
2. pick the vector \mathbf{x} from the FGREP lexicon that is closest to \mathbf{y}
3. shift \mathbf{x} a tiny amount towards \mathbf{y} , using Eq. 7

$$x_i = x_i + \eta(y_i - x_i) \quad (7)$$

A learning rate $\eta = 0.00001$, applying the algorithm 5 times per trained word, does the trick. The algorithm is demonstrated for a set of 2D vectors in Fig. 4.

7 Results

Using dispersion, much improved performance was obtained, as can be seen in Fig. 5. Setting the score threshold to 0.96, about 87% of the training data is accepted as valid,

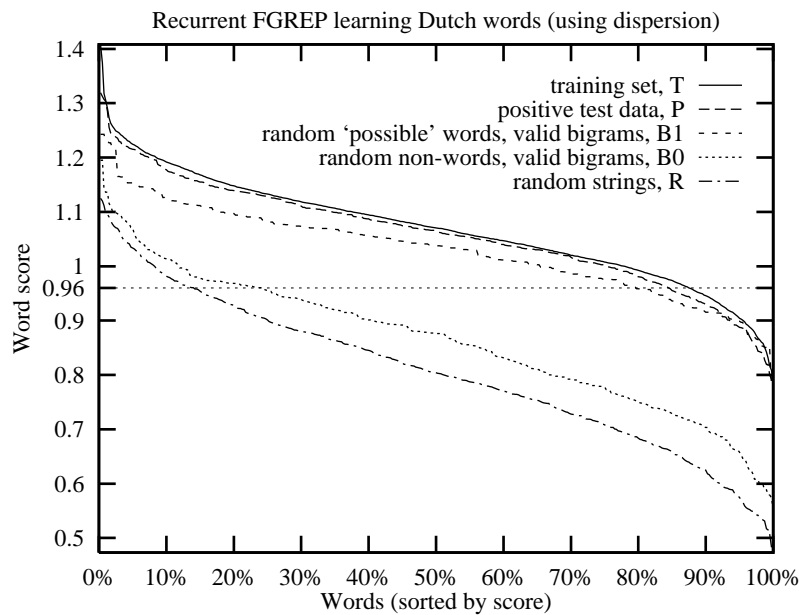


FIGURE 5: Performance of a network trained with dispersion.

and only 13% of the set of random strings.² The score distribution of the positive test data is nearly identical to the training data. Remarkable is the lower performance for the set of ‘words that *might* exist’ (**B1**). The network seems to have become sensitive to dependencies between letters which are separated by one or more other letters. This is in accordance with the design of SRNs. It indicates that the network has a stricter notion of a valid word. It may indicate that the network is sensitive to actual, not just possible words — which seems eminently plausible psychologically.

Nearest neighbor clustering of the FGREP lexicon created using dispersion results, however, in no clear clusters (Fig. 6). Is all order lost?

A Kohonen map [Kohonen 1989] is a *Self-Organizing Map* (SOM) used to order a set of high-dimensional vectors. It’s working is inspired by topological neural maps in the brain. The map is trained to ‘respond’ with a single unit for a particular input vector, and organize itself in a way such that similar input vectors are mapped to units that are close to each other. So, a Kohonen map can be used to clarify relations in a complex set of data.

We created a Kohonen map, displayed in Fig. 7. Line darkness indicates increasing difference- between neighboring units [Kleiweg 1998]. A minimal spanning tree is included to clarify relationships. Close relationships between *b-p*, *g-k* and *l-r* are still visible. Vowels are *not* randomly distributed between consonants.

Table 1 lists statistics for the network performance. Note that more than a standard deviation separates the training data, test data, and “Dutch-like” bigram combina-

2. We got better results by simply using the averaged square letter error: 86% of training data, 8% of random strings.

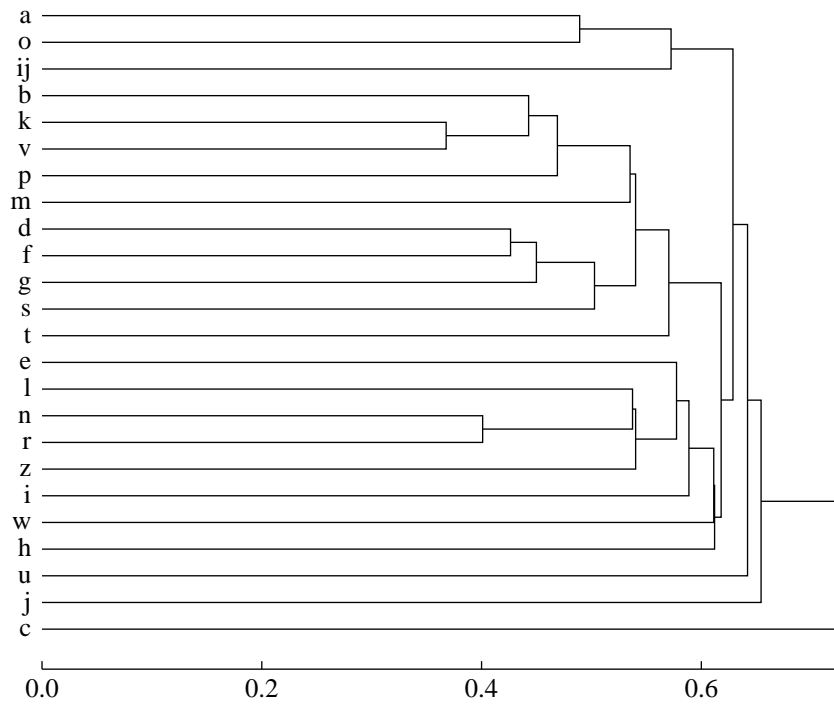


FIGURE 6: Clustering of FGREP data obtained using dispersion.

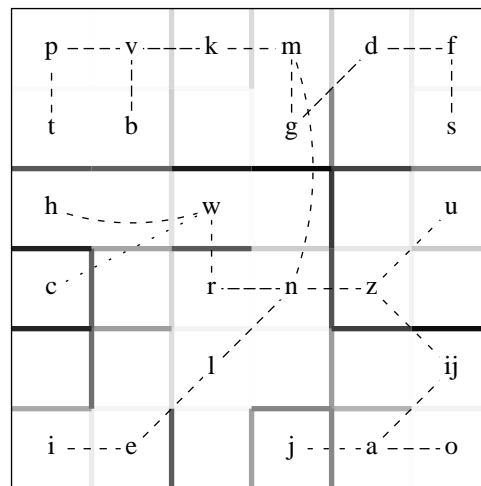


FIGURE 7: A Kohonen map of FGREP data obtained using dispersion.

length	T			P			B1			B0			R		
	n	\bar{x}	s	n	\bar{x}	s	n	\bar{x}	s	n	\bar{x}	s	n	\bar{x}	s
2	61	1.0095	0.0669	8	1.0063	0.0560	32	1.0487	0.0830	39	0.9439	0.0606	17	0.9684	0.0756
3	524	1.0406	0.0739	71	1.0365	0.0737	85	1.0183	0.0679	67	0.8650	0.1232	117	0.9013	0.0961
4	1379	1.0762	0.0910	160	1.0722	0.0926	51	1.0399	0.0958	101	0.8627	0.1155	299	0.8303	0.1211
5	1177	1.0762	0.1024	115	1.0613	0.1075	7	0.9901	0.1038	63	0.8464	0.1249	237	0.7718	0.1266
6	508	1.0686	0.1109	61	1.0561	0.1147				19	0.7486	0.1370	105	0.7020	0.1193
7	132	1.0569	0.1001	8	1.0259	0.0810				3	0.6911	0.1076	25	0.6523	0.1352
8	22	1.0141	0.1003	1	0.9231										
9	4	0.9708	0.0968												
all	3807	1.0681	0.0966	424	1.0585	0.0976	175	1.0290	0.0818	292	0.8614	0.1231	800	0.8039	0.1366

TABLE 1: Statistics for FGREP with dispersion on all data sets, grouped by word length. n : number of words, \bar{x} : average score, s : standard deviation.

T		P		B1		B0		R	
1.408	dracht	1.319	flanst	1.243	dint	1.195	laip	1.125	sdiip
1.399	gracht	1.313	spelt	1.231	pelk	1.142	kaid	1.120	fuit
1.398	kracht	1.307	slecht	1.227	vank	1.136	luld	1.120	iept
1.393	pracht	1.303	brand	1.216	gied	1.116	chigl	1.114	ljef
1.391	tracht	1.285	wacht	1.165	wirt	1.110	ud	1.112	feet
1.391	bracht	1.255	geep	1.164	ijt	1.102	taw	1.106	mojlp
1.388	klacht	1.245	krak	1.163	kreef	1.099	ded	1.099	ded
1.375	placht	1.241	vaat	1.153	ijd	1.096	orl	1.097	eft
1.375	hecht	1.241	zeep	1.153	ijp	1.095	teud	1.096	pe
1.374	specht	1.238	slaat	1.151	ied	1.087	od	1.094	poet
0.781	sinds	0.843	rooms	0.899	voo	0.618	gdto	0.517	vtmo
0.769	duts	0.841	soort	0.897	bu	0.608	nsututs	0.516	fmofmeu
0.767	corps	0.840	doods	0.883	sijl	0.608	nti	0.515	uskmve
0.767	ramsij	0.827	toorts	0.873	eups	0.599	mboogs	0.515	tfibo
0.745	soos	0.819	sol	0.870	afs	0.598	bsemsm	0.507	pfjpibs
0.744	schmink	0.819	saus	0.866	pijs	0.589	trfiks	0.485	vcbfct
0.742	psalm	0.800	koorts	0.853	wuibs	0.589	wtoos	0.482	jtifdu
0.722	soeks	0.792	puts	0.851	kods	0.579	lkijgsj	0.476	hfmso
0.702	soms	0.788	smots	0.848	seu	0.574	lisimu	0.473	fupgmmr
0.639	scouts	0.746	sjeiks	0.781	sij	0.552	bosso	0.452	fbfnmi

TABLE 2: Highest and lowest scores for all data sets.

tions (**T**, **P**, **B1**) on the one hand from the invalid non-Dutch-like bigram combinations and random strings (**B0**, **R**) on the other.

Table 2 lists the words from all data sets with the highest scores and the lowest scores.³ We believe that inspection of the “Dutch”-like column of random words from valid bigrams (**B1**) confirms our decision to evaluate this set separately. It is also the position of many linguists that the notion “possible word” is the cognitively more significant one.

The network trained with dispersion was further tested to determine how well it performed in completing words, given just the first letter. Results are shown in Table 3 and Table 4.

8 Discussion

Our experiments were not aimed at developing a better algorithm for the machine-learning of language. In any case, our results were not better than those obtained by Tjong Kim Sang [1998] or Stoianov et al. [1997].

We intended to capture some aspects of how humans process language. Our agenda is philosophical, not utilitarian. The focus is on whether FGREP is reliable in determining appropriate data representations, and in understanding its strength and weaknesses.

3. Using an alternative error function, such as mentioned in footnote 2, other types of words end up in the top and bottom of this table.

ane*	ip	raep*
baep*	jaf	stin
ch*	keep	tanp*
deep	laep*	ue*
ee	man	vanl*
felp	nanp*	wee
geep	one*	ijke*
hee	penp*	zae*

TABLE 3: Words generated from a single initial letter, copying output to input (invalid patterns marked*).

an	ip	ranv*
banv*	jaan	stan
chl*	keep	tanv*
de	lanv*	uep*
eep	manv*	vanv*
fe*	naan	weel
gelv*	on	ijk
hel	pel	zan

TABLE 4: Words generated from a single initial letter, output replaced by nearest match from lexicon before input (invalid patterns marked*).

The artificial neural network used in our experiments was *not* trained to distinguish between valid and invalid word patterns. The networks was trained to predict the next letter in partially processed words. Naturally, the trained network does a poorer job when processing illegal word patterns. The distinctions the network draws between valid and invalid word patterns is a by-product.

Likewise, people can distinguish between valid word patterns (either existing words or non-words that might exist) and invalid word patterns. In this case too, this ability is probably acquired as a by-product of another sub-function of the human language processor, but perhaps, a very different one than in our network experiments.

In language processing by adult humans, there is perhaps no such task as predicting the next letter. Adult readers don't process words one letter at a time, left to right. Human language processing is a problem of segmenting real-time data, synchronizing different levels of abstraction, such as phonetics, morphology, and syntax. In our network setup, there is only one level of processing, and input/processing/output are already segmented and synchronized. In spite of these differences, human language processing certainly involves dynamic sequence processing at some levels, and it certainly results in an ability to distinguish ill-formed from well-formed sequences.

There are also other reasons to use FGREP models in experiments.

The human language processor is not a collection of independent networks, but an integrated collection of networks that has learned to cooperate. This cooperation

is not limited to processing data collectively, the data itself is modeled when used for internal communication between sub-networks. The data representation must be structured in a manner that facilitates its processing, and relations between data must be represented by similarities within the data.

For network experiments this means the following: the input and output vectors processed by the network weren't fixed orthogonal vectors, but random vectors that change during the learning task to represent items with similar rôles by similar vectors. In Dutch, the letters *l* and *r* perform similar functions within word patterns, so we expect the vector representations developed during FGREP training to become similar. The main question for these experiments was: will a set of vectors develop that is ordered in a manner that is intuitively acceptable? We saw it did.

FGREP is intended as a means of communication *between* networks, as used by Miikkulainen [1993]. Using it in an isolated network, as we did, may be unnecessary. Language processing also involves feedback between levels of processing, which is modeled in work by McClelland & Rumelhart [1981] and Rumelhart & McClelland [1982], but absent in our setup. This said, what can we conclude about the poor prediction results of our network, not using dispersion?

In Miikkulainen & Dyer [1991] and Miikkulainen [1993], either network input or output, or both, are static. In our network, both input and output are temporal.⁴ In basic backpropagation, learning is a task of optimization. In an Elman network, learning suffers from the moving target problem: network weights are adjusted by a learning rule that does not take into account that the input from the context units is faulty. FGREP adds to this problem that *all* input is initially faulty. Despite these inadequacies, in Miikkulainen's experiments, the learning algorithm eventually accomplishes its task. We assume that FGREP with both temporal input and temporal output is too unstable, if used alone.

FGREP is not an optimizing algorithm, such as basic backpropagation. The learning rule adjusts weights and vectors to the task as it is presenting itself *at that moment*. However, as a result of learning, the overall task changes, possibly in a direction that has a worse possible global solution.

FGREP strives for similar vector representations of items that serve similar rôles, but *only in short term processing*. In a temporal to temporal Elman network distinctions with long term effects will get lost.

Adding dispersion overcomes this shortcoming of FGREP. Preliminary experiments indicate that FGREP + dispersion performs better than an Elman network with randomized, fixed input/output vectors.

We find it useful to evaluate using the more rigorous standards of machine learning in order to investigate neural networks more precisely. Note that we would not have been able to appreciate the effect of dispersion without this rigor.

Other improvements of FGREP might be possible. FGREP adjusts vectors through

4. By 'static' we mean that a sequence of symbols is concatenated to a single pattern, and processed without reference to the sequence. By 'temporal' we mean that a sequence is processed one symbol at a time.

the input layer only. *Target learning* could be implemented, using Eq. 7 (page 6), with output vector \mathbf{y} and target vector \mathbf{x} . Another improvement may be a more direct feedback between FGREP modules.

9 WWW resources

Data sets and results are available at:

<http://www.let.rug.nl/~kleiweg/papers/afiip/>

The CELEX Lexical Database is at:

<http://www.kun.nl/celex/>

Clustering was done with software available at:

<http://www.let.rug.nl/~kleiweg/clustering/>

The Kohonen map was created with software available at:

<http://www.let.rug.nl/~kleiweg/kohonen/>

References

- Aldenderfer, Mark S. & Roger K. Blashfield [1984], *Cluster Analysis*, Quantitative Applications in the Social Sciences, Sage, Beverly Hills.
- Cleeremans, Axel [1993], *Mechanisms of Implicit Learning: Connectionist Models of Sequence Processing*, Neural Network Modeling and Connectionism, The MIT Press, Cambridge, MA.
- Elman, Jeffrey L. [1990], 'Representation and structure in connectionist models', *Cognitive Science* **14**.
- Kleiweg, Peter [1998], 'Extended Kohonen maps'. Web article and software at <http://www.let.rug.nl/~kleiweg/kohonen/>.
- Kohonen, Teuvo [1989], *Self-Organization and Associative Memory*, 3rd edn, Springer-Verlag, Berlin.
- McClelland, James L. & David E. Rumelhart [1981], 'An interactive activation model of context effects in letter perception: Part 1. an account of basic findings', *Psychological Review* **88**, 375–407.
- McClelland, James L. & David E. Rumelhart [1988], Training hidden units: The generalized delta rule, in 'Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises', The MIT Press, Cambridge, MA, chapter 5.
- Miikkulainen, Risto [1993], *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*, Neural Network Modeling and Connectionism, The MIT Press, Cambridge, MA.
- Miikkulainen, Risto & Michael G. Dyer [1991], 'Natural language processing with modular PDP networks and distributed lexicon', *Cognitive Science* **15**, 343–399.
- Rumelhart, David E. & James L. McClelland [1982], 'An interactive activation model of context effects in letter perception: Part 2. the contextual enhancement effect and some tests and extensions of the model', *Psychological Review* **89**, 60–94.
- Stoianov, Ivo, John Nerbonne & Huub Bouma [1997], Modelling the phonotactic structure of natural language words with Simple Recurrent Networks, in 'CLIN '97'.
- Tjong Kim Sang, Erik F. [1998], Machine Learning of Phonotactics, PhD thesis, Rijksuniversiteit Groningen.

- Tjong Kim Sang, Erik F. & John Nerbonne [1999], Learning simple phonotactics, *in* 'Neural, Symbolic, and Reinforcement Methods for Sequence Learning', IJCAI-99 Workshop.
- Werbos, Paul J. [1995], Backpropagation: Basics and new developments, *in* M. A. Arbib, ed., 'The Handbook of Brain Theory and Neural Networks', The MIT Press, Cambridge, MA, pp. 134–139.