Linguistic Pattern Matching Capabilities of Connectionist Networks

Antal van den Bosch and Walter Daelemans*
ITK Tilburg University
Email: antalb@kub.nl, walter@kub.nl

February 1992

Abstract

We report on a series of experiments with backpropagation learning in feed-forward and recurrent feed-forward connectionist networks. The target application domain is the assignment of syllable boundaries to orthographic and phonological representations, a linguistic pattern matching problem in which phonological and morphological constraints interact. We investigated how different encoding schemes and network architectures influence the generalisation capabilities of the backpropagation learning rule for this task. We also show how different encodings can be combined in modular networks, improving performance. Our results indicate that a network is capable of learning the task, but not significantly better than symbolic pattern matching approaches or an approach based on table look-up.

1 Introduction

There is a marked difference between the rich inventory of representational and control structures used in "symbolic" approaches to linguistic pattern matching and transformation (production rules, frames, trees, graphs, unification, matching) and the one available in connectionist approaches (activation and inhibition links between simple units), which at first sight suggests that the former approach, because of its expressive power, is more suited for linguistic knowledge representation and processing. On the other hand, it is clear that we need methods for the automatic acquisition and adaptation of linguistic knowledge if we want to achieve real progress in computational linguistics. Connectionist learning algorithms allow us to learn mappings between representations automatically, on the basis of a limited number of examples, and to generalize what is learned to unseen cases. It is instructive in this respect to compare the architecture of a typical symbolic system for grapheme-to-phoneme conversion, which "learns by brain surgery" (Figure 1, from Daelemans, 1988) to a connectionist solution of the same problem, such as the one by Sejnowsky and Rosenberg (1987, Figure 2).

The connectionist approach can be adapted to different languages simply by changing the training set. Weijters (1990) used the architecture for English developed by Sejnowsky and Rosenberg, 1987) to accomplish the grapheme-to-phoneme conversion task for Dutch. Connectionist architectures are more robust, and it is not necessary to invest several manmonths of linguistic engineering to get the rules right. On the other hand, symbolic systems are modular (parts can be reused in other tasks)[1], and the rules and structures used can be inspected and interpreted by domain specialists (in this case linguists).

This paper is concerned with a well-defined phonological instance of linguistic pattern matching problems: the assignment of syllable boundaries to orthographic (spelling) and phonological

*We are grateful to Ton Weijters, Theo Vosse, Erik-Jan van der Linden, David Powers, and Peter Berck for relevant comments and conversation. Thanks also to Arthur van Horck for his LaTeXwizardry.

[1]The modularity argument in favour of symbolic systems applies to most connectionist approaches, but modular networks are possible as well, we present an example later in this paper. Also, it could be argued that in a connectionist approach *reusability* exists at the level of the acquisition technique rather than at the level of the acquired knowledge.

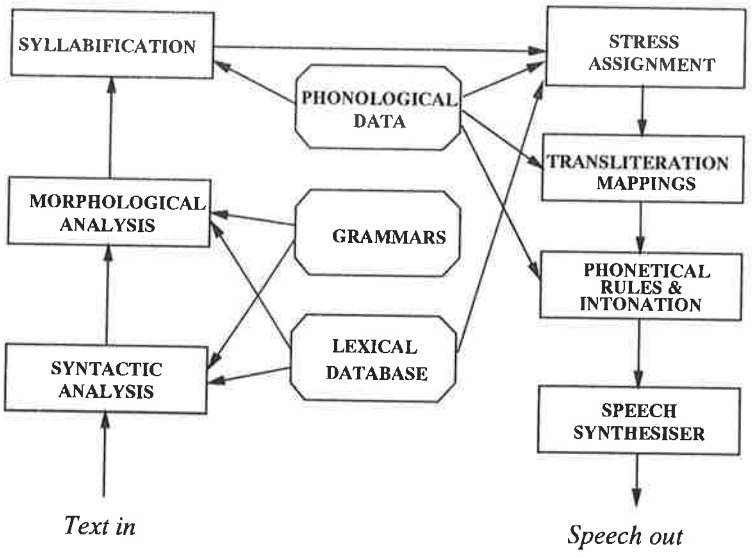

Figure 1: *Interaction between modules in the GRAFON grapheme-phoneme conversion system.*

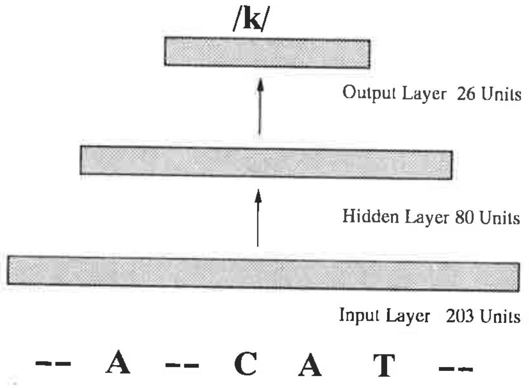

Figure 2: *Topology of the NETtalk grapheme to phoneme conversion network.*

representations of word forms in Dutch. We wanted to investigate whether the currently most popular connectionist learning technique, backpropagation of errors (Rumelhart et al, 1986) on (recurrent) feed-forward networks, is powerful enough to abstract the regularities governing the segmentation of strings of spelling symbols or phonemes into syllable representations. The hypothesis we set out with was that the performance of connectionist solutions to the problem would not be significantly better than that of existing pattern matching approaches, because of the inherent complexity of the task.

> The [connectionist] approach suffers from the same shortcoming as pattern matching approaches: without a dictionary, it is impossible to correctly compute morphological and syllable boundaries (...). We see no way how any network (...) could provide sufficient generalisations to parse or syllabify compound words reliably, whatever the size of the training data (remember that the vocabulary is infinite in principle). [Daelemans, 1988:11].

2 Task Analysis

Dutch syllabification is an interesting problem to test the pattern extraction capabilities of connectionist networks because the process involves phonological and morphological constraints that are sometimes conflicting. Simplifying matters slightly (see Daelemans, 1989 for a full account), we can say that the process is guided by a phonotactic *maximal onset principle*, a principle which states that between two vowels, as many consonants belong to the second syllable as can be pronounced together, and a *sonority principle*, which states that in general, the segments in a syllable are ordered according to sonority (from low sonority in the onset to high sonority in the nucleus to low sonority in the coda). This results in syllabifications like *groe-nig* (greenish), *I-na* and *bad-stof* (terry cloth). However, these principles are sometimes overruled by a *morphological principle.* Internal word boundaries (to be found after prefixes, between parts of a compound and before some suffixes) always coincide with syllable boundaries. This contradicts the syllable boundary position predicted by the *maximal onset principle.* E.g. *groen-achtig* (greenish, *groe-nachtig* expected), *in-enten* (inoculate, *i-nenten* expected) and *stads-tuin* (city garden, *stad-stuin* expected). In Dutch (and German and Scandinavian languages), unlike in English and French, compounding is an extremely productive morphological process which happens through concatenation of word forms (e.g. compare Dutch *spelfout* or German *Rechtschreibungsfehler* to French *faute d'orthographe* or English *spelling error*). Because of this, the default phonological principle fails in many cases (we calculated this number to be on average 6 % of word forms for Dutch newspaper text).

By incorporating a morphological parser and lexicon, a phonologically guided syllabification algorithm is able (in principle) to find the correct syllable boundaries in the complete vocabulary of Dutch (i.e. all existing and all possible words, excluding loan words and semantically ambiguous word forms like *kwarts-lagen* (quartz layers) versus *kwart-slagen* (quarter turns)). Existing symbolic pattern matching approaches that do not use a morphological parser fail miserably on a large proportion of new[2] cases where phonological and morphological constraints conflict.

The task for our connectionist network can be specified more clearly now. It should be able to achieve the following:

- Abstract the maximal onset and sonority principles and apply them to input not present in the training material.
- Abstract some (implicit) notion of morphological boundaries as overriding the phonological principles.
- Recognize loan words as overriding the previous principles.

We designed and implemented[3] a series of simulations to test the performance of networks on this task, using both orthographic (spelling) representations and phonological representations as training material (we expected the task to be more difficult with spelling representations because they contain more anbiguity).

3 Simulations

One of the disadvantages of applying a connectionist approach to any empirical problem, is that the designer of the simulations is confronted with a large search space formed by alternative architectures (number of layers, size of hidden layer, feedback connections from output to input or from hidden to input), training data selection and presentation methods, learning algorithms and activation functions, learning rate and momentum parameters, and data encoding schemes. As the simulation of massively parallel algorithms on serial machines is notoriously slow, we have been able to explore only a small region in this search space. In the description of our results we will focus on those choices that influenced network performance most.

[2]With *new* we mean: not used to derive the rules. The pattern matching rules can of course be tailored to a set of word forms and hyphenate this set with 100% correctness (the approach by Vosse to be discussed later achieves this), but we are concerned with generalization to new cases here.

[3]For the simulations we used PLANET 5.6, a public domain connectionist simulator for UNIX workstations developed by Yoshiro Miyata. We are grateful to Van Dale Lexicografie (Utrecht) for allowing us to use a word form list with hyphens based on *Prisma Handwoordenboek Spelling. Het Spectrum, 1989* for research purposes, and to Henk Kempff (Tilburg University) for making available a list of word forms with phoneme representations.

pattern	left		focus	right		target
1	_	_	z	i	e	0
2	_	z	i	e	k	0
3	z	i	e	k	e	0
4	i	e	k	e	n	1
5	e	k	e	n	h	0
6	k	e	n	h	u	0
7	e	n	h	u	i	1
8	n	h	u	i	s	0
9	h	u	i	s	_	0
10	u	i	s	_	_	0

Table 1: *Window encoding applied to 'ziekenhuis' (hospital).*

3.1 Training and Test Data Encoding

To be able to train a connectionist network for hyphenation, the problem must be translated into a representation suited for connectionist simulation. This amounts to deciding what form the input and target must take and what type of network architecture is needed.

We interpret the hyphenation task as a decision problem: given a certain character position in a word and a left and right context, decide whether it is the first character of a new syllable. This formulation leads to an encoding in which the input is a part of the word, with one character position as the focus decision position. The target is a simple yes/no unit that decides whether the focus position is the start of a syllable. This encoding can be seen as a window being 'moved' along the word. An example of this 'moving window' encoding of *ziekenhuis* (hospital), resulting in 10 patterns, is shown in Table 1. In this example, the position in the middle is the focus position, accompanied by two left context and two right context characters. The word is preceded and followed by two blank characters. We will take a 5-character window to mean a left context of 2, a target character, and a right context of 2.

As mentioned earlier, we need representations for both graphemic (spelling) strings and phonemic strings. We encoded the individual characters randomly using 6 units for phonemes (of which we distinguished 35) and 5 units for graphemes. This random encoding is economical and avoids weakening the results by explicitly encoding linguistic knowledge into the patterns (although we will loosen this restriction in section 3.4). Our results indeed indicate that for this task, there is no need for encoding phonetic features, or using a space-consuming *local coding*[4].

For an optimal training, all possible patterns should be presented in one training set. For practical reasons, we restricted ourselves to pattern sets derived from maximally about 3,000 words, resulting in about 18,000 patterns. For simple feed-forward back propagation networks it is not necessary to present the patterns in their natural order (i.e. the order in which they occur in the word form), because they lack memory, and thus the capacity to remember previous hyphenations in the same word form. Presenting the patterns in random (permuted) order resulted in slightly better results. Permuted presentation of patterns also resulted in a less smooth error graph than in the case of natural order presentation.

For constructing pattern sets, we took a lexical database (195,000 word forms for spelling, 70,000 for phoneme representations), and extracted every nth word, n being a number between 10 and 400 (depending on the desired magnitude of the training and test set). We then split the resulting word list into a training word list and a test word list (so that no training word was contained in the test set). Note that, since a word is transformed into a number of patterns, some training patterns may be contained in the test pattern set, because of partial similarity

[4]One input unit for each possible input character for each pattern position. E.g. for the case phoneme input: 35 units for each character in the input pattern. Tests with local coding show that at best, their hyphenation performance equals that of networks with randomly encoded patterns. An advantage of local coding may be that it is in general easier to interpret trained connection weight matrices. The complexity of the present task is such that local coding is not really helpful however.

between words. E.g., *draadje* (thread) and *paadje* (path) produce the identical patterns [aadje], [adje_] and [dje__] when using a 5-character (2-1-2) window.

3.2 Output Analysis

The activation of the single output unit is interpreted as a decision on the insertion of a syllable boundary before the target position: YES (activation 0.5 or higher) or NO (activation less than 0.5). The activation level could also be interpreted as a probability or certainty factor, but in order to optimize accuracy we chose the threshold interpretation.[5]

The network error on the test set measures the number of incorrect decisions on *patterns*. What we are interested in, however, is the number and type of incorrectly hyphenated syllables and words. To analyse the actual hyphenation performance (as opposed to the rather uninformative network error), we therefore used some additional metrics to determine the different kinds of errors that a hyphenation network made. Four different kinds of errors are distinguished:

1. Omission of a hyphenation. This error can easily be stated as a NO that should have been a YES. It counts as one false hyphenation (a hyphenation missed). E.g. *pia-no* instead of *pi-a-no*.
2. Insertion of a hyphenation. A YES that should have been a NO. This error also counts as one false hyphenation (a hyphen too many). E.g. *pi-a-n-o*.
3. Transposition. Hyphenation on a position to the left or right of the target. This is actually a combination of error type 1 and 2. This error typically occurs on the linking position between the different parts of a morphologically complex word, where additional morphological information would be needed to put the hyphen in the correct place (see the examples in section 2). Two adjacent incorrectly placed patterns count as one incorrectly placed hyphenation. E.g. *daa-rom* instead of *daar-om* (therefore).
4. Marking two adjacent positions as hyphenation positions, creating an impossible one-consonant syllable. Two adjacent patterns may (in isolation) both deserve a hyphen, so without memory it is inevitable that a network tags both positions as a hyphen. E.g. *daa-r-om*. In Dutch it is possible to have a one-vowel syllable, as in *pi-a-no*, so when counting false hyphenations, the type of the isolated phoneme has to be checked. If it is a consonant, the incorrectly processed pattern counts as one incorrectly placed hyphenation.

We will call errors of types 1, 2 and 4 *non-morphological* errors, and errors of type 3 *morphological* errors. For errors of type 4, it is possible to introduce a correction mechanism to solve some instances of this problem. Since the output of the type of network we used is usually not exactly the minimum or maximum target but a floating point value that comes near to it, the two YES outputs involved in this type of error could be matched in the way that the output with the highest value is declared to be the correct output; the other is set to NO. Note that if this decision is not correct, the resulting single hyphenation error has become of error type 3 (e.g., a morphological error).

In the simulations mentioned below, this correction mechanism chose the correct solution in about 60 to 70 % of all cases. Without the correction, all cases of error type 4 count as one incorrectly placed hyphenation of type 2. Note that this correction mechanism is efficient (a linear comparison between pairs), and hardly affects the total time needed to hyphenate a word. In the following performance descriptions we will provide results both with and without this post-processing.

3.3 Optimizing Hyphenation Performance

In the simulations that will be described here, various network features were systematically altered to measure their effect on generalization performance (the degree to which the extracted patterns can be successfully applied to new data not present in the training data). Our simulations can be divided into two groups, the first dealing with phonemic hyphenation, the second

[5]Having an UNKNOWN-group with activations between 0.3 and 0.7 resulted in 10% more incorrect decisions.

type	% incorrect patterns	% incorrect hyphens	% incorrect hyphenations (with postprocessing)
phonemic	2.4	7.7	6.2
graphemic	3.7	12.7	6.9

Table 2: *Results of the best phonemic and graphemic hyphenation networks*

with graphemic hyphenation. This strict division is not maintained throughout all following paragraphs, because most results are valid for both groups of simulations. We start with a short summary of network parameters that we decided not to change systematically after some initial experimentation.

3.3.1 Static network parameters

Hidden layer size. To represent the extracted knowledge necessary for hyphenation, a reasonable number of hidden units must be available. In practice, it turned out to be best to have a number of hidden units that is about 1.5 times the number of input units.

Activation values. We used input and target activation values of 0.9 and 0.1 instead of 1.0 and 0.0, resulting in 2 to 3% less incorrect patterns.

Network parameters. After some exploratory experimentation, we chose to use standard values for the learning rate (0.55) and the momentum (0.5) for all simulations.

Length of training. Because of the inherent ambiguity of some hyphenation patterns, a hyphenation network will never converge to an error of nearly 0.0, but a somewhat higher error. Usually it took about 300 to 400 iterations or epochs (1 iteration is one presentation of the whole training set) to reach that level, when dealing with 15,000-18,000 training patterns. When aiming at hyphenation optimization, it is important to notice that the lowest error on test material is reached much earlier than after 300 or 400 iterations: due to overfitting and overgeneralisation on the training material, which already starts to play a role after a few iterations, the network often performs best on test material after 50-100 iterations. See Figure 3 for a typical result.

3.3.2 Phonemic versus graphemic hyphenation

When comparing results on graphemic and phonemic material, it turned out higher accuracy is achieved on phonemic material than on graphemic material. The best generalization results on both types of hyphenation are shown in Table 2. The best phonemic network uses 6-unit random coding with a 5-character window; the best graphemic network uses 5-unit random coding with a 7-character window.

This result was expected, as phonemic training data contains more linguistic information (e.g. it disambiguates between the different phonemic meanings of the grapheme <*e*>).

3.3.3 Effect of window-width

The maximum phoneme syllable length in Dutch is 5 (or 6 with diphtongs counted as two phonemes); in spelling, syllable length can be up to 8 graphemes. Most syllables do not contain more than 3 phonemes (approximately 50% of all Dutch syllables contain only 1 or 2 phonemes, and approximately 85% has 3 phonemes or less).

To determine the optimal window width, we first determined the importance of each side separately. We trained a network on patterns which had only a right context and another network on patterns which had only a left context. We did this for phonemic as well as graphemic hyphenation. In both cases we used a context of four characters. The results shown in Table

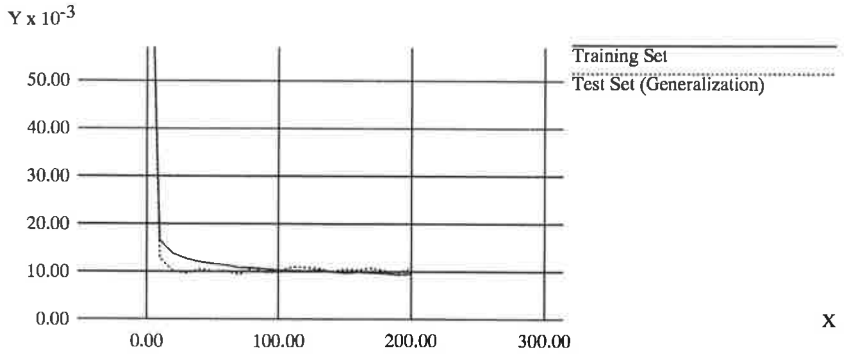

Figure 3: *Network error on training material compared to network error on test material. X = number of epochs, Y = 0.001error.*

3 indicate that the right context contains more information useful in hyphenation.[6] This is consistent with the maximal onset principle.

We expected that in the case of both graphemic and phonemic hyphenation, a window width of 7 (3-1-3) would produce better results than smaller window widths. However, in the case of phonemic hyphenation, window width 5 (2-1-2) turned out to be optimal. Note that this window contains patterns that show the last two characters of a syllable and the first three characters of the following syllable if the focus character is at the beginning of that syllable. This is in accordance with the fact that the right context is more informative. A possible explanation for the counter-intuitive result that a smaller window size produces better generalization, would be that in case of a larger window size, more ambiguous patterns (the same pattern with both a YES and a NO decision) are present in the training material, or that all areas of the larger space of possible training examples is not sufficiently covered by the training material. It may well be possible that the morphological constraints (internal word boundaries) are so unconstrained that they act as noise. This is subject of current research.

Table 4 shows the difference in hyphenation performance between two networks trained with different window widths, in which the other parameters are kept constant. The networks used

[6]The same asymmetry between information content in left and right context shows up in a grapheme-to-phoneme conversion task using table-lookup, described in Weijters, 1991.

Hyphenation type	% Wrong patterns left context	% Wrong patterns right context
Phonemic	53.7	38.2
Graphemic	50.7	35.3

Table 3: *Hyphenation performance on pattern sets with only left or right context.*

window	% incorrect patterns	% incorrect hyphenations	% morpho-logical	% non-morpho-logical
2-1-2	7.6	28.3	34.9	65.1
3-1-3	6.2	22.8	25.1	74.9

Table 4: *Hyphenation performance and error analysis of two graphemic networks using window sizes 5 and 7.*

here are trained on graphemic hyphenation (their performance is relatively bad; postprocessing results are left out here). The network trained on window width 7 hyphenates better than the network that is trained on window width 5 (as noted above). The additional analysis of the types of errors, also in Table 4, indicates that improvement of hyphenation performance is not necessarily linked to an increase in morphological errors.

3.3.4 Network Architectures

Errors of type 4 (marking two adjacent positions as hyphens, isolating a consonant) were made by most networks. The correction mechanism that solved a lot of these errors is obviously not part of the network itself, but only plays a role after the word has been passed through the network. Using standard backpropagation, it was impossible to let the network notice this type of errors, simply because in standard backprop no 'memory' is available to remember that the previous pattern already received a hyphen.

Recently, proposals have been made on the subject of incorporating memory in connectionist networks. The two most used approaches are those of Jordan (1986) and Elman (1988). Jordan proposes an extra *recurrent* copy link from the output layer to a *context* layer, which in its turn is connected to the hidden layer. In the case of hyphenation networks, it can be expected that a previous YES-output, copied back to the context unit, is a sign for the network to suppress marking the following position as a hyphenation position (provided that the current focus character is a consonant). Elman's approach introduces an extra context layer which is a copy of the hidden layer after a pattern has passed the network. Instead of a direct clue about the previous output, the hidden layer activations might indirectly make clear that the current output should not be a syllable boundary by using its memory about previous positions.

We performed four simulations on each architecture, using the same training set in each simulation. The results indicate that there is no evidence for the claim that recurrency improves hyphenation. In fact, Jordan networks seem to perform worse than standard backprop networks.

Table 5 displays the results of the comparison simulations. We performed an addition analysis on the error types made by the three networks. The supposed advantage of an internal memory in avoiding errors of type 4 can only slightly be observed in the case of Jordan architecture, but apparently has no effect on its hyphenation performance.

3.4 Modular versus Internal Combination

Sometimes two networks can have the same error percentage, while producing different types of hyphenation errors. For example, network A can have the habit of leaving out uncertain hyphenations, whereas network B, producing the same error, tends to 'overhyphenate'. If it were possible to somehow combine the solutions of A and B, their shortcomings might be partly

Architecture	% incorrect hyphenations	% morphological	% non-morphological	Number Type 4 errors
Backprop	16.2	36.7	63.3	110
Elman	16.5	36.3	63.7	114
Jordan	19.1	41.6	58.4	133

Table 5: *Hyphenation performance and error type analysis for Backprop, Elman and Jordan architectures. Mean results for four simulations with each architecture.*

corrected against each other. We investigated two different approaches that combine two or more network solutions in order to get better hyphenation performance:

1. **Modular Combination**: combining the outputs of several (two or more) networks that solve the same problem. The array of outputs serves as the input layer for a *top network* that is trained to decide on the basis of its inputs (which may conflict at some points) what is to be the definitive output (see Figure 4).
2. **Internal Combination**: combining different encodings in single patterns. Contrary to modular combination, the hyphenation problem is presented to a single network. Hyphenation performance is augmented by presenting the network with more clues for solving the problem, by extending the encoding.

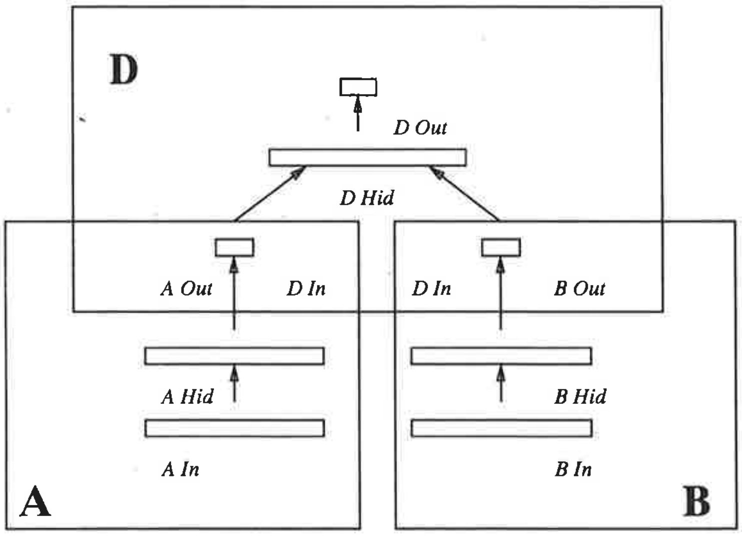

Figure 4: *Composition of networks.*

The main advantage of having a top network to solve this decision problem, is that this network is in principle able not only to extract generalizations about when to amplify or suppress each network output, but also to represent exceptions to these generalizations. For example, the top network will encounter situations where the outputs of the competing networks clearly conflict. It will have to develop some notion of exception to decide in those cases which network is right. A second advantage is that by analyzing the relative influence of each subnetwork on the overall decision, we can get insight into the relative importance of different encodings.

The main advantage of combining different encodings in patterns as opposed to the top network approach is that the problem solving is done within a single network. The solution to the hyphenation problem is not developed separately as in the case of a top network, but proceeds interactively during training.

The accuracy of both optimization methods turned out to be the same. To check this, we performed a series of simulations that involved two different phoneme encodings, which were

phoneme class	phonemes	sonority value
vowel a	a,A	1.0
vowel e,o	e,ε,o,O	0.9
vowel i,u	i,I,y,u,œ,∂	0.8
vowel ø, r-sounds	ø,r	0.7
semi-vocals, laterals	l,w,h,j	0.6
nasals	m,n,η	0.5
voiced fricatives	z,Z,v	0.4
voiceless fricatives	s,S,f,γ,x	0.3
voiced stop	b,d,g	0.2
voiceless stop	p,t,k	0.1
-	blank space	0.0

Table 6: *Sonority encoding.*

Encoding	%Wrong patterns	% Wrong hyphenations	% Wrong hyphenations (with postprocessing)
Identity	3.8	12.5	8.8
Sonority	4.1	10.9	9.2

Table 7: *Test performance of the individual networks*

used in two separate, single phonemic hyphenation networks. The outputs of the best of both networks were combined into a modular combination network, and finally the two different encodings were combined to be trained in an internal combination network.

For all networks we used a 5 character window, applied to a fairly large training set (1,774 phonemic words, forming 15,002 patterns). For the first network, each phoneme was encoded as a random 6-unit string (from now on, the *identity encoding*). For the second network, each phoneme was encoded by its sonority (the *sonority encoding*, a number ranging between 0.0 and 1.0), as seen in Table 6. The hierarchy shown in the table is based on Selkirk (1984).

First, we trained and tested the two competing networks separately to determine their individual performance (Table 7). The test set consisted of 7,465 patterns, extracted from 876 words which were not in the training set. These words contained 1,813 hyphenations. The identity network proved to be the best, but only after postprocessing. This reflects the greater complexity of the training material for the identity network (6 random units per phoneme) than for the sonority network (1 unit per phoneme). Higher information density and more combinations within a pattern led to a higher uncertainty of the identity network, which is partly suppressed by the decision mechanism solving error type 4.

All top networks were trained and tested on the same test set as described above (with 7,465 patterns). It turned out that the top networks reduced the number of incorrect hyphenations by about 30% (see Table 8). The network that produced these results used 2 hidden units. Tests with other hidden layer sizes (1, 3 and 4 units) showed no significant difference.

During experimentation with top networks, it became clear that the best results were not obtained by combining the best networks ever trained, but by combining well performing networks

% incorrect patterns	% incorrect hyphenations	% incorrect hyphenations (with postprocessing)
3.0	8.7	6.5

Table 8: *Performance of example top network.*

network	% incorrect patterns	% incorrect hyphenations	% incorrect hyphenations (with postprocessing)
A	3.0	9.5	6.6
B	3.8	12.9	5.5

Table 9: *Hyphenation performance of internal combination networks*

network type	data type	% incorrect patterns	% incorrect hyphenations	% incorrect hyphenations (with postprocessing)
modular	phonemic	2.1	6.7	5.9
	graphemic	2.4	7.9	5.7
internal	phonemic	2.6	8.2	5.5
	graphemic	2.3	7.9	4.6

Table 10: *Best phonemic and graphemic hyphenation performance of modular and internal combination networks.*

with slightly erroneous networks. This helped the top network in detecting the real network errors and cancel them out. The main thing a top network which is trained on well-performing network outputs does, is to repeat what these networks do, without performing much additional correction.

For the simulations with internal combination, we formed pattern sets on the basis of the same words as with the single experiments. The encoding for each phoneme was formed by simply concatenating the identity and the sonority encoding. Network A uses a 5-character window; network B uses 7-character window. The results of the best two internal combination networks are shown in Table 9. Notice the rather low scores of network B on patterns and hyphenations without postprocessing versus the high score with postprocessing.

Taking into consideration the fact that more extensive testing could produce even better results, it can be concluded that the combination of different encodings in a modular or internal way can lead to a marked improvement in hyphenation performance, although it seems that about 95% correctly placed hyphens is the ultimate accuracy threshold for networks of our kind (see also Table 10). There is a slight hyphenation performance advantage for internal combination versus modular combination. Furthermore, internal combination has the practical advantage of using less space as it results in a single network. The best results obtained by training both modular combination networks and internal combination networks on phonemic and graphemic hyphenation respectively, are shown in Table 10. Notice that the best result on graphemic hyphenation with internal combination is obtained with a noteably larger training set (73,752 patterns) than in the other simulations.

4 Related Research

One of the first applications of connectionist learning to (morpho)phonology was the pattern association (2-layer) network of Rumelhart and McClelland (1986), that learned to map roots to their past tense. The experiment has been replicated with backpropagation learning in a three-layer network by Plunkett and Marchman (1989). To avoid the legitimate criticism that these approaches only work because of the linguistic knowledge that is implicit in the training data (Lachter and Bever, 1988) or don't work because of the wrong linguistic knowledge implicit in the training data (Pinker and Prince, 1988), we performed most of our simulations with random encodings of segments.

In the landmark experiments by Sejnowsky and Rosenberg (1987) on text-to-speech transformation with NETtalk, they also included syllable boundaries (and stress) in the training material. It is unclear whether generalization performance on syllable boundary prediction was

network	% incorrect hyphens	% incorrect words	% morpho-logical	% non-morpho-logical
CHYP (Daelemans)	4.7	8.6	92	8
Table Look-up (Weijters)	2.0	3.7	40	60
PatHyph (Vosse)	1.8	3.0	87	13
Network	6.3	11.6	22	78
Network (with postprocessing)	4.0	7.4	60	40

Table 11: *Hyphenation performance on a Dutch text of alternative pattern matching hyphenation systems vs. the best hyphenation network on spelling (internal combination).*

taken into account in their performance measures (80% generalization), or, if this was the case, what part of the error was due to incorrect hyphenation boundaries. Furthermore, hyphenation in Dutch is of a completely different nature, which makes comparison specious.

Fritzke and Nasahl (1991) report 96.8% correct generalization on connectionist hyphenation for German (which is similar to Dutch as regards syllabification) with a three-layer feed-forward architecture, a window of 8 letters, a hidden layer size of 80, random encoding of graphemes, and one recurrent (feedback) link from output unit to an extra input unit (the approach of Jordan (1986), described in subsection 3.3.3), which makes it a 'sequential' network. In contradiction with our own results, they noted a better result than a comparable architecture without feedback connection. The network was trained on 1,000 words and tested on 200 words not present in the training set. This result should be compared with our error rate on patterns. As far as can be inferred from the text, the error is measured on the percentage of 'incorrectly hyphenated positions' but these positions seem to be interpreted as our 'patterns'. In Dutch words there are on average about four times more characters (and therefore patterns) than hyphenations, and we calculated that a network has at most about 1.3 more incorrect patterns than incorrectly placed hyphens. Assuming German to be similar to Dutch in this respect, this leads to the conclusion that Fritzke and Nasahl would have had a hyphenation error percentage of about 10%.

5 Connectionist versus Symbolic Pattern Matching

As a final comparison of the performance of connectionist networks and symbolic pattern matching systems, we selected a Dutch text[7] and compared the performance of CHYP (Daelemans, 1989), an approach based on the table look-up method of Weijters (1991)[8], an (as yet) undocumented algorithm PatHyph (Vosse, p.c.) and our best spelling hyphenation network. The results are summarized in Table 11. For each approach, we provide the percentage of incorrect hyphenations, the percentage of incorrectly hyphenated word types, and the contribution of morphological versus non-morphological errors to the overall performance.

CHYP is a symbolic pattern matching algorithm based on phonotactic restrictions. It operates in two modes: a *cautious* mode, in which only those syllable boundaries are indicated that are absolutely certain (predictable from phonotactic pattern knowledge), and a *daring* mode, in which apart from the 100% certain hyphens also the most probable uncertain hyphens are provided. For this test, CHYP operated in daring mode.

The table look-up method of Weijters uses the training set as a data base, and computes the similarity of new patterns to each of the items in the database. The decision associated with the most similar data base item(s) is then used for the new pattern as well. The similarity measure takes into account the fact that characters closest to the target character are more important than those further away (this can be interpreted as a domain heuristic).

[7]Foreign words and non-words were removed, but we left loan words and names in the text.

[8]We are grateful to Ton Weijters for his willingness to apply his table-lookup algorithm at very short notice to our hyphenation data.

PatHyph also uses patterns to predict syllable boundaries, but the patterns were continuously and automatically adapted by repeatedly testing them on a large lexical database. Although being symbolic in nature, this method is automatic (no hand-crafting), and the resulting "knowledge" cannot be inspected.

The comparison shows that even our best network cannot compete with the best pattern matching approach, which we predicted in our hypothesis. Especially the good performance of the simple look-up method is surprising.

6 Conclusion

For the problem of finding syllable boundaries in spelling or phonemic strings, solutions using domain knowledge are still superior or comparable in *accuracy* to a connectionist solution. They have an added advantage because of their inspectability and the reusability of developed rules. Once more, it has become clear that backpropagation learning (and arguably connectionist learning in general) does not offer miracle solutions to hard problems, and that when domain knowledge is available a connectionist approach may not be the best way to tackle a problem (see also Weijters, 1991). We have also shown that modular networks or internal combination networks integrating linguistically relevant encodings perform better than each of the networks separately.

As far as *efficiency* is concerned, a connectionist approach achieves accuracy levels comparable to a symbolic pattern matching approach overnight (given sufficient computing power) without a large amount of *linguistic engineering*. The connection weight matrix of a fully trained network, combined with simple code for encoding, activation, decoding, and postprocessing could be combined into a simple and efficient hyphenation module for text processors, comparable in accuracy to existing approaches, but without the overhead of keeping in store large tables of patterns or, even worse, a dictionary.

7 References

Daelemans, W. GRAFON-D: A Grapheme-to-phoneme Conversion System for Dutch. AI Memo 88-5, AI-LAB Brussels, 1988.

Daelemans, W. 'Automatic Hyphenation: Linguistics versus Engineering.' In: F.J. Heyvaert and F. Steurs (Eds.), *Worlds behind Words*, Leuven University Press, 347-364, 1989.

Dennis, S. and S. Phillips. Analysis Tools for Neural Networks. Unpublished paper, 1991.

Elman, J. Finding Structure in Time. CRL Technical Report 8801, 1988.

Fritzke, B. and C. Nasahl. A Neural Network that Learns to do Hyphenation. In: T. Kohonen, K. Mäkisara, O. Simula and J. Kangas (Eds.) *Artificial Neural Networks.* Elsevier Science Publishers, 1375-1378, 1991.

Jordan, M. I. Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society* Hillsdale, NJ, 1986.

Lachter, J. and T. Bever. 'The relationship between linguistic structure and associative theories of language learning.' In Pinker and Mehler (eds.) *Connections and Symbols.* MIT Press, 1988.

Pinker, S. and A. Prince. 'On Language and Connectionism: Analysis of a PDP Model of Language Acquisition.' In Pinker and Mehler (eds.) *Connections and Symbols.* MIT Press, 1988.

Plunkett, K. and V. Marchman. 'Pattern Association in a Back Propagation Network: Implications for Child Language Acquisition.' San Diego, CRL, Technical Report 8902, 1989.

Rumelhart, D. E. and J. McClelland. 'On learning the past tense of English verbs.' In D.E. Rumelhart and J.L. McCleland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of cognition. Volume 2.* Cambridge, MA: Bradford Books.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams. Learning Internal Representations by Error Propagation. In: Rumelhart and McClelland (Eds.) *Parallel Distributed Processing Volume 1*, MIT Press, 318-362, 1986.

Sejnowsky, T.J. and C.R. Rosenberg. Parallel Networks that Learn to Pronounce English Text. *Complex Systems* 1, 145-168, 1987

Selkirk, E.O. 'On the major class features and syllable theory.' In: Aronoff, M. and R.T. Oehrle (eds.) *Language sound structure.* Cambridge, Mass.: MIT Press, 1984, 107-136.

Weijters, A. and G. Hoppenbrouwers. 'NetSpraak: een neuraal netwerk voor grafeem-foneem-omzetting.' *Tabu* 20:1, 1-25, 1990

Weijters, A. 'A simple look-up procedure superior to NETtalk?' In: T. Kohonen, K. Mäkisara, O. Simula and J. Kangas (Eds.) *Artificial Neural Networks.* Elsevier Science Publishers, 1991.