Proceedings 3rd Twente Workshop on Language Technology, 27-38, 1992

GENERALIZATION PERFORMANCE OF BACKPROPAGATION LEARNING ON A SYLLABIFICATION TASK*

Walter Daelemans
walter@kub.nl

ITK
Tilburg University,
P.O.Box 90153
5000 LE Tilburg
The Netherlands

Antal van den Bosch
antalb@kub.nl

Abstract

We investigated the generalization capabilities of backpropagation learning in feed-forward and recurrent feed-forward connectionist networks on the assignment of syllable boundaries to orthographic representations in Dutch (hyphenation). This is a difficult task because phonological and morphological constraints interact, leading to ambiguity in the input patterns. We compared the results to different symbolic pattern matching approaches, and to an *exemplar-based* generalization scheme, related to a k-nearest neighbour approach, but using a similarity metric weighed by the relative information entropy of positions in the training patterns. Our results indicate that the generalization performance of backpropagation learning for this task is not better than that of the best symbolic pattern matching approaches, and of exemplar-based generalization.

1 BACKGROUND

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 for the same problem, such as the one by Sejnowski and Rosenberg (1987, Figure 2).

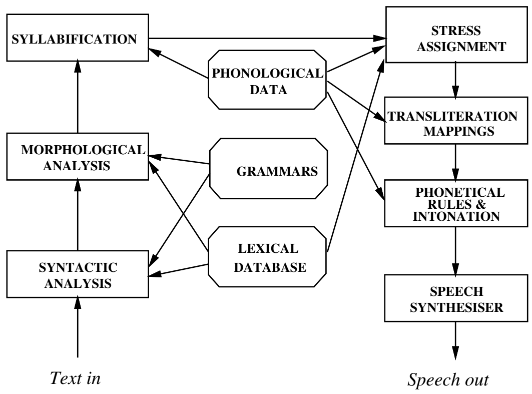

Figure 1: *Interaction between modules in the GRAFON grapheme to phoneme conversion system.*

The latter approach can be adapted to different languages simply by changing the training set. Weijters (1990) used the architecture for English developed by Sejnowski 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

*A shorter version of part of this paper appears in the proceedings of the International Conference on Artificial Neural Networks (ICANN 92). We are grateful to Ton Weijters, Theo Vosse, David Powers, Erik-Jan van der Linden, and Peter Berck for relevant comments and conversation. Thanks also to Arthur van Horck for his LaTeX wizardry.

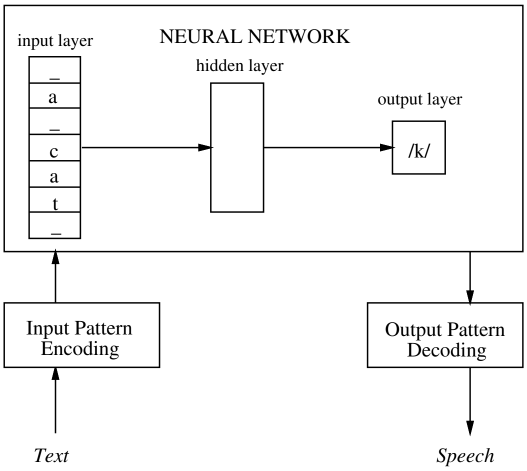

Figure 2: *Topology of the NETtalk grapheme to phoneme conversion network.*

tasks), and the rules and structures used can be inspected and interpreted by domain specialists (in this case linguists). We argue that in connectionist and other learning approaches, *reusability* (which has become the new philosopher's stone of computational linguistics recently) exists at the level of the acquisition technique rather than at the level of the acquired knowledge. This is a form of reusability which is stronger and more useful than what is usually understood by this term.

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

We also wanted to compare the generalization performance of the connectionist approach to that of other statistical induction techniques (which we consider to be a baseline in the evaluation of the results of connectionist generalization).

2 TASK DESCRIPTION

Dutch syllabification is an interesting problem to test the generalization capabilities of connectionist networks because the process involves phonological and morphological constraints that are sometimes conflicting. There are also a number of language-dependent spelling hyphenation conventions that override syllabification rules. 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 principles fail 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 (as described in Daelemans, 1989) is able to find the correct syllable boundaries in the complete vocabulary of Dutch (i.e. all existing and all possible words, excluding some 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[1] 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 and language-specific hyphenation conventions as overriding the phonological principles.

- Recognize loan words as overriding the previous principles.

We designed and implemented[2] a series of simulations to test the performance of networks on this task.

3 CONNECTIONIST 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, training data selection and presentation methods, learning and activation functions, parameters, and encoding schemes. In this section, we report on a series of simulations in which we explored part of this search space for the hyphenation problem. We will focus on those choices that influenced network performance most. Unless otherwise stated, backpropagation learning in a three-layer feed-forward network (Sejnowski et al., 1986) should be assumed.

[1] With *new* we mean: not used to derive the rules. The pattern matching rules can of course be tailored to any 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.

[2] 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.

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

We interpret the hyphenation task as a pattern classification 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 character string (a pattern) representing 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 Table1.

We encoded the individual characters randomly using 5 units for each grapheme. This random encoding is economical and avoids weakening the results by explicitly encoding linguistic knowledge into the patterns (although we will loosen this restriction later in the paper). Our results indeed indicate that for this task, there is no need for encoding orthographic features, or using a space-consuming *local coding*[3].

3.2 Training and Test Set Properties

The training set consisted of 19,451 word forms (containing hyphens indicating syllable bound-

[3] One input unit for each possible input character for each pattern position. I.e. 26 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.

Size	N	T/t	A	O
3 (1-1-1)	6266	0.03	16.2	97.8
5 (2-1-2)	66231	0.32	1.0	75.9
7 (3-1-3)	124309	0.61	0.1	48.7

Table 2: *A comparison of number of pattern types in training set (N), pattern type/token ratio in the training set (T/t), percentage of ambiguous pattern types in the training set (A), and overlap between training set and test set pattern types (O).*

aries) taken from the Van Dale list (containing about 195,000 word form types). The test set consisted of 1,945 words from the same database, not present in the training set. It is useful to keep in mind a number of properties of training and test set when evaluating the generalization performance of networks and other classification algorithms.

- Depending on the window size, the number of training pattern types may differ in orders of magnitude, but also the representativeness of the training set for the problem space (the space of possible and actually occurring patterns) may differ radically. Some information about this can be gained from the average ratio between types of patterns and the number of instances they have in the training set (*type-token ratio*).
- Different pattern sizes may result in different amounts of *ambiguity* in the training set (i.e. the proportion of pattern types for which contradictory decisions can be found in the training set).
- Since a word is transformed into a number of patterns, some test pattern types may be contained in the training pattern set, because of partial similarity 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. We will call this *overlap*.

Table 2 lists number of types, type-token ratio (number of types for each token), percentage of ambiguous pattern types (i.e. patterns with contradictory classifications), and overlap between training and test set (in percentage of pattern types) for three different pattern sizes.

These results show that with increasing pattern length, the training pattern type set becomes increasingly less representative for the problem space (the space of possible patterns, for this problem 26^k, where k is pattern size). Cues for this decreasing representativeness are a.o. a strong increase in number of pattern types, decreasing overlap with test set, and increasing type/token ratio. Ambiguity of the pattern types is already near minimal at pattern size 7. This seems to suggest that increasing pattern size further would not necessarily lead to increasing generalization performance (noise is absent at pattern size 7).

3.3 Output Analysis

The activation of the single output unit in our network architecture is interpreted as a decision on the insertion of a hyphen before the target position of the input pattern that is fed to the input layer: 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.[4]

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 placed hyphens and incorrectly hyphenated words. To analyse the actual hyphenation performance (as opposed to the 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. Two adjacent incorrectly placed patterns count

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

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 syllable boundary. 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 value 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 one 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.

3.4 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). We start with a short summary of network parameters that we decided not to change systematically after some initial experimentation.

3.4.1 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 to 2 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 less incorrectly placed hyphens.

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. More usual values (such as 0.2 for learning rate and 0.9 for momentum) resulted in lower performance and generalization rates.

Length of training. Because of the ambiguity of some training patterns, a network will never converge to an error of nearly 0.0, but to a somewhat higher error. Usually it took about 300 to 400 iterations or epochs to reach that level. The lowest error on *test* material is reached much earlier: due to overfitting and overgeneralisation on the training material, which already starts to play a role after a few epochs, the network often performs best on test material after 50-100 iterations.

3.4.2 Effect of Window Size

In spelling, average syllable length is 4.3 graphemes. To determine the optimal window size, we first determined the importance of each side of the patterns separately. We trained a network on patterns which had only a right context and another network on patterns which had only a left context (using a context of four characters). We obtained an error of 51% incorrectly placed hyphens using left context, and 35% using right context. These results show that the right context contains more information useful in hyphenation but that it is not sufficient for the task. 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) and in our own experiments with exemplar-based generalization to be discussed shortly. It is consistent with the maximal onset principle.

As expected from the analysis of training and test set, window size 7 (3-1-3) produced optimal results. The average *phonological* syllable length in Dutch is 2.8 phonemes. In a different set of simulations (van den Bosch and Daelemans, 1992) in which we tested syllabification of *phoneme* representations instead of orthographic representations, we even found that window size 5 (2-1-2) turned out to produce better results than window size 7. For that task, the optimal trade-off between coverage of the problem space by the training set and ambiguity of the patterns lies at window size 5.

3.4.3 Effect of Network Architecture

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, we 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 implicitly 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. This training set was considerably smaller than the one we used in our primary simulations. 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 3 (hyphenation results without post-processing) displays the results of the comparison simulations. We performed an addition analysis

	Error	Morph	Non-morph	Type 4 errors
Backp	16.2	36.7	63.3	110
Elman	16.5	36.3	63.7	114
Jordan	19.1	41.6	58.4	133

Table 3: *Hyphenation performance (number of incorrect hyphenations), and error type analysis for Backprop, Elman and Jordan architectures. Mean results for four simulations with each architecture.*

on the error types made by the three networks. Instead of a decrease in the number of type 4 errors, the results show that Jordan networks also perform worse in this respect than backprop networks.

3.5 Combination of Network Solutions

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 overall error, tends to 'overhyphenate'. If it were possible to somehow combine the solutions of A and B, their shortcomings might be partly 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 3).

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.

The main advantage of having a top network to solve this decision problem, is that it 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

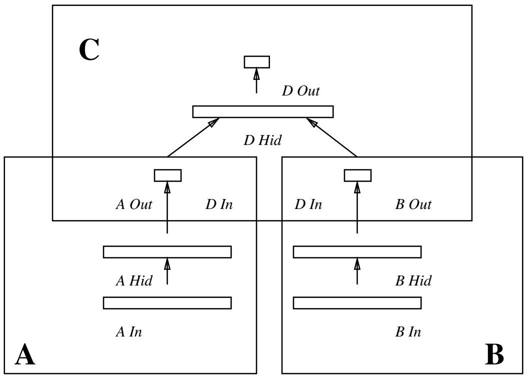

Figure 3: *Composition of networks. The output layers of networks A and B are combined to serve as the input layer of top network C.*

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. During experimentation it also became clear that the best results were not obtained by combining the best networks for each encoding, but by combining well performing networks with slightly erroneous networks.

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. 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 accuracy of both optimization methods turned out to be the same.

In one of the simulations, two different input encodings (the random identity encoding discussed earlier and an encoding representing the sonority of each grapheme as a number) were combined in the input patterns. The accuracy of this method turned out to be better than that of networks using each of the encodings separately. Notice that we introduce a linguistic bias here, in the sense that the sonority encoding is expected (on the basis of linguistic theory) to be useful in finding syllable boundaries[5].

Type	Error pat.	Error hyph.	Post-process.
Single	2.8	9.6	4.7
Combined	2.1	7.2	4.6

Table 4: *Results: performance on test set single versus combined networks. Error on patterns, and on syllables with and without postprocessing is given.*

Table 4 shows the results of the best network trained on an encoding of the identity of graphemes, and the results of the combined network.

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 improvement in hyphenation performance, although it seems that about 96% correctly placed hyphens is the ultimate accuracy threshold for networks of our kind (with postprocessing).

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

[5] Experiments with syllabification of *phonological* representation show a stronger increase of performance when biasing the encodings with sonority information. It is not always possible to assign a clear sonority level to graphemes.

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). In contradiction with our own results, they noted a slightly better result than a comparable architecture without a feedback connection. The network was trained on 1,000 words and tested on 200 words not present in the training set. Their result (an error of 3.2% incorrectly placed hyphenations) 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%.

3.7 Connectionist versus Symbolic Pattern Matching

As a final comparison of the performance of connectionist networks to *symbolic* pattern matching systems, we selected a Dutch text[6] and compared the performance of CHYP (Daelemans, 1989), an approach based on the table look-up method of Weijters (1991)[7], an (as yet) undocumented algorithm PatHyph (Vosse, p.c.), and our best spelling hyphenation network. The results are summarized in Table 5. 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 only. 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 (according to the phonological rules) are provided. For this test, CHYP operated in daring mode.

	Error hyphens	Error words	Morph	Non-morph
CHYP	4.7	8.6	92	8
Table	2.0	3.7	40	60
PatHyph	1.8	3.0	87	13
Net	4.8	9.0	19	81
Net (postp)	3.1	5.8	54	46

Table 5: *Hyphenation performance on a Dutch text of alternative pattern matching hyphenation systems vs. the best hyphenation network (internal combination).*

The table look-up method of Weijters (1991) 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, and is expressed as a set of numbers used to weigh the importance of each position during similarity matching). Using a pattern size of seven, and as weights 1 4 16 4 1, he reported (Weijters p.c.) an error on the test set of 1.66 (error computed on patterns, to be compared with our results in Table 4). Larger pattern sizes (up to size 11) and different weight settings did not significantly improve the score. Interestingly, still with size 7 patterns, weights 1 4 1 already produced a low error on patterns of only 2.93%.

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 with post-processing cannot compete with the best pattern matching approach, confirming our hypothesis. Especially the good performance of the simple table look-up method is surprising, and it incited us to explore this type of *exemplar-based generalization* further, and compare its performance to that of symbolic and connectionist

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

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

pattern matching for this task.

4 EXEMPLAR-BASED GENERALIZATION

The generalization technique which we will call here exemplar-based generalization (EBG) is a variant of statistical classification methods like k-nearest neighbour (see e.g. Weiss and Kulikovsky, 1991), and shares with Case-Based Reasoning (CBR, e.g. Riesbeck and Schank, 1989) and Memory-based Reasoning (MBR, Stanfill and Waltz, 1986) the hypothesis that the foundation of intelligence is reasoning on the basis of stored memories rather than the application of (tacit) rules. In linguistics a similar emphasis on reasoning on the basis of stored examples is present in Skousen's analogical modelling framework (Skousen, 1989; Durieux, 1992).

An EBG system consists of a database of exemplars each with a category assignment (in the case of ambiguous patterns, for each category the frequency of occurrence in the training set is kept), and a metric to compute the similarity between exemplars. An exemplar is a set of features (attribute-value pairs). The training set of our connectionist experiment can be interpreted as a database of exemplars in a straightforward way, with patterns as exemplars (features are positions in the pattern and values the character at that position). When a pattern from the test set is presented as input, it is first looked up in the database. If it is present, the category with highest frequency (in case of ambiguity) is taken. If the pattern is not found in the table, a similarity measure is used to compare the new pattern with each pattern in the table, and of those patterns in the table which have the highest similarity with the new case, the frequencies for each category are summed before a decision based on frequency is taken.

The simplest similarity metric assigns equal weight to all features of patterns when comparing them (*absolute similarity*). Table 6 shows the results for the hyphenation task with different pattern sizes. Training and test set were exactly the same as in the connectionist experiments. Test error is the overall error on the test set (in patterns, to be compared with the results in Table 4). Memory error is the error on database lookup (i.e. the percentage of test pattern types that is explicitly present in the database, but which is incorrectly processed due to an incorrect frequency decision). Generalization error is the error percentage on those patterns in the test set that are not in the pattern database.

Pattern Size	Test Error	Memory Error	Generalization Error
3	8.5	8.2	21.4
5	3.1	1.4	8.4
7	2.4	0.2	4.8

Table 6: *Error on the hyphenation task using a similarity metric based on absolute similarity. Error on patterns is shown.*

Even with this simplistic similarity measure, the approach scores as good as backpropagation learning. We set out to find a more reasonable similarity measure that would be able to assign different importance to different features (not all features are equally important in solving the task). At the same time we wanted this metric to be as domain-independent as possible (unlike the weights assigned by Weijters on the basis of intuitions about the task, or the special-purpose metrics developed in MBR). Our similarity metric was designed by weighing each field with a number expressing its role in decreasing the overall information entropy of the database (an approach inspired by the use of information entropy in ID-3, Quinlan 1986).

Database information entropy is equal to the number of bits of information needed to know the decision (in this case YES or NO) of the database given a pattern. It is computed by the following formula where p_i (probability of category i) is estimated by its relative frequency in the training set. For the training set in this task (with two categories: YES or NO a hyphen), $E(D)$ is equal to 0.78 bits.

$$E(D) = -\sum_i p_i log_2 p_i \qquad (1)$$

For each feature (position in the patterns), it is now computed what the *information gain* is of knowing its value. To do this we have to compute the average information entropy for this feature and subtract it from the information entropy of the database. To compute the average information entropy for a feature, we take the average information entropy of the database restricted to each possible value for the feature. The expression $D_{[f=v]}$ refers to those patterns in the database that have value v for feature f, V is the set of possible values for feature f.

$$E(D_{[f]}) = \sum_{v_i \in V} E(D_{[f=v_i]}) \frac{|D_{[f=v_i]}|}{|D|} \qquad (2)$$

Pattern Size	Test Error	Memory Error	Generalization Error
3	8.3	8.2	10.7
5	2.5	1.4	5.9
7	1.7	0.2	3.4

Table 7: *Error on the hyphenation task using a similarity measure weighed by information gain of features.*

Information gain is then obtained by equation three, and scaled to be used as a weight for the feature in the EBG task.

$$G(f) = E(D) - E(D_{[f]}) \tag{3}$$

In the hyphenation task with pattern size seven, for example, we see the pattern of information gain values of Figure 4. It suggests that the target letter, and even more so the letter immediately following it, should play a primary role in the similarity measurement.

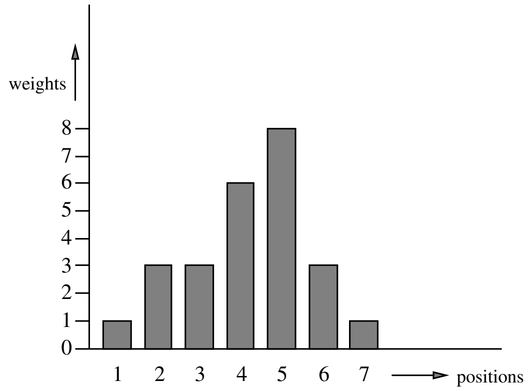

Figure 4: *Information gain for each position in the patterns of the training set. Position 4 is the target position.*

Table 7 shows the improvement when using entropy metrics (to be compared with the results using absolute similarity in Table 6). Notice that performance on memorization stays the same because the similarity metric only plays a role when a pattern is not found in memory.

These results show that a useful similarity metric can be derived automatically from a training set of patterns, obtaining results comparable to more ad hoc metrics based on domain heuristics (as is the case in the work of Stanfill and Waltz, 1986, and of Weijters, 1991). Preliminary results show a poor generalisation performance on the hyphenation task using the similarity metrics developed in Stanfill and Waltz (1986) for grapheme to phoneme conversion, clearly showing the domain-dependence of these metrics.

5 CONCLUSION

For the problem of finding syllable boundaries in spelling strings, solutions using domain knowledge are still superior or comparable in *accuracy* to a connectionist solution, even when the latter is biased with linguistic information. They have an added advantage because of their inspectability and the reusability of developed rules. This suggests that when domain knowledge is available, a connectionist approach may not be the best way to tackle a problem (see also Weijters, 1991).

On the other hand, as far as *efficiency* is concerned, a connectionist approach achieves accuracy levels comparable to a symbolic pattern matching approach automatically, without need for a large amount of *linguistic engineering*, shifting reusability from the acquired knowledge to the acquisition technique. 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.

What is worrying (from the point of view of connectionist research), is the fact that a simple exemplar-based generalization technique with a task-independent information-theoretic similarity measure, achieves better generalization performance than backpropagation in feed-forward networks, even if context memory is available through recurrent links. Further research should make clear whether this result is limited to this particular task.

6 REFERENCES

Bosch, A. van den and W. Daelemans. Linguistic Pattern Matching Capabilities of Connectionist Networks. In: Daelemans and Powers (eds.) *Background and Experiments in Machine Learning of Natural Language. Proceedings First SHOE Workshop.* Tilburg: ITK, 183-196, 1992.

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.

Durieux, G. Analogical Modelling of Main Stress Assignment in Dutch Simplex Words. In: Daelemans and Powers (eds.) *Background and Experiments in Machine Learning of Natural Language. Proceedings First SHOE Workshop.* Tilburg: ITK, 197-204, 1992.

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.

Quinlan, J. R. Induction of Decision Trees. *Machine Learning* 1, 81-106, 1986.

Riesbeck, C. K. and R. S. Schank. *Inside Case-based Reasoning.* Hillsdale, NJ: Erlbaum Assoc., 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.

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

Skousen, R. *Analogical Modeling of Language.* Dordrecht: Kluwer, 1989.

Stanfill, C. and D. L. Waltz. Toward Memory-based Reasoning. *Communications of the ACM*, Vol. 29, 12, 1986.

Weijters, A. and G. Hoppenbrouwers. 'NetSpraak: een neuraal netwerk voor grafeemfoneem-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.

Weiss, S. M. and C. A. Kulikowsky. *Computer Systems that Learn.* San Mateo: Morgan Kaufmann, 1991.