A NEURAL NETWORK THAT LEARNS TO DO HYPHENATION

Bernd Fritzke
Institut für Mathematische Maschinen und Datenverarbeitung
Lehrstuhl für Programmiersprachen
Universität Erlangen-Nürnberg
D-8520 Erlangen, Germany

Christof Nasahl
UB ANL A433-SI
Siemens AG
Gründlacher Straße 248
D-8510 Fürth, Germany

Abstract: Hyphenation of German words is a highly irregular problem. Existing solutions for automatic hyphenation are not very satisfying. We successfully applied a "sequential network" to this problem. The training algorithm was standard backpropagation. The network was trained with a collection of 1000 German words together with their correct hyphenation. In subsequent tests with unknown words we achieved a correctness of 96.8 percent. Analysis of the simulation results indicate, that with further increases of the training data improvements are still possible.

Introduction

Hyphenation is a standard in text processing software. It is needed to prevent the occurrence of wide gaps in the text. In the German language hyphenation is a nontrivial problem, mostly due to the occurrence of compound words. Existing solutions are generally rule based or make use of large dictionaries. In contrary to the English language, the results of such programs are often not satisfying.

Neural networks are able to generalize: In a domain with a certain inherent structure learning a number of examples might suffice to make the network react properly also on unknown inputs of the same domain. Applications of this principle range from speech to text conversion[5] over game playing[6] to truck steering[2].

The difficulties of describing German hyphenation by rules together with the nevertheless inherent structure of the problem suggests, that neural networks might be appropriate here. The general proceeding can be as follows:

1) Training a network with a limited set of words until it is sufficiently able to determine the correct hyphenation for the word corpus.
2) Applying the trained network to unknown words.

The point of interest is, to what degree unknown words can be hyphenated correctly, i.e. how much "generalization" takes place.

1. The Network Architecture

The architecture we used has similarities with the one used by Sejnowski and Rosenberg for NETtalk[5]. It consists of three layers of processing units: input layer, hidden layer and output layer.

The input layer receives the information of 8 adjacent letters (a "window") which are part of a

word. The output layer consists of one single unit, which indicates whether hyphenation is possible or not at the “hyphenation position” between the fourth and the fifth letter of the input window As **hyphenation position** we denote in the following every position between two letters in a word. The complete hyphenation of the word is thus available by shifting the 8-letter-window character by character over the word.
Every word in the training corpus is converted to several pairs of input/output patterns, e.g. the word “ROTATION”, hyphenated “RO-TA-TI-ON” in the German language, is converted to the following four patterns:

	input information	output information
pattern 1:	_ _ R O | T A T I	yes
pattern 2:	_ R O T | A T I O	no
pattern 3:	R O T A | T I O N	yes
pattern 4:	O T A T | I O N _	no
pattern 5:	T A T I | O N _ _	yes

examined hyphenation position

Fig. 1: The input/output patterns generated from the German word “Rotation”

Empty positions in the input patterns are filled with blanks. Letters outside the window are not considered. As in the German language single letters are never hyphenated, we don’t have to regard (and code) the first and the last hyphenation position.
The complete network architecture is shown in figure 2.

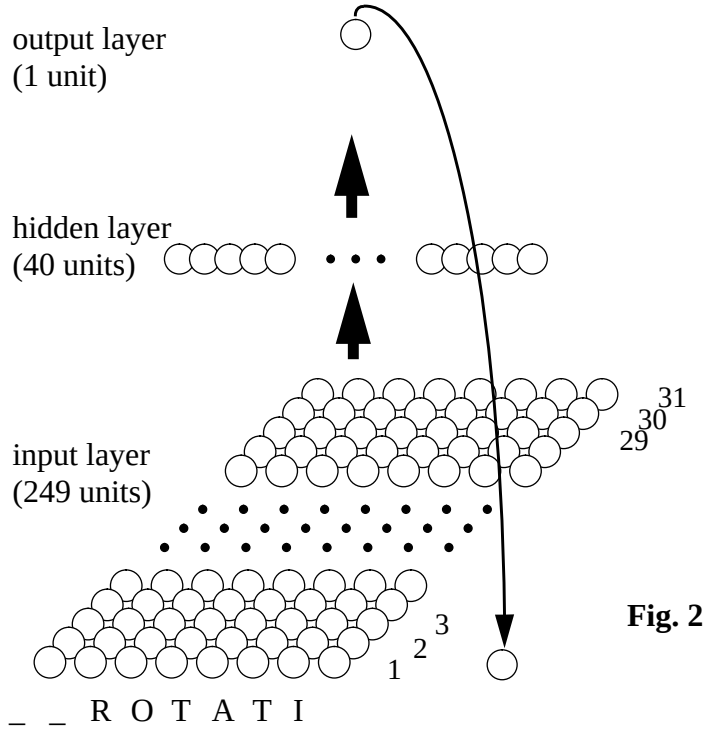

Fig. 2: The Network Architecture

Every character position in the word is coded by *n* neurons with *n* being the size of the alphabet used. In the German language we have the letters a..z plus 4 extra characters. Together with the blank as fill character we have $n = 31$ different letters. To code a letter we set the corresponding neuron to one and the other 30 to zero (1-out-of-n coding). This is done for each of the eight input

positions. Thus $8 \times 31 = 248$ neurons are needed to code the input.
Every input neuron has a connection to every unit in the hidden layer. Every unit in the hidden layer is connected to the output neuron.
There is one single feedback connection going from the output neuron to an extra input neuron. This connection has been introduced to give the network the chance to use the contextual information if hyphenation has been possible at the previous position (For every syllable consists of at least two letters in the German language, adjacent hyphenation positions are not possible!). The feedback connection is realized by copying the previous value of the output neuron to the extra input neuron before every pattern presentation. This type of network is called "sequential network" due to Jordan [1].
We made experiments with and without the feedback connection. Thereby the nets with feedback had significantly (with a significance of 0.06) better results. For this reason we choosed this network architecture for all following investigations.

As learning rule we used standard backpropagation[4] with a momentum. The activation function was the logistic function

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

In the output pattern "Yes" was coded as 1 and "No" as 0.

2. Choosing Word Bases

One important aspect was the choice of the word bases for training and test of the network. We built a word base out of three groups:

- the 325 most frequently used words in the German language (according to [3]).
- 850 words selected from articles in the "Frankfurter Allgemeine Zeitung", a widespread German newspaper.
- 25 words with rare letters or letter combinations.

This makes 1200 words at the whole. Out of these we constituted different training bases W_n with $n \in \{5, 10, 20, 40, 80, 160, 320, 640, 1000\}$ by selecting n words at random. Thereby the smaller training bases are subsets of the larger ones.
As test base W_t we took the 200 words which were not incorporated in W_{1000}.
The words in the word bases were transformed to input/output patterns as described above.

3. Learning to Hyphenate the Training Words

First we trained a network with the standard backpropagation to hyphenate the words in the training bases W_n, $n \in \{5, 10, ..., 1000\}$. One learning cycle consisted of the presentation of all words to the network. The presentation order of the words was random. But for each word the different input/output patterns for that word (m-3 for a word of length m) were presented sequentially. This was necessary to take advantage of the feedback connection.
We performed up to 10 runs for each word base W_n. Fig. 3 shows the convergence behavior of the network.
For each word base is given the number of input/output patterns and the total square error after convergence. The total square error is defined as follows:

$$\text{total square error} = \sum_{i=1}^{\text{number of patterns}} (\text{desired output for pattern } i - \text{actual output})^2$$

base is W_n with n=	5	10	20	40	80	160	320	640	1000
number of I/O patterns	34	60	101	215	468	908	1917	3744	5793
final square error	0	0	0	0.1	1.6	4.7	9	32	39
relative error in %	0	0	0	.046	.342	.518	.470	.855	.673

Fig. 3: Error after learning the different training bases W_n

The relative error is computed as

$$\text{relative error} = \frac{\text{total square error}}{\text{maximal possible error}} = \frac{\text{total square error}}{\text{number of input/output patterns}}.$$

The “maximal possible error” is equal to the “number of input/output patterns” because in the worst case the network would map every input-pattern for which hyphenation should occur on 0 (instead of 1) and vice versa. The error would be 1 or -1 in either case, the square error always 1.

As the table indicates the network is not able to learn the hyphenation perfectly if the size of the training base exceeds 20. In all cases the remaining relative error was below one percent. The error could probably be eliminated by further increase of the hidden layer. However since one training cycle with W_{1000} took already about 44 cpu minutes on our HP-9000/330 workstation we didn’t check that out. Nevertheless the effect we want to show is obvious also with these “erroneous” networks as the next chapter shows.

4. Testing the Network with Unknown Words

Learning the hyphenation of a fixed number of words is not very exciting. This could easily be done with a table. Instead we are interested if the network would acquire some general ability to hyphenate through learning the training words. For this reason we performed various tests with

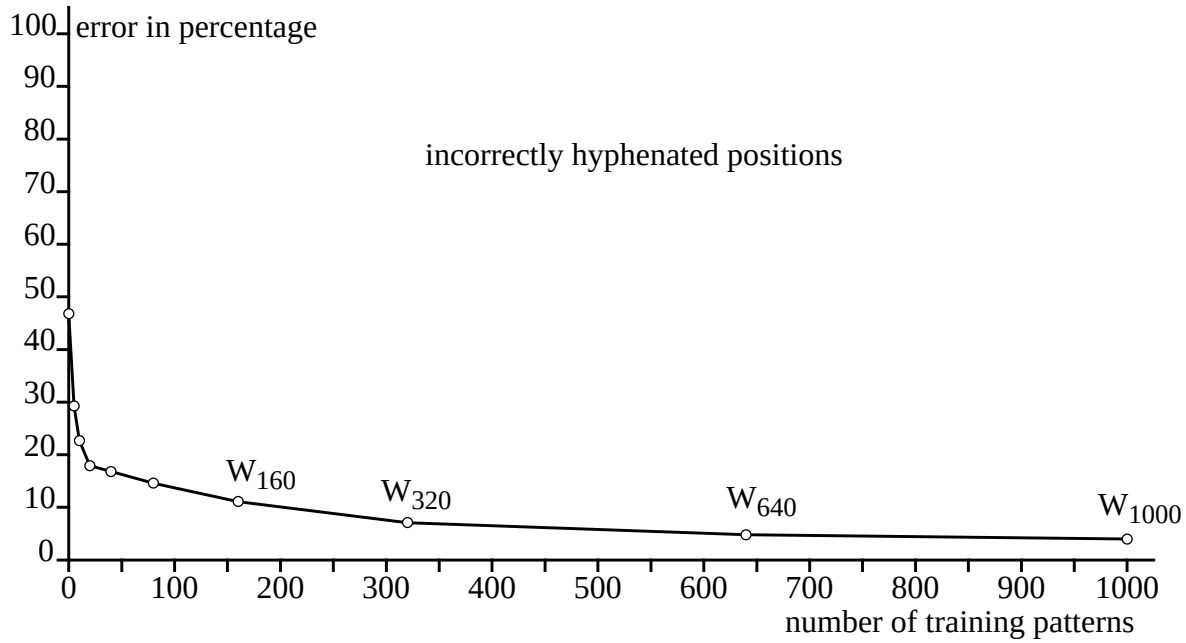

Fig. 4: Hyphenation error for the test base W_t after learning the different training bases W_n

the word base W_t which consists of 200 words which had not been used during the training. Fig. 4

shows the hyphenation error for these new words after training with different word bases W_n. There is a clear correlation between the size of the training base and the hyphenation ability for new words. With a training base of only 10 words 77.8% of the test patterns are hyphenated correctly. This improves to 96% with a training base of 1000 words.
It is interesting to remark, that improvements in the hyphenation of the training words do not always lead to better hyphenation of the test words. In Fig.5 the hyphenation error for W_t is shown in different stages of the training with W_{1000}, the largest training base. While until cycle 25 both the error for W_{1000} and the error for W_t decrease, further training only diminishes the error for the training words.

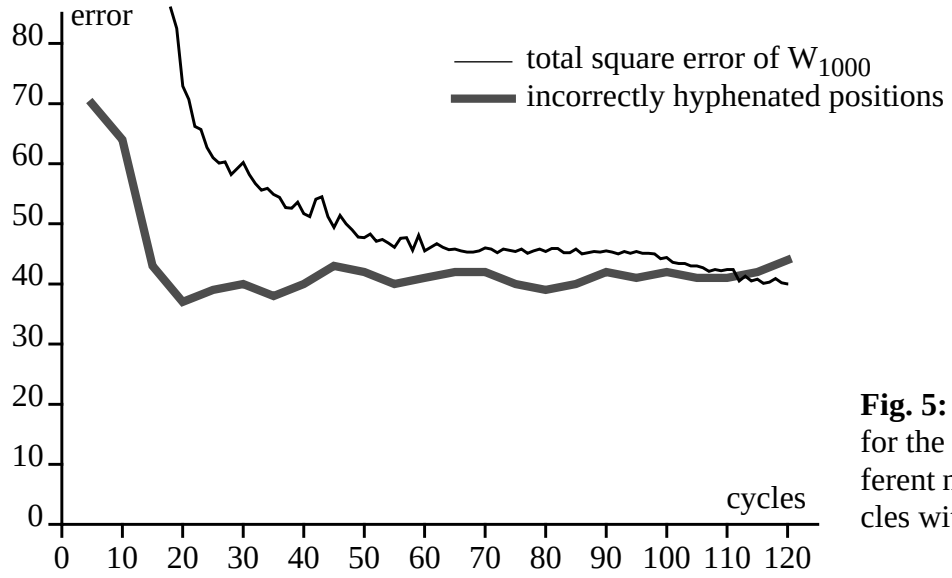

Fig. 5: Hyphenation error for the test base W_t after different numbers of training cycles with W_{1000}

5. Effects of the Hidden Layer Size

To determine the influence of the number of hidden units on the performance of our network, we made simulations also with 10 and 80 hidden units. In each case the training base was W_{1000}. In

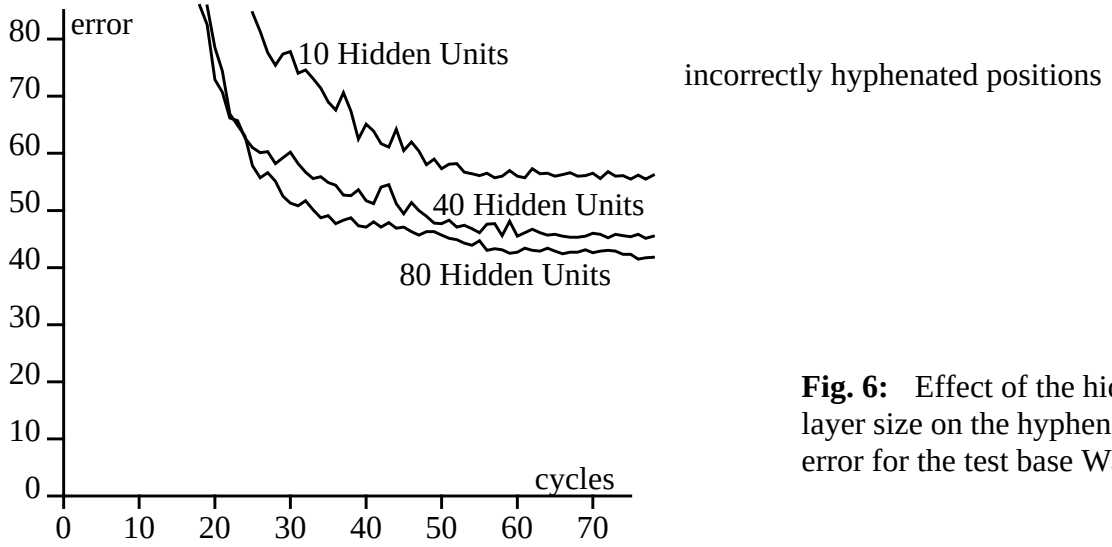

Fig. 6: Effect of the hidden layer size on the hyphenation error for the test base W_t

Fig.6 the square error for the two new networks and for the one with 40 hidden units is shown for a number of training cycles. The convergence seems to be better if the number of hidden units is increased.
The hyphenation ability for our test base W_t improved too. After 35 cycles the 80-hidden-units-

network hyphenated 1024 of 1060 test patterns correctly. This is 96.6% in contrary to 96.04% of the network with 40 hidden units. So the use of larger networks might still improve our results.
On the other hand too many hidden units usually lead to bad generalization:
The training base is learned “by heart” and there is no need for finding efficient representations of the problem. So the increase of the hidden layer beyond a certain point is likely to worsen the hyphenation ability for new words.

6. Conclusions

We applied a sequential neural network with 40 hidden units to the problem of hyphenation in the German language.
In the training phase the network was able to learn 99.3% of the 5793 training patterns which were formed from 1000 German words.
In the test phase the network was able to determine correctly the hyphenation of 96.04% of 1060 patterns formed from 200 German words. During the simulations we made the following observations:

- Increase of the training base leads to better hyphenation of new words.
- Increase of the training time leads to better learning of the training patterns, but after a certain point the hyphenation of new words does not improve anymore.
- Increase of the hidden layer size can improve both the learning of training patterns and the hyphenation ability for new words. (96.8% correct hyphenation of positions in new words with 80 units instead of 96.04% with 40)

So hyphenation with neural networks seems possible. In order to improve our results the word bases used for training should still be larger than the one we used.
One interesting aspect of this approach is the fact that hyphenation for some other languages than German can be done by simply exchanging the training base.

References

[1] 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

[2] Nguyen, D. and B.Widrow, ”*The Truck Backer-Upper: An example of self-learning in neural networks*” Proceedings of the IJCNN, pp.II-357, Washington D.C., 1989

[3] Plickat, H.,”*Deutscher Grundwortschatz*”, Beltz Verlag, Weinheim, 1986

[4] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, “*Learning internal representation by error propagation*” in Parallel Distributed Processing, vol.1, ed. Rumelhart, McClelland, pp. 675-695, MIT Press, San Diego, 1986

[5] Sejnowski, T. J. and C. R. Rosenberg, “*Parallel networks that learn to pronounce English text*” Complex Systems, vol. 1, pp. 145-168, 1987

[6] Tesauro, G. and T. Sejnowski, “*A ‘neural’ network that learns to play backgammon*” First IEEE Conf. on Neural Information Processing Systems, pp. 794-803, Denver CO, 1988