Two approaches to hyphenating Norwegian

Terje Kristensen
Department of Computer Science
Bergen College
E-mail: tkr@hib.no

Dag Langmyhr
Department of Informatics
University of Oslo
E-mail: dag@ifi.uio.no

Bernd Treeck (Bernd.Treeck@terrenviro.no) and
Ronny Falck-Olsen (ronnyfo@student.hin.no)

Abstract

This paper describes some of the problems encountered when trying to hyphenate Norwegian automatically. Two particular approaches are described and compared: the TEX algorithm and the neural network program developed at Bergen College.

Keywords Computer hyphenation, TEX hyphenation algorithm, artificial neural network, Cortex-Pro, back-propagation.

1 Introduction

In theory, hyphenating Norwegian is simple; see for instance the book by Finn-Erik Vinje[8]. The rules are few and reasonably clear; the most important are:

1. Compound words may be hyphenated between the two components: `flagg-stang` and `atmo-sfære`.
2. We may hyphenate after a prefix or before a grammatical conjugation, as in `be-klage` and `skorp-ene`.
3. If a word has several syllables, we may hyphenate it so that the part following the hyphen starts with exactly one consonant followed by a vowel. Examples are `do-ven` and `vans-kelig`. (Note, however, that "`gj`" and "`kj`" may not be split.)

The sequence of the rules is significant: it is better to hyphenate using a rule with a low number.

1.1 Implementing hyphenation

There are basically three ways to implement computer hyphenation, and all three are being used in various systems today.

1.1.1 Logic systems A logic system tries to analyze the word and apply the rules given above. There are both advantages and disadvantages to this approach:

+ The hyphenation algorithm can be quite compact and fast.

– This approach may have problems with a word it has never encountered before.
– The algorithm is complicated to write, and it must be completely rewritten for every new language, as the rules vary extensively.
– Even though they are technically correct, some hyphenations should still be avoided, like bidrag-syter and pils-piss.

1.1.2 Complete word list systems Another approach is to build a list of all possible words in any form in which they may occur:

gutt	gutten	gutter	guttene	gutta
gutts	guttens	gutters	guttenes	guttas

⋮

The main advantages and disadvantages to this approach are the following:

+ This approach usually produces the best results.
– The list of hyphenated words can be huge; for instance, the newspaper *Aftenposten* has one with 950,000 words.
– The algorithm is helpless when it encounters a word it has never seen before, so a backup method is necessary.

1.1.3 Pattern matching systems Yet another approach is to create a set of letter patterns and rules for how to apply these patterns to determine the hyphenation points.

+ The algorithm may be used for any language; only the patterns will vary.
+ Pattern matching usually performs well on words previously not encountered.
– It is impossible to create perfect patterns, i.e. patterns that completely define all legal hyphenation points.

This article describes two such pattern matching systems: the algorithm used in the typesetting program TEX, and a system developed at the College of Bergen using artificial neural networks.

2 The TEX algorithm

TEX[2] uses a pattern matching algorithm developed by Franklin Mark Liang[5]. The principle is easy: For each language one develops a set of patterns on the form

$${}_3\mathrm{o}_6\mathrm{ve}_2\mathrm{r}_3$$

Every odd number indicates a good place to hyphenate the given pattern, and a bigger number is better than a smaller one. Similarly, the even numbers indicate that hyphenation should be avoided. Using this interpretation, the pattern above can be read as follows:

> "Hyphenating as -over or over- is quite acceptable, whereas ove-r is not, and certainly not o-ver."

Finding all possible hyphenation points in a word is then simply a matter of looking at the patterns that apply to the given word, as shown in Figure 1. The maximum value for each position determines the outcome: those with an odd value are the possible hyphenation points.

Since the patterns are stored by TEX in a special data structure called a *trie*, the algorithm is very fast. Hyphenating a Norwegian word takes on the average only 12μs on a Sun Ultra-1, and this varies very little with the number of patterns.

The word:	o v e r t a l e
Pattern 1:	$_2$e r t
Pattern 2:	$_1$l e
Pattern 3:	$_2$o
Pattern 4:	o v$_2$e
Pattern 5:	$_3$o$_6$v e$_2$r$_3$
Pattern 6:	r$_1$t
Pattern 7:	$_1$t a l
Maximum:	$_3$o$_6$v$_2$e$_2$r$_3$t$_0$a$_1$l$_0$e$_0$

Figure 1: Using the TEX algorithm to hyphenate overtale as over-ta-le.

2.1 How to create hyphenation patterns

The hyphenation patterns for a given language may be hand-crafted, but more common is to use the program patgen that comes with the standard TEX distribution. It will read a list of hyphenated words and produce the hyphenation patterns, as described by Lars Gunnar Thoresen[7]. It typically takes 20 minutes on a Sun Ultra-1 to create a set of patterns from a list of 50,000 words.

Most users will never have to create these patterns themselves. Many sites like CTAN[1] and NTUG[2] contain hyphenation patterns for various languages.

3 The neural network approach

The neural network approach has been studied previously[4] and the results of using such a regime are very promising. The research was based on a back-propagation network and a fairly small database of words. The neural network program was developed in Cortex-Pro which is a general purpose neural network simulation environment that can run standard neural network algorithms or new ones which we develop ourselves. A few learning algorithms such as back-propagation and Kohonen learning[3] are built into Cortex-Pro.

3.1 Back-propagation

In a back-propagation network the processing elements are organized in layers. Neurons in one layer receive signals from neurons in the layer directly below and send signal to neurons in the layer directly above. Connections between neurons in the same layer are not allowed. Except for the input layer nodes, the net input to each node is the weighted sum of outputs of the nodes in the previous layer. Each node is activated in accordance with the input to the node and the activation of the node. The net input to a node j in a layer is

$$S_j = \sum w_{ji} a_i \tag{1}$$

where a_i is output from node i in the previous layer. The output of node j is then given by

$$a_j = f(S_j) \tag{2}$$

1. The *Comprehensive TEX Archive Network* (CTAN) can be found at http://tug2.cs.umb.edu/ctan/.
2. The *Nordic TEX Users Group* (NTUG) has a home page at http://www.ifi.uio.no/~dag/ntug/.

where f is usually the *sigmodal* activation function. In the learning phase we present patterns to the network and the weights are adjusted so that the produced output from the output nodes is equal to the target. In fact, we want the network to find a single set of weights that will satisfy all the (input, output) pairs presented to it.

In general, the outputs $\{a_{pi}\}$ (where p is the current pattern) of the nodes in the output layer will not be the same as the target or desired values $\{t_{pi}\}$. The system error or the cost function for the network is defined by:

$$E = \frac{1}{N}\frac{1}{2}\sum_{p}\sum_{i}(t_{pi} - a_{pi})^2 \tag{3}$$

where N is the total number of patterns. A gradient search should be based on minimization of the expression in Equation 3. The weight updating rule then becomes

$$\Delta w_{ji} = -\alpha\delta_j a_i + \beta\Delta w_{ji} \tag{4}$$

where α is the learning rate and β is the momentum constant. For a more thorough discussion, see references [3, 6, 9].

3.2 Theory

One can formulate a hyphenation problem by answering “Yes” or “No” depending on whether it is allowed to hyphenate between two letters or not. A word consisting of n letters will have $n - 1$ such questions. This is the same method that was used by Brunbak and Lautrup[1]. The neural network is given a certain view or window of, for instance, eight letters. The words are rolling through the window and different possibilities for hyphenation are checked. One is asking if there is a hyphenation between two of the letters in the middle of the window or not. An asterisk (‘*’) defines empty letters to fill up the space. The size of the window may vary. A large window will give fewer conflicts than a small one. A larger window will generally require longer training time and use more memory than a smaller one. The words are rolling through the window and different possibilities for hyphenation are checked. In Figure 2 the word `badedrakt` is presented to the network in a window consisting of eight letters. Normally we will choose a symmetric window, but an asymmetric window with three letters to the left and five letters to the right may also be used.

3.3 Data representation

The Norwegian language has 29 letters from which words are created. To handle the beginning and the end of each word a space is used, indicated by ‘*’. Only one neuron is used to represent a single letter of the alphabet. The active neuron is given by the position of the letter in the alphabet. In such a representation each letter will have the same length. Only one neuron is active at a given moment. The letters are represented as follows (the space character first):

*:	100000000000000000000000000000
a:	010000000000000000000000000000
b:	001000000000000000000000000000
	⋮
ø:	000000000000000000000000000010
å:	000000000000000000000000000001

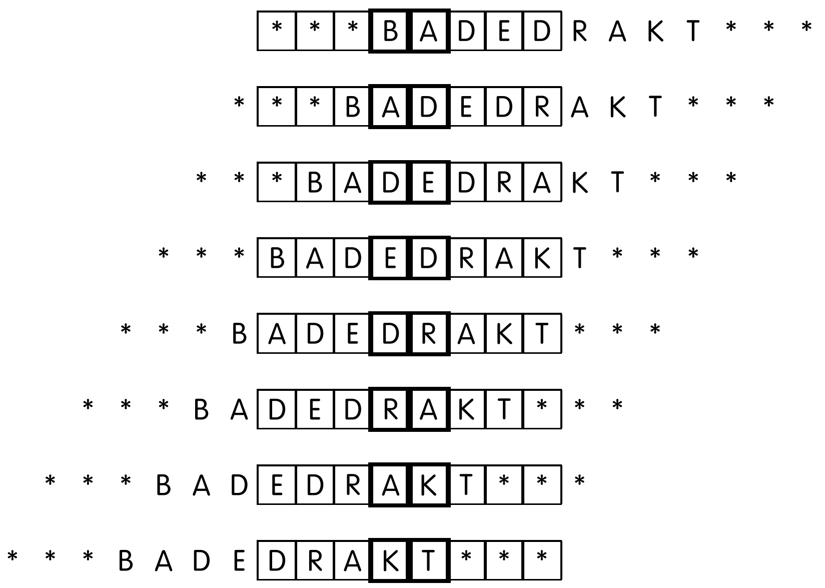

* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *
* * * B A D E D R A K T * * *

Figure 2: In the neural network, the words are rolling through the window and all possibilities for hyphenation are checked.

In the representation above, each letter always has the same number of bits (30). Generally speaking, we see that if the alphabet contains β letters and the size of the window is V letters, the input layer will consist of $V \cdot \beta$ neurons. For $V = 8$, this gives 240 neurons. The output layer consists only of a single neuron. We will accept that hyphenation will take place in the middle of the window if the activity of the neuron is greater than 0.5 ("yes") and not taking place if the activity is less than 0.5 ("no").

In Figure 3 we see how the word TERES*** is represented in the input layer in Cortex-Pro. The image shows the last part of the word in-ves-te-re-es where the question about hyphenation is placed between the two letters in the middle, indicated by a vertical bar, like TER | ES | ***.

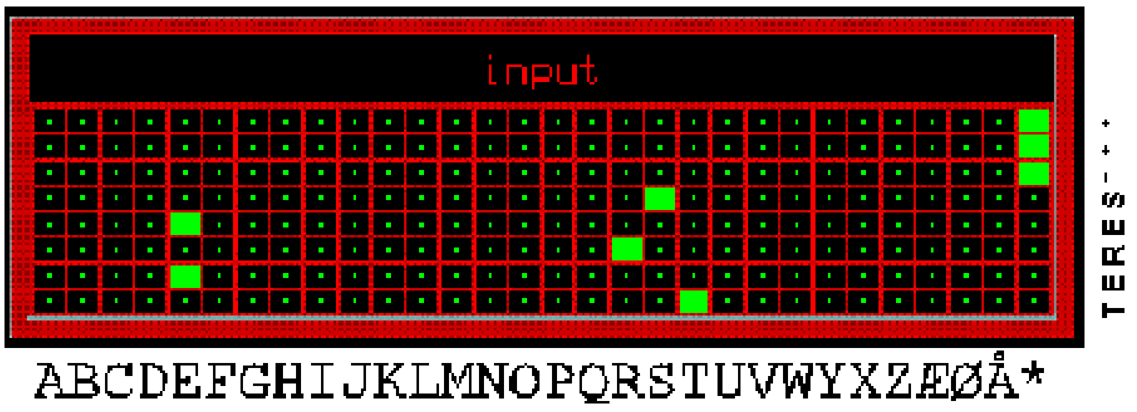


Figure 3: The input layer for the word TER | ES | *** in Cortex-Pro.

3.4 *Training*

Each training pattern in Cortex-Pro will consist of nine lines, eight for the representation of the letter and one for the target. The output node can only have two values: 1 (“yes”) and 0 (“no”) to indicate whether hyphenation between the two letters in the middle is allowed or not. The training pattern for the word picture ***|SE|KVE is given underneath. This is only one example of a training pattern. The data represents the picture of the word ***|SE|KVE and no hyphenation between S and E.

```
000000000000000000000000000001
000000000000000000000000000001
000000000000000000000000000001
000000000000000000100000000000
000001000000000000000000000000
000000000001000000000000000000
000000000000000000000100000000
000001000000000000000000000000
0
```

In a training session, we used a learning regime where the learning rate was changed during training, from 0.3 to 0.05 at the end. The reason for this was to let the network detect more detailed hyphenation structures during the learning phase. For each pass though the training file, the configuration of the network was written to file. In this way we could train and test the network at any stage during training. A bias was introduced to get a more flexible network and faster convergence.

4 Experiments with the neural network

4.1 *A small-scale experiment*

Our training database for this experiment consisted of about 8000 Norwegian words generated from a dictionary of hyphenations developed by NAVF. To test the ability of the network to hyphenate, we constructed two test files, one of words on which the network had been trained, and one consisting of unknown words from the NAVF list. The back-propagation network was trained with one and two hidden layers. The result produced by the network was written to a file. In this way we could compare the result produced by the network with the correct hyphenations. The experiments were done on a 166 Mhz PC with 32 Mbytes memory.

4.1.1 One hidden layer Different numbers of hidden units were chosen to find the right network topology. The number of hidden units were 10, 15, 25 and 40. The training times were very large so we decided to use only 100 iterations in the training session for all the experiments. The execution times were still quite substantial: about 282 hours (12 days) for a network with 40 hidden units. The global error was reduced by increasing the number of hidden units. This was about 1.9% for a network with 40 hidden units. We also noticed that the network performed better on known words than unknown ones, as was expected. During the experiments, we noticed that the number of correct hyphenations increased with more hidden units. The network was only tested with one hidden layer of up to 40 hidden units. We believe that we have still not found an optimum number of hidden units for this topology.

4.1.2 Two hidden layers Two similar experiments were done with the same training database with two hidden layers. The first and second hidden layers consisted of 20 and 10 and 25 and 15 hidden units, respectively. The training time for these two experiments were 172 hours (7 days) and 205 hours (8.5 days) with a global error of 3.5% and 2.5%, respectively. We had expected that the neural network would have improved performance with one extra layer of hidden units, but the experiments did not show that.

4.2 A large-scale experiment

A new training database containing about 40,000 words was generated from the same NAVF list. This time the training was done on a 400 Mhz PC with 128 Mbytes memory. A neural network configuration consisting of two hidden layers was chosen. We still believed that the network would give better results with such a configuration. The number of hidden units were this time 30 and 15 for the first and second hidden layer. The training time was quite large, about 619 CPU hours (nearly 26 days). The total error for 100 iterations was 3.8%.

After 100 iterations the global error is still too large but further training may reduce it. The network configuration was tested on about 10,000 words from the training database and about 10,000 unknown words. The results of the test from both experiments and the the comparison with the TEX algorithm is given in the next paragraph.

We have used a neural network consisting of two hidden layers. After the experiments we believe that one layer of hidden units is better suited for this kind of problem. Further experiments will give us the answer. The main problem with the back-propagation algorithm is the very large training times.

5 Testing the TEX algorithm

To compare the TEX hyphenation algorithm with the neural network, the same word lists were used to generate the required patterns; one set of patterns was created from the list with 8000 words and another from the 40,000 word list.

In addition, a set of patterns based on the complete list of 67,000 words was made. Because of the enormous execution times of the neural network, this list was tested for TEX only.

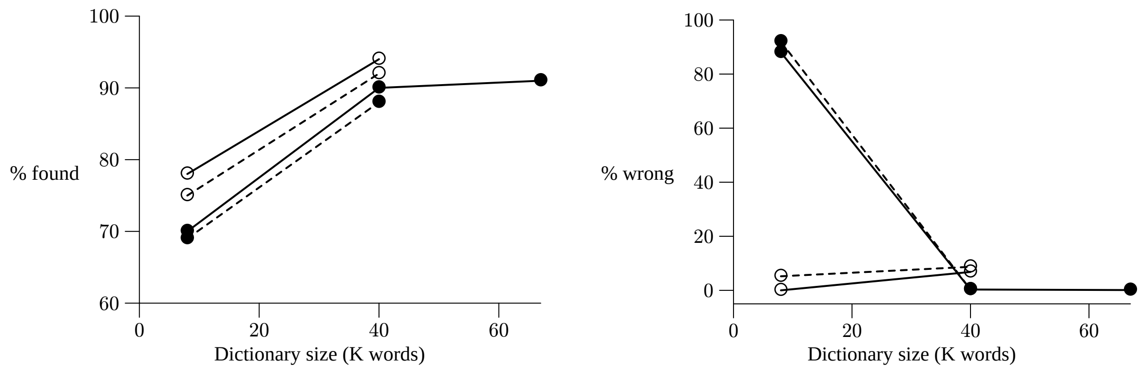
To test the generated patterns, we used them to hyphenate two groups of words: one group containing some of the words used to create the pattern, and another group containing other words from the NAVF list.

5.1 The test criteria

To determine the quality of hyphenation, we looked for two criteria:

1. How many legal hyphenations were found?

   Although this value should be as high as possible, a score of 100% is not essential. Most users will be happy with a system finding sufficiently many hyphenations, for instance 80% or more.



(a) Legal hyphenations found

(b) Illegal hyphenation points found

Figure 4: A comparison between the neural network approach and TEX. The following symbols are used in the graph:

○——○	Neural network; known words
○- - - -○	Neural network; unknown words
●——●	TEX; known words
●- - - -●	TEX; unknown words

2. How many illegal hyphenation points were found?

   This value is much more important than the former, and should be as close to 0 as possible.

5.2 Test results

The results can be found in Tables 1 and 2 and Figure 4, and they seem quite conclusive:

- Using the small dictionary (8000 words), the neural network approach is superior. In fact, the TEX algorithm is completely useless with such a small dictionary, generating 88% illegal hyphenation suggestions.
- With the larger dictionary (40,000 words), the TEX algorithm seems superior. It does not detect quite as many legal hyphenations as the neural network approach (90% vs 94%), but it produces hardly any wrong hyphenation suggestions (0.3% vs 6.8% for the neural network).
- Testing the complete dictionary (67,000 words) with the TEX algorithm seems to confirm the good results, giving only 0.1% illegal hyphenations.
- As can be expected, both approaches perform better on hyphenating the same words as were used to create the patterns, but only slightly so.

6 Conclusion

We have compared the neural network approach developed at Bergen college with the algorithm used by TEX. The results show that the TEX algorithm is still a better choice if you have a big enough dictionary as basis.

Further work remains, however, to explore this more thoroughly. Longer training times and improved neural network configuration may provide better results for this approach.

Dictionary size	**Known words**		**Unknown words**	
	NN	**TEX**	**NN**	**TEX**
8K	3,093 of 3,982 (78%)	2,784 of 3,982 (70%)	2,927 of 3,906 (75%)	2,702 of 3,906 (69%)
40K	23,538 of 25,092 (94%)	22,689 of 25,092 (90%)	23,172 of 25,153 (92%)	22,232 of 25,153 (88%)
67K		122,109 of 134,419 (91%)		

Table 1: Legal hyphenations found by the neural network approach (NN) and TEX.

Dictionary size	**Known words**		**Unknown words**	
	NN	**TEX**	**NN**	**TEX**
8K	1 of 3,093 (0.0%)	2,463 of 2,784 (88%)	152 of 2,927 (5.2%)	2,483 of 2,702 (92%)
40K	1,598 of 23,538 (6.8%)	75 of 22,689 (0.3%)	2,017 of 23,172 (8.7%)	70 of 22,232 (0.3%)
67K		130 of 122,109 (0.1%)		

Table 2: Illegal hyphenation points found by the neural network approach (NN) and TEX.

References

[1] S. Brunbak and B. Lautrup. Linjedeling med et neuralt netværk. *Tidskrift for anvendt og matematisk lingvestik*, 14:55–74, 1993.

[2] D. E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[3] T. Kristensen. *Nevrale nettverk, fuzzy logikk og genetiske algoritmer*. Cappelen akademisk forlag, 1997.

[4] T. Kristensen, B. Treck, and R. Falck-Olsen. Hypehenation by an artificial neural network. In *Norsk informatikkonferanse NIK'97 Voss*. Tapir forlag, 1997.

[5] F. M. Liang. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, Stanford University, Aug. 1983. Stanford report no. STAN-CS-83-977.

[6] J. G. Taylor. *Neural networks*. Alfred Waller limited, London, 1995.

[7] L. G. Thoresen. Virtuelle fonter og norsk orddeling i LATEX. Master's thesis, Universitetet i Oslo, Nov. 1993.

[8] F.-E. Vinje. *Skriveregler*. Aschehoug, 1988.

[9] P. Yoh-Han. *Adaptive pattern recognition and neural networks*. Addison-Wesley, 1989.