

Connectionist Learning of Natural Language Lexical Phonotactics

(*Running head:* Connectionist Learning of Phonotactics)

Ivelin Stoianov and John Nerbonne

Dept. Alfa-informatica, Faculty of Letters, University of Groningen

P.O.Box 716 , 9700 AS Groningen, The Netherlands

Phone: (31-50) 363-5936, e-Mail: {stoianov,nerbonne}@let.rug.nl

Abstract

Connectionist learning of natural language words and their phonetic regularities is presented. The Neural Network (NN) model we employ in this problem is the Simple Recurrent Network, trained with the Backpropagation Through Time (BPTT) learning algorithm. During the training, it was assigned the task of predicting the next phoneme given one phoneme at each moment and keeping information of the past phonemes from a given word in a few context neurons. The phonotactics of the Dutch language was studied among others. The shortcomings of some similar previous implementations are explained and successfully overcome. Among the techniques we employed to achieve the much-improved error rate of 1.1% with monosyllabic words and 3.5% with multisyllabic ones are new methods for network response interpretation, an evolutionary approach in training a set of networks, and the exploitation of the word frequencies in training. Finally, an analysis of the phonotactics rules extracted by a trained network is presented.

Keywords: connectionism, neural networks, SRN, machine learning, linguistics, phonotactics,

1. Introduction.

The connectionist paradigm applied to linguistics does not sound as strange now as it did a few years ago. Probabilistic models – such as stochastic grammars and Hidden Markov Models (HMM) – have been successful, as Neural Networks (NN) promise to be. They have some interesting learning and representational properties, and model some aspects of human intelligence more faithfully than their symbolic competitors.

There is big variety of artificial NN models, but the ones which can be used for linguistic purposes are the networks able to process dynamic patterns, that is, patterns with temporal structure. The network most frequently used in computational linguistics is the Simple Recurrent Network (SRN), developed by J.Elman [*JE90*] in 1990. It is an extension of the most popular NN – the Multilayered Perceptron, with context information provided at the input. This context attaches extra memory for the past static patterns observed at the input.

During the process of language acquisition and processing, a phonological level can be distinguished. It deals with lower-level language events, such as phonemes and syllables. The principles of combination governing how sounds combine is phonology. This paper considers a central phonological phenomenon – phonotactic structure, the allowable combinations of sounds (letters).

The NN used for learning language words and their phonotactics is given the task of predicting the next phoneme to be presented in the input, keeping in mind the previous phonemes, that is, the left context (see Fig.1). Since in general, many phonemes are permitted after a given left context, the more specific task of the NN is to distinguish possible successors by activating the proper neurons.

The main assumption here is that the words of a given language and their structure can be encoded within one connectionist model, and that can be used as a recognizer of words in this

language. After proper training, the network must be able to distinguish words in the training language from random strings, which is a reasonable test of learning the phonotactic structure of the language. The percentage of misclassified strings will result in a classification error. Our SRN model with only 5 to 10 hidden neurons is able to distinguish any multisyllabic language word from general random strings with an error less than 3.5%.

There are a few previous works which have attempted to learn phonotactics with NN's, which will be presented in chronological order. The earliest report is by Shillcock et al [Shillcock93]. The attempt there is to model the phonological space with SRN's. The network there was trained to produce at the output layer, the previous, current and future phoneme, given at the input the present one. The phonemes are encoded in accordance with Government Phonology, in particular using feature-based representation. As the authors write on p.186,

“ Once the network was trained, it was very successful at reproducing the current segment and maintaining the past segment in memory, but was poor at predicting the next segment, as might be expected given that the model only has prior context on which to base its prediction and has no resources for storing discrete representations of words.”

What we claim in the present paper is that memory of the structure of words as encoded in the strength of the connections between neurons is sufficient to predict correctly the possible future phonemes, grounded on the context information. Also, in contrast to their phoneme representation, in the work described here we employ a simple orthogonal representation and the network is expected to develop its own internal representation, having no prior knowledge of the characteristics of the characters.

The research reported by Tjong Kim Sang[TKSang95] consider exactly the same problem we study and moreover, it inspired our research. It was quite successful in its trials to encode

words in Hidden Markov Models, but not in SRN. It was concluded there that the very complex structure of a grammar describing a large number of words (3500 monosyllabic Dutch words) is hardly possible to be learned with SRN. The reason for this was conjectured to be the large number of possible continuations. We will demonstrate that this is really a problem, but we will also describe a method to overcome it. The corpus used there is similar to the monosyllabic corpus we use, so the reported Tjong Kim Sang's 3% error for HMM's misclassification suggests that this is acceptable error for a general stochastic model – the SRN is also a kind of probabilistic machine.

Another report considering connectionist learning the phonetical regularities in Turkish[Rodd97] was published a few months after our first successful experiments. It uses SRN as well, but the goal there is different – to find similar phonemes, formed in the hidden layers during the training. The training corpus is smaller, and no information about the learning and performance are given.

Some other connectionist approaches applied to natural language learning – [Lawrence95,96], [Elman90], [Cleerm89,93], [Schreiber91] among others – consider mainly the grammatical level and some toy-sized problems are reported. As pioneering projects in connectionist language processing, most of them were mainly aimed at comparison between different approaches, including NN's, to language learning. But usually, the attempts at scaling to bigger problems were not successful.

Therefore, another important goal in the research reported on was namely to find out if the SRN has really limited capacity and what NN size is enough to learn the basic regularities in the phonetics of a given natural language. We consider here mainly Dutch, but similar tests were made as well with English, German and other languages.

The next chapter describes the data set used. The the third focuses on SRN's, their

evaluation and some practical comments related to their application. The 4-th chapter consider some advance techniques we developed in order to improve the standard training and SRN interpretation. Analysis of the results and some conclusions can be found in the subsequent chapters.

2. Linguistic data: sources and encoding.

It is not difficult to find the proper data set for evaluation of the ideas we have, especially in linguistics. There are a lot of text corpora for language analysis available on CD-ROM's. One can also find huge amounts of data searching in the Internet. We applied both approaches. The main data set used for training is extracted from the CELEX¹ corpus, which contains a few hundred thousand words from English, Dutch and German languages. This corpus is very suitable for linguistic research, because it has a lot of specific information related to each word, e.g. the number of the syllables in the words, their frequencies, grammatical information, etc. Another data source was the Internet, and it contains words from other languages as well. All the data is orthographically presented. Future experiments will be made with phonetic transcription.

The corpus we used for the basic experiments contains all (4480) monosyllabic Dutch words, as they were extracted from CELEX corpus. It was split to a training corpus (80%) and testing corpus (20%). Another word corpus we extracted contains multisyllabic Dutch words: the first 10,000 most frequent words. It was also split into testing and training sub-corpora.

In order to test how the network would learn the phonotactics of another languages, we

¹ The CELEX corpus is distributed on CD-ROM. For more information see home page <http://www.kun.nl/CELEX/> or Centre for Lexical Information, Nijmegen, the Netheland.

extracted corpuses with the same size containing English and German words.

The characters from the words were presented as input to the NN one by one. At the end of each word, one extra symbol '#' (reading: "end of word") was provided. The character set was reduced to the standard 26 ASCII symbols 'a'..'z'. All characters were orthogonally encoded in a vector of size 27 (one extra for the special end-of-word-symbol).

We selected orthogonal representation because it does not implicitly encode any significant relations among sounds or letters – which the learning should be required to uncover. Also, any non-orthogonal random representation can be transformed to this orthogonal one, and this can be computed with an extra non-recurrent hidden layer in the NN. In addition, this is the shortest orthogonal representation.

The input vectors are encoded as *zeros* at all positions except the one which stands for the particular character number, and which is set to *one*. The output character encoding is similar to the input one, but the *zeros* are replaced with **0.1** and the *ones* with **0.9** in order to increase the effectiveness of the training process (see section 2.4 for details).

An important characteristic of the words for training is their frequencies. The reason why we wanted to use them is that the network should be given realistic input reflecting how many times certain character sequences occur in the language being studied. The frequency determines how often a word is presented to the network during the training. As a result, the network will be trained to respond better to more frequent words, and if it is used to check the correctness of a Dutch text, for instance, it will make fewer errors in total. Still, the difference in the frequencies is too big. Certain words as 'de' occur millions of times more often than some rare words. Therefore, the training frequencies used depend logarithmically on the real ones. This reduces the range of appearance to maximum 15 times in one training epoch.

The result of the training was tested with the 20% of the corpus mentioned above and with

randomly generated strings. The desired behaviour was that the NN would accept all words belonging to the language, and reject all random strings. These random strings were automatically generated and all of them were tested for similarity to words from the trained language. Those, which were within one-character *Levenshtein* distance [Sankoff83, Nerbonne96] to a true word were filtered out.

3. The connectionist model

In this section, a detailed description of the connectionist model we have used – SRN – will be given, as well as technical details related to its implementation. The problem with the large number of successors, observed in [TKSang95] is overcome by better interpretation of the network response. Subsection 2.3 gives details about it. First, however, we introduce some general characteristics of connectionist models and the representational spaces which emerge during their training.

3.1. The Neural Networks - some attractive properties.

We would briefly recall that knowledge and data in the NN are represented distributively [Rumelth86]. The data is sparsely represented as the current activation of a group of neurons, usually organised in ‘layers’. These activations are propagated from layer to layer. There is always an input layer, whose neurons are activated in accordance to external data, and an output layer, whose activations are the product of the network. The knowledge, or the rules of how to process the data is refutantly encoded in the NN as strength of the connections between the neurons. Each neuron is responsible for more than one "rule" and in case damage occurs, the rest of the neurons still are capable of producing correct response. Moreover, they can quickly learn or find out their new tasks, or how to deal with new data. This kind of

representation is the basis for the generalization abilities of the NN, that is, the ability to generate similar output activity for similar input patterns even if they were unseen during the learning. Most of the NN learning algorithms are also geared to this kind of distributed data representation. They build up the strength of the connections in accordance with the natural data clustering (see Fig.7). NN learning algorithms may form "rules" by experience, observing only the data, or they may be told the correct response by a "teacher" (a check on output). The presence of a teacher corresponds to learning an input/output mapping.

It has been proven[Hornik91], that one of the supervised learning algorithms – Backpropagation learning – can approximate any continuous input/output static mapping to any degree of accuracy, by a multilayer neural network, if there are enough hidden neurons. [Doya93] and others have extended this result to include temporal patterns, but using recurrent neural architectures. The more-restricted Simple Recurrent Network was proven [Sperduti97] such able to simulate any frontier-to-root automata, while some other recurrent models as cascade correlation networks and Neural Trees can not. This kind of finite state automata recognizes tree grammars. The phonotactic structure of language words can easily be encoded in such grammar, and therefore we chose this recurrent neural network model for our task.

These nice characteristics attract investigators from different domains to exploit NN models for their specific purposes. But applying NN's is still more an art than a science. The basic algorithms involved in the neuron's activation and training are quite simple, but experience is needed to obtain the best from the nets. The variety of paradigms and parameters involved in the training process raises difficulties with the implementation. Therefore, in most of the reported papers, only toy-sized problems are exploited. The wish to extend results to real-sized tasks runs into a barrier requiring fine NN adjustment. That is why another goal

stated in this paper is to give some practical guidelines how to exploit the NN paradigm in the linguistic domain using large data sets.

3.2. Simple Recurrent Network.

The basic structure of SRN's as originally described by Elman is given in Fig.1, which also depicts the mechanism of a single word processing. The SRN is based on the feedforward Multilayered Perceptron (MLP) and it follows its basic computations. The only difference is the backward, or recurrent connection from the hidden layer to the 'context' layer, which can be seen as a part of the input layer, but fed with internally produced information. This context layer is used to keep the activation of the hidden layer at the previous moment.

The initial state of the context neurons is set to zero. In this subsection, the computations related only to the temporal processing will be given. A full description of MLP's and SRN's can be found in [Haykin94],[EW96].

The forward pass for the hidden layer is computed in accordance with (1,2):

$$(1) \quad \mathbf{net}_i(\mathbf{t}) = \sum_j \mathbf{w}_{ij}^h \mathbf{in}_j(\mathbf{t}) + \sum_k \mathbf{w}_{ik}^h \mathbf{cn}_k(\mathbf{t}) \quad \text{for } j=1 \dots |\mathbf{InputLayer}|, k=1 \dots |\mathbf{ContextLayer}|$$

$$(2) \quad \mathbf{hn}_i(\mathbf{t}) = \mathbf{f}(\mathbf{net}_i(\mathbf{t})) \quad \text{for } i=1 \dots |\mathbf{HiddenLayer}|$$

where $\mathbf{net}_i(\mathbf{t})$ is the sum of activity provided at the input of hidden neuron \mathbf{i} at time \mathbf{t} and $\mathbf{in}_j(\mathbf{t})$, $\mathbf{cn}_k(\mathbf{t})$, $\mathbf{hn}_i(\mathbf{t})$ are the activation of the \mathbf{j} -th input, \mathbf{k} -th context, and \mathbf{i} -th hidden neurons

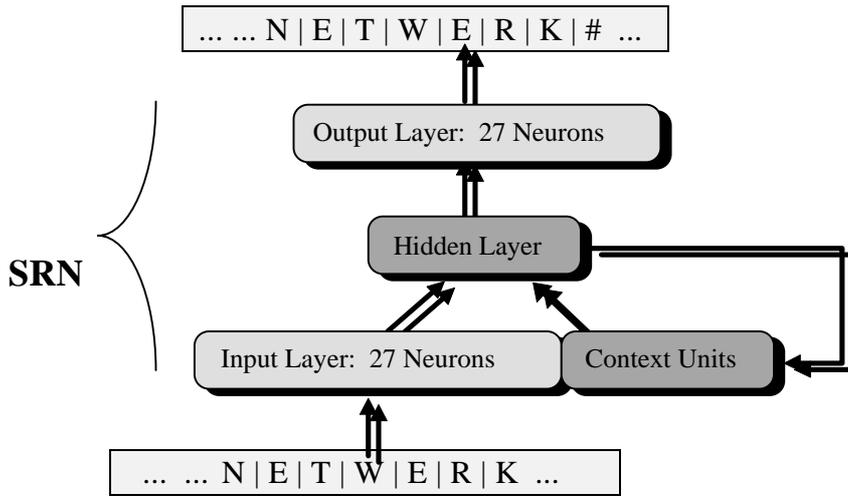


Fig.1. Simple Recurrent Network and mechanism of sequence processing. A character is provided at the input of the SRN and the next one is used for training and in turn, it has to be predicted during the testing phase.

at time t and w_{ij}^h , w_{ik}^h are the weights of the connections between j -th input neurons and i -th hidden neurons, and k -th context neurons and i -th hidden neurons. For convenience, the bias is encoded as an extra input neuron with constant activation $\mathbf{1}$. The activation function $f()$ is of sigmoidal type - logistic or hyperbolic tangent. After the activation of the hidden neurons, their activity is copied to the context neurons in accordance with (3):

$$(3) \quad \mathbf{cn}_i(t) = (\mathbf{1}-\beta) \mathbf{cn}_i(t-1) + \beta \mathbf{hn}_i(t-1) \quad \text{for } i=1..|\mathbf{HiddenLayer}|$$

The transfer coefficient β is advised to be between $\mathbf{0.2}$ and $\mathbf{1}$. The SRN can be trained using both standard BackPropagation algorithm(BP), and Back Propagation Through Time(BPTT) learning. Both of them allow the network to settle to a good point in the weight space, but the second one is much faster, and allows the network to make easily the dependencies between the context, current input and the goal. The difference is that the weights receive delta- or

error-signals not only from the errors in the previous layer, but also from the propagated error in the context units. Therefore, we recommend that the BPTT be used. It differs from the BP only in the training of the weights of the hidden layer, so we will recall the formulae (4,5), that describe the BPTT:

$$(4) \quad \delta_i^H(\mathbf{t}) = \mathbf{f}'(\mathbf{net}_i(\mathbf{t})) \left(\sum_j \mathbf{w}_{ji}^o \delta_j^O(\mathbf{t}) + \sum_k \mathbf{w}_{ik}^h \delta_k^H(\mathbf{t}+1) \right) \quad \text{for } i = 1 \dots |\mathbf{HiddenLayer}|,$$

$$j = 1 \dots |\mathbf{OutputLayer}|, \quad k = 1 \dots |\mathbf{ContextLayer}|$$

$$(5) \quad \Delta \mathbf{w}_{ij} = \eta \sum_t \delta_i^H(\mathbf{t}) \mathbf{n}_j(\mathbf{t}-1) \quad \text{for } t = (t_0 + 1) \dots (t_0 + |\mathbf{word}|),$$

$$i = 1 \dots |\mathbf{HiddenLayer}|, j = 1 \dots |\mathbf{InputLayer}| + |\mathbf{ContextLayer}|,$$

where $\mathbf{n}(\mathbf{t})$ is the joined vector containing both $\mathbf{in}(\mathbf{t})$ and $\mathbf{cn}(\mathbf{t})$. The deltas for neurons for hidden and output and context layers are: $\delta_i^H(\mathbf{t})$ and $\delta_j^O(\mathbf{t})$. The second sum in (4) represents the context layer delta-term $\delta_k^C(\mathbf{t}+1)$, computed by back-propagating the delta $\delta_i^H(\mathbf{t}+1)$ through the weights connecting the context neurons to the hidden neurons.

We note that (4) has to be computed for an earlier time cycle. Also, the weight updating (5) is performed after presenting all characters from a given word and computing the deltas for each time step at which a given word was processed.

A momentum-term also can be applied with (5), which increases by at least twice the training speed. This momentum term helps the net to escape the local minima, which it can meet quite often, running over the error surface. Moreover, highly complex cases like this produce very complex error surfaces, so it is recommended to use all available means in order to achieve the optimal minima for the task at hand.

One might demur that some tricks are not biologically plausible. This is true, but this kind of NN is not biologically plausible at all. The presence of an exact teacher at such low-level is quite unlikely in the human brain. The NN which can be attractive for cognitive modeling, have to be sought among the self-organising models, such as models based on basic Hebbian learning, competitive learning (Kohonen Maps), etc. After all, even with biologically

implausible aspects, NN's are more faithful to human processing than most competing models. (See [Elman96]).

3.3. Evaluating the SRN behaviour. Optimal threshold.

As we said in the introduction, applying NN's is something of an art. The interpretation of neuron responses is a good example of this. How should we decide if the joint response of the output neurons $\{ \mathbf{O}_{C_2}(t_0), \mathbf{O}_{C_3}(t_0+1), \dots, \mathbf{O}_{C_L}(t_0+L-1) \}$ over the whole testing string $[c_1 c_2 \dots c_L]$ accepts it as being a part of the trained language, or not?

The very first ideas put forward were the following: Elman's [Elman90] implementation accepts a sequence if the responses of all the neurons standing for the expected tokens in the sequence have activation greater than 0.3. Tjong Kim Sang's [TKSang95] network accepts a string if all corresponding neurons are more active than the minimal activation value encountered in all training words. The problem with the first approach is that it is very task-specific. The value 0.3 is just experimentally observed: it splits optimally the strings belonging to the simple grammar, from the non-grammatical ones. Applying this method to a task where the possible successors are 27 would fail, because only few of the neurons would have activation larger than 0.3. T.K.Sang met this problem and applied the second rule, but it naturally accepted not only the correct words, but almost all random sequences as well.

The first solution we proposed and used to good effect uses the same principle as in the previous approaches: the activations of the neurons standing for the expected characters/phonemes $\mathbf{O}_{C_i}(t_0+i-1)$ are compared with a threshold \mathbf{t} . If all of the relevant neurons have activation greater than or equal to \mathbf{t} , the string is accepted as being enough close to the words from the training corpus, that is, as phonotactically like the words learned by the SRN

during training. In this case, \mathbf{t} should be between zero and one. In the above notations, the index \mathbf{i} stands for the number of the letter \mathbf{c}_i from the tested word, that is, it has temporal meaning. Since all phonemes from the tested word have to be confirmed by the NN, we can call this rule “all-or-nothing”.

Now, the question is how to find this threshold at which to divide grammatical and ungrammatical input sequences. After complete training, or after each training session, when we want to evaluate training until this moment, we examine the response of the net for all words from the training corpus, and for some randomly generated strings, or words from other languages. The network training is supposed to adjust the

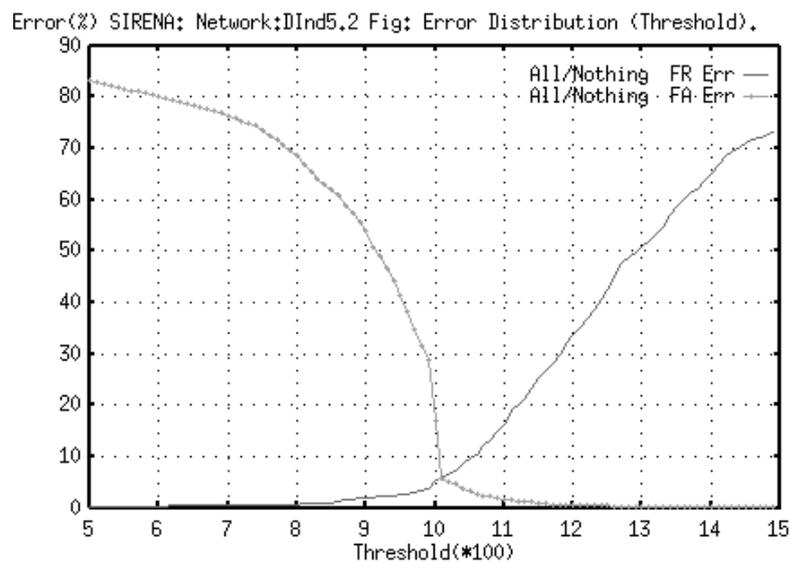


Fig.2. The *falsely rejected word error*(FRE) and *falsely accepted random strings error*(FAE).

The errors (in %) are given against the threshold \mathbf{t} (0.5..0.15).

weights to have a better response for words from the training corpus, and a small error for the testing set. We can test how many words will be accepted and rejected for different values for

the threshold, and to select the one which will minimise the error formed from random strings acceptance and true words' rejection.

Fig.2 depicts both *falsely rejected word error(FRE)* and *falsely accepted random strings error(FAE)*, where the x-axis stands for threshold, and y-axis represents the error, dependent on the threshold. We can see that with the threshold increasing, FRE increases too, because more words are rejected, and on the other hand, FAE decreases, because fewer random strings are accepted. At the point where the two errors are minimal is the threshold which is most interesting. It will be used for character acceptance. The corresponding error at this point is used also for evaluating the training. However, depending on the problem, one of the errors could be preferred and selected to be smaller than the other one.

The method being proposed here is independent of the number of successors, so it can work with 26 characters, as well with, for instance, 50 phonemes, or with some 1500 Japanese characters.

3.4. Implementation details.

The system, described here is implemented in C++ on HP-UNIX. Both BP and BPTT training algorithms were used in order to find the optimal training algorithm. See table 1 for differences. Other training algorithms designed for temporal processing, such as the Temporal Backpropagation Algorithm and Real-Time Recurrent Learning[Haykin94] can not extract any more temporal information, than the BPTT, while BPTT is more suitable for discrete tasks, such as the one we have here.

The training data is organized in a corpus, containing the relative frequencies of each word. The training is split into epochs, during which all words are used for training, but randomly selected. In one epoch, the more frequent words are presented more often to the

SRN. The number of presentations of one word depends logarithmically on the actual frequency, as it is described in section 2.

The random strings used for SRN evaluation, are generated following the structure of the words being used for training, that is, the length and characters are selected following the observed distribution of the training word lengths and the frequencies of each letter.

Orthographic word representation is used. In the Dutch language, the orthographic representation is phonetically motivated, so when we consider this language, the results for both representations will be similar. Each word is presented letter by letter to the input layer. For each letter an input neuron is assigned, which is activated in case its letter occurs. The target activation encodes the letter which follows the current one. Again, one neuron stands for each possible letter, and also one neuron stands for the special 'end-of-word' symbol, which predicts that the current letter is the last in the word. For implementational convenience, the same neuron is attached to the input layer as well. So, 27 input neurons were used, as well as 27 output neurons.

The target activations for each letter are either **0.1** or **0.9**, which stand respectively for non-active and active neuron. The reason for this encoding is that the standard logistic function is being used as an activation function - see (2). To be more specific, its derivative is $f'_i(\text{net}) = \text{on}_i(1 - \text{on}_i)$. So, if the activation of any output neuron approaches **0** or **1**, the weight updating would be quite small (see formulae (4) and (5)) and the training would be slower. Also, the neurons must operate mostly in the linear region of the output function in order to avoid premature saturation - a problem which is described as temporary trapping of the network output units at saturated activation levels (**0** or **1**), during the early stages of the training process[Vitela96].

Experiments to determine the size of the hidden layer were made with 3 to 30 hidden neurons. The size of the context layer is the same as the hidden one. Surprisingly, the variant with 5 hidden neurons achieved the best error rate. The experiments with bigger hidden layer were not as successful, which is strange - a bigger network should have more representational abilities. But such a network meets the same premature saturation problem due to another reason: the very complex weight surface. That is why the SRN with a smaller hidden layer finds the optimal solution faster and more easily. A possible solution to this problem, in case there is a demand for bigger hidden layer, is using some algorithms which gradually increase the number of hidden neurons[Stoianov94].

At the beginning, the weights for each neuron are initialized randomly with regard to the so-called *fan-in*[Haikin94] of this neuron, that is, the number of its inputs. More precisely, the weights were initialized with a random value uniformly selected from the interval:

$$[- 1.5 / fan-in_i \dots + 1.5 / fan-in_i]$$

This range is selected again to avoid premature saturation, but also to provide reasonable initial training speed. The learning rate was found to be optimal at $\eta=0.3$. Both BPTT and BP were tested with and without momentum term α . The former case was significantly better and faster. The optimal value for α was found to be 0.5. Experiments with different values for transfer coefficient β were also conducted. The case with $\beta=1$ is the same as forgetting the previous context state. It was found, that varying β has no big impact on the training.

3.5. Basic experiments:

SRN training evaluation is performed during training after each few epochs (in our experiments - after 3). It is aimed at managing the training and at seeking the optimal threshold. The corpora being used are the training corpus and randomly generated set of strings. The results of this training can be used as final results in case the task does not need generalization, that is, to perform equally well on data similar to the training one. Fig.3 depicts namely this error.

The **SRN testing** verifies the generalization abilities of the trained NN with testing corpus, which is extracted in advance from the available data. Random strings are generated and examined again, to find how the trained SRN classify these sequences. Both corpora are tested at the optimal threshold found during the training.

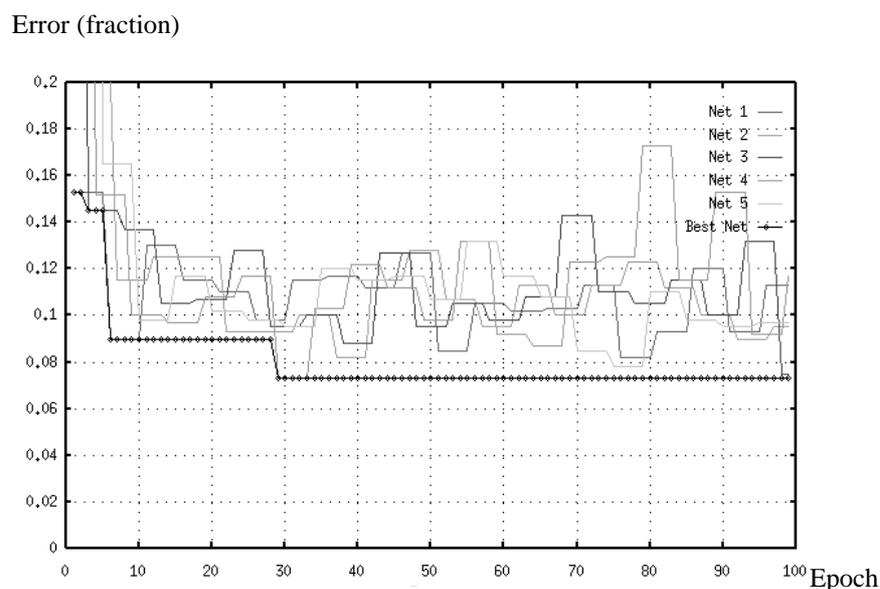


Fig.3. SRN training results with BPTT for 5 independently trained networks. The lowest line gives the errors rate for the best trained network, so far.

Experiments were made with two types of corpora - monosyllabic and multisyllabic words. The monosyllabic words are shorter and have mean length 5 characters, while the mean length of multisyllabic words is 8. The longer words increase the probability of a single character

missprediction which causes the corresponding word to be wrongly classified. This conclusion was confirmed during the tests. The training with monosyllabic Dutch words resulted about 7.5% error, while the performance with multisyllabic words was worse. - 15% of them were wrongly classified.

It was also interesting how the two learning algorithms we described in 3.2 compare. In table 1, an extraction of the training and testing errors for both BP and BPTT learning algorithms are presented. The expectation that BPTT would study better the data was correct. This algorithm makes a weight-updating step after analyzing all dependencies in one training word, rather than only two neighboring steps as BP do, therefore it performs better.

The test errors we achieved were much better than the previous reported experiments, but still, it was far from the desired performance. The ideal one should distinguish without any error words belonging to the language from random strings. The directions in which we have sought improvement was the training process and word evaluation method.

Table 1. Training errors for BP and BPTT algorithms. The tests during the *training* are performed with the training monosyllabic subcorpus, while during the *test* - with the test subcorpus.

Epoch/Test Number	Error (%)	
	BP Algorithm	BPTT
5	18.2	15.0
10	16.9	9.1
20	15.4	9.1
50	11.5	7.8
100	10.0	7.8
200	9.8	7.3
test	10.2	7.5

Another funny test we made just to check the importance of the backward loop was to ‘cut off’ this loop; of course, the performance dropped to about 50% false word rejection.

4. Advanced neural networks techniques.

In the previous section, we saw that the stochastic behavior of the training process, as noted also in many books and papers, might create difficulties in the search for global minima in a very complex error surface. Also, the learning multisyllabic words was not as successful as learning monosyllables. Therefore, we had to search for methods to improve both training and network response interpretation. The first subsection presents an optimization of the learning process, inspired by the processes in natural evolution. In the following subsection, the interpretation of neuron activation as a probability is used as a key to find the best evaluation method. Finally, details about the last SRN model performance are given in Subsection 4.3.

4.1 Evolutionary approach at training Pool of NN's.

As many authors point out [Bengio93], [Gori97], [Haykin94], the stochastic nature of the training process and the complexity of the task doesn't always direct the network toward the absolute minimum, but quite often to local minima. Sometimes, in spite of applying a momentum term, training with coefficient scheduling (see Haykin94 for details), and other “tricks”, the NN can not escape these false minima.

The experiments reported in section 3.5 also confirm that this is a difficult question. The graphics in Fig.3. reflecting the training error do not always tend to a minimum. Even when after a very long training, these networks do achieve an acceptable solution, it is not always the best one, and it takes too much processor time. That is why, for instance, [Lawr95] proposes to generate an initial set of weights, to choose the best of them with respect to the

initial error, and to go on from these. What most investigators do is just train networks a couple of times, on the same problem, and to report the best results.

Searching for an algorithm that supervises NN training and that reduce the incorrect steps made during the stochastic optimisation, we developed an algorithm that follows the basic idea of evolutionary programming [Lankhorst96]. These kinds of algorithms are alternative optimisation techniques which try to emulate Darwin's survival principle of natural evolution. One of their underlying characteristics is that they aim to create conditions in which a steady improvement might occur. For this purpose, they use a population of potential solutions to the problem that is to be solved. An objective function judges the quality of each solution from the population and determines its chance of survival. During a reproduction step, new solutions are produced that inherit the qualities of their parents.

What distinguishes this type of algorithms from the popular genetic ones is that no crossover operator is available. The reproduction phase can include mutations and replications only. This feature makes them more suitable in case the solutions are NN's.

Although, the idea we apply seems quite intuitive, we did not find authors which employ it. The directions we found to be exploited consider mainly architecture optimisation: [Angeline94], but not the training process.

The method we propose trains a set of NN's, as we call it a *pool of networks*, all given the same task, and aiming together at minimal error. It uses an evolutionary-like approach to manage the pool. As in the above described algorithms, we use operators such as replication, deletion, modification and random shift. All of them aim at minimising the errors of the networks in the pool, with exception of the last one - the random shift - which is designed to help the nets and the pool to escape from local minima, in case they are stuck there.

The first two operators - *replication* and *deletion* - are the simplest ones, they just create a copy of a network in the pool or delete a selected one. The second - *modification* - occurs in two cases. The first one is training a selected NN for one epoch. The second case is just a replacement. The last operator - *random shift* - either in a modification of an NN in case the result error is worse than in the previous step, or in a slight massaging of the weights of the network. The strategy of the supervising algorithm that manages the pool is to keep a fixed number of nets improving together with respect to the error position over the weight space. It performs in a loop the following operations:

1. Select randomly a net.
2. Make a replication of the selected net.
3. Modify the copy through one or a few epochs of training.
4. Evaluate the training NN and find the worst network in the enlarged pool and delete it.
5. In case the worst net is the last trained one, with a small probability (about 10-30%) make a random shift.

In addition, network that achieved least error is stored, since, there is some chance to get to a worse position. The pool is initialized with a given number of randomly generated networks. Using this method, the pool "migrates", directed toward the absolute minimum (with regard to the rules of the pool manager) and finally, settles on the minimum, or at least to a good approximation of it.

We compared training with a single net to pool-training, which clearly demonstrated the latter to be faster and more successful. In the following Fig.4, one can see the result of the joining efforts of all nets in the pool, compared to independent trainings (Fig.3).

One could still ask why the pool gets closer to an optimal decision. The answer to this question is based on the diversity which the pool offers to the stochastic-optimization process:

a range of better starting positions in the error surface. Also, after each training step, the pool always keeps the best networks. So, the pool gradually tends to decrease the average error of all nets, still allowing some variation in case it is not possible to find a better position.

In the one-network-case, it is much easier to fall into a local minimum which allows no escape except at the cost of what has been learned thus far. In a pool, the disturbing of one net still doesn't mean disturbing the whole pool, since if random modification to a single net fails, the rest of the networks will quickly restore the 'poor' net to a state no worse than most others.

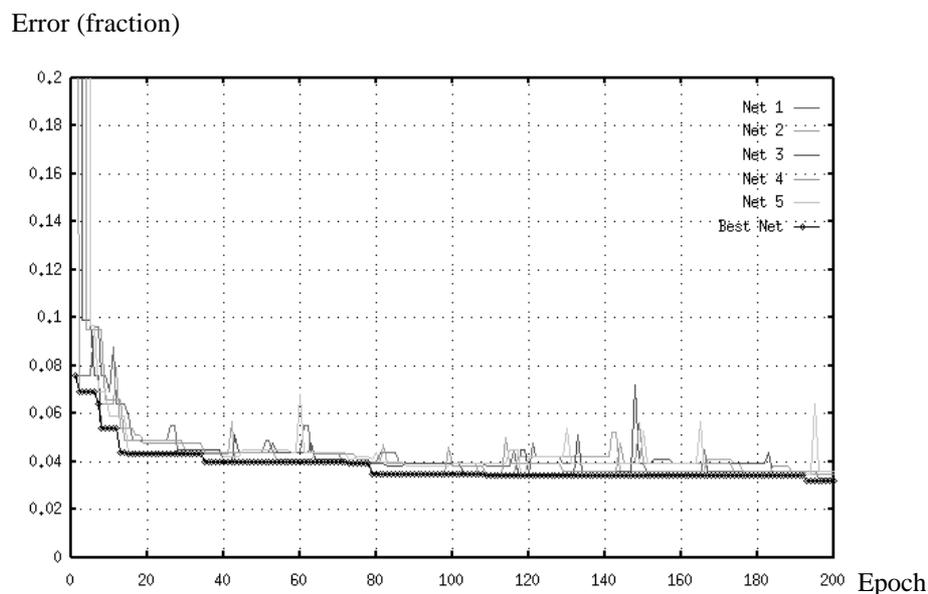


Fig. 4. Training error of 5 SRN's trained in a pool with monosyllabic Dutch words. The lowest line gives the error rate for the best trained network so far.

4.2. Probabilistic approach at SRN evaluation.

In Figure 4, we can see that a network trained in a pool more easily find the optimal position in the weight space, from which to distinguish monosyllabic words from random strings, but the experiments with multisyllabic words didn't achieve this impressive result. One of the reasons for this is the strong dependency of the decision for word acceptance on a single character acceptance. If only one character from a given word is not accepted as being a possible successor to the current context, even due to a very small distance below the threshold, the word would be rejected regardless of how strongly the other characters had been accepted. In longer words, the possibility of a given character failing is much larger. So, many more words are rejected, and there is no way to find a proper threshold in order to split the two testing corpora better using the all-or-none acceptance rule. This was a reason to search for an alternative method, which, moreover should use the neuron activations in the final decision.

Are there alternatives? We propose another rule which tries to get the best from the net, employing a probabilistic approach.

First, we interpret the activation of each output neuron standing for a phoneme or character C_i as the likelihood $P(\text{context} / C_i)$ of the context string provided at the input layer, given this character as a successor. The outputs of the neurons will not always produce activity, which sums to 1.0, so in order to interpret activations as likelihoods, we have to normalise them with respect to the overall sum (6):

$$(6) \quad P(\text{context} / C_i) = \text{on}_i(\mathbf{t}) / \sum_j \text{on}_j(\mathbf{t}) \quad \text{for } j = 1..|\text{OutputLayer}|$$

The *a-priori* probability of each character $P(C_i)$ is simply the observed frequency of C_i which can be encountered during the corpus.

In order to estimate the network's response to the whole string [$C^1, C^2 \dots C^{|\text{word}|}$], we need

to compute the conditional probability $\mathbf{P}^n(\mathbf{C}_i / \mathbf{context})$ of each character \mathbf{C}^n given the context:

$$(7) \quad \mathbf{context} = [\mathbf{C}^1, \mathbf{C}^2 \dots \mathbf{C}^{n-1}],$$

using the Bayesian inversion formula (8):

$$(8) \quad \mathbf{P}^n(\mathbf{C}_i / \mathbf{context}) = \mathbf{P}(\mathbf{context} / \mathbf{C}_i) \mathbf{P}(\mathbf{C}_i) / (\sum_j \mathbf{P}(\mathbf{context} / \mathbf{C}_j) \mathbf{P}(\mathbf{C}_j)) \quad \text{for } j = 1.. 27,$$

and to multiply these probabilities for all characters from the string **Word** , as in formula (9).

$$(9) \quad \mathbf{P}(\mathbf{Word}) = \prod_n [\mathbf{P}^n(\mathbf{C}_i / \mathbf{context})] \quad \text{for } n = 1.. |\mathbf{Word}|$$

The problem with $\mathbf{P}(\mathbf{Word})$ is that it strongly depends on the length of the word. We could never find a proper constant which to split properly two corpora. That is why, we use a threshold which is a function of the length. The function which turns out be successful is (10).

$$(10) \quad \mathbf{t}(|\mathbf{Word}|) = \mathbf{t}_0^{|\mathbf{Word}|},$$

where \mathbf{t}_0 is a threshold which we will find using the same approach as in 3.3. The word is accepted if the probability estimated in (9) is larger than the threshold function (10).

During the experiments, it was observed that this probabilistic rule is more stable, and that it is easier to find (see Fig.5). The first rule needs a very precise estimation of the threshold – the range where both errors are small is much more narrow, while the second – the probabilistic one – is more indifferent to variation in the threshold. The advantage of this is that we can select one of the errors FRE and FAE to be smaller without increasing the other error too much. We might need such imbalanced error, if we are interesting, for instance, in spell-checking.

The main benefit from this rule is that the long words are better evaluated. In the last experiments we made employing this rule, the much improved error of 3.5% was achieved, which is significant success. The SRN performs more successfully also in monosyllabic corpora, as well. In the corresponding experiments we had only 1.1 % average error.

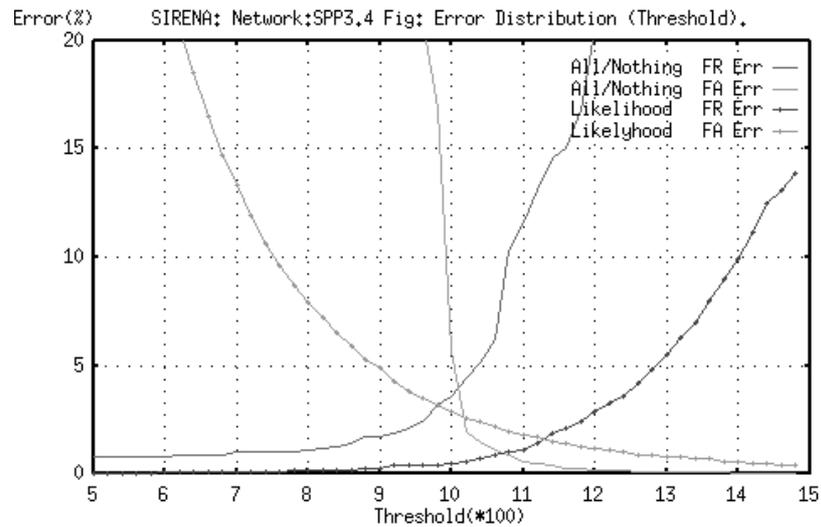


Fig.5 FRE and FAE for both rules interpreting the SRN. The thin lines represent the all-or-nothing rule, crossing at about 4% error ($t=10.2$), while the lines with cross-marks stand for the probabilistic NN estimation, intersecting in this graph at 1.5% error ($t_0 = 11.3$).

4.3. Performance.

Various experiments were made in order to evaluate the performance of the entranced SRN model. The results were repeated in a few independent trials and they are presented in Table 2.

The pooled training outperformed by almost twice the standard training. Both the speed and the minima found were better. The 7.5% training error in the basic experiments with monosyllabic corpus was reduced to about 3%. This improvement is due the better searching strategy of pooled training.

The disadvantages of all-or-nothing evaluation cause instability in determining the category of a long string. The multisyllabic corpora were found difficult even after the pooled training strategy. But applying the probabilistic rule resulted in the much improved error rate of 3.5%.

The basic monosyllabic corpus also was learned much better, missclassifying only 1.1% of the testing words.

Table 2. Training errors for different training and evaluation word strategies, for monosyllabic and multisyllabic words.

Epoch Number / Test	Monosyllabic Words Error (%)			Multisyllabic Words Error (%)		
	Stand.tr. all/nothing	Pool train. all/nothing	Pool train. probabal.	Stand.tr. all/nothing	Pool train. all/nothing	Pool train. probabal.
5	15.0	5.5	2	30	15.1	4.6
10	9.1	4.7	1.9	18.3	12.4	3.8
20	9.1	4.2	1.4	15.1	11.8	3.5
50	7.8	3.9	1.4	14.5	9.6	3.5
100	7.8	3.8	1.2	14.3	9.3	-
200	7.3	3.7	1.1	13.1	9.1	-
500	7.2	3.0	-	13.1	8.5	-
test	7.5	3.0	1.12	14.8	9.1	3.4

Tests with multisyllabic English and German words performed similarly. The least error with English words is 4.3% and the same result with German is 3.1%.

The raw material for this analysis is the strength of the connections from the input and context neurons to the hidden ones, as well as the connections from the hidden neurons to the output ones. The layers are fully connected, so they form two weight matrices. One of the most intuitive presentation of these matrices is the Hinton diagram (Fig.6). It depicts each connection as a node whose diameter is proportional to the magnitude of its weight, and whose sign determines the color. White nodes stand for positive, and black for negative weights. In Fig.6a, each column corresponds to an output neuron and each row to a hidden one. Similarly, the columns in Fig.6b stands for input neurons and the rows for a hidden ones. The numbers represent the context neurons.

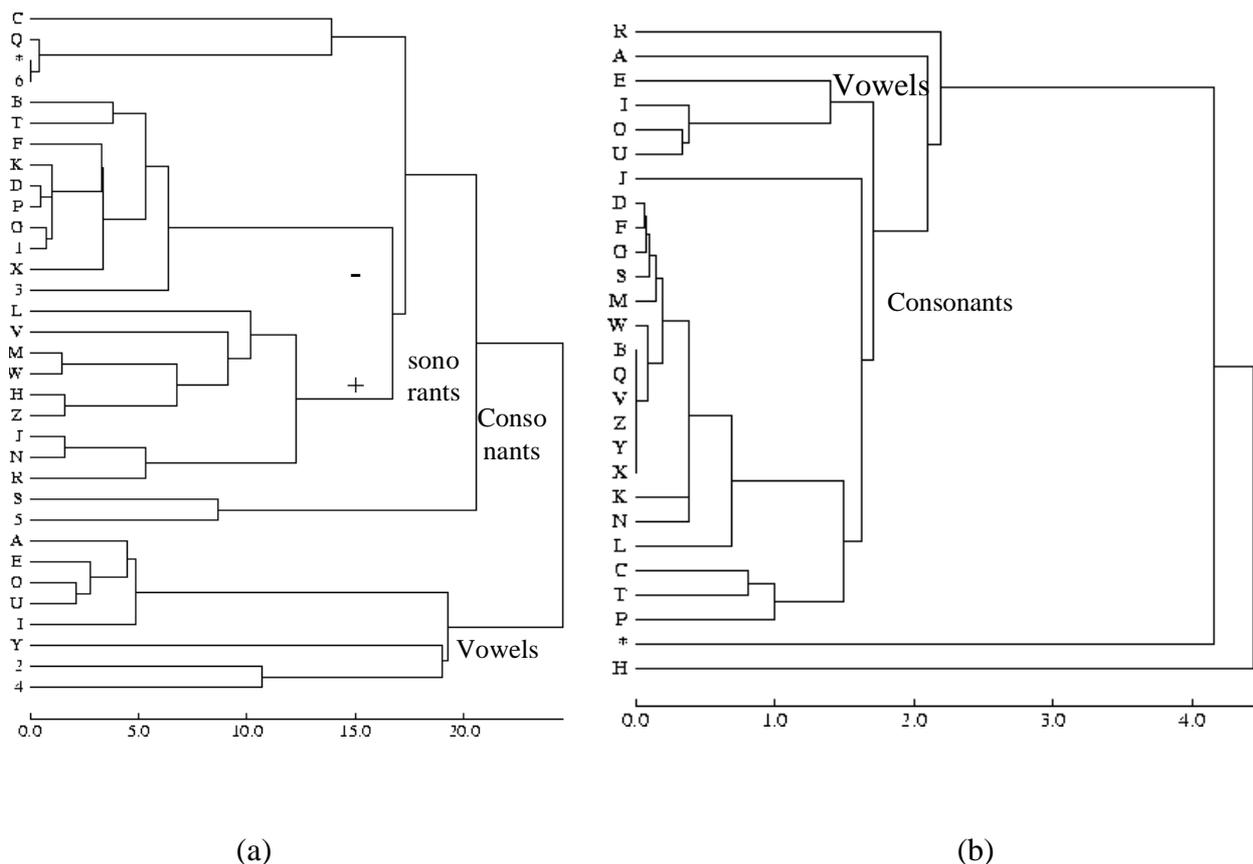


Fig. 7 Cluster analysis of the Input characters (a) and Output characters (b) with respect to weights on the connections to the Hidden Layer. The numbers in (a) represent the context neurons.

It seems difficult to analyze the row data, even presented in this way, so we provide the results from cluster analysis of the two groups of characters, with respect to their vectors. It will help us to understand how input characters are similar with respect to how they influence the production of output characters, and how the output characters are dependent on the input ones. Fig.7 represents the results of this cluster analysis using dendograms.

First, we can see that the weight vectors (and correspondingly the characters) cluster into known natural classes – vowels and consonants – in both input and output character clustering. The vowels are also split with respect to the feature front (I,E,A) and back (O,U). The consonants are clearly distinguished in the input character cluster with respect to sonority. It is possible to find further natural subclusters, but we must keep in mind that this is not a purely phonetical representation; there are 50% more phonemes than orthographic symbols in Dutch.

Obviously, these classes are formed in accordance with the possible transitions between two neighboring symbols. Now, let us take a close look at the possible transitions in Dutch, as documented in “*Fonologie van het Nederlands en het Fries*”[Cohen72] and to check if these transitions are similar to the rules which SRN finds. A monosyllabic word is of the following form in [Cohen72]:

$$[C_3^f [C_2^f [C_1^f]]] V_1[V_2] [[[[C_1^b]C_2^b]C_3^b] C_4^b]$$

where C_i^f and C_i^b stands for front and back consonants, and V_i stands for vowels. Some of the front and back consonants and V_2 can be missing.

According to Dutch phonotactics, the sonorants can occur only at C_1^f and C_1^b positions, that is, there can not be another consonant which intervenes between C_1^f and the vowel V_1 (syllable peak), or between C_1^b and the end of the word.

What the SRN found is exactly the same: they activate the fifth hidden neuron, which activates only vowels, ‘#’-the end of word symbol and ‘R’, which is an exception. They also inactivate the first and second hidden neurons, which control consonants - another correct restriction.

As for the non-sonorants, they activate the first and third hidden neurons in the input layer (Fig.6a), which in turn allows some consonants to be activated in output layer (Fig.6b). This just follows from the possibility in Dutch that the C_2^f positions may be occupied by these phonemes (we recall that there is similarity between the consonant phonemes and the graphemes, which we used in the experiments).

Another rule which can be found in Dutch is that the consonant at C_4^f position only be ‘T’. This phenomena is reflected in Fig. 6a, where the only consonant activated by the third hidden neuron is ‘T’, and this neuron itself is activated only by consonants.

If we consider the vowels, there is a rule in Dutch that a short vowel cannot end syllables, only a long vowel. But there can be consonants after both long and short vowels. In monosyllabic words, these kind of vowels are mostly represented by a single letter (with single exception of the grapheme ‘A’), while the long vowels have two-character orthographic representations. Therefore, we might expect that SRN should have found this rule, and should allow an end-of-word character to appear after a second vowel, but not after only one. In Fig.6b, we can see that the second and fourth neurons are activated positively mainly by vowels. Also, the fourth neuron is influenced positively by all of the above neurons, and inhibited in case its previous activation was positive. We can see as well that ‘A’ activates the second neuron less than the other vowels and that the bias of this neuron diminishes this activation. Looking now at Fig.6a, we can see that the end-of-word symbol can be activated by the second neuron, not by the fourth, because the negative bias again diminishes the

activation. This means that one predicts with great confidence that there are not any more characters in case only one vowel has occurred at the end of the word, which is also the rule found in the phonology.

6. Conclusions and directions for future work.

If we consider one of the questions put in the introduction, whether there are limitations in the SRN abilities to study a complex task as the phonetical regularities of a natural language, the answer one could infer reading the rest of the paper, is negative. The experimental research we made strongly confirms the theoretical studies, that recurrent network models can approximate any temporal relations, and that in particular, that SRN can model any set of sequences.

Still, the complexity of the task demanded some elaborate techniques in NN training and interpretation. Our research was focused mainly on these problems and we developed methods for more successful network training applying evolutionary algorithms as well as optimizing the decision on word acceptance.

The pooled training proposed in section 4.1 uses an evolutionary approach to training a set of networks. This method speeds up the SRN training, which has a stochastic nature and has a chance of being stuck in a local minimum. The networks in the pool independently tend toward better solutions at each step, but the pool supervisor directs them together toward the best solution, avoiding useless steps.

One of the difficult questions related to SRN's is the interpretation of the network response to a whole sequence. This problem was overcome by applying two different approaches - the classical rule all-or-nothing and a probabilistic one. The second method was

much more successful with longer sequences, because it uses probability estimation, rather than yes-or-no decisions.

In this work, we evaluated the NN phonotactics learning, examining how many strings are wrongly classified. It is a good indication how well SRN learned the underlying word structure, but in future work we propose to examine how the output character distribution after each left context is similar to the real distribution in the language. Also, instead of studying orthography, we shall focus on phonetic word representation. Finally, we shall like to find NN's which combine effective learning with cognitively plausible assumptions.

6. References

- [Angeline94] Angeline, P.J., G.M.Saunders, Jordan Pollack. (1994). An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5, 54-65.
- [Bengio93] Bengio Y., P.Frasconi, P.Simard, (1993). "The problem of learning long-term dependencies in recurrent networks", invited paper at Int.Conf.on NN, San Francisco, IEEE Press.
- [Cleerm89] Cleeremans, A., D.Servan-Schreiber and J.L.McClelland. (1989) Finite state automata and simple recurrent networks. *Neural Computation*, pages 372-381.
- [Cleerm93] Cleeremans, Axel(1993). *Mechanisms of Implicit Learning*. MIT Press.
- [Cohen72] Cohen, A., C.L.Ebeling, K.Fokkema, A.G.F.van Holk.(1972). *Fonologie van het Nederlands en het Fries*, Martinus Nijhoff, The Hague.
- [Dorffner91] Dorffner,Georg.(1991).“Radical” Connectionism for Natural Language Processing in *Proc. of Spring Symp. on Connectionist NLP*, Standford,CA, pp.95-106
- [Doya93] Doya, K. (1993). Universality of fully connected recurrent neural networks. Technical report. Univ.of California, San Diego.

- [Elman90] Elman, Jeffrey L.(1990). Finding structure in time. *Cognitive Science*, 14, pp.179-211.
- [Elman96] Elman, Jeffrey L., Bates E.,Johnson M.H.,Karmiloff-Smith A.,Parisi D.,Plunket Kim. (1996). *Rethinking Innates (A connectionist perspective on development)* , A Bradford Book, The Mit Press.
- [Gori97] Gori Marco. (1997). Optimal Learning in Artificial Neural Networks: A Theoretical View. Proc. of Summer School on Rec.NN, Italy. (in press)
- [Haykin94] Haykin, Simon.(1994). *Neural Networks*, Macmillan College Pub.
- [Hornik 89] Hornik, K., M.Stinchcombe and H.White. (1989). "Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, pp.359-366.
- [Lankhorst96] Lankhorst, Marc M., Genetic Algorithms in Data Analysis, Ph.D. thesis, Univ. of Groningen, The Netherland.
- [Laver94] Laver, John. (1994). *Principles of phonetics*, Cambridge Univ. Press,
- [Lawrence95]Lawrence Steve, C.Lee Giles, S.Fong. (1995). On the Applicability of Neural Networks and Machine Learning Methodologies to Natural Language Processing. Technical report UMIACS-TR-95-64 and CS-TR-3479, Univ. of Maryland.
- [Lawrence96]Lawrence Steve, S.Fong, C.Lee Giles. (1996). Natural Language Gramatical Inference: A Comparison of Recurrent Neural Networks and Machine Learning Methods. *Connectionist,statistical and symbolic approaches to learning for Natural Language Processing*, Springer-Verlag, pp.33-47
- [Nerbonne96]Nerbonne John, et al (1996). Phonetic distance between Dutch dialect. In G.Dureux, W.Daellmans & S.Gillis (eds) *Proc. of Computational Linguistics in the Netherlands*, pp.185-202
- [Reber76] Reber A. (1976). Implicit learning of synthetic languages:The role of the instructional set. *Journ.of Experimental Psychology:Human Learning and Memory*,2.
- [Rodd97] Rodd, Jenifer. (1997). Recurrent Neural-Network Learning of Phonological Regularities in Turkish, Proc. of Int.Conf. on Computational Natural Language Learning, Madrid, July, 1997. pp. 97-106.
- [Rumelh86] Rumelhart, D.E., G.E.Hinton and R.J.Williams. (1986). Learning internal representations by error propagation. In David E. Rumelhart and James A. McClelland, (ed). *Parallel Distributed Processing*. volume 1. The MIT Press,

Cambridge, MA.

- [Sankoff83] Sankoff, David and Joseph Kruskal (ed) (1983) *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*, Addison-Wesley Publs, Canda.
- [Schreiber91] Servan-Schreiber D., A.Cleeremans and J.L.McClelland. (1991). Graded state machines: The representation of temporal contiguities in simple recurrent networks. *Machine Learning*, pages 161-193.
- [Sperduti97] Sperduti Alissandro, (1997). "On the Computational Power of Recurrent Neural Networks for Structures". *Neural Networks*, Vol.10, No.3, pp.395-400.
- [Shillcock93] Shillcock, Richard, Joe Levy, Geoff Lindsey, Paul Cairns, Nick Chater, (1993). Connectionis Modelling of Phonological Space, In Mark Ellison & J.Scobbie (eds), *Computational Phonology*, pp.179-195.
- [Stoianov94] Stoianov I.P, I.Baruch and E.Gortcheva. (1994). Off-line Signature verification by Neural Network pressure analysis". *Proc. of Int.Conf. NNACIP'94*, pp.74-79
- [TKSang95] Tjong Kim Sang Erik F. (1995). The Limitations of Modelling Finite State Grammars with Simple Recurrent Networks, *Proc.of the 5-th CLIN*, Enschede, the Netherlands, pp.133-143.
- [Vitela97] Vitela, Javier and Jaques Reifman (1997) Premature Saturation in Backpropagation Networks: Mechanism and Necessary Conditions, *Neural Networks*, Vol.10 No.4 pp.721-735
- [Waibel89] Waibel, Al.et al (1989). Phoneme recognition using time-delay Neural Networks. *IEEE Trans. on Acoustic, Speech and Signal Process.*, ASSP-37, pp. 328-339
- [Wan97] Wan, Erik and F. Beufays, (1996). Diagrammatic Derivation of Gradient Algorithms for Neural Networks, *Neural Computations*, 8.1, pp.182-201.