# AUTOMATIC GENERATION OF N-BEST PRONUNCIATIONS OF PROPER NOUNS

*Neeraj Deshmukh, Joseph Picone*

Institute for Signal and Information Processing (ISIP)
Mississippi State University, Mississippi State, MS 39762

Correspondence:
Neeraj Deshmukh
Department of Electrical and Computer Engineering, PO Box 9571
Mississippi State University, Mississippi State, MS 39762
Email: deshmukh@isip.msstate.edu Phone: (601) 325-8335 Fax: (601) 325-3149

EDICS: SA 1.3.1

## ABSTRACT

The problem of proper noun recognition is key to developing pervasive voice interfaces in applications such as directory assistance and data entry for telecommunications. It is a challenging problem because a large number of proper nouns do not follow typical letter-to-sound conversion rules and their pronunciations are influenced by numerous sociolinguistic factors. The recognition system needs to generate accurate pronunciation networks for correct recognition of such words. Yet, despite being a seemingly intractable problem, humans do amazingly well at generating and recognizing the pronunciation of a name never before encountered.

This paper presents an algorithm based on a Boltzmann machine type of neural network that generates the most likely pronunciations of a proper noun from the text-only spellings of the name. This system does not require any voice data containing the spelling or nominal pronunciation. The generated pronunciation output can be used to build better acoustic models for the proper noun that result in improved recognition performance. The document also describes a corpus comprising more than 18000 proper nouns (surnames) along with 24000 pronunciations developed to train and evaluate the system. While conventional rule-based text-to-speech (TTS) systems provide a single pronunciation for the name, the Boltzmann machine system is perhaps the only one capable of generating multiple pronunciations possible for the proper noun.

## 1. INTRODUCTION

A critical aspect of voice-driven interfaces is their ability to perform accurate recognition of proper nouns. In many applications, particularly those related to medicine [1, 2], the ability to recognize a physician's or patient's name is crucial in providing a usable interface. A comparable problem involving company names and product names exists in voice interfaces for advanced telecommunications services. It is also well-known that a majority of errors in a continuous speech recognition system consist of proper noun words.

Recognition of proper nouns requires an ability to generate accurate pronunciation networks. This problem is very challenging because a large percentage of proper nouns, such as a personal names, have no obvious letter-to-sound mapping rules that can be used to generate the pronunciations. It appears to be an open-ended problem that is constantly evolving as a function of numerous sociolinguistic factors. In spite of being a seemingly intractable problem, humans do amazingly well at generating and recognizing the pronunciation(s) of a yet unencountered name.

Traditional systems for this application require handwriting detailed phonological rule sets and lookup tables to generate an accurate pronunciation of a name. For instance, a commercial product called DECtalk [3] was developed in the mid-1980s that converted unrestricted English text into speech. Such systems were found to be highly labor-intensive and limited in scope to the directory assistance application for which they were initially designed. Since most proper nouns have a number of highly probable pronunciations which can be rarely differentiated from the context of the application, a rule-based system generating only the single-most likely pronunciation essentially attempts to solve an ill-posed problem. For a speech recognition application it is important that all plausible pronunciations be available to the system.

An alternative approach to the rule-based systems is to use massively-parallel network models [4, 5]. Knowledge in such connectionist systems is distributed over multiple processing units and the net exchange of information between these units determines the behavior of the network. Systems like

NETtalk [6] have been fairly successful in capturing most of the significant regularities of the English pronunciations. Recently, systems that employ Hidden Markov Models [9] or decision-tree based statistical modeling [10, 11, 12, 13] have shown the ability to generate more than one pronunciations for regular words. However, no system has succeeded in effectively modeling the peculiarities of proper nouns to generate multiple pronunciations.

In the late 1980s a technique for voice recognition of proper nouns using text-derived recognition models [7] was proposed and subsequently patented at Texas Instruments. The proposed algorithm automatically derives recognition models from the text-only spellings of the name (no voice data) and relies on a particular form of neural network designed to generate multiple outputs for a given input — a Boltzmann machine [6]. It transforms its input to a network of distinctive articulatory features [8] required to produce various pronunciations of the name. However, this system was never implemented.

Besides the implementation and evaluation of this algorithm, we have also addressed the serious issue of an overall lack of proper noun databases by developing a comprehensive training database of almost twenty thousand representative surnames and their many likely pronunciations (using the Worldbet phonetic transcription) to support training and development of the system, as well as application to other speech recognition problems.

## 2. NEURAL NETWORKS FOR PATTERN CLASSIFICATION

Artificial neural networks (ANNs) implement complex operations using massive integration of individual computing units, each of which performs an elementary computation. The output of this basic unit, a neuron, represents a nonlinear transformation of its inputs. Individual neurons link through weighted connections to form various computing machines capable of parallel operation and adaptive learning. Adaptation takes the form of adjusting the connection weights in order to achieve the desired input-output mapping. An advantage of using ANNs for feature extraction and pattern classification is their capability to capture the inherent functionality of the data without any *a priori* statistical characterization or

parameterization [14, 15].

### 2.1. Layered Networks

Multilayered neural networks form a significant class of pattern classification networks. In such systems the external inputs are fed to an initial ‘layer’ of neurons. The remaining cells constitute successive layers which receive as inputs the weighted outputs of the previous layer. The outputs of the final layer are the external outputs of the network. A network can also exist with only a single layer. Two architectures of layered neural networks prevalently used as classifiers are —

**Feedforward networks**: A feedforward or nonrecurrent ANN is one for which neurons in one layer are strictly connected only to those in the immediate next layer. There is no feedback from the outer layers. Such an architecture is also called a multilayered perceptron (MLP) [16, 17]. These require supervised training i.e. the network is updated with regard to achieving the expected output pattern.

**Learning vector quantizers**: A learning vector quantizer (LVQ) [18] is a single layer ANN which automatically adjusts its weights so that input patterns similar in some sense produce similar outputs. These are usually trained in an unsupervised fashion.

### 2.2. Hopfield Network

A Hopfield network [19] contains pairs of neurons interconnected through symmetric weights. During training the neurons update their weights asynchronously based on their local connections with other neurons. The network views each global state as a *hypothesis*. It supports or rejects a hypothesis by assigning positive or negative weights to the associated node connections. Any such global state of the network can be associated with a numerical function called *energy* of the network for that state. This can be interpreted as the extent to which that combination of hypotheses violates the constraints implicit in the problem domain. By minimizing the global energy the system evolves towards interpretations (or classifications) of the input that increasingly satisfy the constraints of the problem space.

Hopfield networks often get trapped in local energy minima via a completely distributed training algorithm. To achieve an interpretation of the inputs that satisfies as many interacting constraints as possible, the system must achieve a globally optimal state of minimum energy.

Both these architectures are limited to a single output per input pattern. To yield multiple possible outputs corresponding to the same input pattern, a stochastic component needs to be introduced in such networks so that the system can generate a different output at different times. Also, it is not possible to reactivate such a network to reach another stable state. A Boltzmann machine is a system capable of these properties and therefore emerges as a viable alternative.

## 3. THE BOLTZMANN MACHINE

A Boltzmann machine is a neural network that is trained to represent the probability density distribution of observables in a particular domain. It is similar to a Hopfield network i.e. we can assign each global state of the network a numerical energy value, and then make the individual units act to minimize the global network energy compatible with each input configuration. However, each neuron has a stochastic activation which is a function of the global energy. Thus each neuron fires with some probability for a particular input pattern; and therefore may produce a different output at different times. It is from this ability to generate multiple outputs for a single input that the Boltzmann machine derives its strength to handle the problem of generating multiple pronunciations of proper nouns. Figure 1 illustrates a typical neuron connection in the Boltzmann machine network and related concepts.

### 3.1. Boltzmann Machine Architecture

A prototype of the neural network architecture for generating proper noun pronunciations is given in Figure 2. It consists of three principal components: an input layer that buffers n-tuples of input letters and maps them to binary-valued inputs, a hidden layer that maps such bit streams into a set of internal states (that derive and store the context-sensitive information regarding the “sounds” such n-tuples produce) and an output layer that mixes long-term and short-term constraints to interpret the groups of letters into a

phonetic representation. The network models the n-tuples of input letters using local and/or long-distance constraints.

The Boltzmann machine training algorithm is a procedure for gradually adjusting the weights on connections between units so that the network comes to model the domain of interest. Once trained, the network can perform pattern completion tasks probabilistically — a subset of its external units are set to values representing the input, and all other units are set randomly. Activations are propagated through the network, with the resulting states of the remaining external units representing the output. If the network has been trained successfully, the set of outputs produced for a given input represents the probability distribution of these outputs for the given input in the domain represented by the network.

The architecture shown in Figure 2 is designed specifically for the problem of name pronunciation/recognition. In a sense, the network is designed to model n-tuples of letters using local and long-distance constraints [20]. To implement this in a manner consistent with a Boltzmann machine, we have devised a shift register structure that is used to buffer characters as they are input to the system one at a time. This approach is similar to other time-delay techniques that have become popular in speech recognition systems [21, 22].

### 3.2. Training Algorithm

The connection weight values associated with the network are derived using the backpropagation training algorithm and a training database of names. The training database contains the spelling and all expected pronunciations of each name. During the training phase, each name in the database is input to the system as a text string. Each output unit is clamped to represent the correct feature sequence, while each input unit is clamped to represent the correct letter combination. The output of the network is compared with this expected output and the weights are updated accordingly. This step is repeated for each expected pronunciation for each letter in each name.

The Boltzmann machine is capable of learning the underlying constraints that characterize a problem

domain given some examples in that domain, simply by modifying the weights of its interconnections to construct an internal model that is capable of producing similar examples with the same probability distributions. By modifying the weights the machine can be made to approach any desired set of probabilities. Since the Boltzmann machine can be viewed as a stochastic version of the multilayered perceptron, the backpropagation method combined with a simulated annealing procedure [24] serves as the most ideal training algorithm for this system. The backpropagation training algorithm is described below in detail for a single context multilayered Boltzmann machine. Figure 3 contains a brief schematic summary of the same.

**Given:** A set of input spellings along with the corresponding phonetic transcriptions.

**To compute:** The set of weights for a network with K multiple layers that maps the inputs onto corresponding outputs.

**Algorithm:**

1. As there are $K$ layers in the network, layer 1 corresponds to the one clamped with the inputs and layer $K$ corresponds to the neuron layers that constitute the system output. Let $N_k$ be the number of neurons in the $k^{th}$ layer. Also, let $N_0$ be the number of bits that are input the first layer of neurons. These bits are the accumulation of the bit-strings corresponding to all the symbols loaded in the input buffer, and hence $N_0$ is fixed once the context size is decided. Let the input bits be denoted by $x_i$, the activation levels of the neurons in the $k^{th}$ hidden layer be denoted as $h_{k_i}$ and the output bits of the $K^{th}$ (output) layer be $o_i$. We indicate the weight connecting the $i^{th}$ neuron in the $k\text{-}1^{th}$ layer to the $j^{th}$ neuron in the $k^{th}$ layer by $w_{k_{ij}}$. $t$ is the index of the number of training loops. $T(t)$ is the temperature in the $i^{th}$ iteration through the training data. Let $\eta$ be the *learning rate* and $\alpha(t)$ be the feedback coefficient or the *momentum* term used to update the connection weights. The learning rate is a fixed constant that characterizes the impact of the output error of a neuron on the weights connected to it. The momentum determines how much

the previous training affects the weight update.

2. For *t = 0*, initialize the weights to random values between -0.1 and 0.1. Set the initial values of the momentum $\alpha(0) = 0$ and the temperature $T(0) = T_0$. The initial temperature $T_0$ is a parameter specified by the user.

For $t = 0, 1, 2, \ldots$

3. For an input vector $\{x_i\}$, the output of a hidden layer neuron clamped to it is calculated in terms of its energy gap —

$$\Delta E_{1_j} = -\sum_{i=0}^{N_0} (w_{1_{ij}} x_i) \quad ; \quad h_{1_j} = \frac{1}{1 + e^{\Delta E_{1_j}/(T(t))}} \tag{1}$$

4. The output of the units in the first hidden layer is propagated through the network to compute the outputs of neurons in the subsequent layers. Thus for all $k = 2,3,\ldots,(K-1)$ —

$$\Delta E_{k_j} = -\sum_{i=0}^{N_{k-1}} (w_{k_{ij}} h_{k-1_i}) \quad ; \quad h_{k_j} = \frac{1}{1 + e^{(\Delta E_{k_j})/(T(t))}} \tag{2}$$

5. Finally, the output bits of the outermost layer are computed.

$$\Delta E_{K_j} = -\sum_{i=0}^{N_{K-1}} (w_{K_{ij}} h_{K-1_i}) \quad ; \quad o_j = \frac{1}{1 + e^{(\Delta E_{K_j})/(T(t))}} \tag{3}$$

6. The output bit-string is compared to the bit-string $\{y_i\}$ that corresponds to the expected or target output phoneme. The error in the system output is computed based on the actual output and the target output. Since this error corresponds to the outermost layer, the error for the *j*th neuron in this layer is denoted as $\delta_{K_j}$.

$$\delta_{K_j} = o_j(1 - o_j)(y_j - o_j) \tag{4}$$

7. The error in the output of a neuron in an earlier layer is computed. The error at the *k*th layer is

calculated by backpropagating the error in the *k+1th* layer and is denoted by $\delta_{k_j}$. For all $k = K-1,\ldots,2,1$ —

$$\delta_{k_j} = h_{k_j}(1-h_{k_j})\sum_{i=0}^{N_k}\delta_{k+1_i}w_{k+1_{ij}} \tag{5}$$

8. The weights are updated using these error values with some feedback from the updates in the previous training pass (see Appendix A for derivation). This feedback is controlled using the *learning rate* $\eta$ and the *momentum* or feedback coefficient $\alpha$. For all $k = K, K-1,\ldots,2,1$ —

$$\begin{aligned}\Delta w_{k_{ij}} &= \eta\delta_{k_j}h_{k-1_j} + \alpha(t)\Delta w_{k_{ij}}\\ \Delta w_{1_{ij}} &= \eta\delta_{1_j}x_j + \alpha(t)\Delta w_{1_{ij}}\end{aligned} \tag{6}$$

9. Steps 3 through 8 are repeated for the next input token. This is continued till all input tokens are exhausted. A complete pass through all the input tokens is called an iteration or an *epoch*.

10. The momentum and temperature parameters are updated for the next iteration through the training data. The momentum term is slowly increased to be small in the beginning and to approach unity as the network runs through more epochs. The temperature is gradually decreased i.e. the system is allowed to cool down as per the simulated annealing paradigm. A most common cooling schedule for such networks follows an exponential function. The cooling exponent $\beta$ is specified by the user to customize the training schedule.

$$\alpha(t) = 1-e^{-\beta t} \quad ; \quad T(t) = T_0e^{-\beta t} \tag{7}$$

11. The machine continues to make passes of the training data till the cumulative mean squared error in the output values drops below a suitable threshold. At this juncture the system is said to have achieved **convergence**. The training may be stopped according to several other criteria as well. These may include stopping the training once some minimum value of the system temperature is reached, or when the largest increment in any of the connection weights falls less than a threshold value etc.

The training algorithm, while applying the simulated annealing technique to an error backpropagation algorithm, essentially implements a gradient-descent kind of error minimization for updating the connection weights. The error function to minimize is a divergence measure [25, 26] between the network energy distributions corresponding to the outputs generated by the network and the desired values for a given input. A mathematical definition of the error function and the derivation of the weight update equations is provided in [27].

This training process is strongly biased towards low energy states at a low temperature, and takes time to achieve equilibrium. At higher temperatures the bias is not so favorable towards energy minima but the learning is faster. A good trade-off is to begin annealing at a high temperature and gradually cool down; performing a coarse-to-fine search for the global minimum.

### 3.3. Evaluation Strategy

The paradigm used to evaluate the system performance is as follows —

1. The Boltzmann machine neural network is loaded according to the specified parameters with trained connection weights.
2. The test word is loaded in the corresponding input buffer(s).
3. The output phoneme string corresponding to this word is evaluated.
4. Steps 2 and 3 are repeated till N-best number of pronunciations are generated. Often the number of total output pronunciations is constrained by the training. In such cases the pronunciation generation is stopped after a certain number of iterations.
5. The pronunciations are sorted in decreasing order of likelihood scores and output to the user.
6. The N-best pronunciations list is compared with the reference list of pronunciations for that word. The system is considered to output an error if *none* of the reference pronunciations feature in the system output list.

Sometimes, all the reference pronunciations will be generated by the system. In other cases, only a fraction of the number of reference pronunciations will appear in the output list of the system. We have termed these cases as *all correct pronunciations* and *some correct pronunciations* respectively. The error case in step 6 above is labeled as the *no correct pronunciations* case. Thus the system error rate is the same as its "no correct pronunciations" performance. We present a detailed analysis of evaluations in Section 5.

## 4. PRONUNCIATION DICTIONARY

A comprehensive database is an essential component of technology development, and we have devoted a significant amount of effort in building one for our task. Previously, numerous data has been collected anecdotally on the problem of alternate pronunciations, but none of this data had ever been incorporated into a publicly available pronunciation dictionary.

Our database currently consists of 18494 American surnames (last names) and a total of 25648 pronunciations. It represents a diverse set of ethnic origins and contains a reasonable mix of common names, names with infrequent occurrence and names that are known to present problems for letter-to-sound conversion because of complex morphology or difficult stress assignments [28, 29]. Each name was transcribed by hand using the Worldbet standard phone set to obtain all the possible correct pronunciations.

The Boltzmann machine network is designed to output a phoneme corresponding to every letter in the input spelling. Since in many cases a single phoneme encompasses a group of letters such one-to-one alignment is not possible. To align the spellings with the corresponding phonetic expression we have developed a dynamic programming algorithm that performs automatic alignment by introducing an *empty phoneme* "_" at appropriate places. For instance, the name 'Wright', where 'Wr' corresponds to a single phoneme '9r' and 'igh' is mapped to 'aI', is transcribed and aligned as '_ 9r aI _ _ t'.

## 5. PERFORMANCE EVALUATION

The performance of the Boltzmann machine system was evaluated on the basis of its capability to

automatically generate accurate multiple pronunciations of the input name. We have followed a simple-to-complex approach in evaluating this system. Our initial experiments with simple alphabet strings provide us with a broad idea of the basic functioning of the system, its dependence on various parameters how the network scales up to a more challenging task. Similarly with real data (proper nouns and pronunciations) we first evaluate on a pilot benchmark of 4 and 5 letter surnames before scaling the system up to handle the larger database.

### 5.1. Calibration Experiments

We conducted a series of basic experiments that involved classification of simple linearly separable character string classes. These helped us in realizing the effect of individual network parameters as well as gaining some insights into the performance pattern of the system. Specifically, the following parameters were the focus of our attention —

- number of hidden layers
- number of neurons in each layer
- effect of shifting letters in the input buffer
- context length

**Two-Class Classifier:** The data used for this set of experiments consisted of 4-letter strings corresponding to the two classes, (e.g. *aaaa* for class 1; *bbbb* for class 2). The training set was created by corrupting such strings in at most two random positions (e.g. *aaxa, bcyb, kaaf* etc.). The evaluation was closed loop i.e. the system was tested on the training data itself.

In the first round of evaluations the system was made to look at the entire 4-letter string and output a class at once in a single step. Since there was no shifting of the input letters the context length was fixed to be 4. The system was trained and tested on 22 strings created as described above. The performance for different number of neuron layers and neurons per layer is given in Table 1. The effect of various parameters on system performance with a single hidden layer and with two hidden layers is displayed in Figure 4 and Figure 5 respectively.

Next, the input strings were fed to the Boltzmann machine buffered through the input shift register. As displayed in Figure 6, a context of at least 5 letters is necessary to completely solve the 2-class problem for data strings of length 4. This is expected as the machine needs a minimum of three letters at each time to map a letter string to a phoneme, and at the end points of the data string (first and last letters) the input buffer can hold 3 letters only for a context length of 5.

With a small number of neurons the error rate as well as the generation rate were high. Thus a large number of spurious pronunciations were generated until an optimal number of hidden layer neurons was crossed. With two layers of hidden neurons the number of neurons per layer required for system convergence was found to be much smaller. The results with a single hidden layer of neurons are summarized in Table 2. The performance as a function of number of hidden neurons for this context length is depicted in Figure 8, and that for the effect of number of training iterations is displayed in Figure 9.

**Alphabet Classifier**: A similar set of experiments was carried out with 26 classes (one corresponding to each letter of the alphabet) to study how the system performance scales up for a larger problem space. The data consisted of pure strings (i.e. consisting of a single letter such as *aaaa* (class *A*), *xxxxx* (class *X*) etc.) with corruption in one or two positions (e.g. *aahad* (class *A*), *vddd* (class D) etc.). It was observed that a large context (length 5) was required to achieve a reasonable classification performance. The system was trained on simple character sequences that corresponded to 26 output phonemes. The results for the pilot runs for fixed-length alphabet strings are described in Figure 7 (cases *i-v*). Performance for variable length strings is depicted as case *vi* in Figure 7 and described in detail in Table 3.

These elementary results show that the context duration is of extreme importance for accurate generation of pronunciations. It can also be deduced that the context duration is a loose function of the randomness associated with the groups of adjacent letters in the spelling of the input word.

### 5.2. Evaluation With Four-letter Proper Nouns

To study the capacity of the Boltzmann machine of generating multiple output pronunciations for a given

input proper noun we devised a pilot experiment that consisted of proper noun data with text spellings of a fixed length 4. As no significant gain in performance was observed for a system using multiple hidden layers compared to that using only a single internal layer, these experiments were carried out only for a single hidden layer system. The training set comprised of 1665 proper nouns and 2022 pronunciations. The results for this evaluation are summarized in Table 4. Even though the closed loop performance of the system differs considerably on this “real” application, there is a significant amount of information that can be gathered from this evaluation.

**Effect of context length:** It can be deduced that as the context length in which each input letter is evaluated (as regards its mapping to a corresponding phoneme) is increased, the performance improves as well. This behavior is described in Figure 10.

**Effect of number of neurons:** It appears that the number of hidden layer neurons has an optimal value at which the performance is maximized, and the error rate increases if the number deviates from this ideal value. This number varies with the type of input data and is therefore difficult to predetermine analytically. Shorter context lengths are more susceptible to changes in the number of hidden neurons than the longer contexts, as evident from Figure 11.

**Training schedule:** The most ideal training schedule for all the experiments performed was found to be the one with an initial temperature of 100 and a temperature decay rate of 0.1 per training iteration. About 60 training iterations were found to be necessary and/or sufficient for reasonable performance.

The rate of generation of pronunciations was found to be loosely bound to the number of neurons and the context length. Also, there was considerable overgeneration of pronunciations compared to the reference pronunciations in the dictionary.

### 5.3. Evaluation On Real Data

The final set of evaluations was an open loop test on the full data set. To achieve a comprehensive

benchmark of performance we divided the complete data set into training and test subsets. This division of data was done thrice to create three overlapping training sets of 15000 names and approximately 20000 pronunciations each. The remaining names constitute the corresponding test set. All test sets were completely disjoint from any of the training sets. The system was trained and evaluated on each of the three data sets for each set of system parameters (such as the number of hidden neurons, number of training iterations and training schedules), thus producing three benchmarks for each case. We found no significant difference in these over any parameter set and therefore present only the overall (average) results here.

The benchmark for a single context system is described in Table 5. Table 6 depicts the performance measured for a system using both the short-term and long-term contexts simultaneously. It can be seen that the performance degrades significantly as the problem is made more complex by introduction of the complete data set for training. In spite of using a large number of hidden neurons and a reasonable number of training iterations, the system proves to be incapable of capturing the underlying letter-to-sound information in the training data. We believe that this breakdown in performance is due to a proliferation of conflicting letter-to-sound mappings inherent in the training data which confuses the Boltzmann machine network. In the previous experiments the training sets were fairly smaller in comparison and hence this effect was not as drastic as it appears now. Also, the network is now presented with text strings of variable length as input. This added dimension of complexity further degrades the network performance.

### 5.4. Computation and Memory Issues

The Boltzmann machine neural network was implemented with an object-oriented data-driven approach in the C++ programming language. The code was highly optimized and the training and evaluation experiments were carried out on a Sun SPARCstation20. The system required on average approximately 160ms to train each proper noun-pronunciation pair per iteration of training and about 5ms for generation of each pronunciation. These figures scaled up almost linearly with contexts length and/or the number of

hidden neurons. This demonstrates a need for faster CPUs to train a reasonably large network in a practical time frame. The memory space required to store the trained network (the connection weights between different layers of the network) was in accordance with the network size.

## 6. CONCLUSIONS

Accurate pronunciation of proper nouns is a key issue in speech recognition and voice interface applications. Since proper nouns follow unconventional letter-to-phoneme transformations, a rule-based generation of pronunciations is not very effective. The Boltzmann machine can generate an N-best list of pronunciations by simply analyzing the text spelling of a proper noun.

Implementation of such a system has proved to be a non-trivial problem. A major task in the design of such a network is the selection of various network parameters such as the optimal number of hidden layers, the number of units in these hidden layers, and the training schedule. These parameters are application-specific functions of quantities such as the amount of training data, the context size and the stopping criteria for training. The Boltzmann machine system performs to a reasonable grade for comparatively small subclasses of proper nouns. However, at a larger scale the performance degenerates as the present network architecture shows limited capacity to adapt to a large training set of variable length proper noun strings.

A significant contribution of this exercise is the pronunciation dictionary which currently consists of 18494 surnames and 25648 pronunciations. We expect it to be a valuable and useful resource to the entire speech research community in fueling further research towards accurate recognition of proper nouns and preparing batter acoustic models corresponding to different pronunciations.

The complete database, as well as all the software for the Boltzmann machine system is available in the public domain at *http://www.isip.msstate.edu/software/n_best_pronunciations/*. A graphical user interface has been developed for this application that allows the user to enter any proper noun spelling, generate the multiple pronunciations and a graphic of its pronunciation network.

Our future work will be focused on overcoming the present problems of the pronunciations system. The proposed modifications include development of alternative architectures such as subnetworks of MLPs for different input string lengths and exploring alternative training paradigms that facilitate confusion-free modeling of the letter-to-phone features. One such possibility is to combine the Boltzmann machine with a time-delayed neural network (TDNN).

We also plan to further develop the pronunciation database by adding more names to it, as well as creating pronunciation sets for different kinds of proper nouns such as geographical names (countries, major cities in the United States of America and other countries, street names etc.), corporate names (companies and products) etc.

## 7. ACKNOWLEDGEMENTS

We wish to thank Dr. Barb Wheatley of the National Security Agency for her invaluable guidance and support in development of the surname pronunciations database, and the in the design of the original system. We are also grateful to Dr. Jack Godfrey at TI for his numerous contributions in the development of the training database, and his continued support of this project. We also wish to acknowledge the continual support and guidance we have received from our sponsors, Texas Instruments, particularly, Dr. Raja Rajasekaran and Dr. Vishu Vishwanathan. We are also thankful to Julie Ngan and Mary Weber for their assistance in building the pronunciation dictionary.

## 8. REFERENCES

[1] B. Wheatley and J. Picone, “Integrating Speech Technologies For Medical Applications,” presented at the Medical Applications of Voice Response Technology Conference in Pittsburgh, PA, Dec. 1989.

[2] J. Picone, B.J. Wheatley and J. McDaniel, “On the Intelligibility of Text-To-Speech Synthesis of Surnames,” Texas Instruments Technical Report No. CSC-TR-91-002, pp. 1-34, Texas Instruments Inc., Dallas, TX, March 13, 1991.

[3] Digital Equipment Corporation, DTC-01-AA, 1985.

[4] G. Hinton and J. Anderson (Eds), *Parallel Models of Associative Memory*, Erlbaum Associates, NJ, 1981.

[5] G. Hinton and T. Sejnowski, "Optimal Perceptual Inference", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 448-453, Washington D.C., June 1986.

[6] T.J. Sejnowski and C.R. Rosenberg, ""NETtalk: A Parallel Network That Learns To Read Aloud," Tech. Rep. JHU/EECS-86/01, John Hopkins University, Baltimore, MD, 1986.

[7] B.J. Wheatley and J. Picone, "Voice Recognition of proper nouns Using Text-Derived Recognition Models," US Patent No. 5212730, May 18, 1993.

[8] D.R. Calvert, *Descriptive Phonetics*, 2nd Edition, Thieme Inc., New York, New York, USA, 1986.

[9] H.M. Meng, S. Seneff, and V.W. Zue, "Phonological Parsing for Reversible Letter-to-Sound/Sound-to-Letter Generation," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. II-1-II-4, Adelaide, Australia, April 1994.

[10] F.R. Chen, "Identification of Contextual Factors for Pronunciation Networks", in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 753-756, Albuquerque, NM, April 1990.

[11] M.D. Riley, "A Statistical Model for Generating Pronunciation Networks", *in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, S11.1, pp. 737-740, Toronto, Canada, May 1991.

[12] G. Tajchman, E. Fosler and D. Jurafsky, "Building Multiple Pronunciation Models for Novel Words Using Exploratory Computational Phonology", in *Eurospeech '95*, 1995.

[13] E. Fosler, M. Weintraub, S. Wegmann, Y-H. Kao, S. Khudanpur, C. Galles and M. Sinclair, "Automatic Learning of Word Pronunciation from Data", presented at the *JHU/CLSP Large Vocabulary Conversational Speech Recognition Workshop '96*, Baltimore, MD, 1996.

[14] J.L. Elman and D. Zipser, "Learning the Hidden Structure of Speech", ICS Report 8701, University of California at San Diego, 1987.

[15] A. Waibel, H. Sawai and K. Shikano, "Modularity and Scaling in Large Phonemic Time-delay Neural Networks", *in IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 30, pp. 1888-1898, December 1989.

[16] F. Rosenblatt, "The Perceptron: A Perceiving and Recognizing Automaton", Cornell Aeronautical Laboratory Report 85-460-1, 1957.

[17] M.L. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.

[18] T. Kohonen, "Automatic Formation of Topological Maps in a Self-organizing System", In E. Oja and O. Simula (Eds.), *Proceedings of the 2nd Scandinavian Conference on Image Analysis*, pp. 214-220, 1981.

[19] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", in *Proceedings of the National Academy of Sciences USA*, Vol. 79, pp. 2554-2558, 1982.

[20] R. Rosenfeld, "A Hybrid Approach to Adaptive Statistical Language Modeling," in *Proceedings ARPA Workshop on Human Language Technology*, pp. 76-81, Plainsboro, New Jersey, USA, March 1994.

[21] T. Robinson, M. Hochberg, and S. Renals, "IPA: Improved Phone Modeling With Recurrent Neural Networks," in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. I-37-I-40, Adelaide, South Australia, Australia, April 1994.

[22] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, no. 3, pp. 328-339, March 1989.

[23] S. Kirkpatrick, C.D. Gellatt and M.P. Vecchi, "Optimization by Simulated Annealing", in *Science*, Vol. 220, pp. 671-680, 1983.

[24] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning Internal Representation by Error Propagation", in D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing — Vol. 1: Foundations*, MIT Press, Cambridge, MA, 1986.

[25] S. Kullback, *Information Theory and Statistics*, Wiley, New York, NY, 1959.

[26] A. Renyi, *Probability Theory*, North Holland Publishers, Amsterdam, Netherlands, 1962.

[27] N. Deshmukh and J. Picone, "Automatic Generation of N-best Proper Name Pronunciations", ISIP Technical Report No. ISIP-96-0002, pp. 1-38, Institute for Signal and Information Processing, August 1996.

[28] E.C. Smith, *American Surnames*, Genealogical Publishing Co. Inc., Baltimore, MD, 1986.

[29] R.A. Cole, M. Fanty and K. Roginski, "A Telephone Speech Database of Spelled and Spoken Names," in *Proceedings of the International Conference on Spoken Language Processing*, Banff, Alberta, Oct. 12-16, pp. 891-893, (1992).

## List of Tables

1. Performance of the Boltzmann machine on the 2-class problem (no shifting of input letters).
2. Performance of the Boltzmann machine on the 2-class problem. In this case the input letters are shifted through the Boltzmann machine shift registers.
3. Summary of the Boltzmann machine performance on the 26-class problem. The first seven rows correspond to closed loop evaluation, the last two rows describe open loop results.
4. Summary of performance of the Boltzmann machine pronunciations generation system evaluated on proper nouns of length 4.
5. Performance of the single-context Boltzmann machine pronunciation generation system.
6. Performance of the both-context Boltzmann machine pronunciation generation system.

## List of Figures

1. Typical neuron connections in a Boltzmann machine network, along with the Boltzmann distribution function that governs the activation probability of a neuron.
2. An overview of a neural network architecture that performs letter to sound conversion. In this example, there are three layers: a layer that converts letters to binary-valued inputs, a layer that converts n-tuples of letters into sounds, and a layer that applies a mixture of short-term and long-term relationships.
3. A schematic overview of the simulated annealing used to train the Boltzmann machine neural network. The backpropagation of error training pass is displayed in detail in the inset.
4. Performance on two-class problem for four-letter strings with no shifting of input data. This case is with a single hidden layer of neurons.
5. Performance on two-class problem for four-letter strings with no shifting of input data. This case is with two hidden layers of neurons. The number of neurons is specified in the form 1st layer, 2nd layer on the horizontal axis.
6. Performance on two-class problem for five-letter strings fed sequentially as input data. This case is with a single hidden layer of neurons.
7. Performance on the 26-class problem. The error rate decreases with a bigger context interval and a larger number of neurons in the hidden layers. This case is with a single hidden layer of neurons.
8. 2-class classification performance as a function of number of hidden neurons.
9. 2-class classification performance as a function of number of training iterations.
10. Effect of context length on performance for the 4-letter proper nouns.
11. Effect of number of hidden neurons on performance for the 4-letter proper nouns.

| # neurons | classification error |
|---|---|
| 2 | 27.27% |
| 8 | 50.00% |
| 64 | 4.55% |
| 75 | 0.00% |
| 80 | 0.00% |
| 125 | 0.00% |

(a) Single hidden layer

| #neurons per layer | | classification error |
|---|---|---|
| layer 1 | layer 2 | |
| 1 | 2 | 100.00% |
| 2 | 1 | 100.00% |
| 2 | 2 | 22.73% |
| 2 | 16 | 50.00% |
| 16 | 2 | 13.64% |
| 16 | 4 | 45.45% |

(b) Two hidden layers

Table 1: Performance of the Boltzmann machine on the 2-class problem (no shifting of input letters).

| context length | # neurons | classification error |
|---|---|---|
| 3 | 70 | 41.00% |
| 3 | 100 | 36.00% |
| 4 | 100 | 33.00% |
| 5 | 16 | 50.00% |

| context length | # neurons | classification error |
|---|---|---|
| 5 | 50 | 5.00% |
| 5 | 80 | 5.00% |
| 5 | 96 | 5.00% |
| 5 | 105 | 0.00% |

Table 2: Performance of the Boltzmann machine on the 2-class problem. In this case the input letters are shifted through the Boltzmann machine shift registers.

| training data string size | # training strings | context type | context size | # hidden layers | # neurons / layer | classification error |
|---|---|---|---|---|---|---|
| 3 | 1000 | both | short 3<br>long 7 | 1<br>1 | 125<br>125 | 22.03% |
| 4 | 10000 | both | short 3<br>long 7 | 1<br>1 | 125<br>100 | 56.47% |
| 4 | 1000 | both | short 4<br>long 7 | 1<br>1 | 200<br>200 | 7.28% |
| 4 | 1000 | short | 4 | 1 | 200 | 8.47% |
| 4 or 5 | 1000 | short | 5 | 1 | 300 | 9.14% |
| 4 or 5 | 1000 | short | 5 | 1 | 500 | 7.76% |
| 4 or 5 | 2000 | short | 5 | 1 | 1000 | 3.77% |
| 4 or 5 | 1000 | short | 5 | 1 | 300 | 37.46% |
| 4 or 5 | 1000 | short | 5 | 1 | 500 | 27.74% |

Table 3: Summary of the Boltzmann machine performance on the 26-class problem. The first seven rows correspond to closed loop evaluation, the last two rows describe open loop results.

| context length | # hidden neurons | #training iterations | training schedule init. temp. / decay rate | pronunciations generation rate | % correct pronunciations | | |
|---|---|---|---|---|---|---|---|
| | | | | | all | some | none |
| 3 | 200 | 50 | 1000 / 0.1 | 1.64 | 17.12 | 13.75 | 69.13 |
| 3 | 200 | 60 | 1000 / 0.1 | 1.87 | 17.60 | 14.35 | 68.05 |
| 3 | 110 | 60 | 100 / 0.1 | 2.93 | 3.24 | 43.60 | 53.15 |
| 3 | 125 | 90 | 100 / 0.05 | 1.37 | 26.49 | 9.13 | 64.38 |
| 3 | 125 | 90 | 1000 / 0.1 | 1.55 | 18.74 | 10.69 | 70.57 |
| 3 | 125 | 60 | 100 / 0.1 | 1.87 | 20.54 | 33.69 | 45.77 |
| 3 | 150 | 70 | 100 / 0.1 | 3.04 | 3.90 | 49.37 | 46.73 |
| 4 | 125 | 60 | 100 / 0.1 | 2.89 | 5.29 | 50.93 | 43.78 |
| 7 | 125 | 60 | 100 / 0.1 | 2.60 | 10.99 | 53.21 | 35.80 |
| 7 | 150 | 60 | 100 / 0.1 | 2.66 | 10.87 | 51.29 | 37.84 |
| 7 | 200 | 60 | 100 / 0.1 | 2.67 | 10.45 | 50.15 | 39.40 |
| 7 | 300 | 60 | 100 / 0.1 | 2.66 | 12.25 | 54.23 | 33.51 |

Table 4: Summary of performance of the Boltzmann machine pronunciations generation system evaluated on proper nouns of length 4.

| context length | # hidden neurons | #training iterations | training schedule init. temp. / decay rate | pronunciations generation rate | % correct pronunciations | | |
|---|---|---|---|---|---|---|---|
| | | | | | all | some | none |
| 3 | 100 | 40 | 1000 / 0.1 | 1.41 | 5.21 | 2.46 | 92.33 |
| 3 | 100 | 90 | 100 / 0.05 | 1.31 | 22.78 | 6.78 | 70.44 |
| 3 | 150 | 70 | 100 / 0.1 | 2.48 | 2.18 | 15.41 | 82.41 |
| 3 | 200 | 90 | 100 / 0.05 | 1.23 | 29.33 | 6.72 | 63.95 |
| 3 | 200 | 40 | 1000 / 0.1 | 2.46 | 2.23 | 14.88 | 82.89 |
| 3 | 500 | 60 | 1000 / 0.1 | 1.38 | 6.07 | 1.37 | 92.56 |
| 7 | 100 | 90 | 1000 / 0.05 | 1.08 | 23.46 | 1.37 | 75.17 |
| 7 | 100 | 60 | 1000 / 0.1 | 1.28 | 10.47 | 2.23 | 87.30 |
| 7 | 200 | 60 | 100 / 0.1 | 2.28 | 5.04 | 11.44 | 83.52 |
| 7 | 200 | 60 | 1000 / 0.1 | 1.30 | 14.13 | 3.29 | 82.58 |
| 7 | 300 | 60 | 100 / 0.1 | 2.26 | 1.77 | 5.12 | 93.10 |
| 7 | 300 | 60 | 1000 / 0.1 | 1.29 | 15.65 | 3.20 | 81.14 |

Table 5: Performance of the single-context Boltzmann machine pronunciation generation system.

| context length | # hidden neurons | #training iterations | training schedule init. temp. / decay rate | pronunciations generation rate | % correct pronunciations | | |
|---|---|---|---|---|---|---|---|
| | | | | | all | some | none |
| 3<br>7 | 1000<br>1000 | 99 | 100 / 0.1 | 1.84 | 3.63 | 2.98 | 93.39 |
| 3<br>7 | 2000<br>2000 | 35 | 100 / 0.1 | 2.30 | 0.00 | 2.14 | 97.86 |
| 3<br>7 | 4000<br>4000 | 62 | 100 / 0.1 | 1.30 | 1.28 | 4.19 | 94.53 |
| 3<br>7 | 8000<br>8000 | 15 | 100 / 0.1 | 2.47 | 0.00 | 0.01 | 99.99 |
| 3<br>7 | 10000<br>10000 | 25 | 100 / 0.1 | 1.26 | 0.00 | 0.01 | 99.99 |

Table 6: Performance of the both-context Boltzmann machine pronunciation generation system.

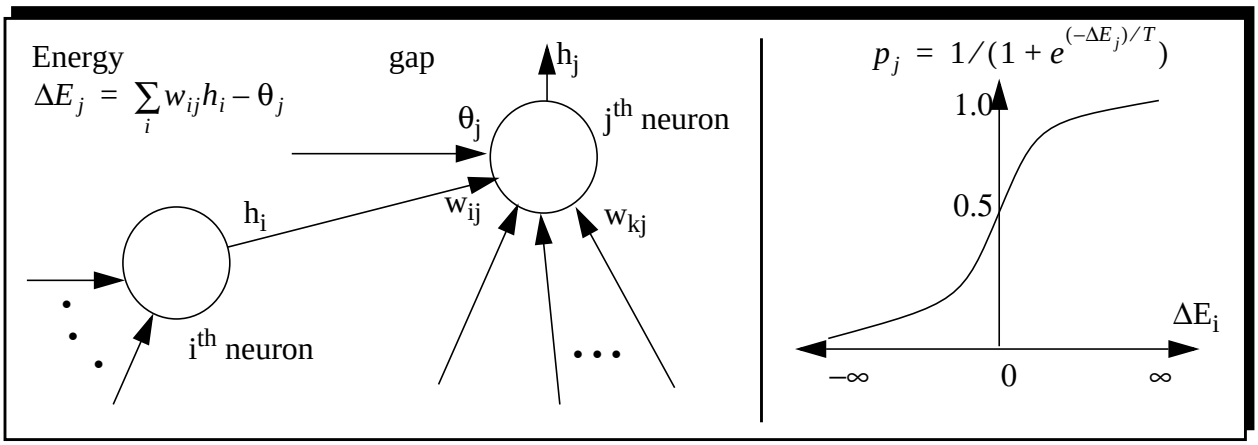



Figure 1. Typical neuron connections in a Boltzmann machine network, along with the Boltzmann distribution function that governs the activation probability of a neuron.

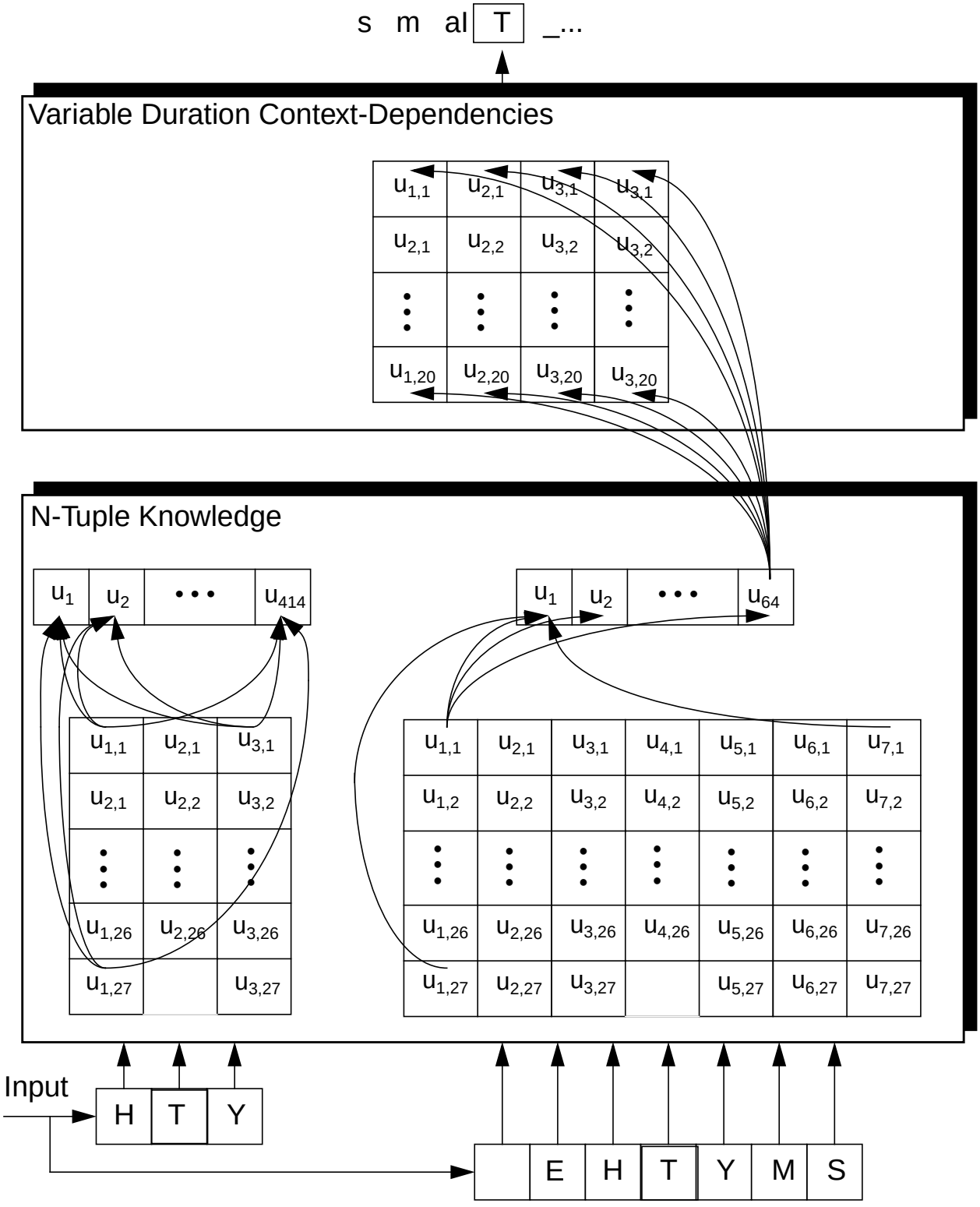



Figure 2. An overview of a neural network architecture that performs letter to sound conversion. In this example, there are three layers: a layer that converts letters to binary-valued inputs, a layer that converts n-tuples of letters into sounds, and a layer that applies a mixture of short-term and long-term relationships.

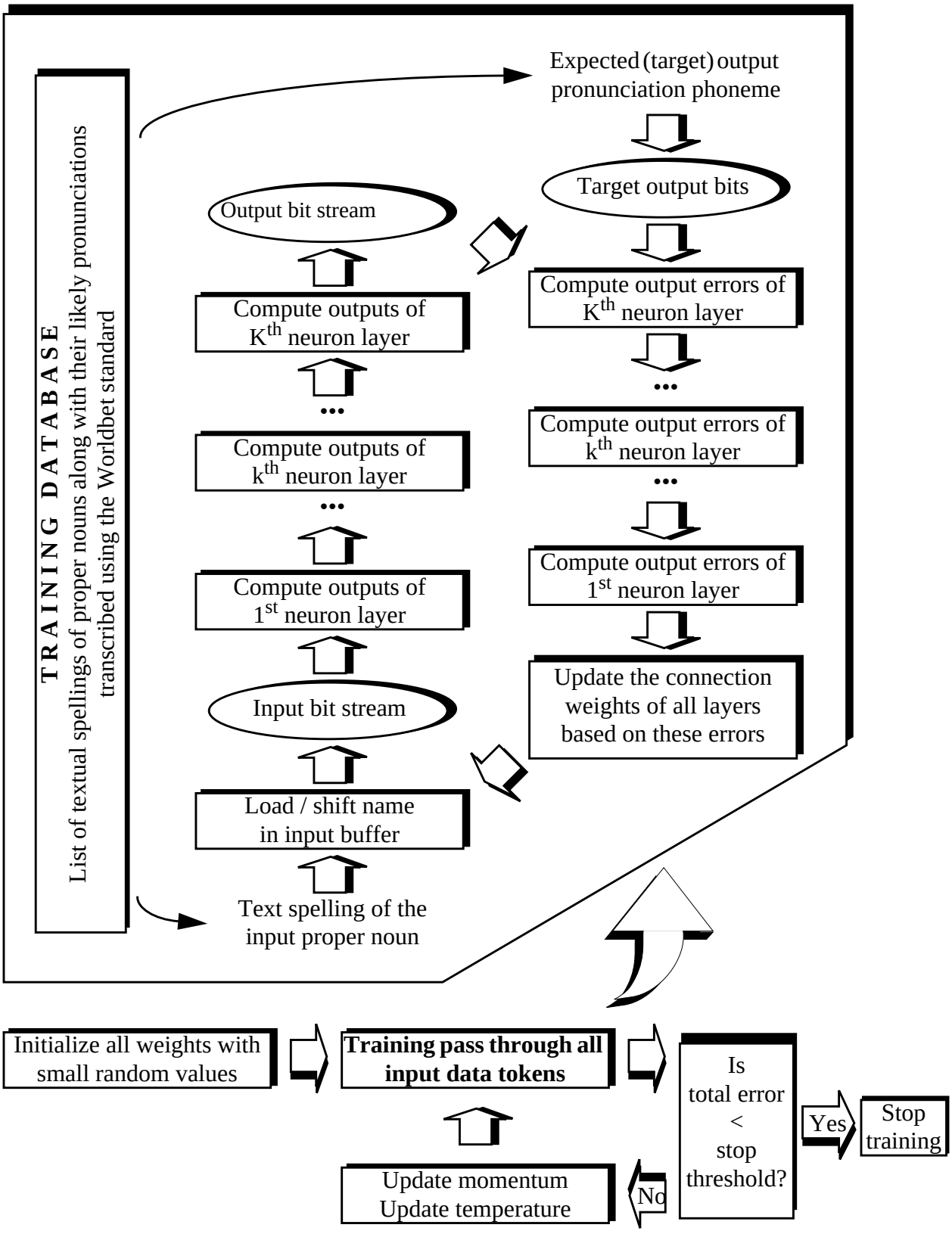
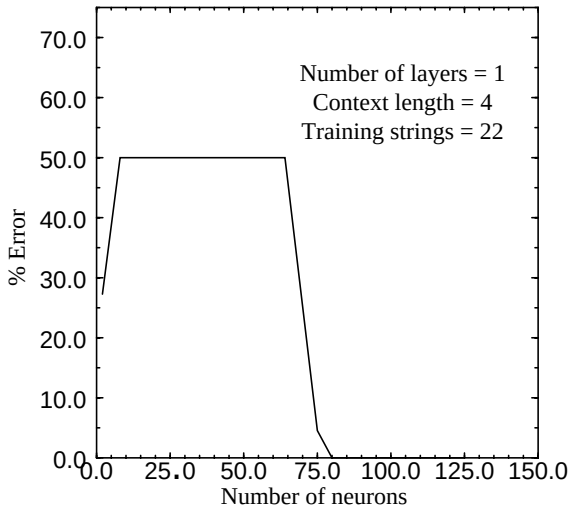
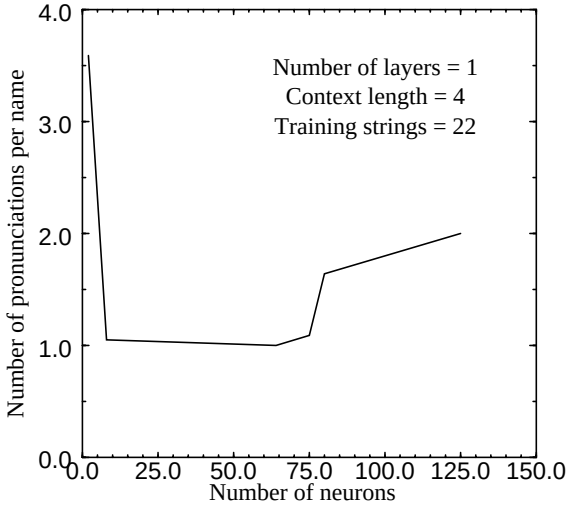



Figure 3. A schematic overview of the simulated annealing used to train the Boltzmann machine neural network. The backpropagation of error training pass is displayed in detail in the inset.

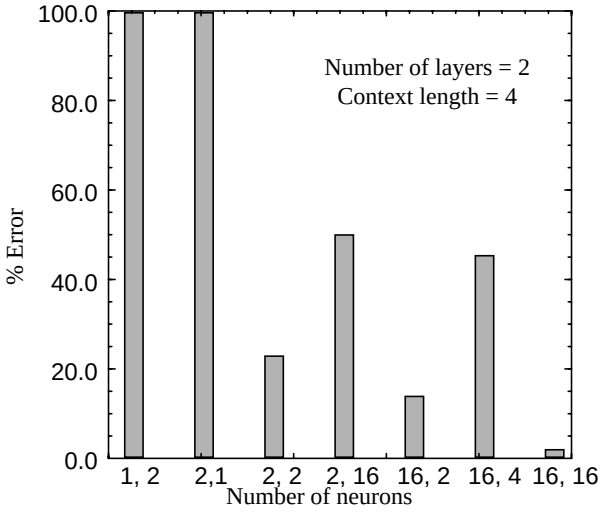



(a) Error rate as a function of number of hidden layer neurons

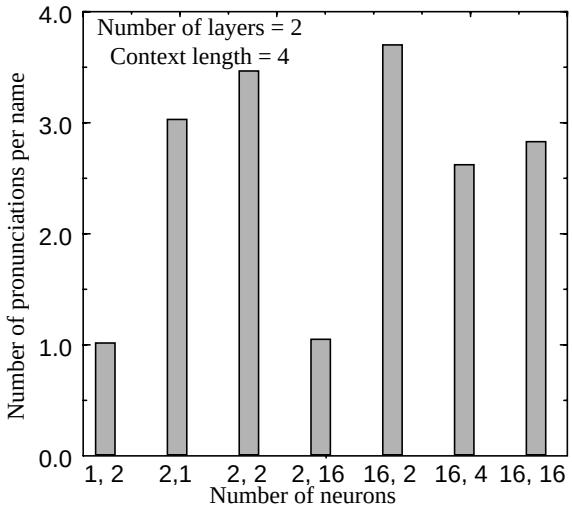



(b) Generation rate as a function of number of hidden layer neurons

Figure 4. Performance on two-class problem for four-letter strings with no shifting of input data. This case is with a single hidden layer of neurons.

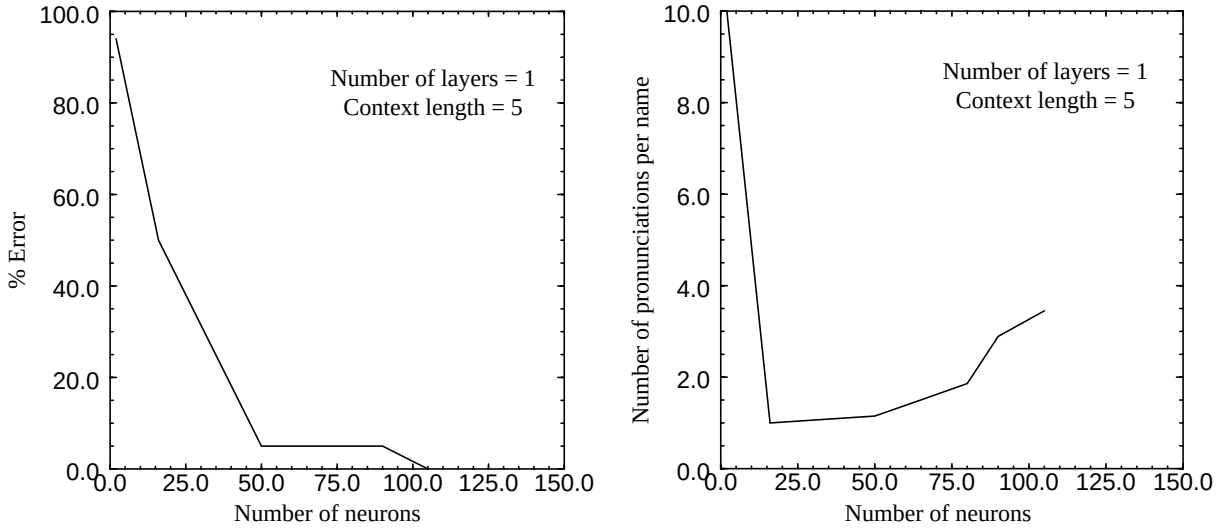



(a) Error rate as a function of number of hidden layer neurons




(b) Generation rate as a function of number of hidden layer neurons

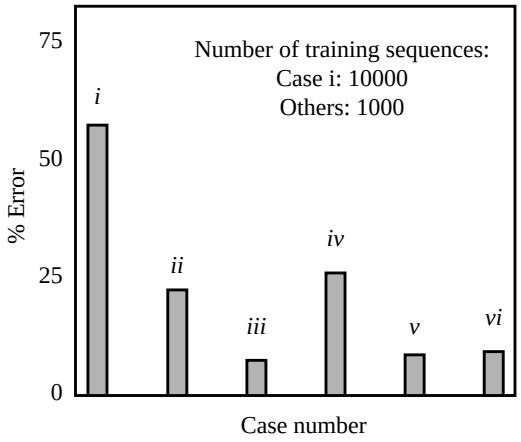
Figure 5. Performance on two-class problem for four-letter strings with no shifting of input data. This case is with two hidden layers of neurons. The number of neurons is specified in the form *1st layer, 2nd layer* on the horizontal axis.




(a) Error rate as a function of number of hidden layer neurons

(b) Generation rate as a function of number of hidden layer neurons

Figure 6. Performance on two-class problem for five-letter strings fed sequentially as input data. This case is with a single hidden layer of neurons.




| Case | Context length | | Hidden neurons | | Input data size |
|---|---|---|---|---|---|
| | Short | Long | Short | Long | |
| *i* | 3 | 7 | 125 | 100 | 4 |
| *ii* | 3 | 7 | 125 | 125 | 3 |
| *iii* | 4 | 7 | 200 | 200 | 4 |
| *iv* | 4 | — | 200 | — | 4 |
| *v* | 5 | — | 300 | — | 4 |
| *vi* | 5 | — | 300 | — | 4 / 5 |

Parameters for experiments

Figure 7. Performance on the 26-class problem. The error rate decreases with a bigger context interval and a larger number of neurons in the hidden layers. This case is with a single hidden layer of neurons.

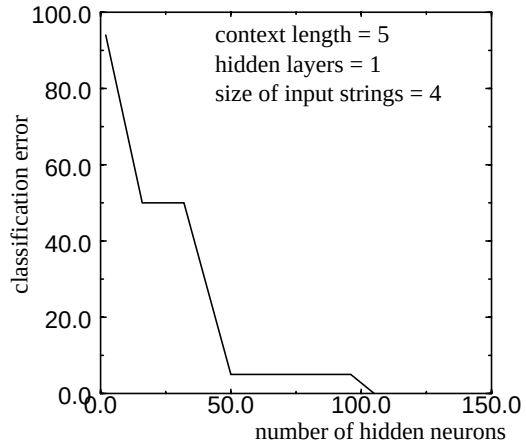



Figure 8. 2-class classification performance as a function of number of hidden neurons.

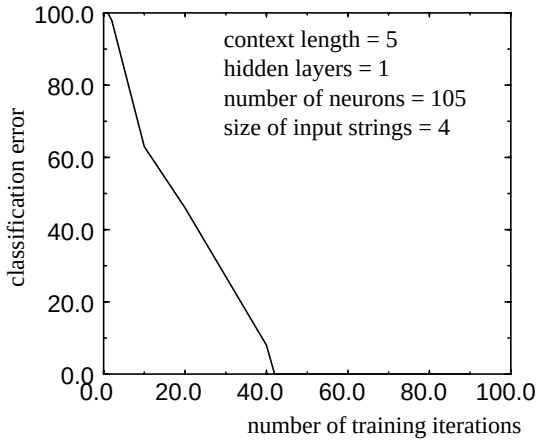



Figure 9. 2-class classification performance as a function of number of training iterations.

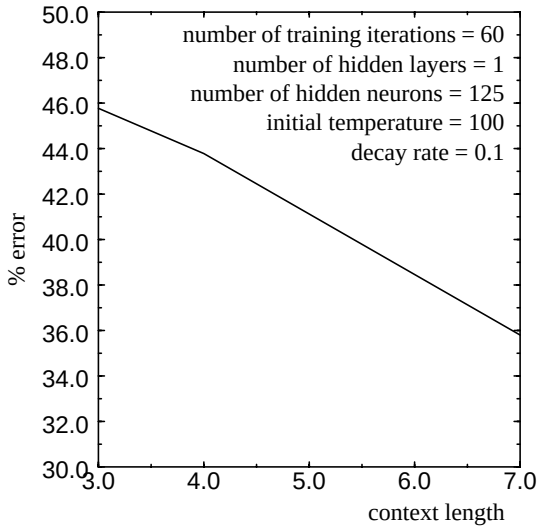



Figure 10. Effect of context length on performance for the 4-letter proper nouns.

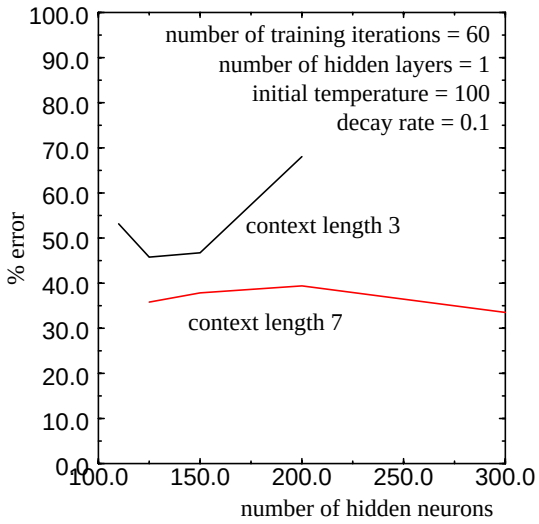



Figure 11. Effect of number of hidden neurons on performance for the 4-letter proper nouns.