Automatic Generation of N-Best Proper Noun Pronunciations

prepared for:

Speech Research Group
Personal Systems Laboratory
Texas Instruments, Inc.
PO Box 655474, MS 238
Dallas, Texas 75265

by:

N. Deshmukh and J. Picone
Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571
413 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
Tel: 601-325-3149
Fax: 601-325-3149
email: {deshmukh,picone}@isip.msstate.edu

EXECUTIVE SUMMARY

The problem of proper noun recognition is key to developing pervasive voice interfaces in applications such as directory assistance and data entry for telecommunications. Recognition of such words 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. Yet, despite being a seemingly intractable problem, humans do amazingly well at generating and recognizing the pronunciation(s) of a name never before encountered.

The Institute for Signal and Information Processing (ISIP) at Mississippi State University has implemented a neural networks algorithm capable of discovering some underlying structure in this problem to provide Texas Instruments (TI) with a software system capable of converting proper nouns in English to a list of N most plausible pronunciations. This research effort involved the following four concurrent phases: development of a representative training database, implementation of a Boltzmann machine type neural network software, design and development of an application-specific architecture, and detailed evaluation of the resulting system.

One aspect of this task that makes it particularly challenging and timely is that there are no existing proper noun databases that include extensive lists of plausible alternate pronunciations for a demonstrative sample of proper nouns. Further, there is no industry-accepted database that represents a comprehensive coverage of all possible phenomena an algorithm must handle. Most of this information is passed word-of-mouth amongst developers of text-to-speech (TTS) synthesis technology. ISIP has developed a training database that consists of almost twenty thousand representative surnames and their many likely pronunciations to support training and development of the system. This dictionary uses the Worldbet notation for phonetic transcriptions to incorporate data from linguistically diverse sources, and is an extremely valuable contribution in itself to the ongoing research in speech recognition and synthesis. It is recommended that this database be transferred to the Linguistic Data Consortium (LDC) and made available as a public resource used to fuel further research into proper noun recognition.

We have extensively evaluated the pronunciation generation performance of this system on such a database. A formal evaluation of a speech recognition system that employs this technology for building models for proper nouns is also planned. This collaborative effort leverages TI's industry-leading speech recognition technology. This opportunity to work jointly on such a demonstration system will be important in identifying and solving many very real aspects of the problem not exposed by the limitations of the training database.

We have provided TI with the complete software and binaries for training the system and generating the N-best pronunciations. The software adheres to the standards of C++ object-oriented data-driven applications at ISIP. A Tcl-Tk graphical user interface (GUI) demo allows the user to select system parameters and outputs an ordered list of the likely pronunciations along with a phonetic network that can generate them for an input name. The pronunciations database, as well as all tools and software developed in this project are placed into the public domain (*http://www.isip.msstate.edu/software/n_best_pronunciations/*) and supported by ISIP.

TABLE OF CONTENTS

1. ABSTRACT 1

2. HISTORICAL BACKGROUND 1

3. NEURAL NETWORKS FOR PATTERN CLASSIFICATION 2

3.1. Layered Networks 3
3.2. Hopfield Network 3

4. THE BOLTZMANN MACHINE 4

4.1. Boltzmann Machine Architecture 5
4.2. Mathematical Foundations 8
4.3. Training Algorithm 9
4.4. Evaluation Strategy 14

5. CHARACTERIZING THE SYSTEM 14

5.1. Two-Class Classifier 14
5.2. Alphabet Classifier 16

6. BUILDING A TRAINING DATABASE 17

6.1. Collection And Transcription 18
6.2. Phone Alignment With Spelling 18

7. USER INTERFACE 19

8. PERFORMANCE EVALUATION 20

8.1. Two-class Alphabet Strings 20
8.2. Multi-class Alphabet Strings 22
8.3. Evaluation With Four-letter Proper Nouns 23
8.4. Evaluation On Real Data 25
8.5. Run-Time Issues 26

9. CONCLUSIONS 27

10. FUTURE DIRECTIONS 27

11. ACKNOWLEDGEMENTS 28

12. REFERENCES 28

APPENDIX A. DERIVATION OF THE TRAINING ALGORITHM 31

A.1. The Weight-update Algorithm Derivation 31

APPENDIX B. ALPHABET AND PHONEME SETS 34

B.1. Input Alphabet Set 34
B.2. Output Phoneme Set 35

APPENDIX C. PUBLICATIONS 36

C.1. Paper Presented At ICASSP ‘96 37
C.2. Paper Proposal Submitted To ICASSP ‘97 41

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 document 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 system 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.

2. HISTORICAL BACKGROUND

The motivation for this work originates from experiences at Texas Instruments in the late 1980s with attempts to field speech recognition technology in medical applications [1, 2]. In many such applications, the ability to recognize a physician’s or patient’s name was crucial in providing a usable interface. For example, using numbers to access patient records is problematic due to the impracticality of remembering such numbers for any significant class of patients. Name recognition is a vital step in transforming medical record access from keyboard input to voice input. 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.

An extensive study was performed in the late 1980s to evaluate the accuracy of text-to-speech systems on pronouncing proper nouns [2]. In this work, it was shown that extensive handwritten rule sets were required to generate an accurate pronunciation of a name. Such rule-based systems, even though deemed fairly accurate for the directory assistance application for which they were designed, generated only the single most likely pronunciation for each proper noun. Since most proper nouns have a number of highly probable pronunciations which can be rarely differentiated from the context of the application, a system generating only the single-most likely pronunciation essentially attempts to solve an ill-posed problem. For a useful speech recognition application it is important that all plausible alternatives be available to the system.

A commercial product called DECtalk [3] was developed in the mid-1980s that converted unrestricted English text into speech, using a set of phonological rules and by handling exceptions with a lookup table. DECtalk used two methods for converting text into phonemes — first a word was looked up in a pronunciation dictionary of common words. If it was not found there then a set of phonological rules was applied to obtain a phonetic transcription along with stress assignments. These were then converted into speech sounds using transition rules and digital speech synthesis.

For novel words for which a correct pronunciation could not be obtained, the dictionary and rule sets required updating through explicit intervention. This approach was found to be highly labor-intensive and very limited in scope.

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. Such networks can be created automatically through incremental learning i.e. by repetitive training on example cases. Multilayered neural networks, in which the internal or hidden units can act as feature detectors that perform a mapping between the input and the output are a class of models ideally suited for such applications of letter-to-phone conversion. The general idea of applying neural network techniques to this problem is loosely based on a feasibility study performed in the mid-1980s that focused on automatically learning text-to-speech letter-to-sound mappings using neural networks [6].

A multilayered neural network model called NETtalk [6] was developed in 1985 as an alternative to DECtalk. It successfully demonstrated that a relatively small network can capture most of the significant regularities of the English pronunciations, as well as absorb a large number of the irregularities. It also had the advantage of being language independent (i.e. it could be trained to be used on any language) and directly implementable in hardware. However, it was found to be limited in its ability to handle ambiguities that require syntactic and semantic levels of analysis.

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. In this technique, an algorithm was proposed to automatically derive recognition models from the text-only spellings of the name (rather than voice data containing a spelling or nominal pronunciation of the name). This system relies on a particular form neural network designed to generate multiple outputs for a given input. This class of networks is known as a Boltzmann machine [6]. It transforms its input, the spelling of a proper noun, to a network of distinctive features [8] describing articulatory movements required to produce various pronunciations of the name. However, this system was never implemented or evaluated.

Recently, similar work at MIT involving Hidden Markov Model-based approaches has shown some promise [9]. Also, a technique that uses decision-tree based statistical modeling to automatically generate detailed phonetic pronunciation networks from a coarse phonemic transcription [10] has been developed at AT&T. These networks are capable of generating more than one pronunciations for regular words. US West has also sponsored some research in automated generation of pronunciations. However, no system has been designed that can effectively model the peculiarities of proper noun pronunciations to generate multiple pronunciations.

3. NEURAL NETWORKS FOR PATTERN CLASSIFICATION

The concept of an artificial neural network (ANN) originates from the notion that complex computing or classification operations can be implemented by massive integration of individual computing units, each of which performs an elementary computation. This basic unit, called a

neuron, is a nonlinear system whose output 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 important application of ANNs is in feature extraction and classification of patterns. An advantage of using ANNs for such problems is their capability to capture the inherent functionality of the data without any *a priori* statistical characterization or parameterization. The network reduces the large input vectors into small output feature vectors that effectively indicate the classes represented by the input patterns. Often, such features constitute the internal activation patterns of a network rather than the output [11, 12].

3.1. Layered Networks

Multilayered neural networks, in which the neurons can be viewed to be connected across different levels, 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 subsequent layers such that each successive layer receives 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.

There are two architectures of layered neural networks prevalently used as classifiers —

Feedforward networks: A feedforward or nonrecurrent ANN is one for which no neuron has a connection leading from its output back to its input. Moreover, neurons in one layer are strictly connected only to those in the immediate next layer. Such an architecture is also called a multilayered perceptron (MLP) [13, 14], and needs to be trained in a supervised fashion i.e. by exposing the network to the input patterns along with the corresponding desired outputs. An appropriately trained network can then reproduce the entire population of the target outputs as closely as possible in some sense.

Learning vector quantizers: A learning vector quantizer (LVQ) [15] is a single layer ANN which typically follows unsupervised learning. The network automatically adjusts its weights so that input patterns similar in some sense produce similar outputs. Thus the input patterns can be classified on the basis of the outputs they produce. The output neurons compete with each other to produce an output that best characterizes the input pattern.

Both these architectures are limited to a single output per input pattern. To yield multiple possible outputs corresponding to a single input pattern, a stochastic component needs to be introduced in such networks so that the system can generate a different output if it is presented with the same input at multiple instances.

3.2. Hopfield Network

A Hopfield network [16] is a powerful classification ANN in which pairs of neurons are connected through symmetric weights. During training the neurons update their weights asynchronously by looking at their local connections with other neurons. The network views each

global state as a *hypothesis*. The node connections in agreement with the hypothesis (or supporting it) are assigned positive weights, while the connections that are incompatible with the hypothesis are assigned negative weights.

Depending on such connections each global state of the network can be associated with a numerical function called *energy* of the network for that state. With the right assumptions, the individual nodes can be made to act so as to minimize the global energy. Thus the system tries to find the minimum energy configuration compatible with each input pattern. The energy of any such configuration can be interpreted as the extent to which that combination of hypotheses violates the constraints implicit in the problem domain, so by minimizing the global energy the system evolves towards interpretations (or classifications) of the input that increasingly satisfy the constraints of the problem space.

The main problem with Hopfield networks is that they 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. If a network reaches a stable energy state it is not possible to reactivate it to reach another stable state. The network must be pushed externally so that several neurons change state simultaneously to slip the network into another energy minimum. A Boltzmann machine is a system capable of this and therefore emerges as a viable alternative.

A Boltzmann machine is a neural network that is trained to represent the probability density distribution of observables in a particular domain. The Boltzmann machine architecture is particularly useful in the context of name pronunciation/recognition because of the need to produce alternative outputs for the same input. This is the single most important differentiating feature of a Boltzmann machine relative to other, more traditional, neural networks used in various aspects of speech and language processing. Traditional networks are trained to produce the single most likely output corresponding to a given input. The Boltzmann machine, on the other hand, is designed to model all observables for a given system, rather than simply the best choice.

4. THE BOLTZMANN MACHINE

The Boltzmann machine is a parallel computational architecture quite similar to a Hopfield network. 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. The architecture is designed to allow efficient searches for various hypotheses (encoded as combinations of network units with binary states) that maximally satisfy the input data and some constraints resulting from weighted interaction between the individual units. An important distinction from the networks discussed so far is that each neuron has a stochastic distribution associated with its activation. This distribution is a function of the global energy of the system. Thus each neuron will activate, or “fire”, with some probability for a particular input pattern; and therefore for the same input 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.

4.1. Boltzmann Machine Architecture

An overview of the neural network architecture for generating proper noun pronunciations is given in Figure 1. 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 architecture shown in Figure 1 is, of course, just one example of how such a network might be configured.

A Boltzmann machine comprises layers of simple interconnected units, at least some of which are external (input/output) units. Each unit can be either "on" or "off" — the state of each unit (if not fixed) is a probabilistic function of the states of the units to which it is connected and the strength of the real-valued weights on the connections. All connection weights between units are symmetrical, representing mutually excitatory or mutually inhibitory relationships. Each configuration of the network has an energy value that is a function of the states and connections for all units. 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. It involves alternate cycles in which

- the states of external units are either clamped (determined externally and held constant) or free (set randomly and allowed to change)
- all internal units are free. For each initial configuration, a conventional simulated annealing procedure is used to bring the network to a state of equilibrium. Connection weights are adjusted to reduce the difference in energy between clamped and free configurations.

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 1 is designed specifically for the problem of name pronunciation/recognition. The configuration is based on two design criteria:

1. Generally, a relatively small amount of contextual information will be sufficient to narrow the range of possible sound correspondences to a small set.
2. Choosing a correct sound from this set may require information occurring at more remote points in the name (such as the identification of the foreign language from which the name was drawn).

In a sense, the network is designed to model n-tuples of letters using local and long-distance constraints [17]. 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

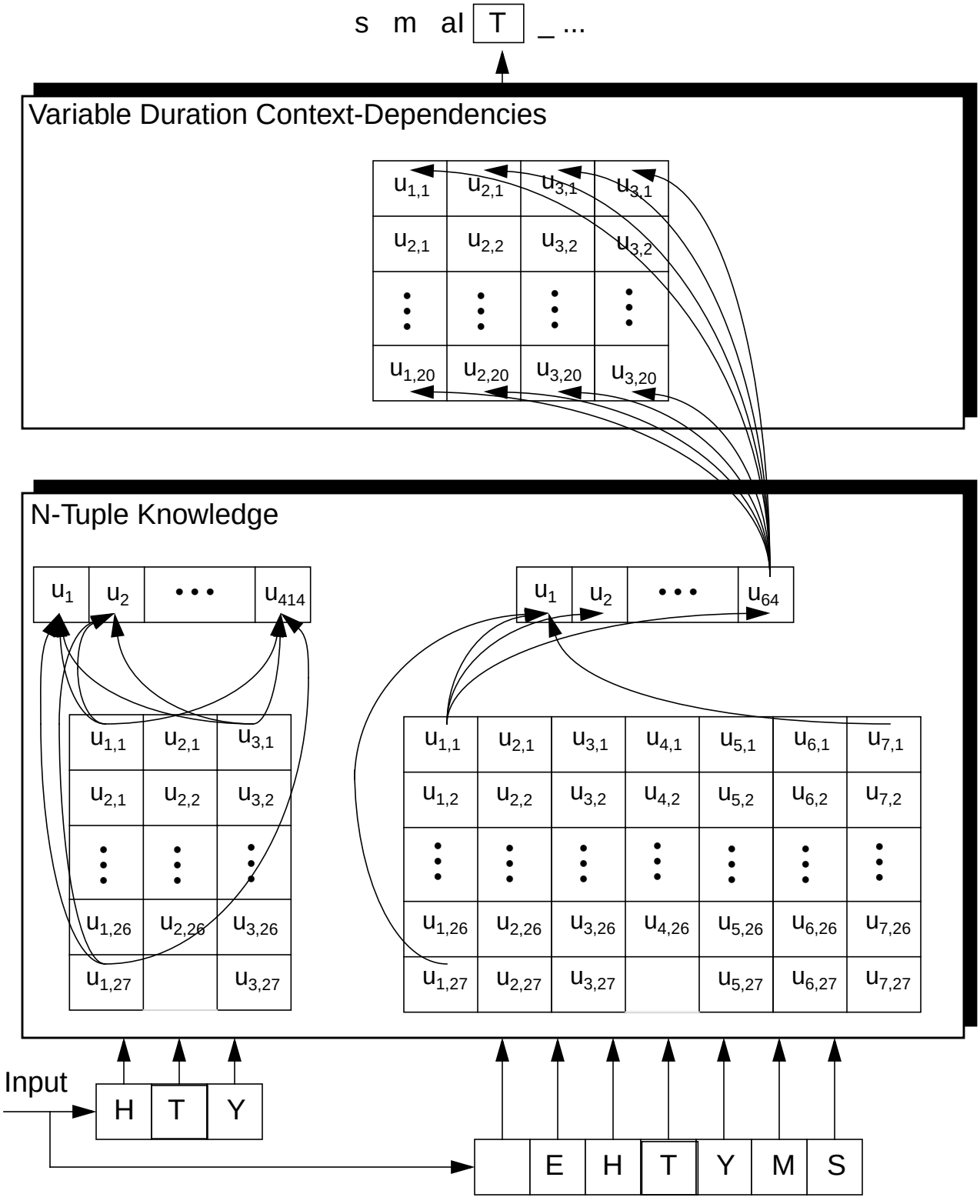


Figure 1. An overview of a neural network architecture that performs letter to sound conversion. In this example, there are three layers: a layer that converts letters to binary-valued inputs, a layer that converts n-tuples of letters into sounds, and a layer that applies a mixture of short-term and long-term relationships. The system need not be restricted to the architecture shown here.

speech recognition systems [18, 19]. A typical network architecture for generating multiple pronunciations consists of three principal components is described below.

The first step in the conversion of the input to a corresponding set of phonemes is to simply map the ASCII input onto a set of binary-valued units corresponding to the 26 letters of the alphabet, plus whitespace and a few other special characters such as the apostrophe and the period (for a total of 30 input characters — see Appendix B.2). Hence, each input character is mapped to a set of 5 bits, and a n-tuple of input letters is mapped onto a bit-string of length 5 times n. The sliding window approach used at this level is key to insuring that a given n-tuple context, no matter where it occurs in the input text, is localized to the same area of the neural network. This avoids a combinatorial explosion that would result from each n-tuple being replicated for each possible position in the input string.

There are two subnetworks in our design of the architecture, in order to capture both the short-distance as well as the long-term contextual information in the textual spelling of the name. Figure 1 illustrates the particular case in which a 3-tuple is used for short-term context while a 7-tuple is buffered to capture the long-context statistics. Since a majority of phones corresponding to any letter arise from the context of its immediate neighbors in the name spelling, a 3-tuple is adequate to capture many of the letter-to-phone mappings.

However, there are still a number of phones that result through a combination of four or more letters (e.g. “ough”), or due to long-range relationships between n-tuples. For instance, in a name like “Hollinshead” the letters “sh” do not result in the phoneme “S”, but instead create two distinct sounds “z h”. Attributes such as the language from which the name originates and knowledge reflecting an understanding of the morphological structure of the name require representation in the network. For example, if the name is known to be of French origin, vowel-vowel-semivowel clusters such as “ier” would most likely need to generate an alternate French pronunciation. Such cases are indirectly handled through the use of a longer context which we capture relationships between 3-tuples. In the implementation of the system the size of the n-tuples is left as a user-defined parameter.

The next step in the conversion is to transform the bit-string output of this input layer into some representation of sounds or features corresponding to the pronunciation of the name. This is performed by the internal (or hidden) units in the network. The connection weights between the input buffers and the hidden layer are used to represent knowledge about n-tuple letter sequences (for example, “str” has a fairly unique pronunciation independent of its larger context). Even though the architecture illustrated has a single hidden layer for each context, it is possible to generate networks with multiple layers of hidden units that can potentially capture the finer constraints in the problem space.

The last level of the system maps internal units into external observables. Rather than deal directly with phone units as outputs, we prefer the use of an indexing system to the output symbols in order to reduce the complexity of the system. As the number of phonemes used to transcribe the pronunciations is about 50 (see Appendix B.3), we have used 6 units at the output layer. The combination of the output bits of these outermost layer neurons specifies the corresponding phoneme output of the network.

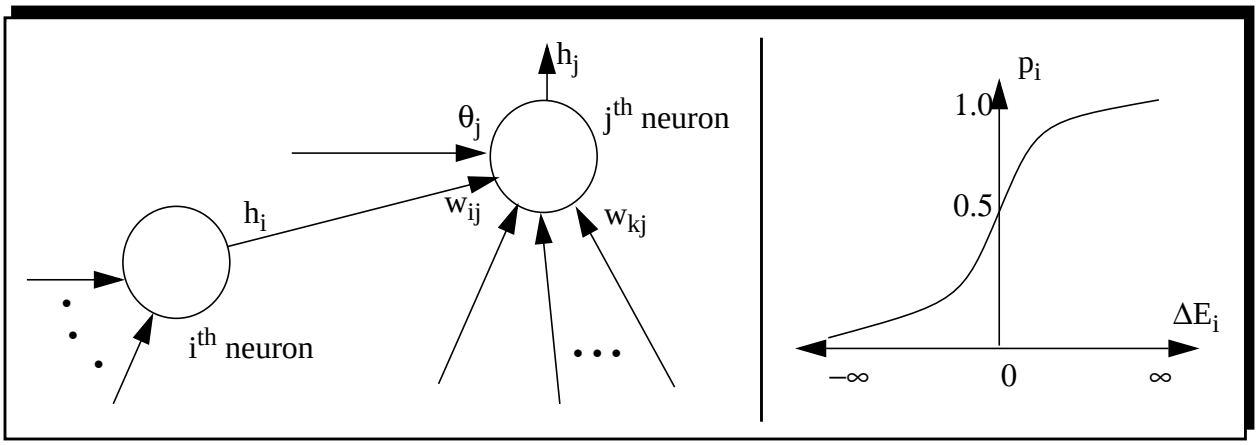


Figure 2. Typical neuron connections in a Boltzmann machine network, along with the Boltzmann distribution function that governs the activation probability of a neuron.

By repeatedly applying the same name as input to the Boltzmann machine, different phonetic feature sequences can be produced, corresponding to alternate plausible pronunciations of the name. Thus, the network succeeds in determining not only a nominal or "correct" pronunciation, but also describing all likely pronunciations of the name. The activation probability of each unit in the network also provides a likelihood measure or "score" for each pronunciation.

4.2. Mathematical Foundations

Figure 2 illustrates a typical neuron connection in the Boltzmann machine network. We denote a connection weight with w, an input to the neuron as h and the activation threshold value for a neuron by θ.

The energy of the network is defined as the sum of the products of all the connection weights with the bits they connect, and the products of all activation thresholds with the respective neuron outputs. Thus

$$E = -\sum_{i<j} w_{ij} h_i h_j + \sum_i \theta_i h_i \tag{1}$$

where w_{ij} is the weight of the link between the units i and j, h_i is 1 if the unit i is *on* and 0 otherwise, and θ_i is a threshold, as shown in Figure 2. This energy value can be interpreted as a metric of the system deviation from the constraints implicit in the data (or alternately, a measure of the mismatch between the statistics of the input data and the statistical model represented in the machine). By minimizing this energy the system is forced to evolve in a fashion that progressively satisfies these constraints.

A simple technique to minimize the global energy function is to switch each individual unit to a state that results in a lower energy value given the current states of the other units [16]. The global *energy gap* of the system at each unit is the difference in energy with the unit hypothesis accepted

and rejected, and is given for the j^{th} unit by

$$\Delta E_j = \sum_i w_{ij} h_i - \theta_j \tag{2}$$

The threshold value can also be incorporated in the sum by assuming it to be another weighted connection which is clamped to a bit that is always set to 1. This simplifies the energy gap computation, reducing it to a scalar product operation. Here w_{ij} consists of both the threshold value as well as the connection weights associated with this neuron.

$$\Delta E_j = \sum_i w_{ij} h_i \tag{3}$$

Thus by adopting the *on* state if the total weighted input to a neuron exceeds the threshold, we get the familiar decision rule of binary thresholding. Binary thresholding ensures that the system comes to rest in some local energy minimum. However, this is not an optimal state of the machine for constraint specification. It is possible to escape from poor local minima by allowing the network to occasionally jump to states of higher energy. This is achieved by employing the simulated annealing technique [20] of stochastic relaxation. If the energy gap between the *on* and *off* states is ΔE_j, then regardless of its previous state the probability of a unit turning *on* is given by the following probability distribution function (see Figure 2) —

$$p_j = \frac{1}{1 + e^{(-\Delta E_j)/T}} \tag{4}$$

where T is a parameter that acts similar to temperature in the simulated annealing algorithm. If the system reaches *thermal equilibrium* in a global state α corresponding to some unit being *on*; and a state β if that neuron is *off*, then the relative probability of the system being in either of the two states is given by

$$\frac{P_\alpha}{P_\beta} = e^{\frac{(E_\beta - E_\alpha)}{T}} \tag{5}$$

which is the Boltzmann density distribution. An interesting feature of a Boltzmann machine is that the equilibrium distribution uses only locally available information at each unit, even though a local change in the weights optimizes a global measure.

4.3. 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 letter by letter. Each output unit is clamped to represent the correct feature sequence, while each input unit is clamped to represent the correct letter combination. The network energy is then computed and a simulated annealing procedure is used to bring the network to equilibrium. 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 [21] 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 $k-1^{th}$ 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 t^{th} iteration through the training data. Let η 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 —

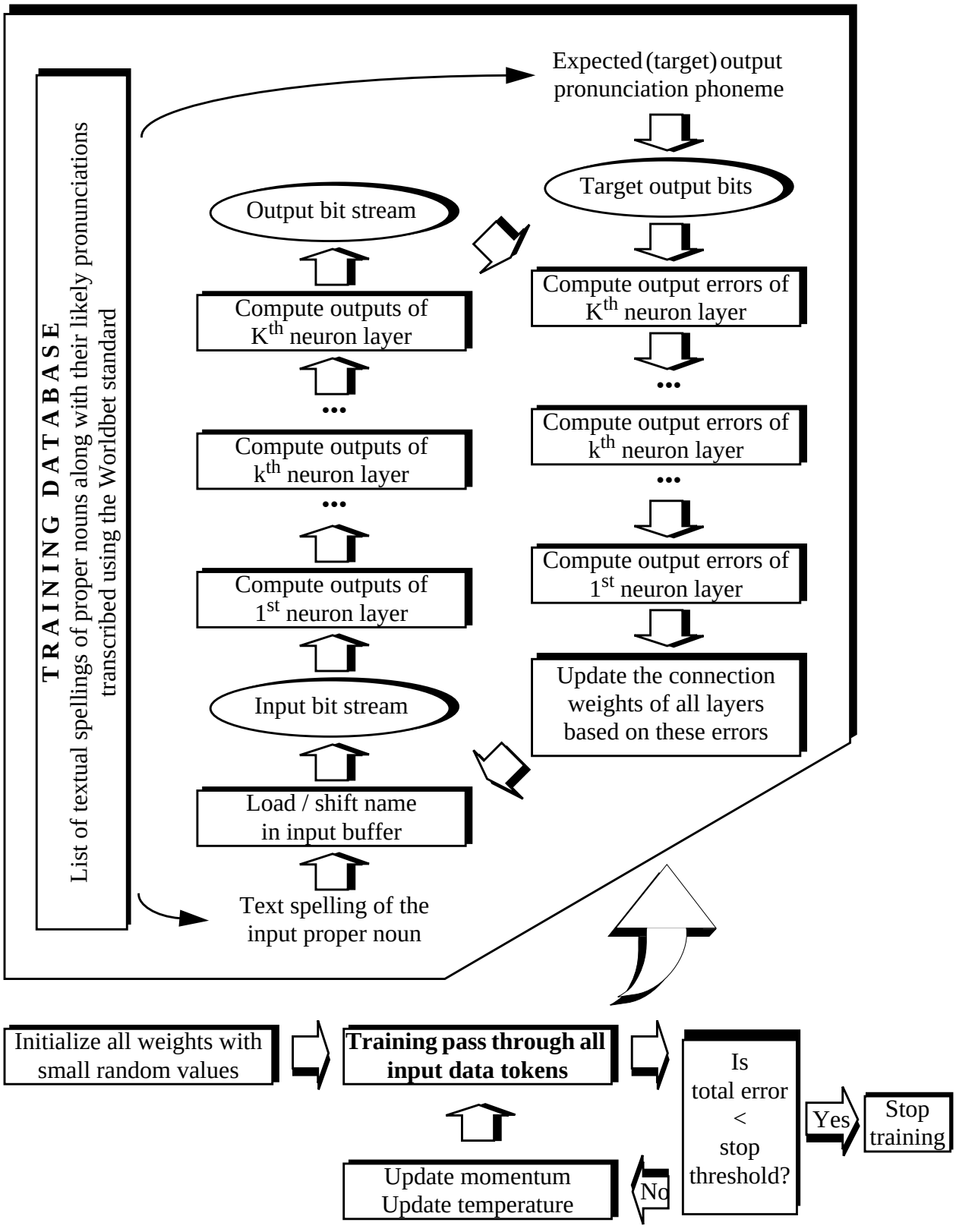


Figure 3. A schematic overview of the simulated annealing used to train the Boltzmann machine neural network. The backpropagation of error training pass is displayed in detail in the inset.

$$\Delta E_{1_j} = -\sum_{i=0}^{N_0} (w_{1_{ij}} x_i)$$
$$h_{1_j} = \frac{1}{1 + e^{\Delta E_{1_j}/(T(t))}} \tag{6}$$

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 $k = 2,3,\ldots,(K-1)$ —

$$\Delta E_{k_j} = -\sum_{i=0}^{N_{k-1}} (w_{k_{ij}} h_{k-1_i})$$
$$h_{k_j} = \frac{1}{1 + e^{(\Delta E_{k_j})/(T(t))}} \tag{7}$$

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

$$\Delta E_{K_j} = -\sum_{i=0}^{N_{K-1}} (w_{K_{ij}} h_{K-1_i})$$
$$o_j = \frac{1}{1 + e^{(\Delta E_{K_j})/(T(t))}} \tag{8}$$

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 δ_{K_j}.

$$\delta_{K_j} = o_j(1-o_j)(y_j - o_j) \tag{9}$$

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 $k+1^{th}$ layer and is denoted by δ_{k_j}. For all $k = K-1,\ldots,2,1$ —

$$\delta_{k_j} = h_{k_j}(1-h_{k_j}) \sum_{i=0}^{N_k} \delta_{k+1_i} w_{k+1_{ij}} \tag{10}$$

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* η and the *momentum* or feedback coefficient α. For all $k = K, K-1,\ldots,2,1$ —

$$\Delta w_{k_{ij}} = \eta \delta_{k_j} h_{k-1_j} + \alpha(t) \Delta w_{k_{ij}}$$
$$\Delta w_{1_{ij}} = \eta \delta_{1_j} x_j + \alpha(t) \Delta w_{1_{ij}} \tag{11}$$

9. Steps 3 through 8 are repeated for the next input token. This is continued till all input tokens are exhausted. A complete training 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 β is specified by the user to customize the training schedule.

$$\begin{aligned} \alpha(t) &= 1 - e^{-\beta t} \\ T(t) &= T_0 e^{-\beta t} \end{aligned} \tag{12}$$

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 [22, 23] 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 Appendix A.

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.

4.4. 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 8.

The system performance is scored automatically using a scoring software that locates the reference pronunciations in the dictionary and compares them with the pronunciations generated by the network. A score file is then created which contains each test case proper noun, the number of reference and hypothesis (generated) pronunciations for it and the number of correct pronunciations generated. The correct hypothesis pronunciations and the incorrect pronunciations are also listed in decreasing order of likelihood scores. A brief summary is provided in the end of this file which contains the overall scoring statistics — the number of test case proper nouns, the number of total reference pronunciations, the number of pronunciations generated and the error statistics as described earlier.

5. CHARACTERIZING THE SYSTEM

As described earlier, the system input is designed to be strings containing any of 30 characters (alphabet and punctuation). For each input token the machine outputs one of 46 phonemes (see Appendix A). Before proceeding with full-scale training with real names and pronunciations, we conducted a series of basic experiments to familiarize ourselves with the functioning of the system. These primarily involved classification of simple character strings that were linearly separable. We provide a brief summary of these exercises that helped us in realizing the effect of individual network parameters as well as gaining some insights into the performance pattern of the system.

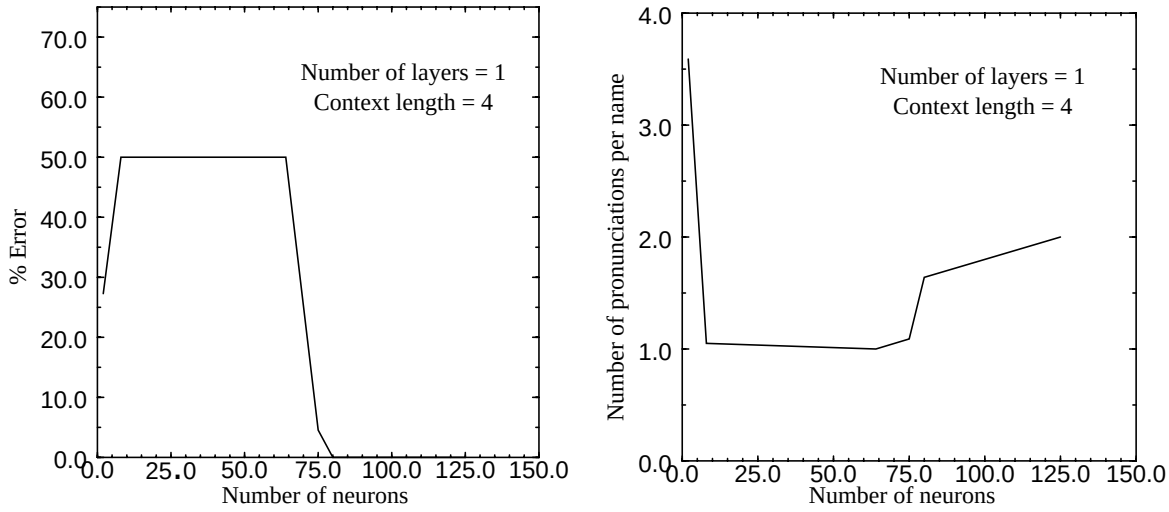
5.1. Two-Class Classifier

The system was trained on simple character sequences corresponding to only two output phonemes to study the effect of the following parameters —

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

In the most basic case, these strings were a fixed length and each letter in the input string corresponded to a phoneme in the output string (no insertions of the blank phoneme). For example, we used the four-letter strings *aaaa* and *bbbb* that corresponded to the phoneme strings *@ @ @ @* and *b b b b.* In each set of experiments the system was trained to completion with an initial temperature of 100 and an exponential temperature decay rate of 0.1 per iteration. Two main sets of experiments were performed with such 4-letter strings.

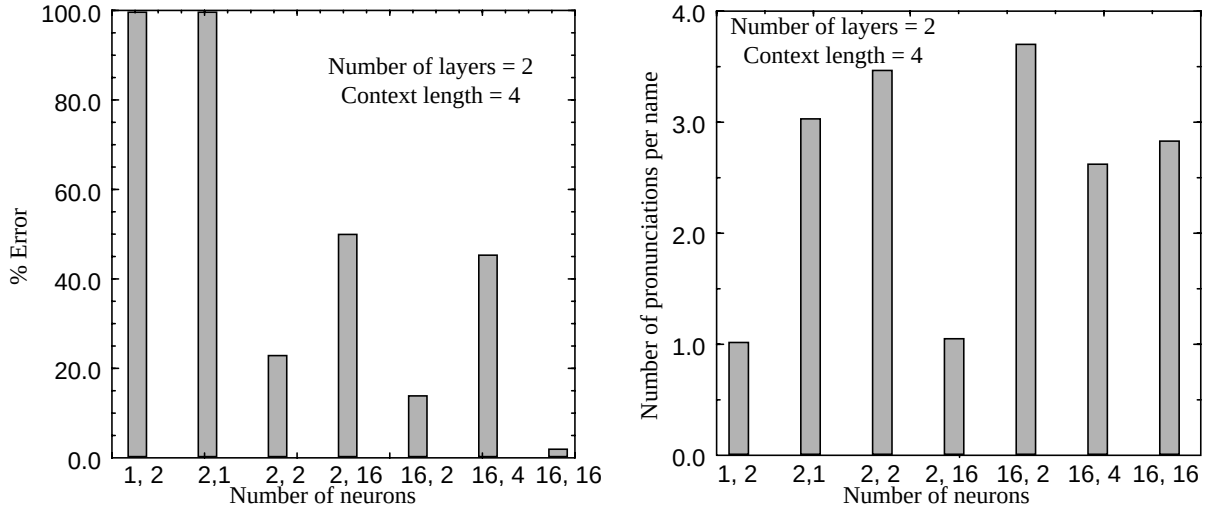
In the first case there was no shifting of inputs and the entire 4-letter word was loaded into the network at once to output a single phoneme class corresponding to that word. The system was



(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. The error rate remains constant after a threshold number of neurons is crossed and the only difference is in the generation rate. This case is with a single hidden layer of neurons.

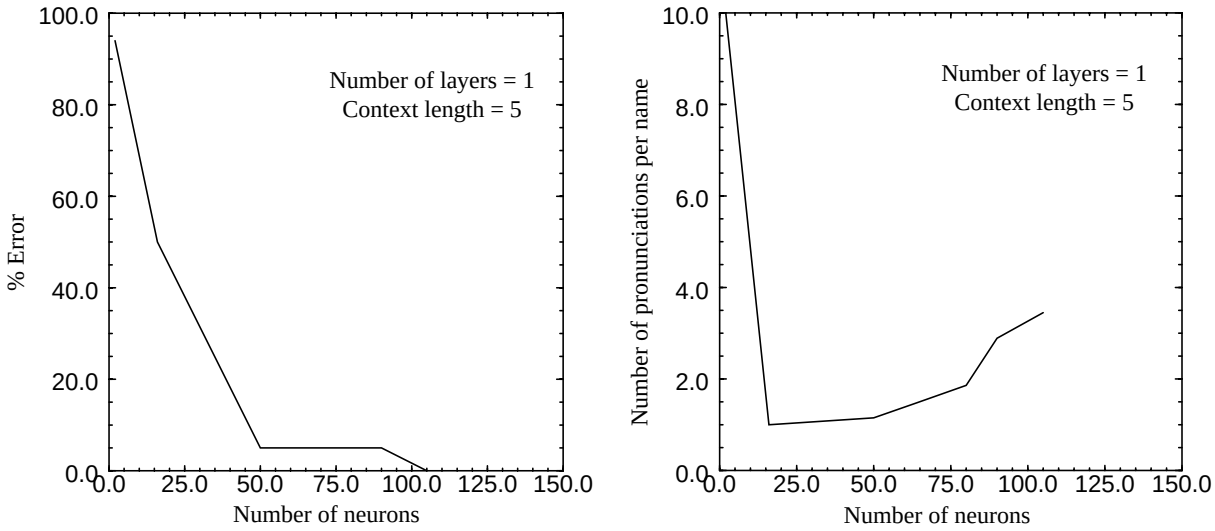


(a) Error rate as a function of number of hidden layer neurons

(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. The error rate remains constant after a threshold number of neurons is crossed and the only difference is in the generation rate. 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.

trained using 22 strings of 4 letters each, with one letter in the input spelling corrupted. For instance, the strings *aaba* and *aaxa* also corresponded to the output phonemes @ @ @ @. Figure 4 describes the effect of various parameters on system performance when a single hidden layer is used. The performance with two hidden layers is displayed in Figure 5.



(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. The error rate remains constant after a threshold number of neurons is crossed and the only difference is in the generation rate. This case is with a single hidden layer of neurons.

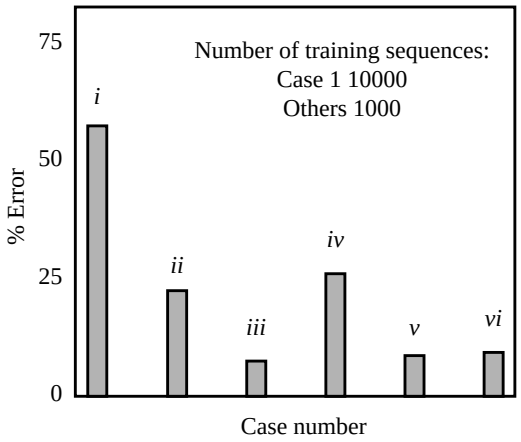
In the second set of experiments the input letters were buffered and shifted in the input register, and phoneme strings output one at a time. It was observed that with shifting, a 3 or 4 letter context was inadequate to handle input strings of length 4. Some results obtained using a 5-letter context are displayed in Figure 6. A single hidden layer network was used for this experiment.

With a small number of neurons the error rate as well as the generation rate was high. Thus, even though a large number of pronunciations were generated most of them were spurious. Only after some optimal number of hidden layer neurons was crossed did performance become meaningful. With two layers of hidden neurons the system converged for a smaller number of neurons to give a comparable performance to the single-layered case.

5.2. Alphabet Classifier

A similar set of experiments as described in the previous section 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 system was trained on simple character sequences that corresponded to 26 output phonemes. In one case, these strings were fixed-length and each letter in the input string corresponded to a phoneme in the output string (no insertions of the blank phoneme). For example, we used the four-letter strings *aaaa* and *bbbb* that corresponded to the phoneme strings @ @ @ @ and *b b b b.* The other set consisted of experiments using variable length (4 or 5 characters) strings as input and output phoneme strings of corresponding length. In each set of experiments the system was trained to completion with an initial temperature of 100 and an exponential temperature decay rate of 0.1 per iteration.

Experiments with fixed-letter strings: We conducted a number of experiments for strings of



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.

fixed length. Experiments *i, ii and iii* employed both contexts to map the input letters to the output sounds and in each case a short-length context of size 3 and a long-range context of size 7 were used for training. A single hidden layer of neurons was used for each context. Cases *i* and *iii* corresponds to a system trained on strings of length 4 and at most a single impurity (such as *aaaa, bbxb, zhzz* etc.) while case *ii* corresponds to a system trained on strings of length 3. The short-context length for case *ii* is 4. Cases *iv* and *v* correspond to a single context Boltzmann machine. For case *iv* the context length was 4 while for case *v* it was 5. The corresponding results are described in Figure 7.

Experiments with variable-length strings: Another experiment was conducted on a single context network with variable length strings (such as *aaaaa, bbxb, zzhzz* etc.) with at most one impurity as input. This is depicted as case *vi* in Figure 7.

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.

6. BUILDING A TRAINING DATABASE

As is the case in most forms of speech research, progress in the development of such a technology is fueled by a comprehensive development database. No appropriate databases existed for training such a system capable of generating multiple pronunciations of a proper noun. Numerous data has been collected anecdotally on the problem of alternate pronunciations, but none of this data has ever been incorporated into a publicly available database.

A significant portion of the effort expended in this work was devoted to the development of a comprehensive training database. This database currently consists of 18494 surnames (last names) found in the United States of America and contain names from numerous ethnic origins. A total of 25648 pronunciations have been transcribed for these names. This database adheres to the Worldbet pronunciation standards and represents a reasonably diverse set of names with which we have developed significant experience and a great deal of confidence.

6.1. Collection And Transcription

Construction of a representative database of surnames presents some peculiar problems. The distribution of surnames is quite skewed — the 2000 most common surnames account for about 15% of the American population, while the remaining 85% population covers almost 1.5 million surnames. In our database we have strived to achieve 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.

A significant part of this database of surnames was taken from a study performed at Texas Instruments [2]. The list of the 2000 most common American surnames came from the Social Security Administration records compiled in 1964 [24]. Bellcore provided a list of medium-frequency names as well as some names that look for somewhat subtle distinctions among not very foreign names and/or common patterns in some foreign names. A number of last names were generated at Texas Instruments as those belonging to the people working at Texas Instruments Computer Science Center circa 1990. A list of surnames was extracted from electronic phone books compiled by the Naval Research Laboratory. Another was created at MIT from people's usernames on the MIT computers.

We also received a list of about 8000 surnames from the 30000 names database being developed at the Oregon Graduate Institute [25]. However, of these only about 2000 were found to be new additions to our dictionary.

The phonetic transcription was performed by hand using the Worldbet standards. Each name was transcribed to obtain all the correct pronunciations possible. Transcription of name pronunciations was a difficult task as the names derive from dozens of source languages, a lot of foreign names have both ethnic as well as anglicized pronunciations and individual pronunciations are often peculiar in defying any kind of text-to-speech rules.

6.2. Phone Alignment With Spelling

The Boltzmann machine network is designed to look at each letter of the input name spelling one by one in context of its nearest neighbors, and output a phoneme symbol in accordance. Thus for training purposes it is necessary that there be 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 (e.g. in 'Wright' the two letters 'Wr' account for a single phoneme '9r'. Similarly three letters 'igh' correspond to the one phoneme 'aI'.).

To satisfy this need to align the spellings with the corresponding phonetic expression we have introduced the concept of a *blank phoneme* denoted by the symbol _ (Appendix B.3). We have developed a dynamic programming algorithm that performs automatic alignment by introducing such an *empty phoneme* at appropriate places. This algorithm assigns numeric scores to all possible letter-to-phone maps. A phoneme corresponding to a group of letters is aligned with one of the letters according to a strategy that maximizes the total alignment score for the entire word. The other letters are aligned with the blank phoneme. For instance, our previous example 'Wright' is transcribed and aligned as '_ 9r aI _ _ t'.

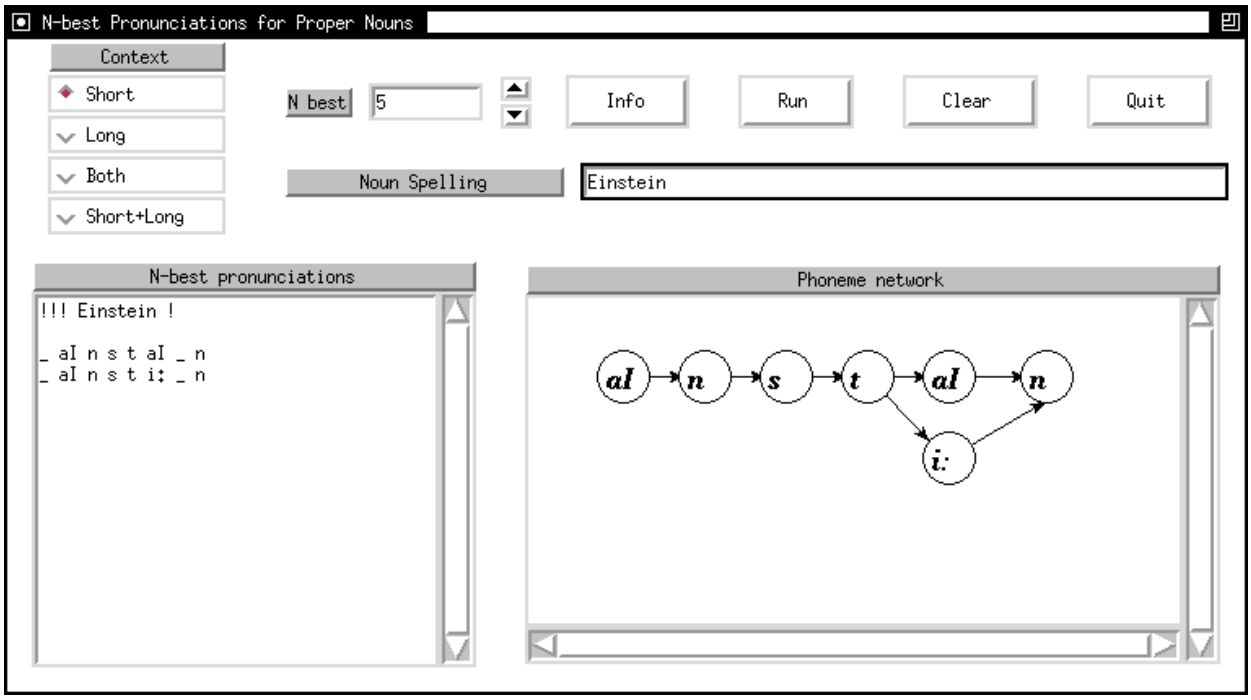


Figure 8. The graphical user interface for the N-best pronunciations system. The output consists of an ordered list of all possible pronunciations along with the corresponding pronunciation network.

7. USER INTERFACE

An important aspect of developing a software application is to provide the user with an easy way of demonstrating its performance. We have developed a Tcl-Tk based GUI that provides a neat demonstration of the Boltzmann machine system generating multiple pronunciations and the corresponding phoneme networks. The user needs to select some system parameters and type in only the spelling of the name for which the pronunciation network is desired. A snapshot of this pronunciation generating tool is shown in Figure 8.

The demo interface allows the user to clear the display area, enter a new proper noun as input and execute the N-best generation of pronunciations. Some online help about the demo and the pronunciation generation system has also been provided.

The top left part of the demo contains parameter options which are used to input the network conditions of context length and the number of N-best pronunciations desired. The context lengths in this initial version of the demo are limited to 3 for short and 7 for long. The user can select one of the four possible context settings —

- short — the system was trained for short context
- long — the system was trained for long context
- both — the system was trained using both short and long contexts simultaneously
- short+long — the system was trained using the short and long contexts individually, but both contexts are used simultaneously for generating the pronunciations

An adjacent field allows the user to enter the desired number for the value of N-best. The system

can then generate at most N-best likely pronunciations. Alternatively, the user may increment or decrement the default setting for the value of N using the up and down arrow buttons. The pronunciations are output in decreasing order of likelihood scores.

The bottom part of the window is for the output display and is divided in two sections. The left portion lists all the generated pronunciations in a decreasing order of their likelihood scores. These pronunciations are aligned with the text spelling by suitable insertions of the blank phoneme. A state diagram indicating the phoneme network which could have generated the various pronunciations is displayed in the bottom right part of the window. The pronunciation network does not include the blank phonemes. The GUI makes efficient use of the screen real estate and computer resources, so that a user can work effectively even from a small 15” black and white monitor.

8. 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. Since the development of the surname-pronunciations database was concurrent with the implementation of the system in software, the system performance has been calibrated at different stages of completion of the database. For instance, a set of pilot experiments was carried out using names with 4-letter spellings. As briefly described earlier in Section 5 a number of early experiments involved classes of alphabet strings. A final set of experiments spanning the entire training database was carried out to extensively study the effect of various system parameters (such as number of neurons, size of context and training schedule). The outcome of all such experiments is documented in the following.

We have followed a simple-to-complex approach in evaluating this system. The initial experiments with two-class alphabet strings provide us with a broad idea of the basic functioning of the system and its dependence on various parameters. The subsequent benchmarks on the multi-class alphabet strings allow us to study how the network scales up to a more challenging task. With real data (proper nouns and their pronunciations) again we first evaluate on a pilot benchmark of 4 and 5 letter surnames before scaling the system up to handle the larger database.

8.1. Two-class Alphabet Strings

The two class problem is fairly simple to solve using a mechanism as powerful as the Boltzmann machine, yet it allows a lot of insight in the behavior of the system and its response to various training parameters. The data used for this set of experiments consisted of 4-letter strings of two alphabets (which corresponded to the two classes), such as *aaaa* for class 1 and *bbbb* for class 2. The training set was created by corrupting such strings in at most two random positions with some random letters (e.g. *aaxa, bcyb, kaaf* etc). The training schedule for this set of experiments had an initial temperature of 100 that decayed exponentially at the rate of 0.1 per training iteration. The evaluation was closed loop i.e. the training data was also used as the test set.

In the first round of evaluations in this experiment the system was made to look at the entire 4-letter string and output a class at once in a single step, without any shifting of the input letters in

the input buffer of the system. Thus the context length was fixed to be 4. The system was trained and tested on 22 strings created as described above. The performance for one and two hidden layers with different number of neurons per layer is summarized in Table 1.

# 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. In this case there was no shifting of input letters.

Next, the input strings were fed to the Boltzmann machine buffered through the input shift register. The machine was trained on different context lengths with 100 4-letter strings equally distributed between the two classes. This experiment was conducted with a single hidden layer of neurons. The performance is summarized in Table 2.

context length	# neurons	classification error
3	70	41.00%
3	100	36.00%
4	100	33.00%
5	2	94.00%
5	16	50.00%

context length	# neurons	classification error
5	50	5.00%
5	64	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.

We observed that a context of 5 letters is necessary to completely solve the 2-class problem for data strings of length 4. This is expected as the machine needs to look at a minimum of three letters at each time to decide which class the letter string belongs to, and at the endpoints of the data string (first and last letters) the input buffer can hold 3 (or more) letters only for a context length of 5 (or higher). The performance as a function of number of hidden neurons for this context length is depicted in Figure 9. Further experiments were performed with a context length of 5 and 105 hidden neurons to study the effect of number of training iterations on performance. The results for this experiment are summarized in Figure 10.

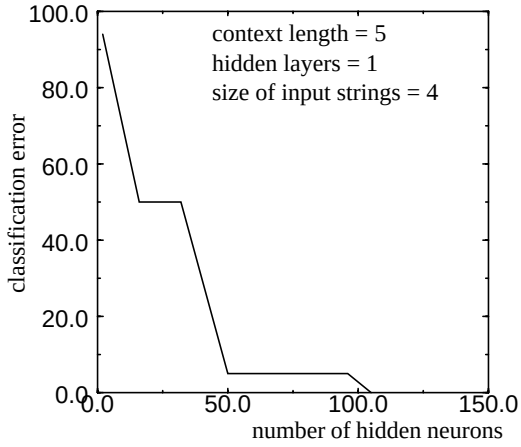


Figure 9. 2-class classification performance as a function of number of hidden neurons.

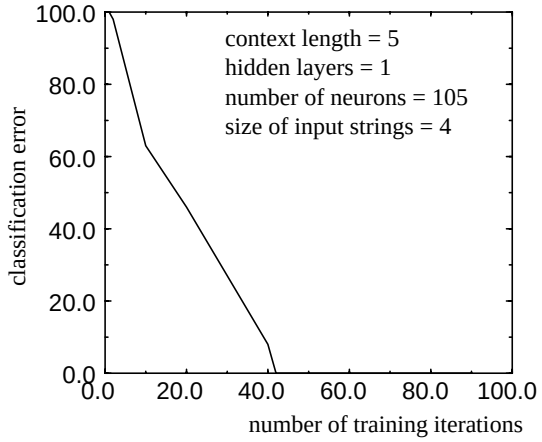


Figure 10. 2-class classification performance as a function of number of training iterations.

8.2. Multi-class Alphabet Strings

The next step in understanding and evaluating the performance of the Boltzmann machine system was to extend the previous experiments to multiple classes. Now the system was trained on variable-length strings of alphabet characters where each letter of the alphabet represented a class. The experiments and the results are described below in Table 3.

training data string size	# training strings	context type	context size	# hidden layers	# neurons / layer	classification error
3	1000	both	short 3 long 7	1 1	125 125	22.03%
4	10000	both	short 3 long 7	1 1	125 100	56.47%
4	1000	both	short 4 long 7	1 1	200 200	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.

The training schedule had an initial temperature value of 100, and a decay rate of 0.1 per training iteration. The data strings consisted of pure strings (i.e. strings 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. While the majority of the experiments involved closed loop evaluations, two experiments were evaluated with an open loop test set i.e. the test set consisted of 1000 strings that were different from those used to train the system.

8.3. Evaluation With Four-letter Proper Nouns

The previous sets of experiments provided us with a broad idea of the functioning of the Boltzmann machine classification system. In order to study its working in the context of the present problem — that 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 case.

We realized that the performance of the system differs considerably on this “real” application compared to that on the pilot evaluations described in the previous two subsections. Yet there is a significant amount of information that can be gathered from the results on this evaluation.

The training set comprised of corresponding 2022 name-pronunciation pairs for a total of 1665 proper nouns (on average 1.22 pronunciations per proper noun). The evaluations were closed-loop i.e. on the same set of data. The results are summarized in Table 4.

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.

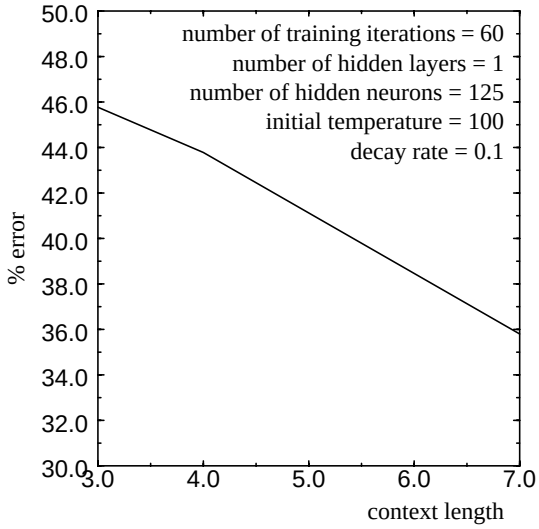


Figure 11. Effect of context length on performance for the 4-letter proper nouns.

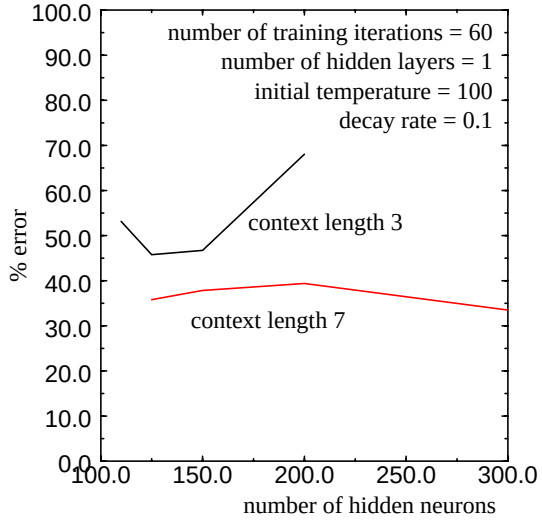


Figure 12. Effect of number of hidden neurons on performance for the 4-letter proper nouns.

As described earlier in section 4.4, the *all correct pronunciations* corresponds to the case where the system output contains all the possible pronunciations of the name. The *some correct pronunciations* corresponds to the situation where some of the likely pronunciations are contained in the system output. The *no correct pronunciations* case means that none of the likely pronunciations were output by the system.

Effect of context length: It can be deduced from Table 4 that the length of the context in which each input letter is evaluated (as regards its mapping to a corresponding phoneme) is of considerable importance. As the context length increases the performance improves as well. This behavior is described in Figure 11.

Effect of number of neurons: The number of hidden neurons made a significant difference in case of the shorter length contexts. For the longer contexts the performance did not vary significantly with number of neurons [Figure 12]. It appears as if the number of hidden neurons has an optimal value at which the performance is maximized, and the error rate increases if the number deviates from this ideal value. However, the optimal number of neurons depends on the type of input data and is therefore difficult to calculate analytically. In these evaluations we have attempted to determine this value empirically and experimentally.

Training schedule: The most idea 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. Typically about 60 iterations through the entire training set 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. There was considerable overgeneration of pronunciations i.e. the number of pronunciations generated by the Boltzmann machine was typically higher than the number of pronunciations for the corresponding name in our reference pronunciation dictionary.

8.4. 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 (which consists of 18494 names and 25648 pronunciations) into a training set and a test set. The training set consists of 15000 names selected at random from the dictionary and their corresponding pronunciations. The remaining 3494 names constitute the test set. This division of data was done three times to create three different training sets of 15000 names each (which have some overlap with each other) and three different test sets (which do not overlap with each other or the corresponding training set at all). Each training set consisted of approximately 20000 pronunciations corresponding to the 15000 names.

The Boltzmann machine system was trained and evaluated on each of the three data sets for each set of system parameters, thus producing three benchmarks for each case. We found no significant difference in these over any parameter set and therefore we will present the only the overall results (averaged over the three data sets) in this section.

The training experiments were conducted for various values of the number of hidden neurons, number of training iterations and training schedules. The results for this benchmark for a single context system are described in Table 5.

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.

It can be seen that the performance fails to scale up as the problem is made more complex by introduction of the complete data set for training. In addition to a larger number of letter-to-sound

mappings the machine is also required to assimilate the effects of a variable-length input, as the proper noun spellings are of different lengths.

Table 6 depicts the performance measured for a system using both the short-term and long-term contexts simultaneously.

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

The performance in this case is found to be extremely degraded. In spite of using a large number of hidden neurons and a fairly reasonable number of training iterations the system proves to be quite 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.

8.5. Run-Time 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. It was found that for an average name length of 6 letters, the system required on average approximately 160 ms to train each proper noun-pronunciation pair per iteration of training with a single context of length 3 and 1000 neurons in the single hidden layer. The generation rate for this case was about 5 ms per proper noun pronunciation. These figures scaled up almost linearly with longer contexts and/or more hidden neurons. For instance, for the case described in the last row of Table 6 (both contexts, 10000 hidden layer neurons for each context) a single training iteration on the complete training data set required almost 30 hours. This strongly demonstrates the need for faster CPUs to achieve a reasonable amount of training in a practical time frame.

The requirements on system memory were extremely reasonable. For the 10000 neurons both-contexts case the network training process required only about 7K of the system dynamic memory, in other cases the memory usage was proportionately smaller. The memory space required to store the trained network (essentially the connection weights between different layers of the network) was in accordance with the network size.

9. CONCLUSIONS

Accurate pronunciation of proper nouns is a key issue in speech recognition and voice interface applications. As proper nouns follow unconventional letter-to-phoneme transformations as compared to regular words, a rule-based approach to automatically pronounce such names is not very successful. The Boltzmann machine — a stochastic neural network capable of producing multiple outputs for a single input — has the potential to generate an N-best list of pronunciations by simply analyzing the text spelling of a proper noun.

Implementation of such a system, in spite of showing fairly encouraging results for small-scale applications, has proved to be a non-trivial problem. A major task in the design of such a network is the selection of the optimal number of hidden layers and the number of units in these hidden layers. Another issue is setting the training schedule to maximize the amount of information captured by the network. This is a complex process as these parameters depend upon application-specific quantities such as the amount of training data, the size of the input shift register and the maximum error (stopping criterion) allowed in training.

While the system implementation is a matter that deserves further research, the pronunciation dictionary is a valuable accomplishment of this project. The dictionary, currently consisting of 18494 surnames and 25648 pronunciations, is the only resource of its kind providing multiple possible pronunciations for various proper nouns. We also intend to augment the database with more names in the future.

We appreciate the need for good public-domain corpora for training various applications in speech recognition. We expect the pronunciation dictionary to be a useful resource to the entire speech research community in this regard. The complete database, as well as all the software developed in course of this project is available in the public domain at *http://www.isip.msstate.edu/software/n_best_pronunciations/*.

10. FUTURE DIRECTIONS

There are a number of ideas we would like to explore in the future to make the implementation of the pronunciation generation system more meaningful. Based on our observation that the system has a better performance on smaller subsets of the training data, we plan to implement a number of subnetworks to handle different parts of the proper noun database which combine to generate the overall pronunciations for the input names.

One such approach is to have a two-stage network, where the first stage is a deterministic machine generating the one best pronunciation for the input noun. This is then fed to a stochastic network that generates variants of this base phoneme string. Another method we intend to explore is a

combination of the Boltzmann machine with time-delayed neural networks (TDNNs).

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.

11. 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.

12. 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] 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.

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

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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-37-40, Adelaide, South Australia, Australia, April 1994.

[19] 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.

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

[21] 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.

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

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

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

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

APPENDIX A. DERIVATION OF THE TRAINING ALGORITHM

We describe here the theoretical derivation of the training algorithm and the weight-update equations discussed in Section 4.3. We present the derivation for the simple case of one set of weights connecting two layers. The extension to multiple sets of connecting weights (more layers) is trivial.

Beneath the simulated annealing and error backpropagation the primary weight-updating algorithm is essentially a gradient descent method, where it tries to minimize the asymmetric divergence (an information-theoretic measure of the distance between two probability distributions) between the network energy distributions generated by the reference pronunciation and the hypothesis phoneme string.

A.1. The Weight-update Algorithm Derivation

Let $\{w_{ij}\}$ be the set of weights connecting units in the two layers, such that $\{h_i\}$ represents the set of output bits of one layer (say the input layer) and $\{h_j\}$ those of the other (output layer). Let α be a global state of this Boltzmann machine neural network corresponding to the case where the outputs represent the hypothesis pronunciation i.e. are set according to the input bits and not clamped externally; and the network is in thermal equilibrium. When the output bits are clamped to their desired values, let the state of the network at thermal equilibrium be represented by $\widehat{\alpha}$. If P_α indicates the probability of the system being in the equilibrium state α, then the asymmetric divergence between the distributions at the hypothesis state α and the desired or ideal state $\widehat{\alpha}$ is defined as

$$\Psi = \sum_{\widehat{\alpha}} P_{\widehat{\alpha}} \log \frac{P_{\widehat{\alpha}}}{P_\alpha} \tag{A.1}$$

Assuming that the threshold values at each unit are included in the summation, the total global energy of this system (as described in Equation (1) in Section 4.2) in state α is

$$E_\alpha = \sum_j \sum_i w_{ij} h_i^\alpha h_j^\alpha \tag{A.2}$$

Therefore differentiating $e^{-E_\alpha/T}$ with respect to a connection weight w_{ij} we get

$$\frac{\partial e^{-E_\alpha/T}}{\partial w_{ij}} = \frac{1}{T} e^{-E_\alpha/T} h_i^\alpha h_j^\alpha \tag{A.3}$$

The probability that the system under thermal equilibrium conditions ends up in the global state α is given by the following equation —

$$P_\alpha = \frac{e^{-E_\alpha/T}}{\sum_\lambda e^{-E_\lambda/T}} \tag{A.4}$$

where λ is any global state of the network. Differentiating Equation (A.4) and substituting Equation (A.3) we get

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \frac{\frac{1}{T} e^{-E_\alpha/T} h_i^\alpha h_j^\alpha}{\sum_\lambda e^{-E_\lambda/T}} - \frac{\frac{1}{T} e^{-E_\alpha/T} \sum_\lambda e^{-E_\lambda/T} h_i^\lambda h_j^\lambda}{\left(\sum_\lambda e^{-E_\lambda/T}\right)^2} \tag{A.5}$$

which simplifies to

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \frac{1}{T} P_\alpha h_i^\alpha h_j^\alpha - \frac{1}{T} P_\alpha \frac{\sum_\lambda (e^{-E_\lambda/T} h_i^\lambda h_j^\lambda)}{\sum_\lambda e^{-E_\lambda/T}} \tag{A.6}$$

We use this relation to compute the gradient of the error function. Taking derivative of Equation (A.1) we get

$$\frac{\partial \Psi}{\partial w_{ij}} = -\sum_{\widehat{\alpha}} \frac{P_{\widehat{\alpha}}}{P_\alpha} \frac{\partial P_\alpha}{\partial w_{ij}} \tag{A.7}$$

Using Equation (A.6) we get

$$\frac{\partial \Psi}{\partial w_{ij}} = -\frac{1}{T} \sum_{\widehat{\alpha}} \frac{P_{\widehat{\alpha}}}{P_\alpha} P_\alpha \left(h_i^\alpha h_j^\alpha - \frac{\sum_\lambda (e^{-E_\lambda/T} h_i^\lambda h_j^\lambda)}{\sum_\lambda e^{-E_\lambda/T}} \right) \tag{A.8}$$

Noting the facts that the sum of probability values over the entire domain is 1 and that the input bits are the same in both the reference and hypothesis cases —

$$\begin{aligned} \sum_{\widehat{\alpha}} P_{\widehat{\alpha}} &= 1 \\ h_i^\alpha &= h_i^{\widehat{\alpha}} \end{aligned} \tag{A.9}$$

If we term the error in the output bit as $\delta_j^\alpha = h_j^{\widehat{\alpha}} - h_j^\alpha$, then on further simplification of Equation (A.8) we get the relationship between the gradient of the error function and the bit error.

$$\frac{\partial \Psi}{\partial w_{ij}} = -\frac{1}{T}(h_j^{\alpha} - h_j^{\widehat{\alpha}})h_i^{\alpha} = \frac{1}{T}(h_j^{\widehat{\alpha}} - h_j^{\alpha})h_i^{\alpha} = \frac{1}{T}\delta_j^{\alpha}h_i^{\alpha} \quad \text{(A.10)}$$

Thus to minimize Ψ it is sufficient to change each weight by an amount proportional to the difference between the expected output and the desired output.

$$\Delta w_{ij} = \eta \delta_j^{\alpha} h_i^{\alpha} \quad \text{(A.11)}$$

Here η is a scaling factor that determines the size of each weight change, and in the context of neural network training is called the *learning rate*.

There are a number of possible modifications to the algorithm derived above that affect the learning in terms of speed of convergence. By keeping the scaling factor fairly small we can minimize the noise in incrementing the weights, but a very small value of η also results in a slow learning rate. This can be partly compensated for by adding some momentum to the training. This involves providing some feedback to the weight updates based on the updates for the previous input-output case. The feedback factor μ is called the momentum coefficient, and can be varied to control the direction of learning to some extent. Now the weight update equation takes the form

$$\Delta w_{ij} = \eta \delta_j^{\alpha} h_i^{\alpha} + \mu \Delta w_{ij} \quad \text{(A.12)}$$

which is the what we have used in training the Boltzmann machine.

APPENDIX B. ALPHABET AND PHONEME SETS

The Boltzmann machine neural network is designed to accept a total of 30 characters as input. These consist of the 26 letters of the English alphabet, as well as some special characters such as white space, apostrophe, hyphen and period. Both uppercase and lowercase alphabet symbols are accepted and treated in an identical fashion i.e. map to the same bit-strings in the input buffers.

The output of the Boltzmann machine for each input set is one of 47 different phonemes. The phonemes are transcribed in the Worldbet symbol set. Each phoneme corresponds to a feature set described by a bit-string corresponding to the output bits of the outermost layer of the network.

The following tables summarize the symbols along with their bit-string equivalents. A typical word example is also given with the phoneme set to illustrate the pronunciation it represents.

B.2. Input Alphabet Set

Character symbol	Bit-string
	00000
-	00001
’	00010
.	00011
a	00100
b	00101
c	00110
d	00111
e	01000
f	01001
g	01010
h	01011
i	01100
j	01101
k	01110

Character symbol	Bit-string
l	01111
m	10000
n	10001
o	10010
p	10011
q	10100
r	10101
s	10110
t	10111
u	11000
v	11001
w	11010
x	11011
y	11100
z	11101

B.3. Output Phoneme Set

The letters that constitute the phoneme are indicated in italics.

Phone symbol	Bit-string	Example name
_	000000	—
i:	000001	H*ee*p
I	000010	M*i*ll
E	000011	B*e*ll
@	000100	*A*dams
A	000101	*A*llman
^	000110	Tr*u*mp
>	000111	The*o*dor
&	001000	Alb*a*
U	001001	B*oo*kman
u	001010	D*u*be
aI	001011	P*i*ne
ei	001100	M*a*te
>i	001101	J*oy*
aU	001110	Clo*u*d
oU	001111	Clo*s*e
iU	010000	L*iu*
3r	010001	Danb*ur*y
&r	010010	East*er*
>r	010011	M*or*ton
p	010100	*P*eabody
b	010101	*B*aggs
t	010110	*T*rump
d	010111	Har*d*y
k	011000	Cul*k*in
g	011001	*G*aston
h	011010	*H*ermit
v	011011	*V*ail
D	011100	Wor*th*y
T	011101	Ro*th*
s	011110	*S*andler
z	011111	Abram*s*
S	100000	Na*sh*
Z	100001	*Z*achry
f	100010	*F*ielding
m	100011	A*m*es
n	100100	I*n*ez
N	100101	Goldi*ng*
dZ	100110	*J*ohnson
tS	100111	*Ch*ow
l	101000	Ca*l*ahan
9r	101001	*R*obinson
j	101010	*Y*oung
w	101011	*W*illis
ks	101100	Fo*x*
&k	101101	M*c*Dowell

APPENDIX C. PUBLICATIONS

C.1. Paper Presented At ICASSP '96

N. Deshmukh, M. Weber and J. Picone, "Automated Generation of N-best Pronunciations of Proper Nouns", in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. I-283-I-286, Atlanta, GA, May 1996.

C.2. Paper Proposal Submitted To ICASSP '97

N. Deshmukh, J. Ngan, J. Hamaker and J. Picone, "An Advanced System to Generate Multiple Pronunciations of Proper Nouns", proposal submitted to *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Munich, Germany, May 1997.

AUTOMATED GENERATION OF N-BEST PRONUNCIATIONS OF PROPER NOUNS

Neeraj Deshmukh, Mary Weber, Joseph Picone

Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, Mississippi 39762
{deshmukh, weber, picone}@isip.msstate.edu

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. Recognition of such words requires an ability to generate reasonably accurate pronunciation networks. This is a very challenging problem because a large percentage of proper nouns, such as personal names, appear to have no obvious (or simple) 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 sociological factors. Yet humans do amazingly well at generating and recognizing the pronunciation of a name never encountered before. We present 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 method does not require voice data containing the spelling or nominal pronunciation.

1. INTRODUCTION

As the voice interface market has grown, so has the demand for simple, intuitive, and user-friendly interfaces. 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.

Traditional systems for proper-noun pronunciation employ extensive handwritten rule sets to generate an accurate pronunciation. Such systems, even though deemed accurate for the directory assistance application for which they were designed, essentially solved an ill-posed problem. Moreover, they claimed to generate only the single-most likely pronunciation of a name. This obviously is not the most useful approach for a speech recognition application. Since most proper nouns have a number of highly probable pronunciations that can rarely be differentiated from the context of the application, it is important that all plausible alternatives be available to the recognizer.

An alternative approach is to use massively-parallel network models [3,4]. 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. Multilayered neural networks, in which the internal or hidden units can act as feature detectors that perform a mapping between the input and the output are a class of models ideally suited for such applications.

The system we present in this paper relies on a particular form of neural network designed to generate multiple outputs for a given input — a Boltzmann machine [5]. This network transforms its input, the spelling of a proper name, to a network of distinctive features describing articulatory movements required to produce various pronunciations of the name. Recently, similar work involving Hidden Markov Model-based approaches has also shown some promise [6].

2. BOLTZMANN MACHINE

The Boltzmann architecture is designed to allow efficient searches for combinations of hypotheses that maximally satisfy the input data and some stored constraints. Each hypothesis is represented in terms of a unit in the network whose binary state represents the truth values of the hypothesis. The interaction between such units implements the stored information about the constraints. Data is supplied to these units in the form of external inputs. Constraints are modified by altering the interaction weights.

The Boltzmann machine is a parallel computational architecture [Figure 1] quite similar to a Hopfield network [7]. We can assign each global state of the network a numerical *energy* value, and then make the individual units act to minimize the global energy compatible with each input configuration. The energy of the network is defined as

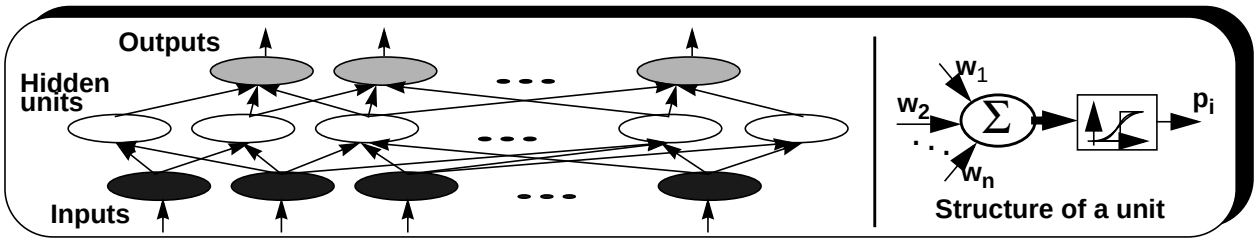


Figure 1. A Boltzmann Machine.

$$= -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i \quad (1)$$

where w_{ij} is the weight of the link between the units i and j, s_i is 1 if the unit i is *on* and 0 otherwise, and θ_i is a threshold, as shown in Figure 1. This energy value can be interpreted as a metric of the system deviation from the constraints implicit in the data (or alternately, a measure of the mismatch between the statistics of the input data and the statistical model represented in the machine). By minimizing the energy the system is forced to evolve in a fashion that progressively satisfies these constraints.

2.1. Minimization of Energy

A simple technique to minimize the global energy function is to switch each individual unit to a state that results in a lower energy value given the current states of the other units [7]. The global *energy gap* of the system at each unit is the difference in energy with the unit hypothesis accepted and rejected, and is given for the i^{th} unit by

$$E_i = \sum_j w_{ij} s_j - \theta \quad (2)$$

Thus by adopting the *on* state if the total weighted input to a unit exceeds the threshold, we get the familiar decision rule of binary thresholding.

2.2. Finding the Global Energy Minimum

Binary thresholding ensures that the system comes to rest in some local energy minimum. However, this is not an optimal state of the machine for constraint specification. It is possible to escape from poor local minima by allowing the network to occasionally jump to states of higher energy. This is achieved by employing the simulated annealing technique [8] of stochastic relaxation. If the energy gap between the *on* and *off* states is ΔE_i, then regardless of its previous state the probability of a unit turning *on* is

$$p_i = \frac{1}{1 + e^{(-\Delta E_i)/T}} \quad (3)$$

where T is a parameter that acts like temperature. Therefore the relative probability of a system in *thermal equilibrium* to be in the two global states α and β (corresponding to some unit being *on* or *off*) is given by $P_\beta = \exp((E_\beta - E_\alpha)/$ which is the Boltzmann density distribution. An interesting feature of a Boltzmann machine is that the equilibrium distribution uses only locally available information at each unit, even though a local change in the weights optimizes a global measure.

2.3. Training the Boltzmann Machine

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. 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 way trade-off is to begin annealing at a high temperature and gradually cool down; performing a coarse-to-fine search for the global minimum.

2.4. The Learning Algorithm

For an input vector x_i, the output of a hidden layer unit is

$$h_i = \frac{1}{1 + e^{\left(-\sum_j w_{ij}^{xh} x_i\right)/T}} \quad (4)$$

where w_{ij}^{xh} are the weights connecting the input units to the hidden units. The output of the units in the hidden layers is propagated to compute the output of the outer layer units.

$$o_i = \frac{1}{1 + e^{\left(-\sum_j w_{ij}^{ho} h_j\right)/T}} \quad (5)$$

where w_{ij}^{ho} are the weights that connect the hidden layer units to the output layer. The errors in the two layers are

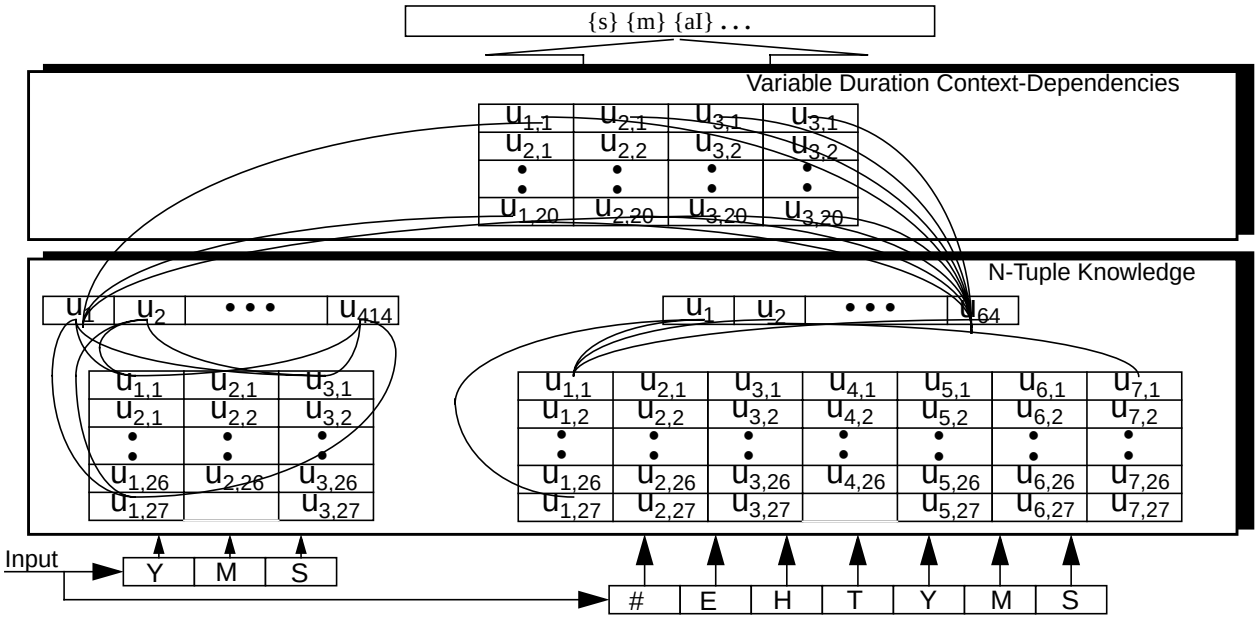


Figure 2. An overview of a neural network architecture that performs letter to sound conversion.

$$\delta_i^{ho} = o_i(1-o_i)(y_i - o_i)$$
$$\delta_i^{xh} = h_i(1-h_i)\sum_j \delta_j^{ho} w_{ij}^{ho} \quad (6)$$

where y_i is the target (or expected) output vector. The weights are updated using these error values with some feedback from the updates in the previous training pass.

$$w_{ij}^{ho}(t+1) = \eta \delta_j^{ho} h_i + \alpha \Delta w_{ij}^{ho}(t)$$
$$w_{ij}^{xh}(t+1) = \eta \delta_j^{xh} x_i + \alpha \Delta w_{ij}^{xh}(t) \quad (7)$$

Here η is the *learning rate* and α the *momentum* or feedback coefficient. The machine continues to make passes of the training data till the mean squared error in the output values drops below a suitable threshold. To allow the machine to choose an appropriate direction of learning, the momentum is initialized at a very low value and is gradually increased to approach unity with the epochs.

3. ARCHITECTURE FOR PROPER NOUN PRONUNCIATIONS

The Boltzmann machine architecture used for the pronunciations consists of three principal components: a layer that maps input letters to binary-valued inputs, a layer that maps n-tuples of letters into a set of internal states using the networks context-sensitive knowledge, and a layer that mixes long-term and short-term constraints to refine the interpretations of groups of letters, as shown in Figure 2. This configuration is based on two design criteria:

1. Generally, a relatively small amount of contextual information will be sufficient to narrow the range of possible sound correspondences to a small set.
2. Choosing a correct sound from this set may require information at more remote points in the name.

In a sense, the network is designed to model n-tuples of letters using local and long-distance constraints. The input layer is 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. The output layer of the machine consists of units that translate the binary-valued outputs into corresponding phonemes. The connection weights are initialized to small random values and modified during training to model the statistical distribution in the training data.

4. TRAINING DATABASE

As is the case in most forms of speech research, progress is fueled by a comprehensive development database. No appropriate databases currently exist for training the proposed system. Numerous data has been collected anecdotally on the problem of alternate pronunciations, but none of this data has ever been incorporated into a publicly available database.

A significant portion of the effort expended in this work was devoted to the development of a small training

database. We have developed a database of 7,000 surnames that we are using in pilot experiments. We are in the process of expanding the database to include 20,000 surnames and several thousand corporate names. This database adheres to the Worldbet pronunciation standards and represents a reasonably diverse set of names with which we have developed significant experience and a great deal of confidence.

5. EXPERIMENTS AND RESULTS

We performed a number of preliminary experiments with a set of 240 names to study the training phenomena in our network. With closed-loop training on 10 names the machine output accurate pronunciations for all ten. When 100 more names were added for training the machine managed to provide plausible alternative pronunciations based on its learning. The performance improved further when trained with all 240 names. Boltzmann machine pronunciations generated for a few words are presented in Table 1.

Spell	Pron	110 wd train	240 wd train
Drew	d 9r u _	d 9r u _	d 9r u _
Teit	t _ ei t	t _ ei t	t _ ei t, g _ ei t
Vega	v E g &	v E g &, v E h &	v E g &, j E g &,v E h &
Wolf	w U l f	w U l f	w U l f

Table 1: Boltzmann machine output pronunciations

Final results on the entire database are not yet available because we have encountered problems with convergence of training on large datasets. These problems exposed the need to reformat some of the database such that it contains spellings aligned with their phonetic representations. Results on the entire training database will be presented at the conference.

6. CONCLUSION

Accurate pronunciation of proper nouns is a key issue in speech recognition applications. A major task in the training process of such a system is the design of the annealing schedule and selection of the number of units in the hidden layer. These parameters were found to significantly affect the performance of the network, and interact with various application-specific parameters such as the amount of training data, the size of the input shift register and the maximum error allowed in training. Optimization of these parameters is the subject of future research.

We also realized the need to align the spellings with the corresponding phonetic expression in our dictionary for ease of implementation. We have developed a dynamic programming algorithm that performs automatic alignment by introducing an *empty phoneme* at appropriate places and are currently updating the dictionary. We will provide a live demonstration with this modified system at the conference. All software and data will be made available at *http://www.isip.msstate.edu*.

7. ACKNOWLEDGEMENTS

We are grateful to the Systems and Information Sciences Laboratory of Texas Instruments, particularly Dr. Raja Rajasekaran, for providing the funding for this work. We also want to thank Dr. Barbara J. Wheatley for her insights into the Worldbet system, and help in the development of early versions of the training database.

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. G. Hinton and J. Anderson (Eds), *Parallel Models of Associative Memory*, Erlbaum Associates, NJ, 1981.
4. 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.
5. 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.
6. 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.
7. 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.
8. S. Kirkpatrick, C.D. Gellatt and M.P. Vecchi, “Optimization by Simulated Annealing”, in *Science*, Vol. 220, pp. 671-680, 1983.

AN ADVANCED SYSTEM TO GENERATE MULTIPLE PRONUNCIATIONS OF PROPER NOUNS

Neeraj Deshmukh, Julie Ngan, John Hamaker, Joseph Picone

Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, Mississippi 39762
{deshmukh, ngan, hamaker, picone}@isip.msstate.edu

1. ABSTRACT

Accurate recognition of proper nouns is a critical component of automatic speech recognition performance. It is a complex 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. The Boltzmann machine N-best pronunciations system is ideally suited for this task as it generates a number of the most likely pronunciations of a proper noun from its text-only spelling. The system output can be used to build better acoustic models for the noun that result in improved recognition performance.

In this paper we present an advanced version of the Boltzmann machine system with powerful architectural features such as multiple hidden layers. We also describe a Worldbet phonetic dictionary comprising of more than 18000 proper nouns (surnames) along with 24000 pronunciation transcriptions developed to train and evaluate the system. We believe that these are valuable resources for speech research and have made them available in the public domain.

2. INTRODUCTION

The quality of a voice interface in many consumer applications is determined by its ability to accurately recognize proper nouns. For instance, recognition of patients' names is a vital step in transforming their medical record access from keyboard input to voice input [1]. A similar situation exists in voice interfaces for advanced telecommunications services with corporate names and product names.

The use of extensive handwritten rule sets to generate pronunciations of proper nouns, such as that in traditional speech recognition systems, has met with only limited success. Since most proper names have a number of highly probable pronunciations which can be rarely differentiated from the context of the application, a system generating only the single-most likely pronunciation essentially attempts to solve an ill-posed problem. For a useful speech recognition application it is important that all plausible alternatives be available to the system.

A multilayered neural network in which the internal (hidden) units act as feature detectors and perform a mapping between the input word and the output pronunciation [2] is ideally suited for this purpose. It has been demonstrated in [3] that a Boltzmann machine network [5] can be successfully used to automatically derive multiple pronunciation models from the text-only spellings of a proper noun. This system transforms the spelling to a network of distinctive features that describe the articulatory movements required to produce various pronunciations of the name.

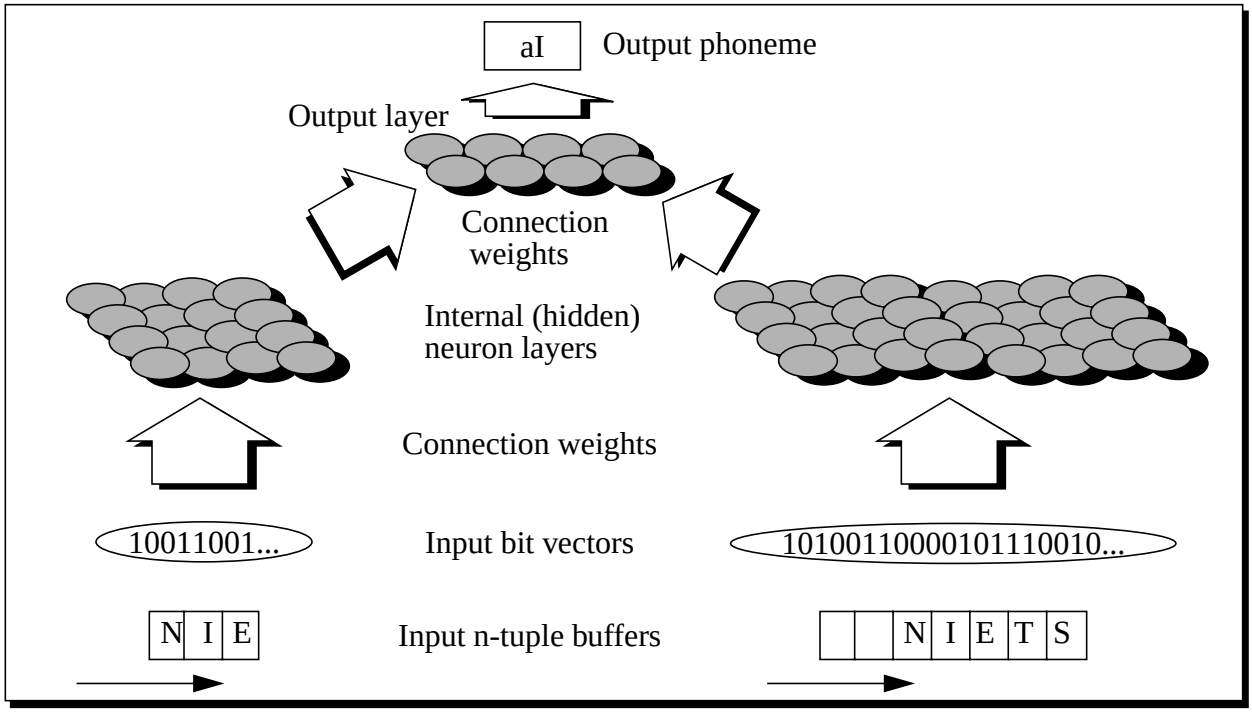


Figure 1: The Boltzmann machine: basic architecture for pronunciation generation

e.g. For a name like ‘Epstein’, the letter ‘i’ maps to the phoneme ‘aI’ due to its context.

3. THE BOLTZMANN MACHINE

The Boltzmann machine is a parallel computational architecture quite similar to a Hopfield network. 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. The architecture is designed to allow efficient searches for various hypotheses (encoded as combinations of network units with binary states) that maximally satisfy the input data and some constraints resulting from weighted interaction between the individual units.

A typical network architecture for generating multiple pronunciations 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) and an output layer that mixes long-term and short-term constraints to interpret the groups of letters into a phonetic representation [Figure Figure 1:]. The network models the n-tuples of input letters using local and/or long-distance constraints.

4. ADVANCED SYSTEM FOR N-BEST PRONUNCIATIONS

A number of architectural and algorithmic features have been added to the early version of the system [3] to create a more powerful system capable of efficiently generating an ordered list of the N most likely pronunciations of the input proper noun.

The new system is capable of supporting multiple hidden layers. The number of such internal layers and the number of neurons in each of them can be specified by the user. The long and short contexts, if used in conjunction, may have a different number of hidden layers. This provides the network with a tremendous ability to model the letter-to-sound mappings in an optimal fashion.

Context	Number of hidden layers	Number of neurons/layer	All correct generation %	Some correct generation %	No correct generation %
Short (3 letters)	1	4000	58.53	31.79	9.68
	2	layer 1 2000 layer 2 1500	64.85	26.91	8.24
Long (7 letters)	1	4000	63.13	28.11	8.76
	2	layer 1 2000 layer 2 1500	70.11	22.71	7.18
Both	1	4000	84.13	11.27	4.60
	2 for each context	layer 1 2000 layer 2 1500	86.24	10.71	3.05

Table 1: Some evaluation results for the advanced pronunciation system

The training algorithm has also been modified to update the connection weights in a more efficient fashion. The training now penalizes larger errors in the output more heavily to expedite convergence. Several other enhancements have improved the efficiency of the network by almost 25% in spite of adding computational complexity in the form of extra neuron layers. There is also an order of magnitude improvement in the system performance.

The Boltzmann machine learns the underlying constraints that characterize a problem domain by looking at examples from that domain. It accordingly modifies the interconnecting weights using a backpropagation algorithm to construct an internal model capable of producing examples with the same probability distributions. Thus a comprehensive development database is essential for training this system. We have developed a proper name dictionary that currently consists of 18494 common surnames with a total of 24040 likely pronunciations to perform pilot experiments. The pronunciations are transcribed in the Worldbet standard to incorporate data from linguistically diverse sources. We are expanding this database with several thousand corporate names.

5. EXPERIMENTAL RESULTS

The key parameters that influenced the performance were identified as the context size used to capture co-articulatory features, the number of internal layers, the number of units in these hidden layers and the training schedule. Numerous experiments were conducted to study the effect of these parameters on system accuracy. Of the 18494 names 15000 were randomly selected for training and the rest were held out for evaluation of the network. The evaluation criterion was based on the ability of the system to generate all, some or none of the likely pronunciations of the proper nouns. The ‘no correct’ case is counted as the error percent. The results were found to be an order of magnitude better than those reported on the earlier version of this system, and are summarized in Table Table 1:.

The source code as well as the dictionary will be placed at *http://www.isip.msstate.edu/* for free public access.

6. REFERENCES

1. B. Wheatley and J. Picone, “Integrating Speech Technologies For Medical Applications,”

presented at the Medical Applications of Voice Response Technology Conference, Pittsburgh, PA, December 1989.

2. G. Hinton and J. Anderson (Eds), *Parallel Models of Associative Memory*, Erlbaum Associates, NJ, 1981.
3. N. Deshmukh, M. Weber and J. Picone, "Automated Generation of N-best Pronunciations of Proper Nouns", *in Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. I-283-I-286, Atlanta, GA, May 1996.
4. T.J. Sejnowski and C.R. Rosenberg, ""NETtalk: A Parallel Network That Learns To Read Aloud," *Tech. Rep. JHU/EECS-86/01*, Johns Hopkins University, Baltimore, MD, 1986.