6.858/18.428 Machine Learning	Lecture 16: November 9, 1994
Lecturer: Ron Rivest	*Scribe: T. J. Hazen*

Outline: Applications of neural nets

- NETtalk - learning pronunciation of English text
- Classifying sonar targets

16.1 NETtalk

16.1.1 Overview

NETtalk is a classic example of a back-propagation trained multi-layer perceptron network applied to a practical application. NETtalk, created by Sejnowski and Rosenberg [1], applies a multi-layer network to the text-to-speech problem. The goal is to develop a system which can convert English text into its underlying sequence of phonemes and stress markers. The string of phonemes and stress markers can then be used by a speech synthesizer to generate an audio realization of the text as seen in Figure 16.1.

In using a network approach to the problem, it was hoped that NETtalk could *learn* a general mapping of spelling to pronunciation. Other current text-to-speech products such as DECtalk, utilize a dictionary lookup for common and irregular English words and apply a set of phonological rules to convert words which don't appear in the

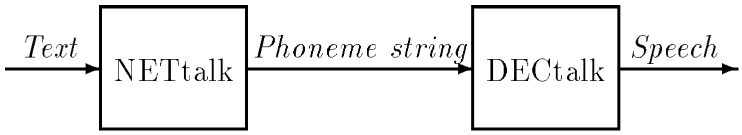

Figure 16.1: A text-to-speech system using NETtalk

dictionary. While a table lookup is the most accurate way to obtain a pronunciation from a spelling, it requires a large amount of storage space. A network, on the other hand, can represent its *knowledge* in a far more compact manner by learning *generalizations* which hopefully apply over many words.

16.1.2 NETtalk Framework

NETtalk uses a two step approach to convert spelling into phonemes. First, NETtalk converts the text, which is represented with 29 different characters (the 26 letters, a comma, a period, and a word-boundary marker), into a sequence of articulatory feature vectors. Each feature vector contains 26 different features which describe the manner in which a sound is to be articulated, such as position in mouth, voicing, nasalization, stress, etc. Second, NETtalk converts the feature vectors into phonemes which can be used by a speech synthesizer.

16.1.3 Predetermined Mappings of Letters to Phonemes

In the conversion from letters to phonemes, NETtalk assumes a one to one mapping. Much of the time, this assumption holds strictly, such as in the mapping:

$$\text{cat} \longrightarrow /\text{k æ t}/$$

Often times the number of letters exceeds the number of phonemes. An example is the word *phone* which has only 3 phonemes (/f o n/) because the letters *ph* are realized as the single phoneme /f/ and the letter *e* is silent. For such cases a *continuation* marker is used to indicate that the current letter is simply part of a continuing letter combination which maps to a single phoneme or is a silent letter which is not realized with a phoneme. Thus, if the symbol "-" is used to mark continuations, the word *phone* is mapped as:

$$\text{phone} \longrightarrow /\text{f} \; - \; \text{o n} \; - /$$

On some occasions a single letter should be mapped to two phonemes. For example the word *six* is realized phonetically as /s I k s/. For these events, the two phoneme sequence is approximated as a single phoneme which is not part of the English phoneme set. In this example, the /k s/ sequence is approximated as the velar affricate /X/. Thus the mapping for the word *six* is:

$$\text{six} \longrightarrow /\text{s I X}/$$

16.1.4 Converting Letters to Feature Vectors to Phonemes

The conversion from letters to feature vectors is performed with a multi-layer perceptron network. The network consists of three fully-connected layers: a 203 element input layer, an 80 element hidden layer and a 26 element output layer. The feature vectors are converted to phonemes with a *best match* procedure. Figure 16.2 displays NETtalk's configuration.

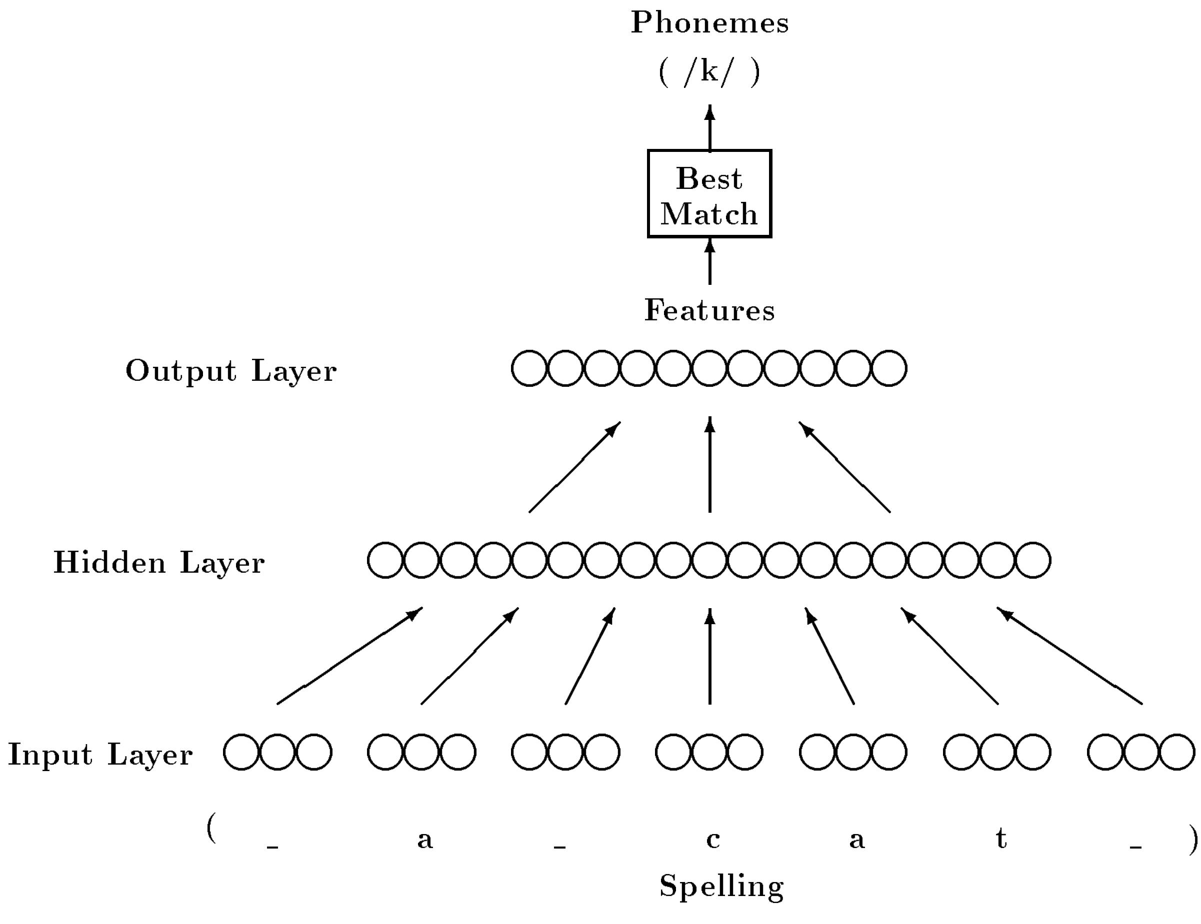

Figure 16.2: NETtalk's fully-connected multi-layer perceptron configuration

The input layer contains a binary representation of the current letter of the text as well as the 3 preceding and 3 following letters. Thus, the input text is clipped by a window 7 letters wide which slides from left to right through the text as the each letter is mapped into a phoneme. The binary representation of a letter is simply a binary vector with 29 elements. Each element in the vector represents one of the 29 possible input characters. Thus, the binary vector for a letter contains a one for the element corresponding to that letter and contains a zero for all other elements. Overall the input layer contains $7 \times 29 = 203$ elements.

The output layer consists of 26 different [0,1] valued outputs. Of the 26 outputs, 21 are used to represent the articulatory features and the other 5 are used to encode stress and syllable boundaries. Because the output feature vector may not yield a perfect match with the desired feature vectors of any of the phonemes, a best match procedure is used to choose the phoneme from the features. This procedure simply chooses the phoneme whose desired feature vector has the smallest angle difference from the network's output vector. It is worth noting that the system uses feature vectors as the output of the network rather than phonemes, which would seem the most natural output representation. Presumably, the authors either discovered empirically or rationalized theoretically that a feature vector output is more accurate and/or robust than a phoneme-based output. However, this is not discussed in their paper.

16.1.5 Training

Two different sets of data were used for training the network. The first set was comprised of entries from *Miriam Webster's Pocket Dictionary.* The second set consisted of text transcribed from informal, continuous speech from a child. During training the dictionary entries were randomized and moved through the window of the network individually. For the child's speech, the text was moved through the window in order with word boundary markers being placed between individual words. The weights in the network were adjusted by *back-propagating* the error for the output feature vector of each phoneme backwards from the output layer to the input layer using a gradient descent methodology.

16.1.6 Performance

When the system was trained on 1024 words extracted from the transcriptions of the child's speech, the system achieved an accuracy of 95% in identifying the correct phoneme on the training data after 50 training iterations through the training set. To see how well the network's results generalize to unseen data, a separate test set of 439 words which were not used during training was tested by the network. The network achieved 78% phoneme accuracy on this 439 word test set.

16.1.7 Analysis

When analyzing a system such as NETtalk, it is helpful to know the goal behind building the system. Is the goal simply to build a useful product, or is the goal to learn more about the underlying problem? The NETtalk system relies on a large set of weights which during training *learn* a mapping from letter strings to phoneme strings. Is there an underlying structure that can be discovered from these weights? Can general spelling to sound rules be discovered from these weights? To answer these questions we must understand what the hidden nodes of the network do and attempt to discover what they represent.

Sejnowski and Rosenberg examined NETtalk's hidden nodes to attempt to determine whether they served as feature detectors of any kind. A cursory examination revealed a few nodes which had obvious feature detecting properties. For example, one node was trained in such a way that it distinguished between vowels and consonants. A hierarchical clustering of the hidden nodes, based on the letter-to-sound correspondences which they were activated by, revealed that different sets of nodes were responsible for different functions. Some sets of nodes were activated for similar sounding consonants while other sets of nodes were activated for particular input vowels. However, the development of systematic methods for analyzing the hidden nodes of a neural net and discovering the underlying rules that exist is an open research topic.

16.1.8 Robustness

The issue of the the network's robustness was also examined by Sejnowski and Rosenberg. Specifically they wanted to answer the question: "Is the network resistant to damage?" They found that adding random noise as large as .25 of the actual magnitude of the weights did not significantly degrade the performance of the system. As a result of this experiment they concluded that the number of bits needed to encode each weight of the network could be relatively small (perhaps as small as 4 bits) since the round off error involved in quantizing the values of the weights would not severely harm the system's performance. Thus, storage of the values of the weights of the entire network would require only $\sim$80,000 bits, while storage of the lookup table for a 20,000 word dictionary requires nearly 2,000,000 bits. Thus, it is clear that the network is somehow capturing and encoding much of the redundancy of English pronunciation within its structure.

16.2 Classifying Sonar Targets

16.2.1 Overview

Another application for which neural networks have been used is the classification of sonar targets. Gorman and Sejnowski performed a study in which a neural net was used to distinguish between rocks and mines on the sea floor from a sonar echo [2]. Thus, this problem be viewed as a concept learning problem with two concepts: rocks and mines.

16.2.2 Sonar Signals

Sonar signals are sent out as an FM ping and returned as a reflected echo of the ping, as seen in the short-time Fourier transform domain in Figure 16.3. The short-time Fourier transform plots in Figure 16.3 use shading to indicate the amount of energy at a particular frequency for a particular point in time. When a sonar ping is transmitted as seen in Figure 16.3(a), the return echo, as seen in Figure 16.3(b), receives different amounts of reflected energy at different frequencies depending on the target off which the sonar ping is reflecting. For their experiments, Gorman and Sejnowski quantized the echo's frequency response into a set of 60 measurements where each measurement is the energy within a different aperture (or time-frequency "box") of the short-time Fourier transform of the return signal (see Figure 16.3(b)).

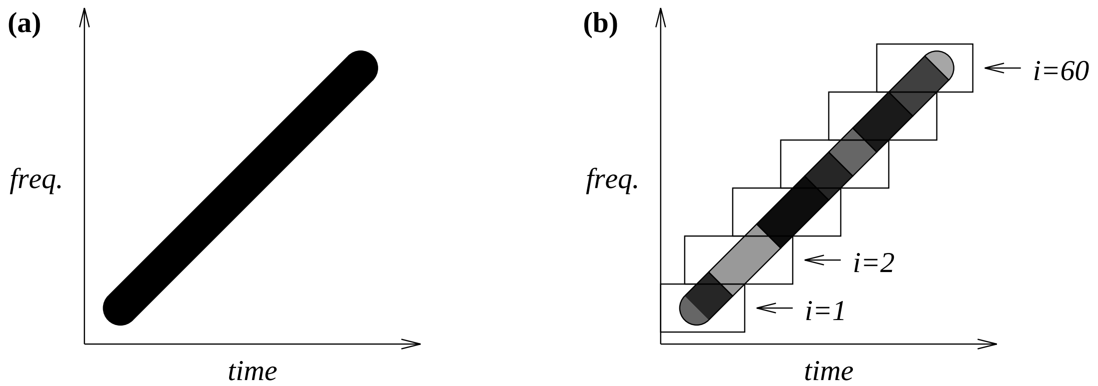

Figure 16.3: A transmitted FM sonar ping (a) and it's returned echo (b).

16.2.3 Neural Net Configuration

The configuration for the sonar target classifier is seen in Figure 16.4. The network is fully connected. It has 60 real valued input units, one output for rocks and one output for mines. The number of hidden units was varied from 0 to 24 to see what effect increasing the hidden layer size had on performance.

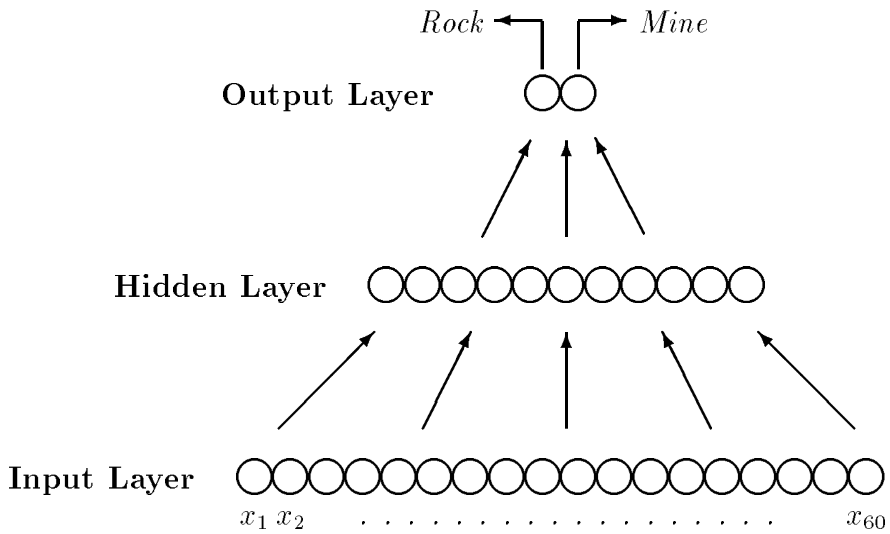

Figure 16.4: Neural network configuration for sonar target classification

16.2.4 Classification Experiments

A total of 208 example vectors were collected for experimentation. Because this is a relatively sparse amount of data, a jackknifing training/testing methodology was used. To do this 13 disjoint sets of 16 examples were randomly created. By leaving out the 16 examples from a particular set, a network could be trained on the remaining 192 samples and then tested on the 16 examples that were jackknifed out of the training set. By doing this for each of the 13 different sets, each of the 208 samples could be used once as a test element. The performance of the system on training and testing data as the number of hidden units was varied is seen in Table 16.1.

Hidden Units	% Correct on Training Data	% Correct on Test Data
0	89.0	77.1
2	96.0	81.9
3	98.8	82.0
6	99.7	83.0
12	99.8	84.7
24	99.8	84.5

Table 16.1: System performance using varying number of hidden units

16.2.5 Analysis

As can be seen in Table 16.1, the system performance improved as the number of hidden units was increased up to 12. The performance peaked with 12 units and fell slightly when the number of hidden units was increased to 24. Most likely this is because there was not enough training examples to properly train the 1488 different weights which are needed when 24 hidden units are used. As a comparison, humans who were trained to listen to the sonar echoes were able to classify the 208 examples correctly 88% of the time.

A nearest neighbor classifier achieved an accuracy of 82.7% on the task. This result is significant because, theoretically, the asymptotic performance of a nearest neighbor classifier, as the number of training examples is increased, should be no worse than 2 times as bad as the *optimal* classifier. Thus, if the nearest neighbor classifier used for these experiments was approaching its asymptotic performance then the optimal classifier should have an accuracy of no better than 91.5 to 92%. This result indicates that the network is doing an adequate job of classifying the sonar targets.

References

[1] Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Journal of Complex Systems,* 1(1):145-168, February, 1987.

[2] R. Paul Gorman and Terrence J. Sejnowski. Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Objects. *Neural Networks,* 1:75-89, 1988.