Automatic speech recognition with neural networks

Raúl Rojas[*]

Martin-Luther-Universität Halle
Fachbereich Mathematik und Informatik

Building computers capable of automatically recognizing speech has been an old dream in the fields of electronics and computer science. Some experiments were conducted as early as the 1950s and in the 1960s some systems were already capable of recognizing vowels uttered by different speakers. But until now all expectations have not been fully met. We all know of several small-scale commercial applications of speech technology for consumer electronics or for office automation. Most of these systems work with a limited vocabulary or are speaker dependent in some way. But current research has as its goal the development of *large vocabulary speaker independent continuous speech recognition*. This long chain of adjectives already underlines the difficulties which still hamper the large-scale commercial application of automatic speech recognition: We would like the user to speak without artificial pauses, we would like that the system could understand anybody, and this without necessarily knowing the context of a conversation or monologue.
Artificial neural networks have been proposed as one of the building blocks for speech recognizers. Their function is to provide a statistical model capable of associating a vector of speech features with the probability that the vector could represent any of a given number of phonemes. Neural networks play the role of statistical machines as we discuss in the second section. But we will see that our knowledge of the speech recognition process is still very limited so that fully connectionist models are normally not used. Researchers have become rather pragmatic and combine the best features of neural modeling with traditional algorithms or with other statistical approaches, like Hidden Markov Models, which we will briefly review. Current state of the art systems combine different approaches and therefore they are called *hybrid speech recognition systems*.

1 Feature extraction

The first problem for any automatic speech recognizer is finding an appropriate representation of the speech signal. Assume that the speech is sampled at constant intervals and denote the amplitude of the speech signal by $x(0), x(2), \ldots, x(n-1)$. For a good recognition the time between consecutive measurements should be kept small. The microphone signal is thus a more or less good representation of speech but contains a lot of redundancy. It would be better to reduce the number of data points but in such a way as to preserve most of the information: this is the task of all *feature extraction methods*.

[*]Extended abstract for the short course on speech recognition at HeKoNN-95

Choosing an appropriate method implies considering the speech production process and what kind of information is 'encoded' in the acoustic signal.
Speech is produced in the vocal tract, which can be thought as a tube of varying diameter extending from the vocal chords to the lips. The vocal chords produce a periodic pressure wave which travels along the vocal tract until the energy it contains is released through the mouth and nose. The vocal tract behaves as a kind of *resonator* in which some frequencies are amplified whereas others are eliminated from the final speech signal. Different configurations of the vocal organs produce different resonating frequencies, so that it is safe to assume that detecting the mixture of frequencies present in the speech signal can provide us with information about the particular configuration of the vocal tract, and from this configuration we can try to deduce what phoneme has been produced.
Many methods have been proposed to deal with the task of the spectral analysis of the speech signal. Some of them have a psychophysical foundation, that is, they are based on physiological research on human hearing. Others have arisen in other fields of engineering but have proved to be adequate for this task. Certainly one of the simplest, but also more powerful approaches, is computing a short term Fourier spectrum of the speech signal.

1.1 Fourier analysis

Given a data set $\mathbf{x} = (x(0), x(2), \ldots, x(n-1))$ it is the task of Fourier analysis to reveal its periodic structure. We can think of the data set as function X evaluated at the points $0, 2, \ldots, n-1$. The function X can be written as a linear combination of the basis functions

$$\begin{aligned} f_0(t) &= \frac{1}{\sqrt{n}}\cos(2\pi t\frac{0}{n}) - i\sin(2\pi t\frac{0}{n}) = \frac{1}{\sqrt{n}}(\omega_n^*)^{0\cdot t} \\ f_1(t) &= \frac{1}{\sqrt{n}}\cos(2\pi t\frac{1}{n}) - i\sin(2\pi t\frac{1}{n}) = \frac{1}{\sqrt{n}}(\omega_n^*)^{1\cdot t} \\ \vdots \;\; & \;\; \vdots \\ f_{n-1}(t) &= \frac{1}{\sqrt{n}}\cos(2\pi t\frac{n-1}{n}) - i\sin(2\pi t\frac{n-1}{n}) = \frac{1}{\sqrt{n}}(\omega_n^*)^{(n-1)\cdot t} \end{aligned}$$

where ω_n denotes the n-th complex root of unity $w_n = exp(-2\pi/n)$. Writing the data set as a linear combination of these functions amounts to finding which of the given frequencies is present in the data. Denote by $\mathbf{F}_n^*$ the $n \times n$ matrix whose columns are the basis functions evaluated at $t = 0, 1, \ldots, n-1$, that is, the element at row i and column j of $\mathbf{F}_n^*$ is $(\omega_n^*)^{ij}$, for $i, j = 0, \ldots, n-1$. We are looking for a vector $\mathbf{a}$ of amplitudes such that

$$\mathbf{F}_n^*\mathbf{a} = \mathbf{x}.$$

The n-dimensional vector $\mathbf{a}$ is the *spectrum* of the speech signal. The matrix $\mathbf{F}_n$ defined as

$$\mathbf{F}_n = \frac{1}{\sqrt{n}}\begin{pmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & & \ddots & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}$$

is the transpose conjugate of the matrix $\mathbf{F}_n^*$. Since the basis functions f_1, f_2, f_{n-1} are mutually orthogonal, this means that $\mathbf{F}_n^*$ is unitary and in this case

$$\mathbf{F}_n\mathbf{F}_n^*\mathbf{a} = \mathbf{F}_n\mathbf{x} \qquad \Rightarrow \qquad \mathbf{a} = \mathbf{F}_n\mathbf{x}.$$

The expression $\mathbf{F}_n\mathbf{x}$ is the discrete Fourier transform of the vector $\mathbf{x}$. The inverse Fourier transform is given of course by

$$\mathbf{F}_n^*\mathbf{F}_n\mathbf{x} = \mathbf{F}_n^*\mathbf{a} \qquad \Rightarrow \qquad \mathbf{x} = \mathbf{F}_n^*\mathbf{a}.$$

The speech signal is analyzed as follows: a window of length n is used to select the data. Such a window can cover for example 10 milliseconds of speech. The Fourier transform is computed and the magnitudes of the spectral amplitudes (the absolute values of the elements of the vector $\mathbf{a}$) are stored. The window is displaced to cover the next set of n data points and the new Fourier transform is computed. In this way we get a short-term spectrum of the speech signal as a function of time, as shown in Fig. 1. Our speech recognition algorithms should recover from this kind of information the correct sequence of phonemes.

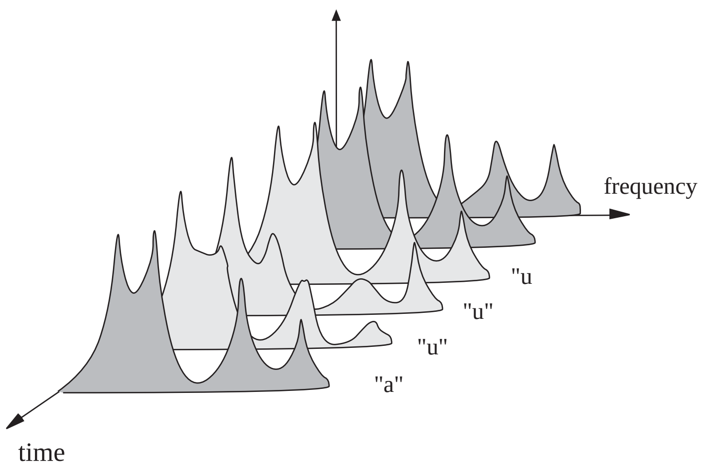

Figure 1: Temporal variation of the spectrum of the speech signal

1.2 Fast transformations

Since we are interested in analyzing the speech signal in real time it is important to reduce the number of numerical operations needed. A Fourier transform computed as a matrix-vector multiplication requires around $O(n^3)$ multiplications. A better alternative is the Fast Fourier transform, which is just a rearrangement of the matrix-vector multiplication. The left graphic in Fig. 2 shows the real part of the elements of the Fourier matrix $\mathbf{F}_n$ (the shading is proportional to the numerical value). The recursive structure of the matrix is not immediately evident, but if the even columns are permuted to the left side of the matrix and the odd columns to the right, the new matrix structure is the one shown on the right graphic in Fig. 2. Now the recursive structure is visible. The matrix $\mathbf{F}_n$ consists of four submatrices of dimension $n/2 \times n/2$, which are related to the matrix $\mathbf{F}_{n/2}$ through a simple formula. In order for the reduction process to work, n must be a power of two. This rearrangement of the Fourier matrix is the basis of the Fast Fourier Transform which we will discuss briefly in the short course.
Many speech recognition systems use some kind of variation of the Fourier coefficients. The problem with the short-term spectra is that the base frequency of the speaker should be separated from the medium term information about the shape of the vocal tract. We will discuss two popular alternatives: cepstral coefficients and linear predictive coding.

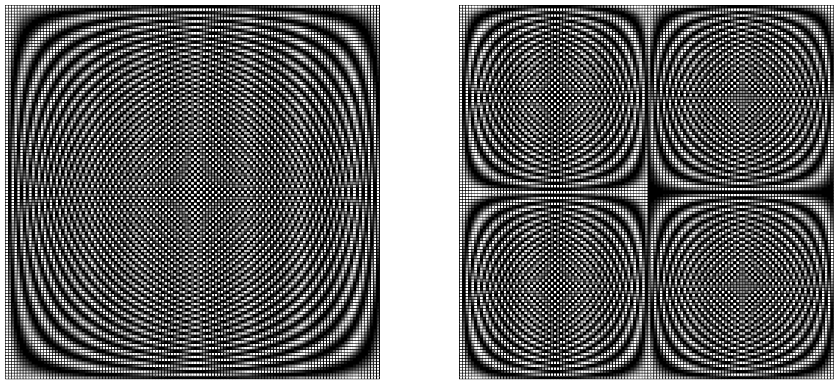

Figure 2: The Fourier matrix and the permuted Fourier matrix

2 Markov Models and Neural Networks

In speech recognition researchers postulate that the vocal tract shapes can be quantized in a discrete set of states roughly associated with the phonemes which compose speech. But when speech is recorded the exact transitions in the vocal tract cannot be observed and only the produced sound can be measured at some predefined time intervals. These are the emissions and the states of the system are the quantized configurations of the vocal tract. From the measurements we want to infer the sequence of states of the vocal tract, i.e. the sequence of utterances which gave rise to the recorded sounds. In order to make this problem manageable, the set of states and the set of possible sound parameters are quantized.
A Hidden Markov Model has the structure shown in Fig. 3. The state transitions remain invisble for the observer. The only data that is provided are the emissions (i.e. the spectrum of the signal) at some points in time.

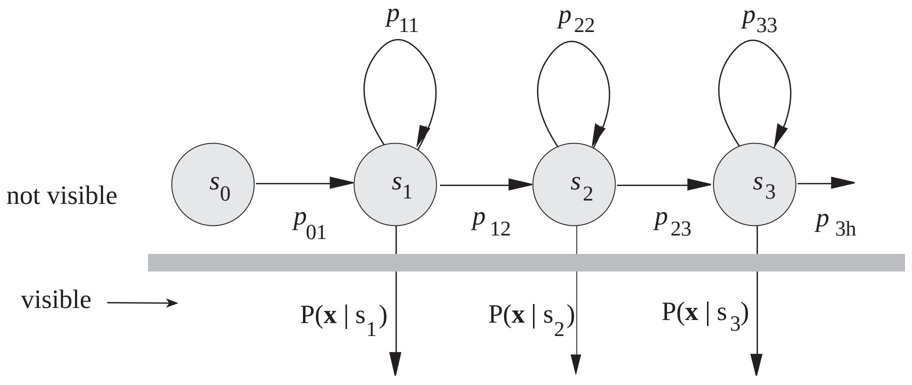

Figure 3: A hidden Markov model

The general problem when confronted with the recorded sequence of output values of a HMM is to compute the most probable sequence of state transitions which could have produced them. This is done with a recursive algorithm.

2.1 Hidden Markov Models

A first-order Markov model is any system capable of assuming one of n different states at time t. The system does not change its state at each time step deterministically but according to a stochastic dynamic. The probability of transition from the i-th to the j-th

state at each step is given by $0 \le a_{ij} \le 1$ and does not depend on the previous history of transitions. These probabilities can be arranged in an $n \times n$ matrix A. We also assume that at each step the model emits one of m possible output values. We call the probability of emitting the k-th output value while in the i-th state b_{ik}. Starting from a definite state at time $t = 0$, the system is allowed to run for T time units and the generated outputs are recorded. Each new run of the system produces in general a different sequence of output values. The system is called a HMM because only the emitted values can be observed but not the state transitions.
The state diagram of a HMM can be represented by a network made of n units (one for each state) and with connections from each unit to each other. The weights of the connections are the transition probabilities (Fig. 4).

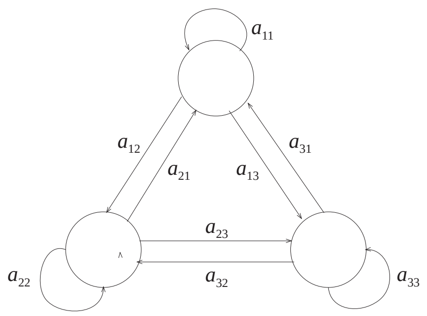

Figure 4:

As in the case of backpropagation through time, we can unfold the network in order to observe it at each time step. At $t = 0$ only one of the n units, say the i-th produces the output 1, all others zero. State i is the the actual state of the system. The probability that at time $t = 1$ the system reaches state j is given by a_{ij} (only some of these values are shown in the diagram to avoid cluttering). The probability of reaching state k at $t = 2$ is

$$\sum_{j=1}^{n} a_{ij} a_{jk}$$

which is just the net input at the k-th node in the stage $t = 2$ of the network shown in Fig. 5.
Consider now what happens when we can only observe the output of the system but not the state transitions (refer to Fig. 6). If the system starts at $t = 0$ in a state given by a discrete probability distribution $\rho_1, \rho_2, \ldots, \rho_n$, then the probability of observing the k-th output at $t = 0$ is given by

$$\sum_{i=1}^{n} \rho_i b_{ik}.$$

The probability of observing the k-th output at $t = 0$ *and* the m-th output at $t = 1$ is

$$\sum_{j=1}^{n} \sum_{i=1}^{n} \rho_i b_{ik} a_{ij} b_{jm}.$$

The rest of the stages of the network compute the corresponding probabilities in a similar manner.
How can we find the unknown transition and emission probabilities for such an HMM? If we are given a sequence of T observed outputs with indices $k_1, k_2, \ldots, k_T$ we would like to

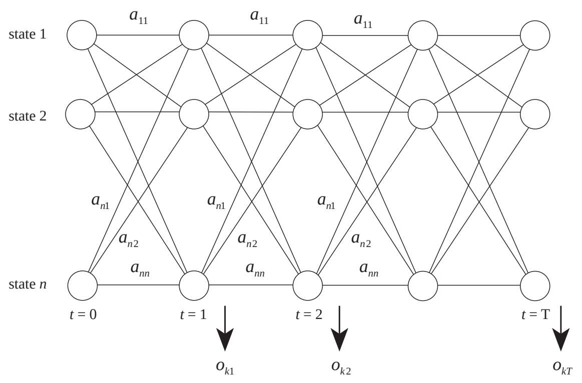

Figure 5: Unfolded Hidden Markov Model

maximize the likelihood of this sequence, i.e. the product of the probabilities that each of them occurs. This can be done by transforming the unfolded network as shown in Fig. 6 for $T = 3$. Notice that at each stage h we introduced an additional edge from the node i with the weight b_{i,k_h}. In this way the final node which collects the sum of the whole computation effectively computes the likelihood of the observed sequence. Since this unfolded network contains only differentiable functions at its nodes (in fact only addition and the identity function) it can be trained using the backpropagation algorithm. However care must be taken to avoid updating the probabilities in such a way that they could become negative or greater than 1. Also the transition probabilities starting from the same node must always add to 1. These conditions can be enforced by an additional transformation of the network (introducing for example a "softmax" function) or by using the method of Lagrange multipliers. We give only a hint of how this last technique can be implemented so that the reader completes the network by himself.

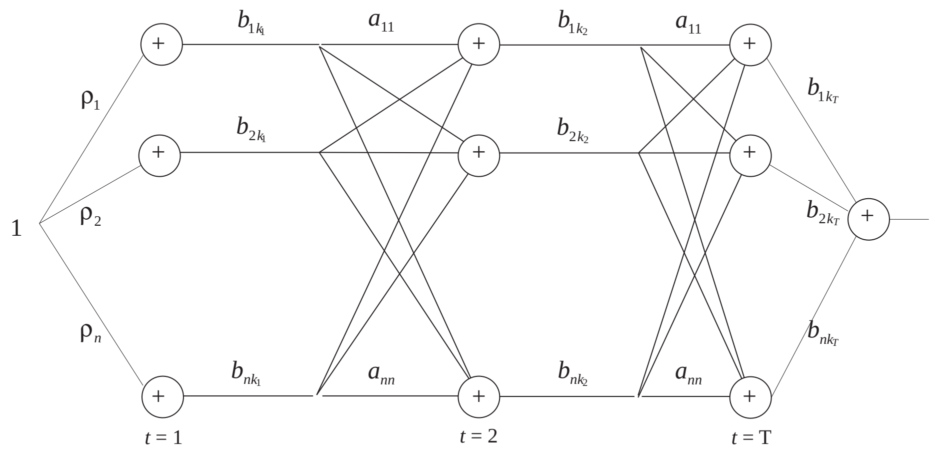

Figure 6: Computation of the likelihood of a sequence of observations

Assume that a function F of n parameters $x_1, x_2, \ldots, x_n$ must be minimized, subject to the constraint $C(x_1, x_2, \ldots, x_n) = 0$. We introduce a Lagrange multiplier λ and define the new function

$$L(x_1, \ldots, x_n, \lambda) = F(x_1, \ldots, x_n) + \lambda C(x_1, \ldots, x_n).$$

To minimize L we compute its gradient and set it to zero. To do this numerically, we follow the negative gradient direction to find the minimum. Note that since

$$\frac{\partial L}{\partial \lambda} = C(x_1, \ldots, x_n)$$

the iteration process does not finish as long as $C(x_1, \ldots, x_n) \neq 0$, because in that case the partial derivative of L with respect to λ is nonzero. If the iteration process converges, we can be sure that the constraint C is satisfied. Care must be taken when the minimum of F is reached at a saddle point of L. In this case some modifications of the basic gradient descent algorithm are needed. Fig. 7 shows a diagram of the network (a *Lagrange neural network*) adapted to include a constraint. Since all functions in the network are differentiable, the partial derivatives needed can be computed with the backpropagation algorithm.

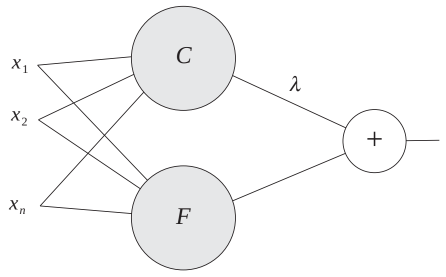

Figure 7: Lagrange neural network

2.2 Classifier networks

It is now well known that neural networks trained to classify an n-dimensional input x in one out of M classes can actually learn to compute the Bayesian a posteriori probabilities that the input x belongs to each class. Several proofs of this fact, differing only in the details, have been published [1], [3], but they can be simplified. In this section we offer a shorter proof of the probability property of classifier neural networks.

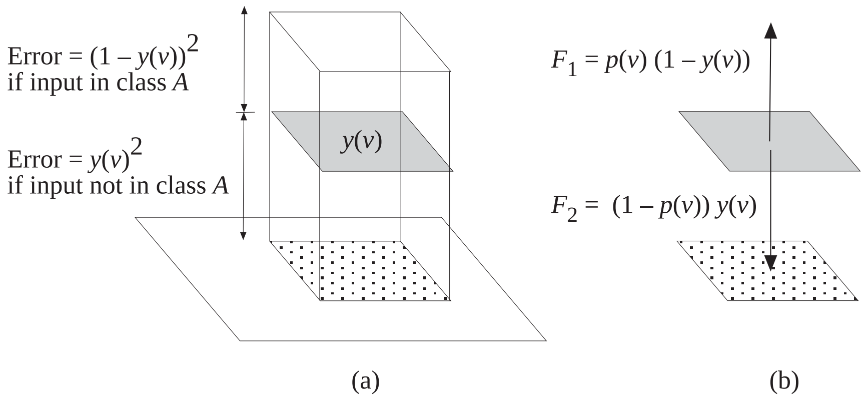

Figure 8: The output $y(v)$ in a differential volume

Part (a) of Fig. 8 shows the main idea of the proof. Points in an input space are classified as belonging to a class A or its complement. This is the first simplification: we do not have to deal with more than one class. In classifier networks, there is one output line for each class C_i, $i = 1, \ldots, M$. Each output C_i is trained to produce a 1 when the input belongs to class i, and otherwise a 0. Since the expected total error is the sum of the expected individual errors of each output, we can minimize the expected individual errors independently. This means that we need to consider only one output line and when it should produce a 1 or a 0.

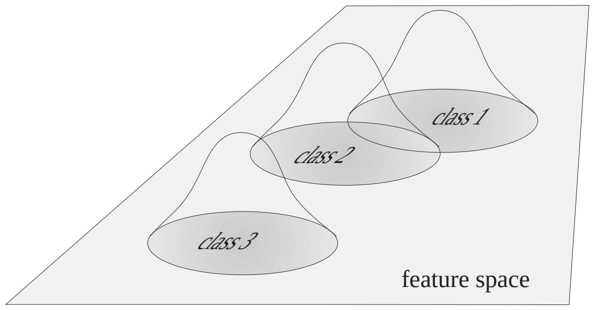

Figure 9: Probability distribution of several classes on input space

Assume that input space is divided into a lattice of differential volumes of size dv, each one centered at the n-dimensional point v. If at the output representing class A the network computes the value $y(v) \in [0, 1]$ for any point x in the differential volume $V(v)$ centered at v, and denoting by $p(v)$ the probability $p(A|x \in V(v))$, then the total expected quadratic error is

$$E_A = \sum_V \{p(v)(1 - y(v))^2 + (1 - p(v))y(v)^2\} dv,$$

where the sum runs over all differential volumes in the lattice. Assume that the values $y(v)$ can be computed independently for each differential volume. This means that we can independently minimize each of the terms of the sum. This is done by differentiating each term with respect to the output $y(v)$ and equating the result to zero

$$-2p(v)(1 - y(v)) + 2(1 - p(v))y(v) = 0.$$

From this expression we deduce $p(v) = y(v)$, that is the output $y(v)$ which minimizes the error in the differential region centered at v is the a posteriori probability $p(v)$. In this case the expected error is

$$p(v)(1 - p(v))^2 + (1 - p(v))y(v)^2 = p(v)(1 - p(v))$$

and E_A becomes the expected variance of the output line for class A.
Note that extending the above analysis to other kinds of error functions is straightforward. For example, if the error at the output is measured by $\log(1 - y(v))$ when the desired output is 1 and $\log(y(v))$ when it is 0, then the terms in the sum of expected differential errors have the form

$$p(v) \log(1 - y(v)) + (1 - p(v)) \log(y(v)).$$

Differentiating and equating to zero we again find $y(v) = p(v)$.
This short proof also strongly underlines the two conditions needed for neural networks to produce a posteriori probabilities, namely *perfect training* and *enough plasticity* of the network, so as to be able to approximate the patch of probabilities given by the lattice of differential volumes and the values $y(v)$ which we optimize independently of each other.

It is still possible to offer a simpler visual proof "without words" of the Bayesian property of classifier networks, as is done in part (b) of Fig. 8. When training to produce 1 for the class A and 0 for A^c, we subject the function produced by the network to an "upward force" proportional to the derivative of the error function, i.e. $(1 - y(v))$, and the probability $p(v)$, and a downward force proportional to $y(v)$ and the probability $(1 - p(v))$. Both forces are in equilibrium when $p(v) = y(v)$.
This result can be visualized with the help of Fig. 9. Several non-disjoint clusters represent different classes defined on an input space. The correspondence of each input vector to a class is given only probabilistically. Such an input space could consist for example of n-dimensional vectors, in which each component is the numerical value assigned to each of n possible symptoms. The classes defined over this input space are the different illnesses. A vector of symptoms corresponds to an illness with some probability. This is illustrated in Fig. 9 with the help of Gaussian shaped probability distributions. The clusters overlap, because sometimes the same symptoms can correspond to different ailments. Such an overlap could only be suppressed by acquiring more information.

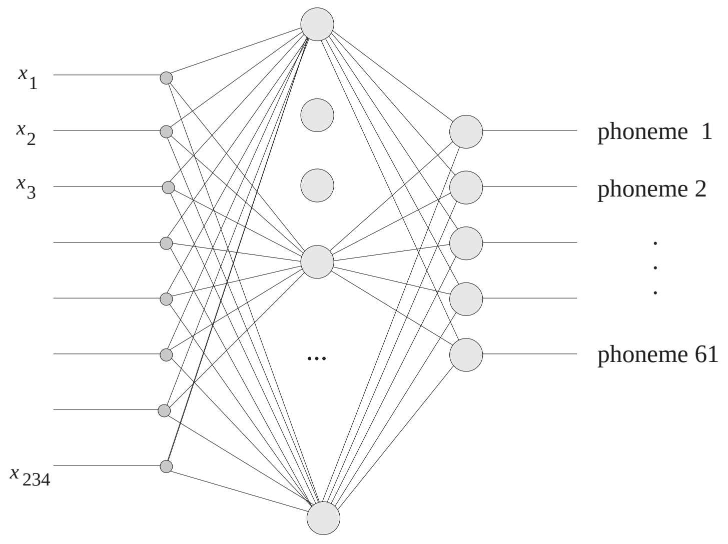

Figure 10: Network for the classification of feature vectors

This is a nice example of the kind of applications that such classification networks can have, namely in medical diagnosis. The existent data banks can be used als training set for a network and to compute the margin of error associated with the classifications. The network can compute a first diagnosis, which is then given to a physician which decides if he takes this information into account or overrides the system.

2.3 Statistical networks

Neural networks are used as classifier networks to compute the probability that any of a given set of phonemes could correspond to a given spectrum and the context of the spectrum. The speech signal is divided in windows of for example 10ms length. For each window the short term spectrum is computed and quantized, using for example 18 coefficients. We can train a network to associate spectra with the probability of each phoneme of occurring in a speech segment. A network like the one shown in Fig. 10 is used. The coefficients of the six previous and also of the six following windows are used together with the coefficients of the window we are evaluating. The dimension of the input vector is thus 234. If we consider 61 possible phonemes we end with the network of Fig. 10.

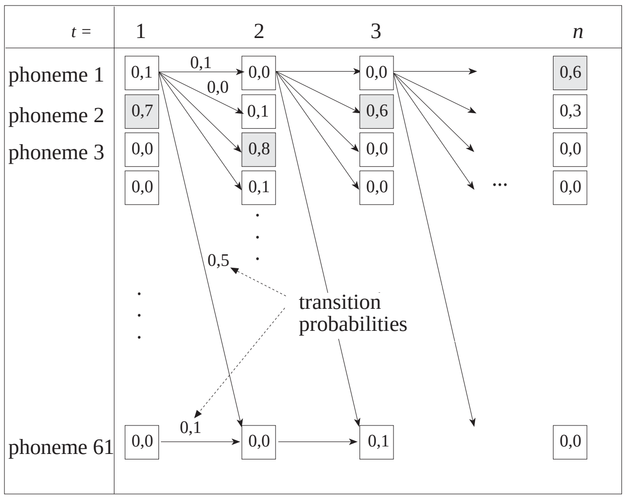

Figure 11: Computation of the most probable path

The network is trained with labeled speech data. There are several data bases which can be used for this purpose, but we will also discuss semiautomatic methods for speech labeling. Once the network has been trained it can be used to compute the emission probabilities of phonemes.

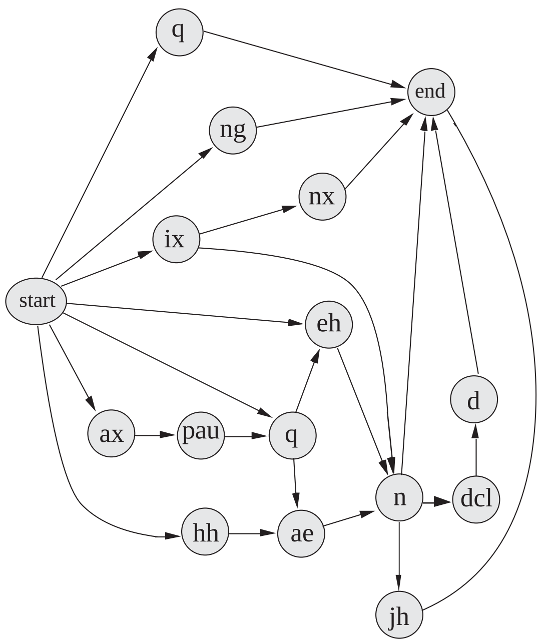

Figure 12: Markov model for the word "and"

Once a set of emission probabilities has been computed for several time frames $1, 2 \ldots, m$ it is necessary to compute the most probable path of transitions of the vocal tract and emissions. This can be done using dynamic programming methods of the same type as those generically known as *time warping*. Fig. 11 shows a matrix of phoneme probabilities for which we want to compute such a most probable path. The result could be the set of shaded nodes.

Finally, we will discuss how to integrate HMMs and neural networks in speech recognition tasks. The role of the neural network is to provide probabilities of emission. The role of the Markov model is to reduce the number of alternatives that we want to consider.

Fig. 12 shows an example of a Markov model for the word "and". The model contains so many possibilities because this is a very common word and we want to consider several possible variations. Other words which are not so common can be modelled in a more simple manner.
I will provide a more extensive list of references at HeKoNN-95.

References

[1] Bourlard, H., Morgan, N. (1993), *Connectionist Speech Recognition*, Kluwer.

[2] Rabiner, L., Bing-Hwang, J. (1993), *Fundamentals of Speech Recognition*, Prentice-Hall International, London.

[3] Richard, M. D., Lippmann, R. P. (1991), "Neural Network Classifiers Estimate *a posteriori* Probabilities", *Neural Computation*, Vol. 3, N. 4, S. 461-483.

[4] Rojas, R. (1995), *Artificial Neural Networks - Theoretical Foundations*, Springer-Verlag, New York.