NEURAL NETS FOR SPEECH PROCESSING

Neural networks (NNs) have historically been used in many speech processing applications: speech recognition, speaker recognition, language recognition, speech coding, and speech synthesis. In many cases, NNs are used as general models for supervised learning tasks (1–3) to represent a certain abstract input/output mapping, or for unsupervised learning tasks, here often to map high-dimensional data into structured lower-dimensional spaces (3,4), or for blind source separation (5). While unsupervised techniques with NNs are not yet commonly used in speech applications, using supervised techniques with NNs has got some attention especially in speech recognition, which is discussed to some extent in this article.

Currently, successful use of NNs for speech processing is mainly limited to speech recognition by machines—that is, the problem of automatically transcribing spoken words or sentences (utterances) that can be input to a microphone or similar input devices. Speech recognition is a very broad topic that includes as applications for example: (1) isolated word recognition for understanding simple control commands in a car, (2) continuous speech recognition for a dictation machine to write a letter without typing it in, or (3) spontaneous speech recognition as a part of a fully automatic translation system for regular conversations over the telephone between people speaking different languages. These three tasks seem to be of very different nature, but the approach for a practical implementation is based on a theory for treating a whole category of problems called *statistical pattern recognition* problems, which include besides speech, speaker, and language recognition also problems occurring in speech synthesis, image recognition, time series prediction, character recognition, and so on. This category of problems is often solved by statistical approaches using the principle "supervised learning from examples," which has been used successfully for speech recognition since about 1975 and has been used for many other problems of engineering interest as well.

A finite amount of examples or *training data* (for speech recognition a number of recorded waveforms reduced to a feature vector sequence $\boldsymbol{x}_1^T$ of length T frames plus their known and correct transcriptions) is used to train a *model* $\boldsymbol{M}$, which can later be used to transcribe new, previously unseen data (waveforms). The model corresponds to a given structure and a number of W parameters combined in a vector $\boldsymbol{w}$ which are estimated during training to maximize some predefined optimality criterion, which, for speech recognition, would ideally be the percentage of correct words.

Neural networks can be used as general tools or black boxes to solve statistical pattern recognition problems by learning from examples either solely or for a part of the problem, and they have been used for that purpose since around 1988 for all kinds of applications. Speech recognition for an arbitrary task is a very complex process and cannot be expected to be solved solely by neural networks. Successful use of them is currently mainly limited to one specific subproblem—the estimation of the probability $P(c_t|\boldsymbol{x}_1^T)$ of a certain phoneme c_t at time point t in the given waveform in its preprocessed form $\boldsymbol{x}_1^T$. Based on these local phoneme probability estimates, further steps can be taken to search for more useful outputs like words or sentences. These systems are often referred to as *hybrid* systems, because NNs are used in combination with other techniques like Hidden Markov Models (HMMs).

Neural networks are not the only way to approach the subproblem discussed above, and in speech recognition they have to compete with other techniques—for example, unconditional mixture density estimation with Gaussian kernels. Although most of the current state-of-the-art speech recognition systems don't use neural networks, systems based on NNs have a number of advantages, which include the following: (1) For similar recognition rates, NN systems are often faster than systems using the traditional techniques, often by a factor of 2 to 5; (2) for similar recognition rates, NN systems use, because of implicit parameter sharing, less parameters in the model, often by a factor of 5 to 10, which leads in turn to a system with lower memory requirements; and (3) NNs are, because of their in general very regular structure, believed to build easier in hardware than other model types. Most notable disadvantages of NN-based systems compared to their traditional counterparts are currently as follows: (1) NN-based state-of-the-art systems have slightly worse word recognition results, (2) training of NN-based systems with large amounts of data is complicated and time-consuming, often 10 to 20 times slower than using non-NN-based systems. It is likely that further effort in research will result in substantial improvements of the listed disadvantages.

SPEECH RECOGNITION THEORY

Speech recognition using standard statistical methods (e.g., HMMs) is well-documented in several books (6,7), and an introduction to speech recognition using neural networks is given in Ref. (8). To introduce some necessary notation within the context of using neural networks, the speech recognition problem can be written as

$$C^* = \arg\max_C P(C|X) \tag{1}$$

with $X = \boldsymbol{x}_1^T = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$ being the input vector feature sequence (frames) calculated from the observed waveform, in practical systems around 100 40-dimensional vectors/s input speech, with $C = c_1^T = \{c_1, c_2, \ldots, c_T\}$ being any valid symbol sequence and C^* being the recognized symbol sequence with the highest probability among all possible sequences. In the case of word recognition, valid symbol classes c_t are any words which are listed in a *pronunciation dictionary* that contains all words to be recognized as phoneme sequences; and in the case of phoneme recognition, valid symbol classes c_t are all possible phonemes, which are, depending on the language to recognize, usually around 50. Since the principal usage of neural networks is the same for word and phoneme recognition, discussion here is limited to the latter in order to simplify notation.

In principle, *training* a speech recognition system corresponds to estimating the probability distribution $P(C|X)$, which includes (1) defining an appropriate model $\boldsymbol{M}$ and (2) estimating its parameters $\boldsymbol{w}$ maximizing some predefined optimality criterion. In practice the model $\boldsymbol{M}$ consists of several modules, with each one being responsible for a different part of $P(C|X)$. Usage of the trained system or *recognition* for a given input sequence X corresponds principally to the evaluation of $P(C|X)$ for all possible symbol sequences to find the best one C^*. This procedure is called the *search* for which efficient algorithms are known (9,10).

Using Bayes' rule $P(B|A) = P(A|B)P(B)/P(A)$ and the product rule of probability $P(A, B) = P(A)P(B|A)$, the conditional sequence probability $P(C|X)$ can for a simple example be broken down to three terms as

$$C^* = \arg\max_C P(c_1^T|\boldsymbol{x}_1^T) \tag{2}$$

$$= \arg\max_C \{P(\boldsymbol{x}_1^T|c_1^T) \cdot P(c_1^T)\} \tag{3}$$

$$= \arg\max_C \left\{\left[\prod_{t=1}^T P(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{t-1}, c_1^T)\right] \cdot \left[\prod_{t=1}^T P(c_t|c_1, c_2, \ldots, c_{t-1})\right]\right\} \tag{4}$$

$$\approx \arg\max_C \left\{\left[\prod_{t=1}^T P(\boldsymbol{x}_t|c_t)\right] \cdot \left[\prod_{t=1}^T P(c_t|c_{t-1})\right]\right\} \tag{5}$$

$$= \arg\max_C \left[\prod_{t=1}^T \frac{P(c_t|\boldsymbol{x}_t)}{P(c_t)} \cdot P(c_t|c_{t-1})\right] \tag{6}$$

making some simplifying approximations, which for this example were as follows: (1) Every output state class c_t depends only on the previous state c_{t-1} and not on all previous state classes, making it a first-order Markov model:

$$P(c_t|c_1, c_2, \ldots, c_{t-1}) \Rightarrow P(c_t|c_{t-1}) \tag{7}$$

(2a) The feature frames are assumed to be statistically independent in time:

$$P(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{t-1}, c_1^T) \Rightarrow P(\boldsymbol{x}_t|c_1^T) \tag{8}$$

(2b) The likelihood of feature vector x_t given the complete symbol sequence c_1^T is assumed to depend only on the symbol found at t and not on any other ones, making it a *context-independent* model or *monophone,* here with only one state per HMM:

$$P(\boldsymbol{x}_t|c_1^T) \Rightarrow P(\boldsymbol{x}_t|c_t) \tag{9}$$

The remaining three probability expressions are: (1) $P(c_t|\boldsymbol{x}_t)$, the posterior probability of phoneme c_t given input vector $\boldsymbol{x}_t$ (also *observation probability*), to be estimated by a neural network; (2) $P(c_t)$, the prior probability of phoneme frame c_t, to be approximated by the relative frequency of observing phoneme frame c_t in the training data; and (3) $P(c_t|c_{t-1})$, the transition probability for the HMMs, also to be estimated by counting from the training data. Often the expression $l_t =$

$P(c_t|\boldsymbol{x}_t)/P(c_t)$ is referred to as the *scaled likelihood,* because it is proportional to the real likelihood $P(\boldsymbol{x}_t|c_t)$.

It is important to notice that although C^* is the symbol sequence with the highest probability, it is not guaranteed that it corresponds to the symbol sequence with the highest word or phoneme recognition rate (often called accuracy), which is defined by $r = (N - I - D - S)/N$ (N, number of symbols in correct sequence; I, number of insertions in recognized sequence; D, number of deletions; S, number of substitutions) and has to be found by a dynamic programming procedure (7). This mismatch is a well-accepted fact and doesn't seem to be a problem in practical systems.

Since choosing an appropriate structure for the neural network estimating $P(c_t|\boldsymbol{x}_t)$ and training it is a substantial part of using NNs for speech recognition, basics of neural networks, which are important for training and using an NN-based speech recognition system, are reviewed here.

NEURAL NETWORKS

Artificial neural networks (see Ref. 1 for an excellent introduction) can be used for many supervised learning tasks. Given as training data N input/target data vector pairs $\boldsymbol{D} = \{\boldsymbol{x}_n, \boldsymbol{t}_n\}$ (a mapping from input to target data), with dimensions M and K, respectively, the aim of a supervised learning process is to learn how to predict output data given new input data, which is written as a K-dimensional function $y^k(\boldsymbol{x}_n;\boldsymbol{w})$ depending on the current input vector $\boldsymbol{x}$ and the NN parameter value vector $\boldsymbol{w}$ with W weights. The weights are combined in *structures,* whose different types are discussed in more detail below. Inputs and targets can, in general, be continuous and/or categorical variables, thereby defining the two categories of supervised learning problems. When targets are continuous, the problem is known as a *regression problem;* when they are categorical (class labels), the problem is known as a *classification problem.* In this article, the term *prediction* is used as a general term which includes regression and classification.

Unimodal regression. For unimodal regression or *function approximation,* the components of the output vectors are continuous variables. The NN parameters are estimated to minimize some predefined error criterion—for example, maximize the likelihood of the output data $P(\boldsymbol{D},\boldsymbol{w}) = \prod_{n=1}^{N} P(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{w})$. When the distribution of the errors between the desired target and the estimated output vectors is assumed to be a single Gaussian with zero mean and a fixed global data-dependent variance, the likelihood criterion reduces to the convenient Euclidean distance measure between the desired and the estimated output vectors or the *squared-error function:*

$$E = \sum_{N}\sum_{K}(y^k(\boldsymbol{x}_n;\boldsymbol{w}) - t_n^k)^2 \tag{10}$$

which has to be minimized during training. It has been shown that neural networks can estimate the conditional average of the desired target vectors at their network outputs; that is $y^k(\boldsymbol{x}_n;\boldsymbol{w}^*) = E[t^k|\boldsymbol{x}]$, where $E[\cdot]$ is an expectation operator and $\boldsymbol{w}^*$ is the parameter (weight) vector at the minimum of the error function.

Classification. In the case of a classification problem, one seeks the most probable class out of a given pool of K classes for each input vector $\boldsymbol{x}_n$. To make this kind of problem suitable to be solved by an NN, the categorical target variables are usually coded as vectors as follows. Consider that k is the desired class label for an input vector $\boldsymbol{x}$. Then construct a K-dimensional target vector $\boldsymbol{t}$ such that its kth component is 1 and other components are 0. Any of the K components can be interpreted as the probability of $\boldsymbol{x}$ belonging to class k. The target vectors $\boldsymbol{t}_n$ constructed in this manner along with the input vectors $\boldsymbol{x}_n$ can be used to train the NN under some optimality criterion, usually the cross-entropy function,

$$E = -\sum_{N}\sum_{K} t_n^k \log(y^k(\boldsymbol{x}_n;\boldsymbol{w})) \tag{11}$$

which results from a maximum likelihood estimation assuming a multinomial output distribution (1). It has been shown that the kth network output can be interpreted as an estimate of the conditional posterior probability of class membership ($y^k(\boldsymbol{x}_n;\boldsymbol{w}^*) = P(c = k|\boldsymbol{x})$), with the quality of the estimate depending on the size of the training data and the complexity of the network.

Neural Network Training

Training of neural networks is equivalent to adjusting the weights $\boldsymbol{w}$ iteratively such that an error function is minimized, which in the case of speech recognition for the estimation of $P(c_t|\boldsymbol{x}_t)$ is usually the cross-entropy error function. Function minimization is a problem occurring in many disciplines of science, and standard procedures are well-documented (see Refs. 1 and 11 for an introduction).

Usual approaches for neural networks are (1) first-order methods, which use the first derivative of the error function $[(\partial/\partial\boldsymbol{w})E]$ to be minimized (for example gradient descent, gradient descent with momentum, RPROP, Quickprop) and (2) second-order methods, which use also the second derivative (Hessian) or approximations to it (e.g., quasi-Newton, conjugate gradient, Levenberg–Marquardt). The first (and also second) derivative of the error function in feed-forward neural networks can be calculated efficiently with a procedure called *back-propagation* (1), which requires a forward pass (calculate $y^k(\boldsymbol{x}_n;\boldsymbol{w})\ \forall\ k$) and a backward pass [calculate $(\partial/\partial\boldsymbol{w})E$ vector] through the network for each of the N training vector pairs.

All training procedures can be (1) off-line or batch methods, for which the weights $\boldsymbol{w}$ are updated after all N training samples have been used to calculate the first derivative, or (2) on-line methods, for which only a part of the training samples is used to get an estimate of the first derivative which is then used to update the weights.

The use of neural networks for speech recognition added two practical problems to training: (1) The number of parameters W (weights) is often in the range of 10,000 to 2 million, being on average much higher than in other disciplines. (2) The number of training data vectors N is often in the range of 1 million to 60 million, being also much higher than for the average NN application. These two problems rule out many of the theoretically superior and more sophisticated second-order training algorithms because of insufficient memory resources and/or a too complicated implementation. Algorithms used in practice for large-scale problems are currently mostly first-order methods—for example, (1) on-line gradient descent and (2) on-line RPROP procedures.

Gradient Descent Training. Gradient descent training refers to adjusting the weight vector $\boldsymbol{w}$ after each iteration i by a small vector $\Delta\boldsymbol{w}$ proportional to the negative gradient $-(\partial/\partial\boldsymbol{w})E^{(i)} = (\partial/\partial\boldsymbol{w})E(\boldsymbol{D}, \boldsymbol{w}^{(i)})$:

$$\Delta\boldsymbol{w}^{(i)} = -\eta\frac{\partial}{\partial\boldsymbol{w}}E^{(i)} \tag{12}$$

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} + \Delta\boldsymbol{w}^{(i)} \tag{13}$$

This procedure can be refined by making the weight change $\Delta\boldsymbol{w}$ linearly dependent on the previous change

$$\Delta\boldsymbol{w}^{(i)} = -\eta\frac{\partial}{\partial\boldsymbol{w}}E^{(i)} + \rho\cdot\Delta\boldsymbol{w}^{(i-1)} \tag{14}$$

which leads often to a considerable speed-up. Good values for η and ρ depend heavily on the used NN structure, the training data, and the initialization of $\boldsymbol{w}$ (which is often random using small values) and have to be found by experiments. If an on-line procedure is used, then the estimated gradient to be used for one update depends only on a small part of the available training data, which might lead to a large fluctuation of the gradient and therefore to slower training. In this case the local gradient estimate may be smoothed and improved by

$$\frac{\partial}{\partial\boldsymbol{w}}E^{(i)} := (1-\alpha)\cdot\frac{\partial}{\partial\boldsymbol{w}}E^{(i-1)} + \alpha\cdot\frac{\partial}{\partial\boldsymbol{w}}E^{(i)} \tag{15}$$

with $0 \le \alpha \le 1$ controlling the amount of smoothing. As an additional improvement α can be made variable, slowly increasing toward 1 during training.

RPROP Training. A procedure that has been named RPROP in Ref. 12 is a simple, heuristic first-order procedure that has been proposed in many variations by different researchers [see Ref. (1)], and that works reasonably well also for large-scale problems. The idea is to keep a stepsize δ_w for each weight individually and make the update dependent only on the sign of the wth component of the gradient $(\partial/\partial w)E^{(i)}$ as

$$\begin{array}{lll} \text{If} & \dfrac{\partial}{\partial w}E^{(i)} > 0, & \text{then}\quad w_w^{(i+1)} := w_w^{(i)} - \delta_w^{(i)} \\ \text{Else if} & \dfrac{\partial}{\partial w}E^{(i)} < 0, & \text{then}\quad w_w^{(i+1)} := w_w^{(i)} + \delta_w^{(i)} \end{array}$$

The stepsize itself is updated depending on the gradient component change as

$$\begin{array}{lll} \text{If} & \dfrac{\partial}{\partial w}E^{(i)}\cdot\dfrac{\partial}{\partial w}E^{(i-1)} > 0, & \text{then}\quad \delta_w^{(i+1)} = \delta_w^{(i)}\cdot\tau^{+} \\ \text{Else} & & \qquad\quad\ \delta_w^{(i+1)} = \delta_w^{(i)}\cdot\tau^{-} \end{array}$$

with good values being $\tau^{+} = 1.2$ and $\tau^{-} = 0.5$ for many problems. It is useful to limit δ_w to not exceed a certain range, which is not very critical and often set to $0.000001 < \delta_w < 50$. A good initial start value for δ_w is often $\delta_w = J/10$, with J being the number of input weights to a certain neuron. For speech recognition problems, RPROP is often applied on-line using gradient smoothing like shown above.

Neural Network Architectures

For speech recognition, several different neural network architectures are in use—for example, multilayer perceptrons (MLPs), recurrent neural networks (RNNs), and hierarchical mixtures of experts (HMEs), which are briefly discussed below. Common architectures like radial basis functions (RBFs) and time–delay neural networks (TDNNs), which are used for other problems, have interesting properties but have become rare in speech recognition applications.

The type of neural networks discussed here have as elements neurons connected by directed connection weights representing scalar parameters w, which are combined in a structure to provide an M- (input) to K-dimensional (output) mapping. Each neuron has one output o and many (e.g., J) inputs connected to outputs of other neurons or the input vector itself. The output o of each neuron is a function of its activation a, so $o = f_{\text{act}}(a)$, with the activation calculated as a sum of all inputs to the neuron multiplied by its corresponding weight, $a = \Sigma_J\, o_j\, w_j$. Usually there is also a bias with its own weight which acts as an additional input constantly set to 1 and in general treated like one of the J inputs. The neurons are often organized in layers as groups of neurons, with consecutive layers being usually fully connected, meaning that each neuron of a layer is connected to all neurons of the next layer. When neurons' outputs are at the same time one of the K neural network outputs, they belong to the output layer; otherwise they belong to one of the hidden layers. Activation functions for hidden layer neurons are commonly the sigmoid function $f_{\text{act}}(a) = 1/(1 + e^{-a})$ or its equivalent by a linear transformation, the tanh function $f_{\text{act}}(a) = (e^{a} - e^{-a})/(e^{a} + e^{-a})$, with the latter one often leading to slightly faster convergence using commonly used training procedures. The choice of the sigmoid activation function is motivated by its distinct property of being the discriminant function for a two-class classification problem that makes the output the posterior probability of class membership, if the input distributions are Gaussian with equal covariance matrices (1). The choice of activation functions for the output layer depends on the problem to be solved. If it is a regression problem, usually the linear activation function $f_{\text{act}}(a) = a$ is used; but if it is a classification problem the softmax function; $f_{\text{act}}(a) = e^{a}/\Sigma_J\, e^{aj}$ is used, which can be interpreted as the generalized sigmoid for the K-class classification problem.

Multilayer Perceptrons. Multilayer perceptrons (MLPs) are the most common type of architecture, in many practical applications only with two layers of weights; a hidden layer and an output layer (Fig. 1). More layers are possible but not necessary, since there are proofs that any mapping can be approximated with arbitrary accuracy with only two layers (Ref. 1 and references there in), although using more layers *can* be a more efficient realization of a certain mapping. In practice, however, more than two layers are rarely used because of little expected performance gain and practical problems during training.

For speech recognition, it is common to use not only the current input vector $\boldsymbol{x}_t$ but also information from its $2L$ neighboring vectors $\boldsymbol{x}_{t-L}, \boldsymbol{x}_{t-L+1}, \ldots, \boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \ldots, \boldsymbol{x}_{t+L}$ from a *window* as input to the MLP to relax the independence assumption equation [Eq. (8)]. Common values are $L \le 4$, but there are also systems that use up to $L = 15$. The size of

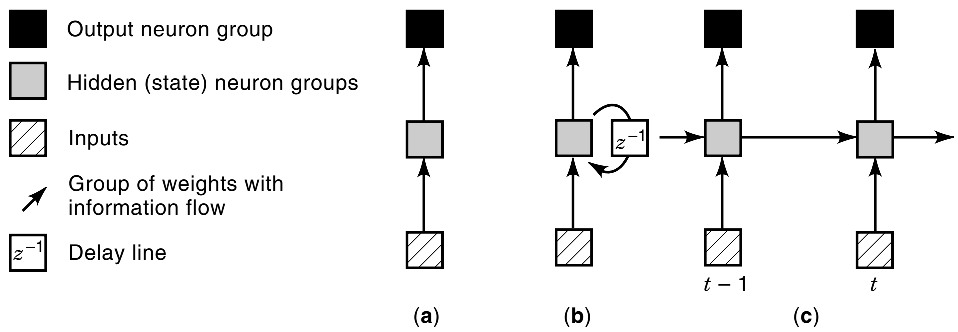

Figure 1. General structure of (a) a multilayer perceptron (MLP) and (b, c) a recurrent neural network (RNN) shown (b) with a delay line and (c) unfolded in time for two time steps, like the RNN used for speech recognition applications.

the hidden layer is in general between 64 and 4096 neurons, depending on the amount of available training data, which results in about 10,000 to 2 million weights.

Time-delay neural networks (TDNNs) (13) have the same structure as a regular MLP, but they have a reduced number of total weight parameters and have proven to be a useful improvement over regular MLPs in many applications, where the amount of training data is low compared to the number of parameters to estimate. This is achieved by a user-defined hard-tying of parameters, meaning forcing certain parameters to have the same values. Which parameters are useful to tie depends heavily on the used data and can only be found by experiments.

Recurrent Neural Networks. For many applications the data $\boldsymbol{D}$ are not a collection of vector pairs in arbitrary order, but the data come in sequences of vector pairs, where the order is not arbitrary. Speech recognition is a typical example for this case: Every preprocessed waveform is an array of vectors $\boldsymbol{x}_1^T$ that is to be mapped to an array of target classes c_1^T in the form of K-dimensional vectors $\boldsymbol{t}_1^T$.

One type of recurrent neural networks (RNNs) provides an elegant way of dealing with this kind of problem. Figure 1 shows a basic RNN architecture with a delay line and unfolded in time for two time steps. In this structure, the input vectors $\boldsymbol{x}_t$ are fed one at a time into the RNN. Instead of using a fixed number of input vectors from a window as done for the MLP and TDNN structures, this architecture can make use of all the available input information up to the current time frame t_c (i.e., $\{\boldsymbol{x}_t, t = 1, 2, \ldots, t_c\}$) to predict $\boldsymbol{y}_{t_c}$. Future input information coming up later than t_c is usually also useful for prediction. With an RNN, this can be partially achieved by delaying the output by a certain number of S time frames to include future information up to $\boldsymbol{x}_{t_c+S}$ to predict $\boldsymbol{y}_{t_c}$. Theoretically, S could be made very large to capture all the available future information, but in practice it is found that prediction results drop if S is too large. For speech recognition, S is commonly set to around 3 to 6 frames, corresponding to a delay of about 30 to 60 ms. One possibility to get around this user-defined delay is to use bidirectional recurrent neural networks (BRNNs) (14).

Because of the recurrent connections of RNNs, the training of RNNS is slightly more complicated than for feed-forward neural networks such as MLPs. An often-used training procedure is back-propagation through time (BPTT). For BPTT, first the RNN structure is unfolded up to the length of the training sequence like shown for two time steps in Fig. 1, which transforms the RNN in a large feed-forward neural network. Now regular back-propagation can be applied; but at the beginning and the end of the training data sequence, some special treatment is necessary. The state inputs at $t = 1$ are not known, and in practice they can be set to an arbitrary, but fixed, value. Also, the local state derivatives at $t = T$ are not known and can be set to zero, assuming that input information beyond that point is not important for the current update, which for the boundaries is certainly the case.

The RNNs used for speech recognition (9,15) have, in general, less parameters than their MLP counterparts for obtaining the same performance. It is common to have between 64 and 1024 hidden units, leading to about 10,000 to 1 million weights.

Hierarchical Mixtures of Experts. Hierarchical mixtures of experts (HMEs) (16) provide an elegant way of dividing large problems into many smaller ones and have been applied successfully to speech recognition problems since 1994. An extensive introduction to HMEs is beyond the scope of this article, but a short discussion with respect to their use for speech recognition is given here.

HMEs consist of a number of expert and gating networks, which are combined in a tree structure with expert networks at the leafs and gating networks at the nonterminal nodes. The overall output at the root node is a weighted average of the expert network outputs, with the weighting factors determined by the gating networks which are directly connected to the input. The structure is called *hierarchical* when there is more than one layer of gating networks. Gating networks always have a softmax output function, which allows their outputs to be interpreted as posterior probabilities conditioned on an input vector $\boldsymbol{x}$. The output activation function of the expert networks depends on the type of problem to be solved: In the case of regression they should be linear, whereas in the case of classification they are networks with a softmax output function. In general, gating and expert networks can be any of the structures introduced so far—for example, simple one-layer networks or MLPs, but also RBFs, RNNs, and TDNNs.

For a part of the training of HMEs the Expectation-Maximization (EM) algorithm (1,16) is used, which consists of two steps, the E-step (expectation) and the M-step (maximization). In the case of HMEs the E-step corresponds to calculating intermediate target vectors for each individual gate and expert for the complete training data set $\boldsymbol{D}$. The M-step corresponds to solving a number of subproblems for each individual gate and expert using the targets from the E-step. These subproblems are equivalent to regression or classification problems of regular structures like MLPs or RNNs, and they can be solved with any of the procedures known for these (e.g., any variation of gradient descent). After a weight update, new intermediate targets with a new E-step can be calculated.

For large databases like those used for speech recognition (100 h of recorded training data correspond to approximately 36 million training vectors), this procedure is used in its on-line version with an update after around 50 to 200 vectors. Practical experiences with HMEs for large databases are reported, for example, in Ref. 17.

CONTEXT DEPENDENT MODELS

Using the assumption equation [Eq. (9)] made the models context-independent one-state models, which is valid for simple tasks and to introduce basic concepts. State-of-the-art speech recognition systems usually make less severe assumptions by introducing context-dependent models (depending on a context class ϕ) and also more than one HMM state per model denoted by s. How to determine the optimal set of context classes and number of states per model for a given task is a current research issue and is beyond the scope of this article. Detailed procedures can be found, for example, in Ref. 9 and in references there in. The scaled likelihood with time t dropped in notation then becomes $l = P(c, \phi, s|\boldsymbol{x})/P(c, \phi, s)$ instead of $l = P(c|\boldsymbol{x})/P(c)$. This representation is not useful for use in an NN-based system, since the number of different output classes for all combinations of phonemes, context classes, and states is generally large (5000 to 30,000) and would lead to an NN with a huge output layer that couldn't be trained in practice. It is possible to decompose the scaled likelihood, for example, as

$$l = \frac{P(c, \phi, s|\boldsymbol{x})}{P(c, \phi, s)} \tag{16}$$

$$= \frac{P(\phi, s|c, \boldsymbol{x})}{P(\phi, s|c)} \cdot \frac{P(c|\boldsymbol{x})}{P(c)} \tag{17}$$

$$= \frac{P(s|\phi, c, \boldsymbol{x})}{P(s|\phi, c)} \cdot \frac{P(\phi|c, \boldsymbol{x})}{P(\phi|c)} \cdot \frac{P(c|\boldsymbol{x})}{P(c)} \tag{18}$$

which results in several terms that can be estimated independently. The last term $P(c|\boldsymbol{x})/P(c)$ is the regular monophone scaled likelihood. The denominator of the middle term $P(\phi|c)$ and the first term $P(s|\phi, c)$ can be estimated by the relative frequencies of the events in the training data. The numerators $P(s|\phi, c, \boldsymbol{x})$ and $P(\phi|c, \boldsymbol{x})$ represent like $P(c|\boldsymbol{x})$ classification problems conditioned on a continuous input $\boldsymbol{x}$, but they depend also on the discrete inputs c and ϕ, which could be treated as additional input vector components that could, for example, be set to one and zero depending on their discrete input state. For estimation of each of these terms there are two possibilities: (1) with one NN that takes also discrete inputs as part of an enlarged input vector allowing parameter sharing between different context-dependent models or (2) with many smaller NNs for each discrete possibility occurring on the right-hand side of the terms [for example, K networks for the estimation of $P(\phi|c, \boldsymbol{x})$ if there are K monophone classes c], which allows greater control over the encapsulated context-dependent models and faster execution. Currently common is the latter approach, which is, for example, discussed in Refs. 18, 19, and 20.

SYSTEM TRAINING

In the discussion up to now, it has been assumed that frame-labeled training data are available, meaning for each input vector $\boldsymbol{x}$ there is a known target class c, which is usually not the case. Instead, there is often only a transcription of the utterance, which might include word boundary or phoneme boundary information but not complete state alignments. Complete state alignments have to be built in incremental steps. Training all acoustic parameters of a complete system (NN weights, transition probabilities, and prior weights) involves a number of iterative steps, which can be summarized as

1. Assign a target class c to each frame of the training data, which is done by aligning the known word transcriptions to the waveforms using the acoustic models from the previous iteration. In the beginning there are no acoustic models available, and the initial state alignment has to be done by hand (or by using another existing speech recognizer) for at least a few sentences in order to bootstrap the system.
2. Calculate the state priors $P(c^{(i)}) = N(i)/N_{all}$ and the transition probabilities $P(c^{(i)}|c^{(j)}) = N(j,i)/N(j)$, with $N(\cdot)$ denoting the occurences of the corresponding frames (or frame pairs) in the training data.
3. Train the NN using the assigned target classes.
4. Goto 1, until there is no significant change in the alignments anymore. In general it is found that around four iterations are sufficient.

This procedure is called *Viterbi training,* because a distinct target class is assigned to each frame. It is also possible to perform a more general but also more memory consuming *Forward–Backward training,* where each frame gets assigned to all target classes with a certain probability.

ACOUSTIC ADAPTATION

Acoustic adaptation refers to improving the acoustic models with new data after they have been trained. Adaptation can be either (a) supervised, where the correct transcriptions, but not the exact alignments (targets $\boldsymbol{t}_1^T$), of the data $\boldsymbol{x}_1^T$ used for adaptation are known, or (b) unsupervised, where they are unknown. Supervised adaptation is, for example, used for a dictation system, that was originally trained for many speakers, but is now to be adapted for one specific speaker who is going to use the system. This is usually done by reading a text (which the dictation system provides) that is automatically aligned while the text is read. Unsupervised adaptation is used to improve the models based on acoustic evidence (inputs $\boldsymbol{x}_1^T$) alone, and it has to rely on a recognized alignment given the complete dictionary, which can, and usually will, include errors. It can be useful to assign a confidence score between 0 and 1 to every frame of the recognized alignment to express the degree of belief in the correctness of it. This confidence score can then be used to improve an unsupervised adaptation procedure.

For NN-based speech recognition systems a common framework for adaptation is to use a transformation for the

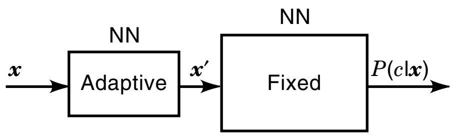

Figure 2. Example setup for acoustic adaptation in neural-network-based speech recognition systems.

feature vectors like shown in Fig. 2 instead of adapting all parameters of the original model (21). After training, the parameters of network "Fixed" are not changed anymore; only the parameters in network "Adaptive" are changed. Often "Adaptive" is a simple linear network corresponding to a linear transformation ($\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x}$), although principally any NN structure can be used. Unsupervised adaptation for one or several utterances $\boldsymbol{x}_1^T$ is done as follows:

1. Initialize "Adaptive" to produce an identity mapping ($\boldsymbol{x}' = \boldsymbol{x}$).
2. Run $\boldsymbol{x}_1^T$ through "Adaptive" and "Fixed" to calculate output probabilities $P(c_t|\boldsymbol{x}_t)\ \forall\ t, K$ and calculate the scaled likelihoods $\boldsymbol{l}_1^T$.
3. Use the local scaled likelihoods $\boldsymbol{l}_1^T$ to search for the best phoneme or word sequence alignment given the complete dictionary, which gives a distinct target class for every frame, $\boldsymbol{t}_1^T$.
4. Backpropagate the error between targets $\boldsymbol{t}_1^T$ and outputs through "Fixed" and "Adaptive" and update weights only in "Adaptive" (using, for example, gradient descent or RPROP). Weight the error of each frame by its confidence score if it was assigned.
5. Go to step 2 until alignment doesn't change anymore.

For supervised adaptation, step 3 is changed to an alignment of the already given phoneme or word sequence.

CONCLUSIONS

Neural networks can be used as general tools to solve statistical pattern recognition problems based on a firm mathematical theory, and they have been used successfully for speech processing since around 1988. In some speech recognition systems, neural networks have been used to replace the calculation of observation likelihoods by Gaussian mixture models, which has led to compact systems with fewer parameters than their more complex traditional counterparts. Although likely to improve in the near future, the major drawback of using NNs for large tasks (more than 100,000 weights and more than 1 million training vector pairs) is the relatively complicated training procedure and the necessary training time, which is currently between days and weeks and doesn't allow extensive experiments which would be necessary to make significant progress in the field.

Since neural networks are used in many other disciplines as well, their use in speech processing can benefit from research in completely different areas. Current issues in neural network research include, among others: (1) useful combination of knowledge represented as a data set of examples and other prior knowledge about the problem to solve, (2) use of Bayesian methods to model the underlying generator of the data more accurately than with maximum likelihood methods (Ref. 1 and references therein), (3) adaptation of model parameters on-line based on very few training data examples, (4) provision of a useful framework to measure and compare complexity of completely different models, and (5) use of unsupervised methods to separate, filter, and organize data based on the statistical properties of the data itself (4,5). All of these research areas, although not yet mainstream, are likely to enlarge the usage of neural networks in speech processing applications.

BIBLIOGRAPHY

1. C. M. Bishop, *Neural Networks for Pattern Recognition,* Oxford: Clarendon, 1995.
2. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* New York: Wiley, 1973.
3. T. Kohonen, *Self-Organizing Maps,* 2nd ed., Berlin: Springer-Verlag, 1997.
4. C. M. Bishop, M. Svensén, and C. K. I. Williams, GTM: The generative topographic mapping, *Neural Comput.,* **10**: 215–234, 1998.
5. A. J. Bell and T. J. Sejnowski, An information-maximization approach to blind separation and blind deconvolution, *Neural Comput.,* **7**: 1129–1159, 1995.
6. X. D. Huang, Y. Ariki, and M. A. Jack, *Hidden Markov Models for Speech Recognition,* Edinburgh: Edinburgh Univ. Press, 1990.
7. L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
8. H. Bourlard and N. Morgan, *Connectionist Speech Recognition,* Boston: Kluwer Academic Press, 1994.
9. C. H. Lee, F. K. Soong, and K. K. Paliwal, *Automatic Speech Recognition: Advanced Topics,* Boston: Kluwer, 1996.
10. S. Renals and M. Hochberg, Decoder technology for connectionist large vocabulary speech recognition, Tech. Rep. CUED/F-INGENG/TR186, Cambridge University Engineering Department, Cambridge, England, 1995.
11. W. H. Press et al., *Numerical Recipes in C,* 2nd ed., Cambridge: Cambridge Univ. Press, 1992.
12. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, *Proc. IEEE Int. Conf. Neural Netw.,* 1993, pp. 586–591.
13. A. Waibel et al., Phoneme recognition using time-delay neural networks, *IEEE Trans. Acoust., Speech, Signal Process.,* **37**: 328–339, 1989.
14. M. Schuster and K. K. Paliwal, Bidirectional recurrent neural networks, *IEEE Trans. Signal Process.,* **45**: 2673–2681, 1997.
15. A. J. Robinson, An application of recurrent neural nets to phone probability estimation, *IEEE Trans. Neural Netw.,* **5**: 298–305, 1994.
16. M. I. Jordan and R. A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, *Neural Comput.,* **6**: 181–214, 1994.
17. J. Fritsch, Context dependent hybrid HME/HMM speech recognition using polyphone clustering decision trees, *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.,* **3**: 1997, pp. 1759–1762.
18. H. Franco et al., Context-dependent connectionist probability estimation in a hybrid Hidden Markov Model—Speech Recognition, *Comput. Speech Language,* **8**: 211—222, 1994.
19. D. J. Kershaw, M. M. Hochberg, and A. J. Robinson, Context-dependent classes in a hybrid recurrent network-HMM speech recognition system, Tech. Rep. CUED/F-INGENG/TR217, Cambridge University Engineering Department, Cambridge, England.

20. J. Fritsch and Michael Finke, Acid/HNN: Clustering hierarchies of neural networks for context-dependent connectionist acoustic modeling, *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.,* **7**: 505–508, 1998.
21. J. Neto et al., Speaker-adaptation for hybrid HMM-ANN continuous speech recognition systems, *Proc. Eur. Conf. Speech Commun. Technol.,* Madrid, Spain, 1995.

MIKE SCHUSTER
ATR Interpreting Telecommunications Research Laboratories