A Connectionist Approach to Speech Recognition

Yoshua Bengio

AT&T Bell Laboratories,

HO-4G312, Crawfords Corner Rd, Holmdel, NJ 07733.

Abstract

The task discussed in this paper is that of learning to map input sequences to output sequences. In particular, problems of phoneme recognition in continuous speech are considered, but most of the discussed techniques could be applied to other tasks, such as the recognition of sequences of handwritten characters. The systems considered in this paper are based on connectionist models, or artificial neural networks, sometimes combined with statistical techniques for recognition of sequences of patterns, stressing the integration of prior knowledge and learning. Different architectures for sequence and speech recognition are reviewed, including recurrent networks as well as hybrid systems involving hidden Markov models.

Paper to appear in the special issue of IJPRAI on Neural Nets.

Keywords: neural networks, speech recognition, sequence recognition, recurrent networks, hidden Markov models, hybrid systems.

1 Introduction

Integrating prior knowledge and learning in flexible ways is one of the attractive features [35,1] of connectionist models (or artificial neural networks). This paper presents some examples of this flexibility in applications to speech recognition. Because of the sequential nature of speech, systems that map input sequences to output sequences are studied in this paper. We consider in particular recurrent networks and hybrids combining neural networks and systems based on dynamic programming (DP), such as hidden Markov models (HMM). The connectionist models considered here are mostly multi-layer networks and recurrent networks, usually trained by stochastic gradient descent with a back-propagation algorithm [44].

In Section 2, several aspects of the integration of prior knowledge with learning are discussed with examples. We classify a priori constraints into three categories[1]: constraints on the architecture, constraints that guide training, and the use of prior knowledge to initialize free parameters. In Section 3, we describe different basic architectures for neural networks (NN) applied to speech or sequence recognition. We present experimental results and arguments indicating that recurrent networks can often outperform static networks but that training them optimally to capture long-term dependencies remains an important challenge. This motivates the introduction of the systems described in Section 4 that combine connectionist models and statistical models, in particular hidden Markov models.

2 Integrating Prior Knowledge and Learning

Learning can be seen as combining two different types of information: prior knowledge (or *hints* [35,1]) and training data. It generally amounts to searching in a set F of functions $f : X \rightarrow Y$ from an input set X to an output set Y. In that case the *training data* would be a set of (x, y) pairs with $x \in X$ and $y \in Y$. In practice, for parameterized functions we search in a parameter space, e.g., R^n where n is the number of free parameters. Here, prior knowledge is meant as any information about the problem that can be used to impose constraints on the learning system or help it arrive at a "better" solution. These correspond to restrictions in the search space, i.e., prior knowledge can be used to reduce the size of F. *Generalization* is defined

[1] originally suggested by Patrick Haffner (personal communication).

as the expected performance on new instances $(x, y) \in X \times Y$ (which may be different from those in the training set but that are assumed sampled from the same underlying distribution). Performance on a given input/output sample is usually defined by a local criterion $d(y, f(x))$, e.g., for the mean square (MS) criterion, $d = \frac{1}{2}(y - f(x))^2$. Prior knowledge and constraints decrease the size of F and thus decrease the number of alternative solutions that *satisfy*[2] a training set of a given size, which may improve generalization. However, since the "prior knowledge" may be partially incorrect, it may also introduce some undesirable bias. The constraints can be classified in the following three categories (which can be combined in a particular system): architectural constraints, constraints that guide training, and constraints used for initialization.

Architectural constraints can be introduced in several ways. The network architecture itself can be constrained by reducing connectivity, using share weights, or more generally using a weight space transformation [28]. The choice of input preprocessing and coding, as well as output representation and coding are also important. Finally, the system can be decomposed into modules by decomposing the problem into subproblems (e.g., see [51,6,41]), sometimes creating intermediate variables. Prior knowledge can be used to decide on a representation for these intermediate variables.

Of the constraints that guide training, the first one is in the choice of cost function and possible *penalty* terms[3]. A particularly interesting type of constraint that guides training is one which imposes some *invariance* on the function computed by the network. For example, [47] shows how to train a back-propagation network such that the classification it performs is constrained to be locally invariant[4] to input image transformations such as translation, rotation, scaling and thinning.

[2] i.e., for which an error criterion is satisfied.

[3] such as weight decay [44].

[4] along the hyperplane tangent to the invariance manifold.

Finally, the initialization of the parameters of the learning system can also use prior knowledge and constraints. The NN criterion function is the function of the tunable parameters that the learning algorithm explicitly attempts to minimize or maximize (e.g. the Mean Square Error criterion). Since a NN criterion function usually has several local minima and a limited time is given to train the NN, initial values of the parameters can influence their final value. A module of the system may be constructed in a non-connectionist way, for example a hidden Markov model [42], and its parameters (if any) may be initialized in a different way from the network weights. A more general way to use prior knowledge to initialize system parameters is to set the initial values of the parameters such that the network (or parts of it) perform desirable transformations which are known to be useful for the task at hand. In particular, several researchers have designed systems in which weights are initialized so that the initial activity of the network corresponds to the successive application of rules [23,49,37], which may come from an expert system. This allows for the combination of prior knowledge in the form of rules with training from examples.

2.1 Input Representation and Coding

Neural networks are like other classifiers in that selecting appropriate preprocessing schemes and input representation is very important. Considering, for example, automatic speech recognition, our work (see [6,5]), as well as the results of others [43], indicate that the choice of signal processing significantly influences the performance of a recognition system. Thus, in experiments described in [17], we found an auditory model to perform better as a preprocessor for a feedforward neural network than a Mel-scaled filter bank based on the discrete Fourier transform. Other examples of the introduction of input features and preprocessing that influenced performance improvements in speech recognition tasks are detailed in [8,6,21,43]. For example, in consonant recognition experiments [8], we found the second-order derivative of spectral energy with respect to time and frequency to be useful for characterizing formant transitions. In addition to the choice of input features, one has to decide on how to code these features with the activations of the input neurons of the network. This choice may influence generalization. For example, in [6] we found that *smoothing* the input "image" improved generalization. This is consistent with results from [25] on how smoothing the input reduces the "effective capacity" of the system. In our experiments, the inputs represented formants with a two-dimensional grid of energy and frequency, where an individual cell represented the presence of a formant at a given energy and frequency. This input "image" was smoothed by distributing some of the activation of each cell to its neighbors.

For some tasks, one may be able to decompose the problem into multiple parallel problems, for example, discriminating among different sets of phonemes. It may be suspected a priori that some particular input features or preprocessing are relevant to some of these subtasks and not to the others. In that case, it would probably be advantageous to use *specialized preprocessing for specialized networks.* We adopt such an approach in the experiments described in [3] on E-set classification as well as those described in [8] on plosive recognition in continuous speech. For example, in [3] we use different temporal and frequency resolution for different classification tasks (see also modular architectures, section 2.3).

2.2 Output Coding

Most NNs designed for phoneme recognition have a simple output coding scheme consisting of one output unit per phoneme, with a high desired output for the output unit corresponding to the target phoneme and a low desired output for the other units (see for example [50]). In these cases, one can interpret the output activations of the network as representing degrees of evidence. Using phonetic knowledge, we have explored coding schemes based on phonetic features related to speech production. Examples of such features are horizontal and vertical place of articulation, voicing and nasality. Such a representation is in general more compact than the “one-output-per-phoneme” representation. Furthermore, it describes a more general space of phonetic characteristics, allowing a network trained with some phonemes to generalize to new phonemes [17] that were not part of the training set[5].

Another way to use prior knowledge for the design of output representation is to use ‘hint’ output units. For example, in [17], in addition to the units representing each vowel, supervision was given to extra output units representing phonetic features. These additional constraints on the network guided it towards computing these phonetic features, and yielded an improvement in generalization. In [52], adding extra output units in order to improve generalization is also suggested. In [8,21], context-dependent output units are used to impose additional constraints, asking more of the network than necessary but in this way helping its performance.

Experiments on output coding schemes are described in [6,8]. It was found that a “one-output-per-phoneme” coding for plosives and nasals was worse than using horizontal place, vertical place and voicing of the current phoneme. Furthermore, using context-dependent output units improved performance even more. As described in [21], for the recognition of the 10 classes of plosives and nasals /p,t,k,b,d,g,dx,m,n,ng/ in the TIMIT database, generalization error was 36.4% with one output node per phoneme, but only 28.7% by coding with 7 nodes representing the following phonetic features: labial, alveolar, velar, flap, voiced, stop, nasal. By adding context-dependent output nodes, generalization error was further reduced to 27%. For each of 4 horizontal places of articulation of plosives, 3 right contexts were considered (front vowel, back vowel, non-vowel). In addition to these 12 nodes for horizontal place of articulation, 2 nodes indicated voiced or unvoiced phoneme, and 6 nodes were used for broad classification of phonemes (liquid, front vocalic, non-front vocalic, nasal, fricative, silence). The above experiments were performed with a 32-filter Bark-scale spectral input. By adding other features such as second-order derivatives of energy [8,21] the error was further reduced

[5] However, such as representation may not be always appropriate, e.g., see experiments in section 3.5.

to 24.9%.

2.3 Architecture

In designing a network's architecture, we attempt to satisfy two conflicting goals. First, we want the network to be able to approximate the desired function as well as possible. Second, we want to reduce the size of the hypothesis space F, i.e., we want to reduce the number of free parameters. One way to reduce the number of parameters is to reduce the connectivity of the network. Another is to use shared weights. In networks for sequence recognition, weights can be shared both in feature space and in time, i.e., weights that are associated to different (e.g., translated) input features may be shared, and weights that are associated to different (e.g., translated) times may be shared. See for example [28,29] and Section 3 of this paper.

A way to reduce connectivity is to break the network into subnetworks or modules, where inter-module connectivity is low but intra-module connectivity is high. A practical question concerning modular systems is how to integrate all the modules together in the "best" possible way. An answer to this question comes from a *common* definition of what is *best*: a global error criterion or cost function, i.e., a common goal of learning for all the modules. If such a global cost function is defined, then in order to allow the parameters of the modular system to be optimal for this cost function, one can perform a *global optimization*[6] of all these parameters. It is easy to imagine cases in which each module in a modular system is trained separately, which is sub-optimal for the given complete architecture, even though each module may have reached a local optimum of its local cost function (see [6] for such a demonstration). In many instances, we would be able to perform a separate training using prior knowledge of what each module should do. In that case, we propose doing an initial separate a priori training, followed by a global tuning with respect to a single global criterion function. A particular way

[6] The use of this term here should not be confused with "global search" techniques, including random search and simulated annealing.

to take advantage of modularization is to use specialized preprocessing for different modules acting in parallel. See section 2.1 above on examples of this approach. In some cases, we don't know at all how to modularize the network. In [27], an algorithm is proposed that achieves self-organization of the modules by automatically performing a corresponding partition in the input space.

3 Connectionist Architectures for Speech Recognition

Various basic structures for speech recognition are described in this section and illustrated in Figure 1. The architectures described here can generally be constructed by appropriately connecting "time-delay neurons" performing the following operation at each time instant t:

$$y_{i,t} = f(\sum_{l \in I_i} w_l \, y_{p_l, t-d_l}) \tag{1}$$

where f is a squashing function (e.g., $tanh$), $y_{i,t}$ is the output of neuron i at time t, I_i is the set of incoming links for neuron i, p_l is the predecessor neuron for link l, and d_l is the discrete delay associated to link l.

3.1 Static Isolated Word Recognizer

This architecture is illustrated in Figure 1.1. As in [17], the input sequence is scaled to have a constant length L. The first hidden layer is fully connected to the $L \times D$ inputs (where D is the number of input features per time frame[7]). Usually, there is one output unit per word. There are several problems with this approach. First, it works only for isolated words and a small vocabulary. Second, it is very sensitive to variations in the input such as change of *alignment*[8] and non-uniform *time warping*[9]. It thus requires many examples of how each word

[7]Here, a frame denotes one discrete time step worth of data.

[8]i.e., translation of the input in time.

[9]i.e., locally scaling the temporal dimension by different amounts at different times.

can be distorted or shifted. Finally, such an architecture generally implies a large number of free parameters. In comparison to the architectures described in the next subsections, this unstructured architecture is seen to require many more free parameters because it doesn't take into account inherent *regularities* in the data.

3.2 Static Fixed Window

For this architecture, illustrated in Figure 1.2, the network "sees" only a few frames at a time and produces an output at every frame (see for example NETtalk [46]). It can be used in continuous speech recognition but requires a higher level post-processor that interprets its output sequence to generate a phoneme sequence or a word sequence. It is insensitive to time alignment and less sensitive to time warping than the static isolated word recognizer. It also requires much fewer free parameters because the same network is shared over many parts of the input. One way to represent this architecture is with multiple connections with different delays between every input node and every node on the first hidden layer. In this case the input represents a single frame. The drawback of this architecture is that it does only part of the work: the dependencies among separate parts of the input sequence that are not "seen" by the network in a single input window remain to be modeled. This task can be performed for example by a hidden Markov model (see Section 4).

3.3 Time-Delay Neural Network (TDNN)

This is a generalization of the static fixed window network, in which there are multiple time delays between successive layers[10]. On can also view a TDNN [50] as an ordinary isolated word (or phoneme) recognizer constrained by shared weights and local connectivity. This is a popular model used by several groups [50,51,12,26,6] and extended to recognition of images

[10]Hence the higher level layers capture some of the temporal structure in the previous layers.

in [36]. The last layer may be used to perform an integration of the outputs of the previous layer over time. In Figure 1.3, this is represented by a recurrent connection, although in general, TDNNs are represented by unfolding them in time (i.e., the actual output is a sum of contributions for all time frames of the sequence). This architecture still has some of the problems of the static fixed size window network: the delays are fixed a priori and a post-processor is required to perform continuous speech recognition.

A related architecture is based on the gamma memory [18], in which the hidden units perform a computation equivalent to a temporal convolution of their input with a kernel that is a gamma density function. This convolution is computed incrementally. Different forms of short-term memory are reviewed in [39].

3.4 Segment-Based Classification

In this approach, one assumes a preprocessor that proposes segments or segment boundaries. An unstructured network similar to the static isolated word recognizer may be applied to each segment. Each input segment is scaled to constant length. The simpler approach assumes a single segmentation hypothesis (a sequence of adjacent segments), as in [17]. It is much faster than a frame-based method (as those described above) but is unfortunately very *sensitive to errors of the segmentation* preprocessor. A more sophisticated approach considers multiple segmentation hypotheses, with each segment constructed from a pair of boundaries satisfying some constraints, as in [31,16, 45]. The network produces as output a classification score for each hypothetical segment. A post-processor interprets those scores[11]. In [31], segment probabilities are estimated and incorporated to phonetic evidence computed by the network. In [16,45] (for off-line and on-line handwriting recognition, respectively), the K-best paths (among sequences of legal interpretations) are used to provide both positive and negative

[11] e.g., a dynamic programming post-processor looks for the allowed sequence of segments with the best total score, usually obtained by the product or sum of local scores for each segment.

training examples to the network. Finally, another approach (in this issue 53) uses a HMM to propose several segmentation hypotheses. A NN can then be used for rescoring each of these hypotheses. See also Section 4 on hybrids.

3.5 Recurrent Nets

As illustrated in Figure 1.4, a recurrent network has *cycles* in its graph that allow it to store information about past inputs for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data. The type of recurrent networks considered here can be used either for sequence recognition, production or prediction: units are not clamped and we are not interested in convergence to a fixed point. Instead the recurrent network is used to transform an input sequence (e.g., speech spectra) into an output sequence (e.g., degrees of evidence for phonemes). The main advantage of such recurrent networks is that the relevant (past) context can be represented in the activity of the hidden units and then used to produce to compute the output at each time step. In theory, the network can *learn* how to extract the relevant context information from the input sequence. In contrast, in network with time delays such as TDNNs, the designer of the network must decide a priori (by the choice of delay connections) which part of the past input sequence should be used to predict the next output. According to the terminology introduced in [39], the memory is static in the case of TDNNs but it is adaptive in the case of recurrent networks.

We will now discuss speech recognition experiments on recurrent NNs trained using stochastic gradient descent and back-propagation through time [44] or the BPS algorithm [24,6], in both cases computing the full gradient (derivative of the criterion function with respect to the parameters). We compare several architectures, including architectures with no recurrence (a TDNN or a network with static fixed size window), to evaluate the improvements brought by recurrence in the network.

In [21,8], experiments on the recognition of plosives and nasals in the TIMIT database are reported: better generalization was obtained on that task with a recurrent network than with a static network. In [6], we compare recurrent and static networks with experiments on the speaker-independent recognition of vowels in the TIMIT database. The task is the discrimination of 11 vowel classes in every context. The training set was obtained by extracting vowels (based on the database segmentation) from 1024 training SI and SX sentences from the TIMIT database. The 192 SI and SX sentences from the core test set were used for evaluating generalization. At each time frame, the input to the neural network is a 32-filter Bark-scale spectrogram (see [54]). The results are summarized in Table 1. Figure 2 shows the architecture that yielded the best results (see also [6] for the other architectures). An important aspect of this architecture is that it combines recurrence with delays. Another interesting point is that it is not a fully recurrent network. Note the improvement in using recurrence: from more than 61% frame error with static networks to around 51% error with recurrent networks (for a comparable architecture). One should also notice that the use of articulatory features in the vowel targets does not seem to be as successful in the case of vowels as we found it to be with consonants [8], when compared to the simpler “one-output-per-class” scheme. This may be explained as follows. For vowels, the mapping from articulatory position to speech acoustics (or formant frequencies) is known to be many-to-one [2,9]. This makes the inversion problem particularly difficult. It could imply that the choice of articulatory position as an intermediate representation for a vowel recognizer is not appropriate.

Another static versus recurrent comparison was done, for the BPS algorithm described in [24,6]. This algorithm can be used only for architectures in which recurrence is limited to self-loops on the units of the first hidden layer. It has the advantage that the gradient is computed on-line (no back-propagation through time is required) and thus one can perform stochastic gradient descent with update after each frame. In these experiments as well the recurrent network yielded better generalization than the static network.

3.5.1 Problems with Training of Recurrent Networks

Although recurrent networks can in many instances outperform static networks, they appear more difficult to train optimally. Our experiments tended to indicate that their parameters settle in a sub-optimal solution which takes into account short-term dependencies but not long-term dependencies. For example, in experiments described in [7], we found that simple duration constraints on phonemes had not at all been captured by the recurrent network. Hence its performance was much worse than that of a NN / dynamic programming hybrid in which these constraints were incorporated. The criterion function (a function of the free parameters) appears to have many *local minima, plateaus* and *abrupt changes.*

Experiments with a recurrent network on a minimal task involving one bit of state information showed [11] that learning becomes more and more difficult for longer term dependencies. We demonstrate [11] several theoretical results on parametric dynamical systems (such as recurrent networks) where bits of state information are stored with hyperbolic attractors. For such systems, it is shown that perturbation resistance is incompatible with efficient learning by gradient descent. The demonstration [11] relies on the fact that if the 2-norm of the Jacobian of the system dynamics around attractors is greater than 1, then the system may be very sensitive to noise. On the other hand, if it is less than 1, then the gradients with respect to a future error vanish exponentially fast. This situation is illustrated in Figure 3. This result is for real-time recognizers based on a parametric dynamical system. A real-time recognizer takes a decision on its output at each time step t given only its current input and current state. Its state is updated using its current input and is thus derived from past inputs.

Although this is a negative result, a better understanding of this problem could help in designing alternative systems for learning to map input sequences to output sequences with long-term dependencies, e.g., for learning finite-state machines, grammars and other language related tasks. Since gradient-based methods appear inadequate for this kind of problem, we want to consider alternative optimization methods that give acceptable results even when the criterion function is not smooth. Global search methods such as simulated annealing can be applied to such problems, but they are generally very slow. One way to help in the training of recurrent networks is to set their connectivity and initial weights using prior knowledge. For example, this is accomplished in [23] using prior rules and sequentiality constraints. Another way to introduce prior knowledge about temporal structure is with hidden Markov models (HMM), which have been used for speech recognition for many years, and that can be combined with NNs. Another advantage of HMM or DP (dynamic programming) recognizers is that they can take context into account and yet do not suffer from the problem described above. This is because the HMM and DP recognizers are not real-time recognizers. Their algorithm can wait for the *end of the sequence* and then go *backward in time*. Hence a decision on the output at time t can be taken based on both past and future inputs. Note that the fact that these non real-time algorithms are not local in time could be undesirable for some applications.

4 Hybrids of Connectionist Models and Statistical Models

Given the unsatisfying performance of recurrent networks, a reasonable alternative is the combination of connectionist models with non-connectionist tools. In particular, methods based on DP (dynamic programming), e.g., HMMs (hidden Markov models), have been shown to be very useful in speech recognition [34,30]. An advantage of HMMs is that they allow one to incorporate (1) prior knowledge about the *sequential structure* of the data with (2) learning from examples. Hence, several researchers have explored such hybrids, the majority of which are constructed by sending the output of a NN to a DP or HMM post-processor [14,4,32, 22,38,6, 26,20,48, 16,45]. Several others propose a NN architecture that can emulate a HMM [15,40] and/or integrate DP in a connectionist unit [26,33]. Alternatively, [53] uses the NN to rescore the N-best hypotheses produced with a HMM. In [6,26,20,48,16] the outputs of the

NN are not interpreted as probabilities, but rather are used as scores and generally combined with DP. On the other hand, [14,22,38, 45] interpret the outputs of the NN as posterior probabilities[12] and use the Viterbi algorithm [42].

The systems proposed in [4,6,7,32] combine a NN with a HMM. In particular, in [4], the outputs of the NN are quantized, hence no global optimization of the hybrid can be performed. In the system introduced in [6,7] (and discussed at the end of this subsection), a global optimization of all the parameters of the hybrid is performed by back-propagating the derivative of the training criterion from the HMM to the NN. In [32,48], a network per class or per state is trained to predict the next input frame given only a few previous frames. In [48] the difference between predicted and actual input is used to compute a local distance whereas in [32] it is used to compute a state conditional observation likelihood (that can be used in the HMM). In the latter system, the Viterbi algorithm is used to obtain a segmentation and train the network. Hence the system is trained to approximately maximize the likelihood of the observations (given the predictive network model). It is an approximation because the Viterbi algorithm is used instead of the Baum-Welsh algorithm [42].

A different kind of hybrid consists in constructing a HMM with connectionist elements. For example [15] shows how the Baum-Welsh forward pass [34] in a HMM can be realized by combining sums, products, and recurrences. It is also shown how the backward pass is equivalent to computing error gradients with the back-propagation algorithm when applied to this architecture. A similar model is proposed in [40], which includes "log" and "divide" elements to perform normalization (required to prevent numerical underflow). In the more recent work of [33], DP is incorporated in the operation of each unit, which allows the authors to propose interesting generalizations of HMMs which contain multiple layers of such units.

Following the methodology introduced in [12,6,13], we will now consider modular systems built

[12] hence the outputs of the NN are normalized to sum to 1

by putting together different modules, e.g., NN (neural network), GM (Gaussian mixture) and HMM (hidden Markov model). As argued in section 2.3, a global optimization of the whole system with a common criterion is preferable. We can associate with a module or group of modules a training criterion such as ML (maximum likelihood), MAP (maximum a posteriori probability), MMI (maximum mutual information), or MS (mean square error). A NN produces as output a parameterized transformation of its inputs, and its parameters are the weights and biases of the network. A GM takes as inputs some observations and computes one or more Gaussian mixture pdfs (probability density functions). Its parameters are means, covariances and mixture weights. For the purpose of this paper, what we call an HMM module takes as inputs numbers associated at each frame to each state (or transition)[13] and produces as output a single scalar[14]. Its parameters are transition probabilities and initial state probabilities. The output of the HMM module may be computed with the forward pass of the Baum-Welsh algorithm [42].

In [7], we present a system built as the concatenation NN + GM + HMM. Experiments were performed on the recognition of plosives with the TIMIT database. First, the NN (which is recurrent, here) is trained to recognize phonetic features. Second, the parameters of the NN are fixed and the parameters of the GM + HMM (Gaussian mixture + hidden Markov model) subsystem are estimated for the ML criterion with the associated efficient EM algorithms [34]. Finally, a global tuning can be performed with the MMI criterion or the ML criterion and stochastic gradient descent on the whole system. The MMI criterion is a discriminant criterion and it can be defined as the maximization of the logarithm of the ratio of two likelihoods. The first one is the likelihood of the recognition model constrained by the correct interpretation. The second one is the likelihood of the recognition model. Optimization with the ML criterion is simpler (maximizing the log likelihood of the constrained model) but can

[13]normally, these numbers are interpreted as likelihood conditioned on a state or transition.

[14]normally interpreted as the likelihood of the observations given the model.

lead to a global maximum of the criterion for which the NN computes useless quantities (e.g., constant outputs, since the likelihood of a Gaussian model can then be arbitrarily large). However, if the system modules are initialized separately, global optimization can bring the system to a local maximum of the ML criterion which corresponds to improved performance. In fact, in our experiments [7] using the ML criterion, the error rate improved from 19% to 14% after global optimization, and was much better than that of the NN alone (47%) or the GM+HMM alone (24%) (i.e. with the input observations directly modeled by the GM+HMM system).

Even though it is the correct way to perform maximum likelihood estimation on the hybrid, using the ML criterion on the *whole* NN + GM + HMM system is more complicated. Indeed, it is not sufficient to maximize the output of the HMM module[14]: in order for the HMM module output to represent the *likelihood of the system inputs*, we have to modify the NN + GM subsystem so that it computes the likelihood of the *NN inputs* (rather than the likelihood of the *NN outputs*). Otherwise the learning algorithm would maximize the likelihood of the NN outputs, which could be achieved in a trivial way when the NN outputs are constant[15]. In order to modify the NN + GM system such that it computes a pdf, we have devised a new approach to density estimation with NNs.

4.1 A New Approach to Density Estimation with Neural Networks

In order to characterize the likelihood of the NN inputs, one must establish the link between the pdf of the network input, X, and the pdf of the transformed data, Y (which could for example be modeled by a GM). The pdf of the random variable $X \in R^n$ is written $f_X(x)$. The random variable $Y \in R^m$, $m \leq n$, is related to X by a class C_1 deterministic function $Y = y(X)$ (the network transformation). We assume that the law induced on Y by the

[15] they can then be easily modeled by a very peaky Gaussian, yielding a very high likelihood.

function y can be defined by a continuous density $f_Y(y)$. The Jacobian matrix of a function f at a point x_0 is referred to as $\frac{\partial(f)}{\partial(x)}(x_0)$.

Consider the case in which $m = n$ and $y : x \mapsto y(x)$ is a diffeomorphism, *i.e.* a bijection with both y and y^{-1} class C_1 functions. The following classical result can be shown [10]:

$$f_X(x_0) \;=\; f_Y(y(x_0)) \left| \text{Determinant} \, \frac{\partial(y)}{\partial(x)}(x_0) \right| \tag{2}$$

Hence the output of the NN + GM system should be scaled by the determinant of the Jacobian of the NN. In [10], we generalize this result to the case $m < n$, in which the network produces a compressed representation. In [10], we consider how to use these results to perform density estimation with a hybrid system consisting of a neural network followed by a pdf such as a Gaussian or a Gaussian mixture. The main drawback of this system is the expensive computation of the gradient (the partial derivatives of f_X with respect to the network weights), which requires performing a singular value decomposition of the Jacobian $\frac{\partial(y)}{\partial(x)}$ and computing the second-order derivatives $\frac{\partial^2 f_X}{\partial x_i \partial w_j}$ where x_i is the i^{th} input and w_j is the j^{th} network weight.

4.2 A Hybrid of Neural Network and Discriminant Hidden Markov Model

Let us now consider a discriminant hybrid built as the concatenation NN + HMM, in which we do not require the inputs of the HMM to represent likelihoods[16]. Instead we will write each of these inputs $f(t, O_\tau)$ (e.g., the output of a NN), a score associated to transition t of the HMM[17] and to the observation O_τ at time τ. Hence the outputs of the NN module are the inputs of the HMM module. Note that unlike the system proposed in [14,38], the outputs of the NN are *not* interpreted as probabilities and hence are not required to sum to 1. Instead,

[16] This model is similar to one proposed in [12] and was developed in discussions with L. Bottou and J. Denker (see also [19]).

[17] or similarly, we could write $f(t, O_\tau) = t.\text{pr}\; g(t.\text{dest}, O_\tau)$ where $t.\text{pr}$ is the transition probability and $g(t.\text{dest}, O_\tau)$ is the output of the NN associated to the destination state $t.\text{dest}$.

the *normalization* is performed at the "model" level, by defining $P(M_k \mid O)$ as in equation 4. We consider a recognition HMM, M, which allows a set of possible interpretations. The recognizer has to select one of these interpretations (a legal string of labels). The local scores $f(t, O_\tau)$ are combined to produce a score $R(O, M_k)$ for the model M_k [18]:

$$R(O, M_k) = \sum_{p \in P_k} \prod_{\substack{t \in p \\ \tau = 1, 2, \ldots L_p}} f(t, O_\tau) \tag{3}$$

where P_k is the set of paths[19] in M consistent with interpretation k. L_p is the length[20] of path p. $R(O, M_k)$ can be computed efficiently by dynamic programming as in equation 9, with $R(O, M_k) = \alpha_{\text{final state, final time}}$ and a restriction to transitions and states of M_k. A discriminant criterion for this system is the MAP, where we define the posterior probability of an interpretation (to be maximized, for the correct interpretation k) as follows:

$$P(M_k \mid O) = \frac{R(O, M_k)}{\sum_i R(O, M_i)} = \frac{R(O, M_k)}{R(O, M)} \tag{4}$$

Hence, during training we maximize $R(O, M_k)$ (for the correct model M_k) while minimizing $R(O, M)$ (for the unconstrained model M). The recognized interpretation should thus be selected as follows:

$$\text{recognized interpretation} = \underset{k}{\text{argmax}}\, P(M_k \mid O) \tag{5}$$

and since the denominator in equation 4 is the same for all k, we can select

$$\text{recognized interpretation} = \underset{k}{\text{argmax}} \sum_{p \in P_k} \prod_{\substack{t \in p \\ \tau = 1, 2, \ldots L_p}} f(t, O_\tau) \tag{6}$$

The HMM in this system is very similar to the one in the NN + GM + HMM with the MMI criterion discussed earlier [7]. The main difference is that in the MMI system the local scores

[18] M_k is a restriction of M which represents the k^{th} possible interpretation in the recognition HMM M.

[19] A path is a sequence of transitions from state to state.

[20] If there is no input segmentation, all paths should have the same length.

are decomposed as $f(t, O_\tau) = t.\mathrm{pr}\ P(O_\tau \mid t.\mathrm{dest})$ where $t.\mathrm{pr}$ and $t.\mathrm{dest}$ are the transition probability and destination state of transition t, and $P(O_\tau \mid t.\mathrm{dest})$ is the likelihood of the observation at time τ conditioned on the destination state, and is usually obtained from a GM. In that case, the quantity $R(O, M_k)$ can thus be interpreted as the likelihood $P(O \mid M_k)$. Finally, if the different interpretations are equiprobable, $P(M_k \mid O)$ is defined as in equation 4, otherwise $P(M_k \mid O) = \frac{R(O,M_k)P(M_k)}{\sum_i R(O,M_i)P(M_i)}$. Hence the computation of $R(O, M_k)$ is formally the same in both cases, and the training criterion is the same when interpretations priors are equal.

Unfortunately, the required computation for performing recognition is very expensive in general, because of the possibly huge number of allowed interpretations (e.g., words, sequences of words) to consider in equation 6. In practice, for the MMI systems, speech recognizers are built by taking approximations of equation 6. The most widespread such approximation is the Viterbi [42] algorithm[21]. It computes $s = \operatorname{argmax}_{p \in M} \prod_{\substack{t \in p \\ \tau=1,2,\ldots L_p}} f(t, O_\tau)$ with an efficient recursive formula similar to that in equation 8. Since each path p is uniquely associated to an interpretation k, this interpretation is the one the algorithm produces as the recognized interpretation. Hence the Viterbi algorithm approximates equation 6 by considering only the maximum term in the sum of equation 6. A better approximation is obtained when computing the N-best paths, or (even better) the N-best interpretations.

However, there is a special (but very common) case of HMM topology where an efficient algorithm exists to compute the correct recognized interpretation and its associated score (equations 6 and 3). In order to compute these quantities, we introduce some intermediate variables $\alpha_{i,\tau}$ associated to state i at time τ.

$$\alpha_{i,\tau} = \max_{k \text{ consistent with } i,\tau} \sum_{p \in P_k} \prod_{\substack{t \in p \\ \tau=1,2,\ldots L_p}} f(t, O_\tau) \tag{7}$$

[21]For example, we used it in [7].

We also associate with each (state,time) pair the best interpretation k that is consistent with it (argmax instead of max in the equation above). The algorithm we propose here is valid when the following conditions hold: first, states and transitions are organized in groups associated to the same label; second, each group has an input and an output bottleneck[22]. To compute $\alpha_{i,\tau}$ efficiently, we consider two cases: if i is an entering state for a group, then all its predecessors are associated to a different group, hence

$$\alpha_{i,\tau} = \max_{t:\, t.\text{dest}=i} \alpha_{t.\text{source},\tau-1} \;\; f(t, O_\tau) \quad ; \tag{8}$$

otherwise all predecessor transitions of i are associated to the same interpretation, hence

$$\alpha_{i,\tau} = \sum_{t:\, t.\text{dest}=i} \alpha_{t.\text{source},\tau-1} \;\; f(t, O_\tau) \; . \tag{9}$$

The best interpretation associated to (i, τ) is obtained by keeping the argmax in equation 8. The recognized interpretation is then the one associated to the final state at the final time[23].

Another possibility for approximately obtaining the best interpretation is studied in [19] in the context of image segmentation and recognition. They use the same criterion as proposed in this section (equation 4), corresponding to a recognition criterion equivalent to that defined in equation 6. They approximate equation 6 as follows. First, the N-best paths are obtained (using a variation of the Viterbi algorithm). Then for each one of the interpretations, say k, corresponding to these paths[24], the exact score $R(O, M_k)$ is computed (using the forward pass of the Baum-Welsh algorithm). Each of these forward passes is rather inexpensive in comparison to computing 6 exactly. Finally, the interpretation k yielding the best score is selected. Comparative experimental results [19] on zip-code recognition confirm the advantages of the hybrid and training criterion discussed in this section.

5 Conclusion

[22] a single entering state whose input transitions come from outside the group, and a single exiting state.

[23] or if there are many final states, the interpretation associated to the best scoring final state.

[24] a single interpretation can be associated to a given path.

After a review of the various ways in which prior knowledge can be integrated with learning in NNs in general and in particular for speech recognition, we have studied different structures for sequence recognition. The four types of architecture discussed here vary in their *expressive power* and in the *number of parameters* required to yield that expressive power in speech recognition problems. Networks with *delays* are generally better suited than the static isolated word recognizers because they take advantage of temporal translation invariance. Recurrent networks are even more powerful in their ability to represent past *context*, often outperforming networks without recurrence. In particular, comparatively good results were obtained with a network including recurrence and delays. However, gradient descent was found *inappropriate to learn long-term dependencies* in recurrent networks. This has motivated our presentation of several hybrids of neural networks and statistical techniques for sequence recognition. Various modules (such as a NN module, a HMM module) can be combined in several ways, with different training criteria. In particular, one such combination yielded a new approach to estimating a probability density function with neural networks.

Acknowledgments

The author would like to acknowledge the helpful discussions with L. Bottou, J. Denker, P. Frasconi, P. Simard and P. Haffner, as well as the support of the National Research Council of Canada, and the McDonnel-Pew Center for Cognitive Neuroscience at MIT (while the author was at the Massachusetts Institute of Technology).

References

1. Y.S. Abu-Mostafa, "Learning from Hints in Neural Networks," *J. of Complexity* 6, 1990, pp. 192-198.

2. B.S. Atal, J.J. Chang, M.V. Mathews, J.W. Tukeys, "Inversion of articulatory-to-acoustic transformation in the vocal tract by a computer sorting technique," *J. Acoust. Soc. Am.* **63** (5), 1970, pp. 1535–1555.

3. Y. Bengio, R. Cardin, R. De Mori and E. Merlo, "Programmable execution of multi-layered networks for automatic speech recognition," *Communications of the Association for Computing Machinery*, vol.

32, no. 2, 1989, pp. 195–199.

4. Y. Bengio, R. Cardin, R. De Mori and Y. Normandin, "A hybrid coder for hidden Markov models using a recurrent neural network," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, NM, 1990, pp. 537–540.

5. Y. Bengio Y., R. Cardin R., R. De Mori, "Speaker-independent speech recognition with neural networks and speech knowledge," *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky, Morgan Kaufmann Publishers, 1990, pp.218–225.

6. Y. Bengio, "Artificial Neural Networks and their Application to Sequence Recognition," Ph.D. Thesis, McGill University, (Computer Science), 1991, Montreal, Qc., Canada.

7. Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global Optimization of a Neural Network - Hidden Markov Model Hybrid," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, 1992, pp. 252–259.

8. Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Phonetically motivated acoustic parameters for continuous speech recognition using artificial neural networks," *Speech Communication*, **11**(2–3), 1992, pp. 261–271.

9. Y. Bengio, J. Houde, M. Jordan, "Representations based on articulatory dynamics for speech recognition", presented at the Neural Network for Computing Conference, 1992.

10. Y. Bengio and L. Bottou, "A new approach to estimating density functions with artificial neural networks," Computational Cognitive Science, Massachusetts Institute of Technology, T.R. 92.2, 1992.

11. Y. Bengio, P. Frasconi and P. Simard, "The problem of learning long-term dependencies in recurrent networks", invited paper to appear in the *Proceedings of the International Conference on Neural Networks*, March 1993, San Francisco, CA.

12. L. Bottou, "Une approche théorique de l'apprentissage connexioniste; applications à la reconnaissance de la parole". PhD thesis, Université de Paris XI, 1991.

13. L. Bottou and P. Gallinari, "A framework for the cooperation of learning algorithms", *Advances in Neural Information Processing Systems 3*, eds. Lippmann, Moody and Touretzky, publ. Morgan Kaufmann, 1991, pp. 781–788.

14. H. Bourlard and C.J. Wellekens, "Speech pattern discrimination and multilayer perceptrons," *Computer, Speech and Language*, vol. 3, 1989, pp.1–19.

15. J.S. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," *Advances in Neural Information Processing Systems 2*, (ed. D.S. Touretzky) Morgan Kaufmann Publ., 1990, pp. 211–217.

16. C. Burges, O. Matan, Y. Le Cun, J. Denker, L. Jackel, C. Stenard, C. Nohl and J. Ben, "Shortest Path Segmentation: A Method for Training a Neural Network to Recognize character Strings," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 1992, Baltimore, vol. III, pp.165–172.

17. P. Cosi, Y. Bengio and R. De Mori, "Phonetically-based multi-layered networks for acoustic property extraction and automatic speech recognition," *Speech Communication* **9**(1), 1990, pp. 15–30.

18. B. de Vries & J.C. Principe, "The gamma model – A new neural net model for temporal processing", *Neural Networks, 5*, pp. 565–576.

19. J.S. Denker, C. Burges, "Image segmentation and recognition: an overview", Technical Memorandum 11389-921212-30TM, AT&T Bell Laboratories, Adaptive Systems Research Department, December 1992.

20. X. Driancourt, L. Bottou and P. Gallinari, "Learning vector quantization, multi layer perceptron and dynamic programming: comparison and cooperation," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 1991, vol. II, pp.815–819.

21. G. Flammia, *Speaker Independent Consonant Recognition in Continuous Speech with Distinctive Phonetic Features*, M.Sc. Thesis, McGill University, School of Computer Science, 1991.

22. M. Franzini, K.F. Lee and A. Waibel, "Connectionist Viterbi training: a new hybrid method for continuous speech recognition," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, 1990, pp. 425–428.

23. P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Unified Integration of Explicit Rules and Learning by Example in Recurrent Networks," *IEEE Trans. on Knowledge and Data Engineering*, in press.

24. M. Gori, Y. Bengio and R. De Mori, "BPS: a learning algorithm for capturing the dynamic nature of speech," *Proc. IEEE Int. Joint Conf. on Neural Networks*, Washington DC, 1989, pp. II.417-II.424.

25. I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S.A. Solla, "Structural risk minimization for character recognition," *Advances in Neural Information Processing Systems 4*, (eds. Moody, Hanson & Lippmann) Morgan Kaufmann Publ., 1992, pp. 471–479.

26. P. Haffner, M. Franzini and A. Waibel, "Integrating time alignment and neural networks for high performance continuous speech recognition," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (ICASSP-91), Toronto, 1991, pp. 105–108.

27. R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, "Adaptive mixtures of local experts", *Neural Computation*, **3** (1), 1991, pp. 79–87.

28. Y. Le Cun, "Generalization and network design strategies,". In *Connectionism in Perspective*,

Elsevier Science Publ., 1989, pp. 143–155.

29. Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, 1989, pp.541–551.

30. K.F. Lee, *Automatic Speech Recognition: the development of the SPHINX system.*, Kluwer Academic Publ., 1989.

31. H.C. Leung, J.R. Glass, M.S. Phillips and V.W. Zue, "Phonetic classification and recognition using the multi-layer perceptron," *Advances in Neural Information Processing Systems 3*, eds. Lippmann, Moody and Touretzky, publ. Morgan Kaufmann, 1991, pp. 248–254.

32. E. Levin, "Word recognition using hidden control neural architecture," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, 1990, pp. 433–436.

33. E. Levin, R. Pieraccini and E. Bocchieri, "Time-warping network: a hybrid framework for speech recognition," *Advances in Neural Information Processing Systems 4*, eds. Moody, Hanson, and Lippmann, publ. Morgan Kaufmann, 1992, pp. 151–158.

34. S.E. Levinson, L.R. Rabiner and M.M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *Bell System Technical Journal*, vol. 64, no. 4, 1983, pp. 1035–1074.

35. K.A. Al-Mashouq and I.S. Reed, "Including Hints in Training Neural Nets," *Neural Computation*, Vol. 3, No. 4, 1991, p. 418.

36. O. Matan, C.J.C. Burges, Y. Le Cun and J.S. Denker, "Multi-digit recognition using a space displacement neural network," *Advances in Neural Information Processing Systems 4*, eds. Moody, Hanson, and Lippmann, publ. Morgan Kaufmann, 1992, pp. 488–495.

37. C. McMillan, M.C. Mozer and P. Smolensky, "Learning explicit rules in a neural network," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 1991, vol. II, pp.83–88.

38. N. Morgan and H. Bourlard, "Continuous speech recognition using multilayer perceptrons with hidden Markov models," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, 1990, pp. 413–416.

39. M. C. Mozer, "Neural net architectures for temporal sequence processing", to appear in: A. Weigend & N. Gershenfeld (Eds), *Predicting the future and understanding the past*, Redwood City, CA: Addison-Wesley Publ., 1993.

40. L.T. Niles and H.F. Silverman, "Combining hidden Markov models and neural network classifiers," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque,

NM, 1990, pp. 417–420.

41. L.Y. Pratt and C.A. Kamm, "Improving a phoneme classification neural network through problem decomposition," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 1991, vol. II, pp. 821–826.

42. L.R. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, Jan. 1986, pp. 257–285.

43. M.L. Rossen *Speech Syllable Recognition with a Neural Net,* Ph.D. Th., Brown University, 1989.

44. D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representation by error propagation," *Parallel Distributed Processing* volume 1. Rumelhart D.E. and McClelland J.L. (eds.), Bradford Books, MIT Press, 1986, pp. 318–362.

45. M. Schenkel, H. Weissman, I. Guyon, C. Nohl, D. Henderson, "Recognition-based segmentation of on-line hand-printed words", to appear in *Advances in Neural Information Processing Systems 5*, (ed. S. Hanson), Morgan Kaufmann, 1993.

46. T. Sejnowski and C.R. Rosenberg, "NETtalk: a parallel network that learns to read aloud," Technical report JHU/EECS-86/01, John Hopkins University, Electr. and Comp. Sc. Dept., 1986.

47. P. Simard, B. Victorri, Y. Le Cun and J. Denker, "Tangent Prop - a formalism for specifying selected invariances in an adaptive network," *Advances in Neural Information Processing Systems 4*, eds. Moody, Hanson, and Lippmann, publ. Morgan Kaufmann, 1992, pp. 895–903.

48. J. Tebelskis, A. Waibel, B. Petek and O. Schmidbauer, "Continuous speech recognition using linked predictive networks," *Advances in Neural Information Processing Systems 3*, eds. Lippmann, Moody and Touretzky, publ. Morgan Kaufmann, 1991, pp. 199–205.

49. G. Towell and J.W. Shavlik, "Interpretation of artificial neural networks: mapping knowledge-based neural networks into rules," *Advances in Neural Information Processing Systems 4*, eds. Moody, Hanson, and Lippmann, publ. Morgan Kaufmann, 1992, pp. 977–984.

50. A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. on ASSP*, vol. 37, no. 3, March 1989.

51. A. Waibel, H. Sawai and K. Shikano, "Modularity and scaling in large phonemic neural networks," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-37, 1989, pp. 1888–1898.

52. Y.H. Yu and R.F. Simmons, "Extra output biased learning," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 1990.

53. G. Zavaliagkos, S. Austin, J. Makhoul and R. Schwartz, "A hybrid continuous speech recognition system using segmental neural nets with hidden Markov models ", in this issue of the *International*

Journal of Pattern Recognition and Artificial Intelligence, 1993.

54. Zwicker E. and Terhardt E. (1980). Analytical expressions for critical band rate and critical bandwidths as a function of frequency. *Journal of the Acoustical Society of America*, vol. 68, no. 5, pp.1523-1525.

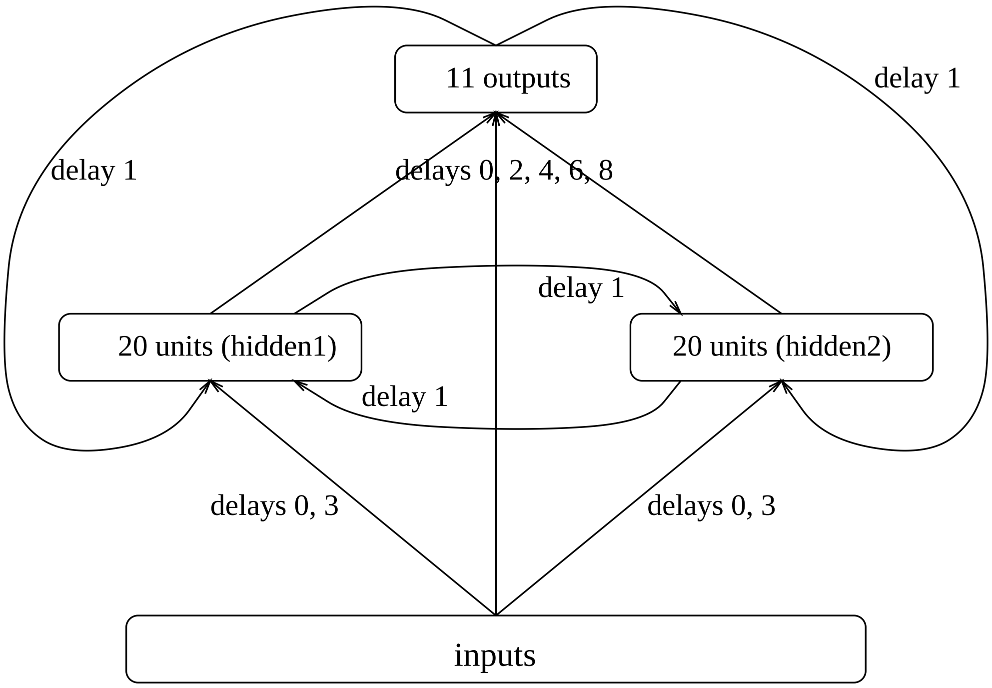

Figure 2: Best architecture obtained for vowel recognition in continuous speech on TIMIT (experiment #5 in Table 1).

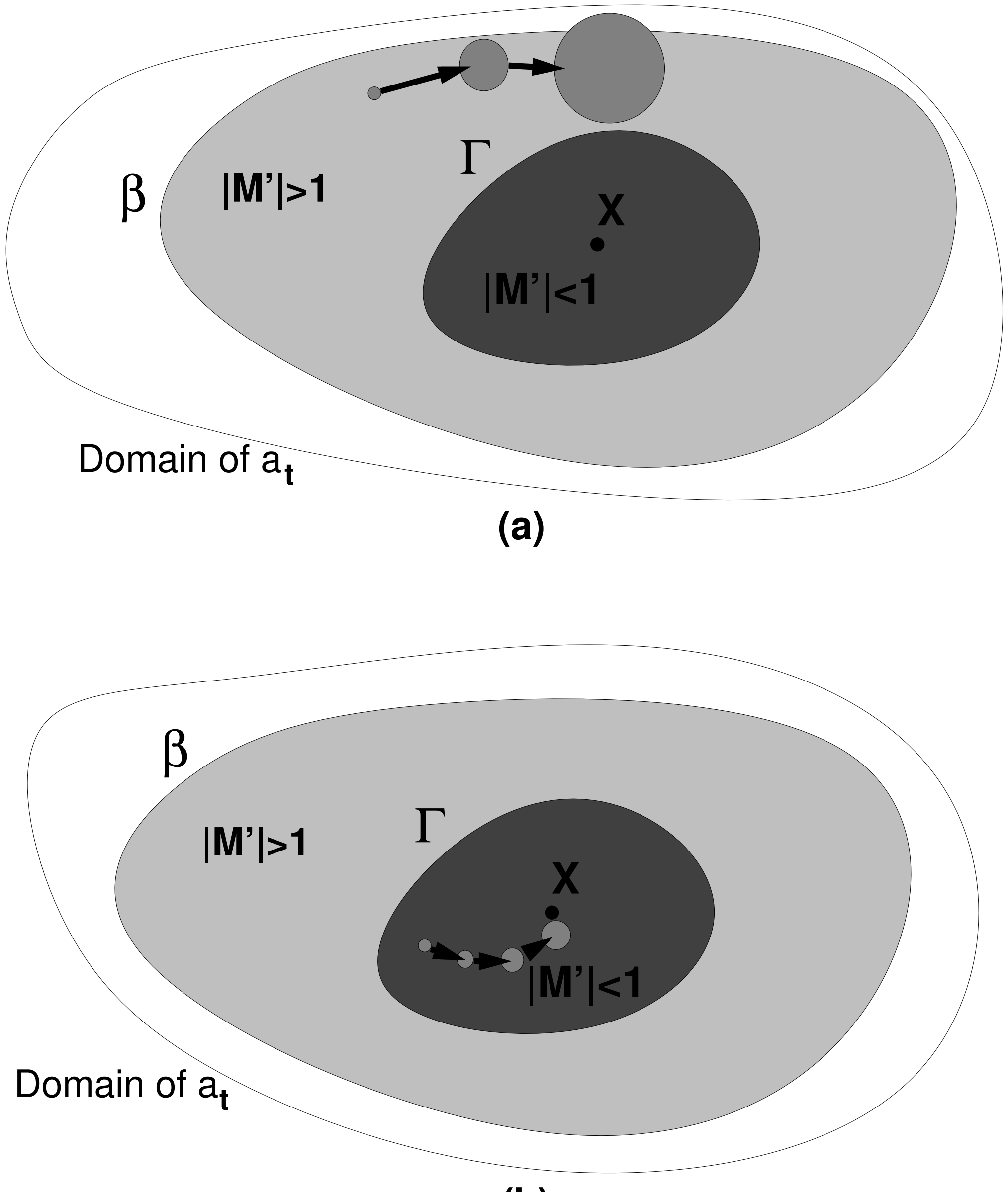

Figure 3: Illustration of the problem with using gradient descent to train a recurrent network to robustly store bits of state information. β is the basin of attraction of an attractor X. Γ is the region where the 2-norm of the Jacobian of the system dynamics map is less than 1. A ball of uncertainty grows exponentially (a) outside Γ, but is bounded (b) inside Γ.