# Pattern Recognition and Neural Networks

Yann LeCun
Rm 4G332, AT&T Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07733
yann@research.att.com

Yoshua Bengio
Dept. Informatique et Recherche
Opérationnelle, Université de Montréal,
Montreal, Qc, Canada, H3C-3J7
bengioy@iro.umontreal.ca

RUNNING HEAD: Pattern Recognition and Neural Networks

Correspondence:

Yann LeCun

Rm 4G332, AT&T Bell Laboratories, 101 Crawfords Corner Road

Holmdel, NJ 07733 , phone: 908-949-4038, fax: 908-949-7322

email: yann@research.att.com

# 1 INTRODUCTION

Pattern Recognition (PR) addresses the problem of classifying objects, often represented as vectors or as strings of symbols, into categories. The difficulty is to synthesize, and then to efficiently compute, the *classification function* that maps objects to categories, given that objects in a category can have widely varying input representations. In most instances, the task is known to the designer through a set of example patterns whose categories are known, and through general, a priori knowledge about the task, such as: "the category of an object is not changed when the object is slightly translated or rotated in space".

Historically, the field of PR started with the early efforts in Neural Networks (Perceptrons, Adalines...). While in the past, NNs have somewhat played the role of an outsider in PR, the recent progress in learning algorithms (and the availability of powerful hardware) have made them the method of choice for many PR applications.

Because most PR problems are too complex to be solved entirely by hand-crafted algorithms, machine learning has always played a central role in PR. Learning automatically synthesizes a classification function from a set of labeled examples. Unfortunately, no learning algorithm can be expected to succeed unless it is guided by prior knowledge. The traditional way of incorporating knowledge about the task is to divide the recognizer into a feature extractor, and a classifier. Since most learning algorithms work better in low-dimensional spaces with easily separable patterns, the role of the feature extractor is to transform the input patterns so that they can be represented by low-dimensional vectors, or short strings of symbols, that (a) can be easily compared or matched, and (b) are relatively invariant to transformations that do not change the nature of the input objects. The feature extractor contains most of the prior knowledge and is rather specific to the task. It also requires most of the design effort, because it is often hand-crafted (although unsupervised learning methods such as PRINCIPAL COMPONENT ANALYSIS can sometimes be used). The classifier, on the other hand, is often general-purpose and trainable. One of the main problems with this approach is that the recognition accuracy is largely determined by the ability of the designer to come up with an appropriate set of features. This turns out to be a daunting task which, unfortunately, must be redone for each new problem.

One of the main contributions of Neural Networks to PR has been to provide an alternative to this design: properly designed multi-layer networks can learn complex mappings in high-dimensional spaces without requiring complicated hand-crafted feature extractors. Networks containing hundreds of inputs and tens of thousands of parameters can be trained on databases containing several 100,000 examples. This allows designers to rely more on learning, and less on detailed engineering of feature extractors. Crucial to success is the ability to tailor the network architecture to the task, which allows incorporating prior knowledge, and therefore, learning complex tasks without requiring excessively large networks and training sets.

The success of multi-layer networks relies on one surprising fact: gradient-based minimization techniques can be used to learn very complex non-linear mappings. Generalizations of the concept of gradient-based learning have allowed to view many PR techniques, neural and non-neural, in a unified way, including not only traditional multi-layer feed-forward nets with sigmoid units and dot products, but also many other structures such as Radial Basis Functions, Hidden Markov Models (HMMs), vector quantizers, etc.. Many recent efforts have been directed at combining adaptive modules of different types into a single system, and training them cooperatively by propagating gradients through them, particularly for recognizing composite objects such as handwritten or spoken words.

# 2 LEARNING AND GENERALIZATION

Due to the presence of noise, the high dimension of the input, and the complexity of the mapping to be learned, PR applications create some of the most challenging problems in machine learning. Most learning methods are trained by minimizing a *cost function* computed over a set of training examples. The cost function is generally of the form:

$$C(W) = \sum_X Q\left(X, F(X, W)\right) + H(W) \tag{1}$$

where $X$ is a training example, $F(X, W)$ is the recognizer output for pattern $X$ and "parameters" $W$, $Q(X, F(X, W))$ is a cost function (the training error), and $H(W)$ is a measure of "capacity" of the recognizer (the REGULARIZER). Such cost functions attempt to model the real measure of performance, i.e., the *testing error* (error rate on a test set disjoint from the training set) (see also LEARNING AND GENERALIZATION).

System designers have to strike the right balance between learning the training set (by using powerful learning architectures), and minimizing the difference between the training error and the test error (by limiting the capacity of the machine). Large machines can learn the training set but may perform poorly if the training set is not large enough, a problem known as overparametrization, or overfitting. On the other hand, too little capacity yields to underfitting, i.e., large error on both training and test sets.

Most adaptive recognizers stand between two extremes of a continous spectrum. At one end, parameter-based methods, in which a set of learned parameters determines the input-output relation, put the emphasis on minimizing the first term in equation 1 with a fixed $H$ (e.g., multilayer neural networks). At the other end, memory-based methods, which rely on matching, or comparing, the incoming pattern with a set of learned or stored prototypes, keep the first term close to zero, and attempt to minimize the regularizer (e.g., nearest-neighbor algorithms).

Although, in principle, any appropriate functional form for $F$, $Q$, and $H$ can be used, the choice is largely determined by (a) the belief that it is well suited to the task, and (b) the efficiency of the available minimization algorithms. There is a strong incentive to choose smooth and well-behaved functions whose gradient can be computed easily, so that gradient-based minimization algorithms can be used, as opposed to inefficient combinatorial search methods. Preferably, $F$ will be a smooth real-valued function (e.g. layers of sigmoid units), rather than a discrete function (e.g. layers of threshold units); $Q$ is often chosen to be the mean-squared error between the actual output and a target, rather than the number of misclassified patterns, which would be more relevant, but which is practically impossible to minimize.

# 3 A FEW BASIC CLASSIFICATION METHODS

## 3.1 Linear and Polynomial Classifiers

A linear classifier is essentially a single neuron. An elementary two-class discrimination is performed by comparing the output to a threshold (multiple classes use multiple neurons). Training algorithms for Linear Classifiers are well studied (see PERCEPTRONS, ADALINES, AND BACKPROPAGATION). Their limitations are well known: the likelihood that a partition of $P$ vectors of dimension $N$ be computable by a linear classifier decreases very quickly as $P$ increases beyond $N$ (Duda and Hart, 1973). One method to ensure separability is to represent the patterns by high-dimensional vectors (large $N$). If necessary, the dimension of original input vectors can be enlarged using a set of basis functions $\phi_i$:

$$F(X, W) = \sum_i w_i \phi_i(X) \tag{2}$$

A simple example is when the basis functions are cross products of $K$ or fewer coordinates of the input vector $X$ ($F$ is a polynomial of degree $K$). Such polynomial classifiers have been studied since the early 60's, and have been "renamed" in the context of NNs as Sigma-Pi units or high-order nets. Unfortunately polynomial classifiers are often impractical because the number of features scales like $N^K$. Nevertheless, *feature selection* methods can be used to reduce the number of product terms, or to reduce the number of original input variables.

## 3.2 Local Basis Functions

Another popular kind of space expansion (equation 2) uses *local* basis functions, that are activated within a small area of the input space. A popular family are the Radial Basis Functions (RBFs): $\phi_i(X) = e^{-(X-P_i)^2}$, where the $P_i$ are a set of appropriately chosen "prototypes". Methods based on such expansions can cover the full spectrum between parameter-based and purely memory-based methods by varying the number of prototypes, the way they are computed, and the classifier that follows the expansion (which can be more complex than a simple weighted sum). At one extreme, each training sample is used as a prototype, to which the sample's label is attached. In the K-Nearest Neighbors algorithms, the $K$ nearest prototypes to an unknown pattern vote for its label. In the Parzen windows method, the normalized sum of all the $\phi_i(X)$ associated with a particular class is interpreted as the conditional probability that $X$ belongs to that class (Duda and Hart, 1973). In the RBF method the output is a (learned) linear combination of the outputs of the basis functions. Associating a prototype to each training sample can be very inefficient, and increases the "complexity" term. Therefore, several methods have been proposed to *learn* the prototypes. One way is to use unsupervised clustering techniques such as K-means to put prototypes in regions of high sample density, but supervised methods can also be used (see RADIAL BASIS FUNCTIONS). An important one is LVQ2 in which prototypes that are near a training sample are moved away from it if its assigned class differs from the sample's, and moved towards it if its class is equal to the sample's (see LEARNING VECTOR QUANTIZATION). Another important supervised method for RBF networks is simply gradient descent: the partial derivatives of the cost function with respect to the parameters of the basis functions (the prototype vectors) can be computed using a form of back-propagation: the same way gradients can be back-propagated through sigmoids and dot products, they can be back-propagated through exponentials and Euclidean distances. The parameters can then be adjusted using the gradient. It has been argued that the local property leads to faster learning than standard multi-layer nets, and good rejection properties (Lee, 1991). Several authors enhance the power of prototype-based systems by using distance measures that are more complex than just Euclidean distance between the prototypes and the input patterns (such as general bilinear forms with learned coefficients). Methods that add prototypes as needed have also been proposed, notably the RCE algorithm (see COULOMB POTENTIAL LEARNING).

### 3.3 Maximum Margin Classifiers

A recently-proposed elegant way of avoiding the curse of dimensionality in polynomial and local classifiers rests on the fact that, if the $w_i$ in equation 2 are computed to maximize the *margin* (the minimum distance between training points and the classification surface), the $W$ obtained after training can be written as a linear combination of a small subset of the expanded training examples (Boser, Guyon and Vapnik, 1992). Points in this subset are called *support points.* This leads to a surprisingly simple way of evaluating high degree polynomials in high dimensional spaces *without having to explicitly compute all the terms of the polynomial.* For example, maximum-margin polynomials of degree $K$ can be computed using:

$$F(X) = \sum_{j \in S} \alpha_j (X \cdot P_j + 1)^K \tag{3}$$

where the $P_j$ are the support points (subset of the training set), and the $\alpha_j$ are coefficients that uniquely determine the weights $W$. Learning the $\alpha_j$ amounts to solving a quadratic programming problem with linear inequality constraints. Excellent results on handwritten digit images have been obtained with a 4th degree polynomial computed with this method

(Bottou et al., 1994). The number of multiply-adds per recognition was a few 100,000's, much less than the $O(400^4)$ multiply-adds required to directly evaluate the polynomial.

## 3.4 Complex Distance Measures

Although many memory-based methods use simple distance measures (Euclidean distance), and large collections of prototypes, some applications can take advantage of more complex, problem-dependent, distance measures, and use fewer prototypes. Ideal distance measures should be invariant with respect to transformations of the patterns that do not change their nature (e.g. translations and distortions for characters, time or pitch distortion for speech). With invariant distances, a single prototype can potentially represent many possible instances of a category, reducing the number of necessary prototypes. An important family of invariant distance measures is *elastic matching*. Elastic matching comes down to finding the point closest to the input pattern on the surface of all possible deformations of the prototype. Naturally, the exhaustive search approach is prohibitively expensive in general. However, if the surface is smooth, better search techniques can be used: gradient descent (Burr, 1983), or conjugate gradient (Hinton, Williams and Revow, 1992). If the deformations are along one dimension (like in speech), dynamic programming can find the best solution efficiently. In an interesting technique, recently proposed in (Simard, LeCun and Denker, 1993), the surface of a deformed prototype is approximated by its tangent plane at the prototype. The matching problem reduces to finding the minimum distance between a point and a plane, which can be done efficiently. This has been applied to handwritten character recognition with great success.

# 4 MULTI-LAYER NETWORKS, GRADIENT-BASED LEARNING

The vast majority of applications of NNs to PR are based on multi-layer feed-forward networks trained with back-propagation. At first, it seems almost magical that an algorithm as simple as gradient descent *works at all* to learn complex non-linear mappings (non convex, ill conditioned error surfaces). Minsky and Selfridge's warning about the limitations of "hill-climbing" methods for machine learning in the late fifties is an indication of the general belief that it could not work. Surprisingly, experiments show that local minima are rarely a problem with large networks. As evidence to the success of back-propagation, all but two of the entries in the last NIST character recognition competition used some form of back-propagation network.

PR problems are often characterized by large and redundant training sets with high-dimensional inputs, which translates into large networks, and long learning times. Much effort has been devoted to speeding up training using refined non-linear optimization methods (conjugate gradient, quasi-Newton methods,...). These are essentially batch methods (the weights are updated after a complete pass through the training set), which can rarely compete with "carefully tuned" stochastic (on-line) gradient descent (where the weights are updated after each pattern presentation). This is due to the presence of redundancy in large, natural training sets. On typical large-scale image or speech recognition tasks, stochastic gradient descent converges in one to a few dozen epochs. To avoid overlearning, a validation set should be set aside, and training should be stopped when the error rate on the validation set stops decreasing. An important limitation to the popularity of NN techniques for PR is that certain simple tricks must be used and many common pitfalls must be avoided that are part of the "oral culture" rather than scientific facts (LeCun, 1989).

Once back-propagation with feed-forward networks of sigmoid units and dot-products established the value of gradient-based learning, it seemed natural to extend the idea to other structures. Minimizing a cost function through gradient-based learning can be seen as the unifying principle behind many methods: Radial Basis Functions or mixtures of Gaussians, Learning Vector Quantization, HMMs, and many prototype-based methods using various distance measures. Experiments have shown the advantage of using different types of modules in different parts of a learning system. In particular, sigmoids and dot-products seem better for processing large amounts of high-dimensional and low-level information (early feature extraction), while RBF or other more local modules seem better suited for final classification, a more memory-intensive task. With the gradient-based learning framework, modules of different types can be connected in any configuration, and trained cooperatively by back-propagating gradients through them. To achieve this, one only needs to be able to compute the partial derivatives of each output of a module with respect to each input and each parameter of the module (see MULTI-MODULAR SYSTEMS). In addition, many *cost functions* can be considered as just another module (with a scalar output) through which gradients can be back-propagated. Examples include the Mean-Squared Error, modified LVQ cost functions, Maximum Likelihood, Maximum Mutual Information, Cross Entropy, Classification Figure of Merit, and several types of statistical post-processors.

## 4.1 Local/Global and Modular Methods

It has recently been suggested that good PR systems should behave differently in different parts of the input space. For example, parts of the input space may be very sparsely populated, requiring a low-capacity learner, while denser areas may require a more complex one. A simple idea is to use a collection of modules, each of which is activated when the input lies in a particular region. A separate module, called a "gater", decides which module should be activated. When the gater is differentiable, the whole system (modules + gater) can be trained cooperatively (see HIERARCHICAL NEURAL NETWORK LEARNING). In such multi-modular systems, parameters are relatively decoupled across modules, which is believed to allow for faster training (or better scaling of training time). In another interesting "semi-local" method, a simple network (e.g., single layer) is trained *each time* a new test pattern is presented, using training patterns in the neighborhood of this test pattern; training is done "on demand" during recognition (Bottou and Vapnik, 1992).

In general, local methods learn fast, but they are expensive at run-time, in terms of memory and, often, of computation. In addition, they may not be appropriate for problems with high-dimensional inputs. Global methods, such as multi-layer networks, take longer to train, but they are quite compact, and they execute quickly. They can handle high-dimensional inputs, particularly when specialized architectures are used.

## 4.2 Specialized Architectures, Convolutional Networks

The great hope that multi-layer networks brought with them was the possibility of eliminating the need for a separate hand-crafted feature extractor, relying on the first layers to automatically learn the right set of features. Although fully-connected networks fed with "raw" character images (or speech spectra) have very large numbers of free parameters, they have been applied with some success (Martin and Pittman, 1991). This can be explained as follows. With small initial weights a multi-layer network is almost equivalent to a single-layer network (each layer is quasi-linear). As incremental learning proceeds, the weights gradually increase, thereby progressively increasing the effective capacity of the system (to the authors' knowledge, this explanation was first suggested by Léon Bottou in 1988).

Nevertheless, using a specialized network architecture, instead of a fully-connected net, can reduce the number of free parameters, and facilitate the learning of invariances. In certain applications, the need for a separate hand-crafted feature extractor can be eliminated by wiring the first few layers of the network in a way that forces it to learn relevant features and eliminate irrelevant variability. CONVOLUTIONAL NETWORKS (see article in this volume), including Time-Delay Neural Networks, are an important class of specialized architectures, well suited for dealing with 1D or 2D signals such as time-series, images, or speech. Convolutional networks use the techniques of local receptive fields, shared weights, and subsampling (loosely based on the architecture of the visual cortex) to ensure that the first few layers extract and combine local features in a distortion-invariant way. Although the wiring of the convolutional layers is designed by hand, the values of all the coefficients are learned with a variant of the backpropagation algorithm. The main advantage of this approach is that the feature extractor is totally integrated into the classifier, and is produced by the learning process, rather than by the hand of the designer (LeCun et al., 1990). Due to the weight sharing technique, the number of free parameters in a convolutional network is much less than in a fully-connected network of comparable power, which has the effect of reducing the "complexity" term in equation 1, and improving the generalization. The success of convolutional nets of various types has had a major impact on several application domains: speech recognition, character recognition, object spotting. On handwriting recognition tasks, they compare favorably with other techniques (Bottou et al., 1994) in terms of accuracy, speed, and memory requirements. Character recognizers using convolutional nets have been deployed in commercial applications. A very promising feature of convolutional nets is that they can be efficiently replicated, or scanned, over large input fields, resulting in the so-called "Space Displacement Neural Net" (SDNN) architecture (see below, and CONVOLUTIONAL NETWORKS in this volume).

Networks with recurrent connections can be used to map input sequences to output sequences, while taking long-term context into account.The main advantage of recurrent networks over TDNNs for analyzing sequences is that the span of the temporal context that the network can take into account is not hard-wired within a fixed temporal window by the architectural choices, but can be learned by the network. However, theoretical and practical hurdles (Bengio et al., 1994) limit the span of long-term dependencies that can be learned efficiently.

# 5 RECOGNITION OF COMPOSITE OBJECTS

In many real applications the difficulty is not only to recognize individual objects but also to separate them from context or background. For example, one approach to handwritten word recognition is to *segment* the characters out of their surrounding, and recognize them in isolation. A typical handwritten word recognizer uses heuristics to form multiple, possibly overlapping, character candidates by cutting the word or by joining nearby strokes. Then, the recognizer must either classify each candidate as a character, or reject it as a non-character. In many applications, such as cursive handwriting or continuous speech, it is difficult, or even impossible, to devise robust segmentation heuristics. One approach to avoid explicit segmentation is to simply scan the recognizer over all possible locations on the input (character string or spoken sentence) and collect the sequence of corresponding recognizer outputs. Although this is very computationally expensive in general, replicated convolutional networks (SDNN or TDNN) can be used to do that very efficiently. In the case of handwriting recognition, an SDNN output will contain a well identified label when centered on a character. Between characters, the output should indicate a reject. However, combinations of off-center characters may cause ambiguous outputs (e.g. "cl" labeled as "d"). Since both methods (explicit segmentation, and scanning) generate many extraneous candidates, a post-processor is required to resolve ambiguities and pull out the most consistent interpretation, retaining genuine characters and rejecting erroneous stroke combinations, possibly taking linguistic constraints into account (a lexicon or grammar). For this to succeed, the recognizer must be trained not only to classify characters, but also to reject non-characters. The search for the best interpretation is easily done within the framework of Hidden Markov Models. A graph is built in which each path corresponds to a possible interpretation of the input, and in which each node is given probabilities of matching recognizer outputs. Dynamic programming can be used to find the path of highest probability, which yields the most likely interpretation.

## 5.1 Multi-module Architectures and Cooperative Training

Such combinations of neural networks and HMMs (or other graph-based post-processors) have been proposed by several authors, mostly for speech recognition (see SPEECH RECOGNITION AND NEURAL NETWORKS), but also for handwriting recognition (see HAND WRITTEN DIGIT RECOGNITION, and CONVOLUTIONAL NETWORKS in this volume).

The main technical difficulty is in *training* such hybrid systems. Training the recognizer exclusively on pre-segmented characters is neither sufficient, nor always possible, since (a) the recognizer must be trained to reject non-characters and (b) in many cases, such as cursive handwriting, segmented characters are not available, only whole words are. The solution is to simultaneously train the recognizer and the post-processor to minimize an error measure at the *word level.* This means being able to back-propagate gradients through the HMM, down to the recognizer, or to generate desired outputs for the recognizer using the best path in the graph (see SPEECH RECOGNITION AND NEURAL NETWORKS and (Franzini, Lee and Waibel, 1990)). Simultaneous training of such hybrids has been reported to yield large reductions in error rates over independent training of the modules in speech recognition (TDNN/dynamic time warping (Driancourt, Bottou and Gallinari, 1991; Haffner, Franzini and Waibel, 1991), TDNN/HMM (Bengio et al., 1992)), and in on-line handwriting recognition (SDNN/HMM (Bengio, LeCun and Henderson, 1994)).

# 6 DISCUSSION

Neural Networks, particularly multi-layer back-propagation NNs, provide simple, yet powerful and general methods for synthesizing classifiers with minimal effort. However, most practical systems combine NNs with other techniques for pre- and post-processing. On isolated character recognition tasks, multi-layer nets trained with variants of back-propagation have approached human accuracy, at speeds of about 1000 characters per second using NN hardware. NNs have allowed workers to minimize the role of detailed engineering, and maximize the role of learning. Despite the recent advances in multi-module architectures and gradient-based learning, several key questions are still unanswered, and many problems are still out of reach. How much has to be built into the system, and how much can be learned? How to achieve true transformation-invariant perception with NNs? convolutional nets are a step in the right direction, but new concepts will be required for a complete solution (see the DYNAMIC LINK ARCHITECTURE). How to recognize compound objects in their context? the accuracy of the best NN/HMM hybrids for written or spoken sentences can not even be compared with human performance. Topics, such as the recognition of 3D objects in complex scenes, are totally out of reach. Human-like accuracy on complex PR tasks such as handwriting and speech recognition may not be achieved without a drastic increase in the available computing power. Several important questions may simply resolve themselves with the availability of more powerful hardware, allowing the use of brute-force methods and very large networks.

## Acknowledgements

The authors wish to thank Leon Bottou, Chris Burges, John Denker, Isabelle Guyon, Larry Jackel, Patrice Simard, Vladimir Vapnik, and the other members of the Adaptive Systems Research Department for their support and comments.

# References

Bengio, Y., , Simard, P., and Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks.* Special Issue on Recurrent Neural Networks, March 94.

Bengio, Y., LeCun, Y., and Henderson, D. (1994). Globally Trained Handwritten Word Recognizer using Spatial Representation, Space Displacement Neural Networks and Hidden Markov Models. In *Advances in Neural Information Processing Systems*, volume 6, pages 937–944.

Bengio, Y., Mori, R. D., Flammia, G., and Kompe, R. (1992). Global Optimization of a Neural Network-Hidden Markov Model Hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259.

Boser, B., Guyon, I., and Vapnik, V. (1992). An algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh.

Bottou, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Jackel, L., LeCun, Y., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. (1994). Comparison of classifier methods: a case study in handwritten digit recognition. In *International Conference on Pattern Recognition*, Jerusalem, Israel.

Bottou, L. and Vapnik, V. (1992). Local Learning Algorithms. *Neural Computation*, 4(6):888–900.

Bridle, J. (1990). Alphanets: a Recurrent 'Neural' Network Architecture with a Hidden Markov Model Interpretation. *Speech Communication.*

Burr, D. (1983). Designing a handwriting reader. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(5):554–559.

Denker, J. and LeCun, Y. (1991). Transforming neural-net output levels to probability distributions. In Lippman, R. P., Moody, R., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 853–859, San Mateo CA. Morgan Kaufmann.

Driancourt, X., Bottou, L., and Gallinari, P. (1991). Learning Vector Quantization, Multi Layer Perceptron and Dynamic Programming: Comparison and Cooperation. In *International Joint Conference on Neural Networks*, volume 2, pages 815–819.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis.* Wiley, New York.

Franzini, M., Lee, K., and Waibel, A. (1990). Connectionist Viterbi Training: a New Hybrid Method for Continuous Speech Recognition. In *International Conference on Acoustics, Speech and Signal Processing*, pages 425–428, Albuquerque, NM.

Haffner, P., Franzini, M., and Waibel, A. (1991). Integrating Time Alignment and Neural Networks for High Performance Continuous Speech Recognition. In *International Conference on Acoustics, Speech and Signal Processing*, pages 105–108, Toronto.

Hinton, G., Williams, C., and Revow, M. (1992). Adaptive elastic models for hand-printed character recognition. In Moody, J., Hanson, S., and Lipmann, R., editors, *Advances* *in Neural Information Processing Systems 4*, pages 512–519, San Mateo CA. Morgan Kaufmann.

LeCun, Y. (1989). Generalization and Network Design Strategies. Technical Report CRG-TR-89-4, Department of Computer Science, University of Toronto.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1990). Handwritten Digit Recognition with a Back-Propagation Network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404, Denver 1989. Morgan Kaufmann, San Mateo.

Lee, Y. (1991). Handwritten digit recognition using K nearest neighbor, radial-basis function, and backpropagation neural network. *Neural Computation*, 3(3):441–449.

Martin, G. and Pittman, J. (1991). Recognizing hand-printed letters and digits using back-propagation learning. *Neural Computation*, 3(2):258–267.

Simard, P., LeCun, Y., and Denker, J. (1993). Efficient pattern recognition using a new transformation distance. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems*, volume 5, pages 50–58, Denver 1992. Morgan Kaufmann, San Mateo.

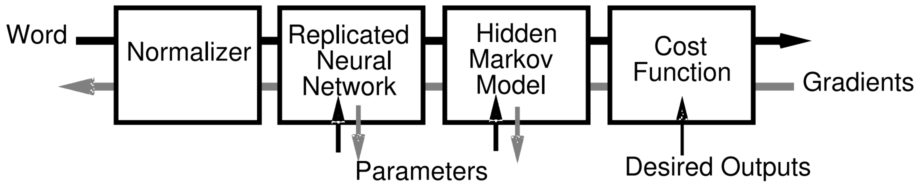


Figure 1: A multi-module architecture combining a convolutional NN with a HMM post-processor.