Direct Transfer of Learned Information Among Neural Networks

Lorien Y. Pratt and **Jack Mostow**
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

and **Candace A. Kamm**
Speech Technology Research
Bellcore, 445 South Street
Morristown, NJ 07962-1910

Abstract

A touted advantage of symbolic representations is the ease of transferring learned information from one intelligent agent to another. This paper investigates an analogous problem: how to use information from one neural network to help a second network learn a related task. Rather than translate such information into symbolic form (in which it may not be readily expressible), we investigate the direct transfer of information encoded as weights.

Here, we focus on how transfer can be used to address the important problem of improving neural network learning speed. First we present an exploratory study of the somewhat surprising effects of pre-setting network weights on subsequent learning. Guided by hypotheses from this study, we sped up back-propagation learning for two speech recognition tasks. By transferring weights from smaller networks trained on subtasks, we achieved speedups of up to an order of magnitude compared with training starting with random weights, even taking into account the time to train the smaller networks. We include results on how transfer scales to a large phoneme recognition problem.

Introduction

Recently, many empirical comparisons (surveyed in [Shavlik *et al.*, 1991]) have been performed between neural network and symbolic machine learning methods on a variety of tasks. Back-propagation neural networks [Rumelhart *et al.*, 1987] have been shown to perform competitively. At present, one reason to prefer symbolic representations is that they are more readily *portable* – we can simply copy axioms, rules, definitions, etc. between intelligent agents for reuse on new tasks. At least in principle, information learned by one agent is easy to share with others. It is less clear how to transfer information encoded in neural networks.

A brute force approach to information transfer in neural networks is to store and communicate the entire set of data used to train the network. Besides the obvious storage costs, this approach suffers from requiring the network to be retrained from scratch every time additional data is received.

An indirect approach is to extract information in symbolic form from one network and insert it into another. While both the extraction [Fu, 1990] and insertion [Towell *et al.*, 1990] processes are receiving some research attention, this indirect approach is limited by the fact that information encoded in the original network may be infeasible to express in symbolic form.

In contrast, this paper investigates how information encoded as learned network weights can be directly transferred between neural networks, short-cutting the indirect approach, as shown in Figure 1. By exploiting previous learning, such a process has the potential to increase network performance and also to improve learning speed, which has been found to be much slower in general for neural networks than for symbolic methods.

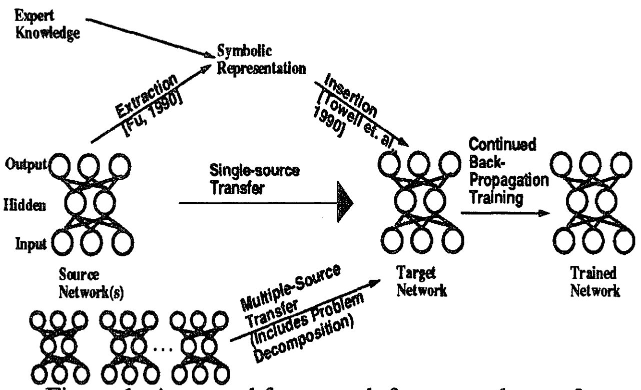


Figure 1: A general framework for network transfer

We use the back-propagation algorithm for neural network learning. This algorithm iteratively modifies a set of initial weights, with the goal of building a network that gives correct answers on a corpus of training examples. Usually, the starting weights are chosen randomly. Here, we show how using previously learned information to initialize weights non-randomly can speed up network learning.

Following a brief introduction to back-propagation, we present a pilot study that explores transfer from a single source to a single target network, as shown in the center of Figure 1. Then we describe two studies on more complex tasks that demonstrate transfer from multiple source networks to different portions of the target network.

Back-Propagation

Back-propagation is an inductive algorithm for concept learning and regression. A back-propagation neural network contains several layers of computational units, connected via weighted arcs. A typical network has an *input layer*, a single *hidden layer*, and an *output layer* of units, with full connectivity between adjacent layers. Each unit has an associated state, called its *activation*, typically a real number in the interval [0, 1]. A distinguished unit called a *threshold* is usually included in the input and hidden layers. Its activation is always 1, and its weight is called a *bias*.

When using a learned network, input unit activations are set externally and used to calculate activations in subsequent layers (this process is called *feedforward*). The network's solution to the problem represented by the input activations is read from output layer activations. For a given connection topology, network behavior is determined by the arc weight values. To determine its activation, each unit first calculates its *input*, via an *input function*. This is typically a linear combination of incoming unit activations times connecting weights. The input is fed through a *squashing function*, usually sigmoidal in shape, which maps the input value into [0, 1].

Learning in back-propagation networks consists of modifying weight values in response to a set of *training data patterns*, each of which specifies an activation for each input unit and the desired *target* value for each output unit. Network weights are initialized randomly, usually in some small-magnitude range such as [−.5, .5]. Then the first training data item is used to set input unit activations, and feedforward calculates output unit activations. They are compared to target values, and an error is determined for each output unit. This error is used to determine the change to each network weight. This process is repeated for all training patterns – one pass through them all is an *epoch*. Typically, learning requires thousands of epochs, during which weights are updated by small increments until output vectors are close to target vectors. Weight increment size is determined in part by the user-supplied parameters η (*learning rate*) and α (*momentum*). For more details, see [Rumelhart *et al.*, 1987].

To use back-propagation for concept learning tasks, a *unary encoding* is usually used for output units, each of which represents a different category. Target vectors contain a 1 in the position corresponding to the desired category, and 0's elsewhere. When the network is used to classify an input vector, the highest output unit activation is used to determine the network's chosen category.

Pilot study: Weight Magnitudes and Network Initialization

In this section, we present an exploratory study of the dynamics of networks that have some of their initial weights pre-set. Consider a single-hidden-layer network that is fully connected in the input-to-hidden (IH) and hidden-to-output (HO) layers, and trained for concept learning using unary encoding. Let the input function be linear, so the input to unit j is $I_j = \sum_i y_i w_{ij}$, where y_i is the activation of incoming unit i, and w_{ij} is the weight between units i and j. Let the squashing function be the *step function*:

$$\text{activation } a_j = \begin{cases} 1 & \text{if } I_j + \text{bias} > 0 \\ 0 & \text{if } I_j + \text{bias} \leq 0 \end{cases}$$

This function is a simplification of the sigmoid. It is commonly used for analysis, since for high weight values it approximates the sigmoid.

Consider a space of n dimensions, where n is the number of input units. As shown in Figure 2, training data define points in this space which can be labelled by corresponding target values. Weights leading from input units to a particular hidden unit, along with the hidden unit bias, determine a hyperplane-bounded decision region in this space. For $n = 2$, the hyperplane is a line. Input vectors on one side of this hyperplane cause a hidden unit activation of 0; vectors on the other side cause an activation of 1.

One condition for successful back-propagation learning is that IH hyperplanes separate the training data such that no decision region contains training data items with different target values. Another condition for learning is that IH hyperplanes are placed such that there is a correct configuration of HO hyperplanes possible (i.e. that hidden layer activations are linearly separable).

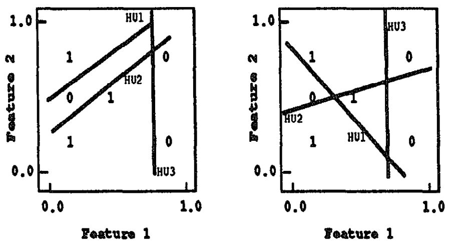


Figure 2: Two examples of hyperplane sets that separate training data in a small network.

Since separating training data items for different targets by IH hyperplanes is one condition for learning, it is reasonable to expect that pre-setting IH weights to produce such hyperplanes will lead to faster learning. We performed a series of experiments to test this hypothesis.

Conditions and Results

All networks studied had a 2-3-1 (2 input units, 3 hidden units, 1 output unit) topology. They were trained on a small set of hand-chosen training data that was not linearly separable and that contained 6 patterns (Figure 2). Each training data target was either 0 or 1. A manual search for values of η and α (10 pairs tried, on networks with random initial weights) resulted in locally optimal values of $\eta = 1.5$, $\alpha = .9$, which were used for all experiments. Standard back-propagation [Rumelhart *et al.*, 1987] was used, with a sigmoidal activation function, training after every pattern presentation, and training data presented sequentially.

A number of experiments were performed, summarized in Table 1. Each row of this table shows results from a set of 30 trials (each with different random initial weights) of the conditions shown. Every network was trained for 2000 epochs. Experiments fell into the following five categories:

Random low magnitude initial weights (Exp. 1): In our control experiment, 30 networks were initialized with random weights in the range: [−0.5, 0.5] (average IH, HO mag-

Exp#, IH source	IH avg. mag.	HO avg. mag.	#con-verg-ing	Mean Epochs to Con-verge	Sig. diff from #1?
1: Random	.25	.25	28	528	—
2: Preset	.25	.25	26	501	N
3: Preset	2.5	.00625	26	228	Y
4: Preset	5.0	.00625	23	151	Y
5: 0.6-perturbed	2.5	.00625	24	199	Y
6: Centroid	.25	.25	28	591	N
7: Centroid	1.25	.00625	30	859	Y
8: Centroid	2.5	.00625	20	898	Y
9: Random	2.5	.00625	23	329	N

Table 1: Conditions and results for the pilot study.

nitudes of 0.25). 28 trials converged to a solution. Convergence time was measured as the number of epochs of training before the total squared error over all patterns (TSS) dropped below 0.1. As shown, the mean time to converge for the 28 networks was 528 epochs.

Pre-set separating IH weights, random HO weights (Exps. 2,3,4): 30 converging networks were generated, and their separating IH weights were extracted and used to initialize a new set of 30 networks (not all of which converged). Note that since a hyperplane position is determined by the ratio, not the magnitude, of its defining weights, it is possible to adjust hyperplane magnitudes arbitrarily while retaining their positions. IH and HO magnitudes were rescaled in a variety of ways, as shown. As the IH magnitude was raised, fewer networks converged, but those that did had shorter training times (significantly shorter in studies 3 and 4, with $p < .0001, df = 50, 44$, according to a t-test). [1]

Perturbed (Exp. 5): The same weights as in the previous experiments were used, but each IH weight was modified to be $w = w + r * w$, where r was a random variable in $[-0.6, 0.6]$. This produced hyperplanes that didn't separate training data completely, but were in the proximity of effective final positions. Perturbed networks trained significantly faster ($p < .0001, df = 46$) than randomly initialized networks.

Centroid initialization (Exp. 6,7,8): To test the generality of the idea of placing hyperplanes near training data, IH hyperplanes were initialized to pass through the training data centroid (median value of each input dimension). As shown, at best this produced training times no different from the random case. At worst, learning time was *longer*.

Random high-magnitude initial weights (Exp. 9): This experiment verified that the speed-up found with high weight magnitudes was due to both their positions and their magnitudes, instead of just their magnitudes. As shown, the high-magnitude random networks did not converge significantly faster ($df = 44$) than low-magnitude initialized networks.

[1] In order to perform a fair significance test in comparisons with randomly initialized networks, when only some subset $k < 28$ of networks converged, the $(30 - k)$ largest times were removed. For Experiment 7, the two worst centroid-initialized scores were removed, since it converged more often than the random case.

Discussion

These results yield several observations that can be used as hypotheses for investigation on more complex tasks:

Learning was faster in networks with pre-set weights than when weights were randomly initialized. In Experiments 3, 4, and 5, learning speed was significantly faster ($p < .0001, df = 50, 44, 46$) than when weights were initialized randomly.

Learning was faster in networks with pre-set weights specifying near-correct hyperplane positions than when weights were randomly initialized. This was shown in Experiment 5.

Surprisingly, correctly pre-set hyperplanes moved out of position. This happened when IH magnitudes were too low, as in Experiment 2. Hyperplanes were observed to diverge rapidly in early epochs, losing their preset positions.

Raising weight magnitudes made hyperplanes more retentive. In Experiment 4, hyperplanes moved out of position to a much smaller degree than in Experiment 2. The mean number of training data patterns crossed by moving hyperplanes during learning was 24.6 in Experiment 4, compared to 36.4 for Experiment 2. The difference was significant with $p < 0.01, df = 58$.

Fastest learning was obtained when weights were preset in the correct positions and weight magnitudes were raised to make them retentive. This was shown in Experiment 4.

Networks with pre-set hyperplanes tended not to converge as often. This was shown by Experiments 2-5,8.

Additional small experiments (only 10 runs, no significance testing) were performed on this same task to determine whether these hypotheses are also supported when a different learning rate is used: in this case $\eta = 0.1$ was tried (instead of 1.5). The results of these experiments were consistent with the above observations.

Problem Decomposition in a Vowel Recognition Task

In the previous section, we showed that hyperplanes initialized near correct positions may move into position, thus speeding up learning. Here we extend those results to a more complex task. We demonstrate a technique for pre-setting network weights that produces faster learning, even taking into account the time to learn pre-set weights. This is because the original problem is decomposed into subproblems, thereby reducing search combinatorics. In this and the next section we borrow a decomposition technique introduced by [Waibel *et al.*, 1989], who showed improved learning speed and performance in networks for consonant recognition.

Using data from [Robinson, 1989], we trained a network to perform a vowel recognition task. It had 10 input units (representing a preprocessed speech signal) and 11 output units, each representing a vowel sound. Training was on 528 vowels pronounced by 8 speakers, and the test set was 462 vowels from 7 speakers not included in that set. Four baseline networks were trained, each with 26 hidden units and no decomposition

We next studied the performance of networks in which IH and HO hyperplanes were pre-set through problem decomposition. The architecture used is illustrated in Figure 3. The number of weights in these networks was approximately the same as that in the non-decomposed networks.

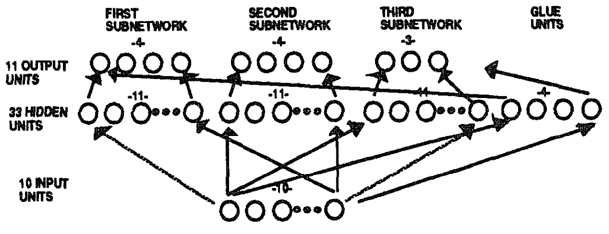


Figure 3: Architecture of the decomposed vowel recognition network. Arrows delimit regions of full connectivity. Numbers indicate unit counts.

The training methodology used by [Waibel *et al.*, 1989] for decomposed networks included these steps:

1. **Subnetwork training**: Divide network into subsets; train each individually.
2. **Glue training:** Combine subnetworks into a larger network using additional "glue" units. These are trained while subnetwork weights are frozen.
3. **Fine tuning:** All weights are modified during further training.

Although fine tuning may be useful in some problems, there are a few reasons to question whether it will always improve network performance. First, for fine tuning to be useful, it would seem to require that, as in Experiment 5 of the previous section, subnetwork training develops a set of roughly positioned hyperplanes, and fine-tuning moves them into place. This won't necessarily happen in all decomposed networks – there may be local minima that are optimal for subproblems but are not near an optimum solution for the full problem. Secondly, weights often grow during back-propagation learning, so it may be important to set relative magnitudes between glue and subnetworks systematically. This was not part of the fine-tuning step in [Waibel *et al.*, 1989]. Finally, for some networks, if glue training is stopped before overfitting begins, fine-tuning has the potential to cause overfitting, reducing network performance.

We sought both to test whether problem decomposition was useful in decreasing training time and to explore whether fine tuning did indeed improve performance on this task. Five different decomposed networks were trained, each with different initial random weights; they were compared to the four non-decomposed networks. We used an "oversized" network training technique [Weigend *et al.*, 1990] to control for overfitting: networks were trained until their errors on the test set ceased to improve. We trained each subnetwork on all input vectors in the training set, with all-zero target vectors for input patterns in classes outside of those corresponding to a subnetwork's output units. We performed t-tests of significance on the learning time and performance difference between the two populations of decomposed and non-decomposed networks. Although space limitations preclude reporting all experimental details, major results are outlined below:

Performance: No significant difference in test set score was found between performance of the decomposed and non-decomposed networks ($p > 0.05, df = 7$). Performance scores, in percent correct on the test data, were 53, 58, 55, 56, 55 for the decomposed and 58, 61, 59, 54 for the non-decomposed networks.

Learning Speed: Decomposed networks learned faster than monolithic networks. The mean decomposed time was 40% of the mean non-decomposed time (learning significantly faster with $p < .01, df = 7$). The mean decomposed learning time was 1.348×10^9 operations (3096 epochs).

Fine Tuning: For one arbitrarily chosen set of subnetwork weights, four different networks were trained, starting with different random glue weights. Generalization scores were calculated every 10 epochs. The best scores observed were 47%, 35%, 43%, and 52%. These are markedly *lower* than the scores at the end of glue training. Substituting further glue training for fine tuning did not cause such a drop in performance.

Note that this study differs from [Waibel *et al.*, 1989] in that it used a network with one hidden layer instead of two. It also uses different training data than their consonant recognition task. Under these different conditions, our experiments also show improved learning speed from problem decomposition. We have also explored the utility of fine tuning and found, in contrast to the previous work, that it may not always be useful. Finally, we have established that the network training time speedup is statistically significant.

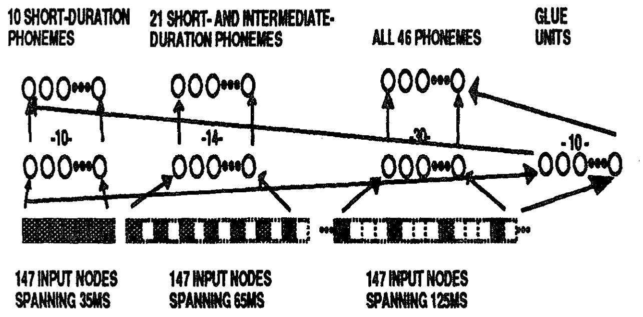
In the following section, we explore a further modification of the problem decomposition model. As in the vowel recognition task, this network is decomposed into subnetworks which are trained separately. However we also decompose the set of input units.

Problem Decomposition in a Large Speech Recognition Problem

[Kamm and Singhal, 1990] describe a neural network for learning part of an acoustic-to-phoneme mapping task. This network maps sequences of a spectral representation of a speech signal into sequences of phoneme classes. Although further high-level processing is necessary to determine the exact word pronounced (by disambiguating phonemes via context and other constraints), this initial phoneme determination is a critical phase of speech recognition.

The training set for this experiment was the DARPA acoustic-phonetic corpus [Fisher *et al.*, 1987], which contains recordings of continuous utterances from a large number (630) of talkers. This corpus is used extensively in phoneme recognition research.

The decomposition strategy and training methodology described in the previous section were applied to this more complex task. A network was built with the architecture shown in Figure 4. Here, each of three subnetworks view



10 SHORT-DURATION PHONEMES
21 SHORT- AND INTERMEDIATE-DURATION PHONEMES
ALL 46 PHONEMES
GLUE UNITS
-10-
-14-
-30-
-10-
147 INPUT NODES SPANNING 35MS
147 INPUT NODES SPANNING 65MS
147 INPUT NODES SPANNING 125MS

Figure 4: Architecture of the decomposed phoneme classification network

input data spanning a different duration of a speech signal. [Kamm and Singhal, 1990] reported the construction of three non-decomposed networks, one for each input duration. Their results showed that networks with input durations of 35ms and 65ms had much higher performance for phonemes with short average duration than for the longest phonemes (diphthongs). Based on this finding, we decomposed the problem on the output units so that the two subnetworks with short-duration input spans "specialized" on subsets of phonemes with short and intermediate average durations.

The three subnetworks were trained individually on a 200-sentence training set (108983 patterns), and then their weights were placed into the combined network, along with glue weights that were randomly initialized in $[-0.3, 0.3]$. These low magnitudes were chosen to give the network flexibility in moving glue hyperplanes. Glue training and fine-tuning were then performed.

As described in more detail in [Pratt and Kamm, 1991], network performance was evaluated by calculating a normalized average hit rate (AHR) and false alarm rate (AFA) among the top 3 phonemes out of the 46 target classes. Learning time was measured as the number of arithmetic operations required to calculate unit activations and weight update values. The table in Table 2 shows training conditions and results (as tested on an independent 200-sentence set), as well as those for [Kamm and Singhal, 1990]'s best network (with 125ms-duration input). The second and third columns in Table 2 show values of η and α selected (by training several networks to a couple of epochs) for training.

If we measure overall performance as the top-3 AHR minus top-3 AFA, then the final network obtained after fine tuning has $\frac{62.4-3.5}{61.7-3.1} > 100\%$ of that of the [Kamm and Singhal, 1990] network, after $\frac{1.1\times10^{11}}{10.62\times10^{11}} \approx 10\%$ of the operations.

Network	α	η	Epochs	Total op's $\times10^{11}$	top 3 AHR	top 3 AFA
K&S net.	.1	.1	300	10.62	61.7	3.1
35ms subnt	2.5	.1	7	.06	70.1	7.4
65ms subnt	1.2	.2	3	.04	65.2	5.8
125ms subnt	3.0	.01	10	.35	33.1	4.3
glue training	2.0	.5	9	.46	56.5	4.0
fine tuning	2.0	.5	2	.19	**62.4**	**3.5**
Decomposed Network Total			31	1.1		

Table 2: Conditions and results for the phoneme recognition network

This is a substantial speed-up over the previous training time.

More details about the acoustic-phonetic network can be found in [Pratt and Kamm, 1991].

Discussion

The vowel recognition and phoneme recognition problem decomposition experiments demonstrated some of the same major findings as the less complex pilot study, despite fundamental differences in the nature of the task decomposition among the studies. In all three studies, time to convergence was shorter when some of the weights in the network were pre-set than when all weights were set to random initial values. Furthermore, when the pre-set weights had relatively high magnitudes, the networks were able to retain them during further training. In addition, the vowel task indicated that fine-tuning of a decomposed network may not always improve performance.

Note that, by considering transfer of partially incorrect weights, we are addressing a more complex issue than if multiple network weight sets were simply glued together in a modular system at run time. In contrast, we have explored how back-propagation learning uses both error-free and errorful pre-set weight subsets.

These experiments leave many open questions for further research, including the following topics.

- More systematic studies of back-propagation dynamics on complex tasks should be done in order to further explore the pilot study hypotheses.
- More work is necessary to establish how problem decomposition can best be combined with the oversized training methodology used on the vowel recognition task.
- The speed-up observed in decomposed vs. non-decomposed networks might be due not to the decomposed training methodology, but to the fact that that networks were trained using a constrained topology. This possibility should be explored empirically.
- It is interesting that weight magnitudes from prior training worked so well in the problem decomposition tasks. This may have been due to the high network weights generated by subnetwork training. For example, the average subnetwork weight magnitude in the vowel study was 15.5. For source and target network tasks that differ substantially,

careful magnitude tuning may be necessary, to avoid local minima. Furthermore, for a technique like that described in [Towell *et al.*, 1990] (which uses weight sets obtained by means other than prior network training) more attention to weight magnitudes may be helpful in dealing with potentially incorrect initial weights.

- The network decomposition on the vowel recognition task was chosen arbitrarily. It is important to characterize the nature of decompositions for which speedup occurs. Careful analysis of subproblem interactions should aid in this endeavor. Also, further experiments with different arbitrary decompositions should indicate sensitivity to particular decompositions. It should also be fruitful to explore automated methods for guiding problem decomposition using domain knowledge.
- When training data is impoverished (noisy, incorrect, incomplete), it may be possible to achieve a performance improvement by using pre-set weights. Although this question has been explored in related contexts [Towell *et al.*, 1990], [Waibel *et al.*, 1989], an important open issue is whether direct network transfer produces significant performance improvement over randomly initialized networks.
- The model of transfer used here decouples initial weight determination from learning. Therefore, the learning algorithm can probably be changed (for example to Conjugate Gradient [Barnard and Cole, 1989]) without changes to the transfer process. A study should be performed to verify that transfer remains effective with this and other learning algorithms.
- Finally, our most active current area of research explores transfer between networks trained on different but related populations of training data, for source and target networks with the same topology. For example, speaker-dependent training may be sped up by transferring weights from a network trained on multiple speakers. The effectiveness of transfer should be evaluated under conditions of different relationships between source and target training data (i.e. superset$\rightarrow$ subset, subset$\rightarrow$ superset, disjoint but related populations, etc.).

Summary

We have addressed the question of how information stored in one neural network may be transferred to another network for a different task. We explored the behavior of back-propagation when some weights in a network are preset, and we studied the effect of using weights from pre-trained subnets on learning time for a larger network. Our results demonstrated that the relative magnitudes of the pre-set weights (compared to the untrained weights) are important for retaining the locations of pre-trained hyperplanes during subsequent learning, and we showed that learning time can be reduced by a factor of 10 using these task decomposition techniques. Techniques like those described here should should facilitate the construction of complex networks that address real-world problems.

Acknowledgments

Thanks to Steve Hanson, Siemens, Inc., and Bellcore for financial support during 1990. Thanks also to David Ackley, Sharad Singhal, and especially the anonymous reviewers, who provided helpful comments on previous drafts. Also, Michiel Noordewier and Haym Hirsh provided critical support and encouragement for the research program on which this paper is based.

References

Barnard, Etienne and Cole, Ronald A. 1989. A neural-net training program based on conjugate-gradient optimization. Technical Report CSE 89-014, Oregon Graduate Center.

Fisher, W. M.; Zue, V.; Bernstein, J.; and Pallett, D. 1987. An acoustic-phonetic data base. *J. Acoust. Soc. Am. Suppl.* 81(1):S92.

Fu, Li-Min 1990. Recognition of semantically incorrect rules: A neural-network approach. In Proceedings of Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE '90. Association for Computing Machinery.

Kamm, C. A. and Singhal, S. 1990. Effect of neural network input span on phoneme classification. In Proceedings of the International Joint Conference on Neural Networks, 1990, volume 1, 195–200, San Diego. IEEE.

Pratt, L. Y. and Kamm, C. A. 1991. Improving a phoneme classification neural network through problem decomposition. In Proceedings of the International Joint Conference on Neural Networks (IJCNN-91). IEEE. Forthcoming.

Robinson, Anthony John 1989. *Dynamic Error Propagation Networks.* Ph.D. Dissertation, Cambridge University, Engineering Department.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1987. Learning internal representations by error propagation. In Rumelhart, David E. and McClelland, James L., editors 1987, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press: Bradford Books. 318–362.

Shavlik, J. W.; Mooney, R. J.; and Towell, G. G. 1991. Symbolic and neural net learning algorithms: An experimental comparison. *Machine Learning* 6(2):111–143.

Towell, Geoffrey G.; Shavlik, Jude W.; and Noordewier, Michiel O. 1990. Refinement of approximate domain theories by knowledge-based neural networks. In Proceedings of AAAI-90, 861–866. AAAI, Morgan Kaufmann.

Waibel, Alexander; Sawai, Hidefumi; and Shikano, Kiyohiro 1989. Modularity and scaling in large phonemic neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37(12):1888–1898.

Weigend, Andreas S.; Huberman, Bernardo A.; and Rumelhart, David E. 1990. Predicting the future: A connectionist approach. Technical Report Stanford-PDP-90-01, PARC-SSl-90-20, Stanford PDP Research Group, Stanford, California 94305-2130.