INVERTING A CONNECTIONIST NETWORK MAPPING BY BACK-PROPAGATION OF ERROR

Ronald J. Williams
Institute for Cognitive Science
University of California, San Diego

ABSTRACT

The *back-propagation* learning algorithm (Rumelhart, Hinton, & Williams, 1986) for connectionist networks works by adjusting the weights along the negative of the gradient in weight space of a standard error measure. The back-propagation technique is simply an efficient and entirely local means of computing this gradient. Using what is essentially the same back-propagation scheme, one may instead compute the gradient of this error measure in the space of input activation vectors; this gives rise to an algorithm for inverting the mapping performed by a network with specified weights. In this case the error is propagated back to the input units and it is the activations of these units — rather than the values of the weights in the network — that are adjusted so that a specified output pattern is evoked. This technique is illustrated here with a small network which is a much simplified version of the NETtalk text-to-speech network studied by Sejnowski and Rosenburg (1986). The idea is to run this network backward so that it attempts to spell words based on their phonetic representations. This example further illustrates the use of this technique in a sequential interpretation setting in which phonemes are presented to the system one at a time and the system must refine its previous guess at the correct spelling as each new phoneme is presented.

INTRODUCTION

This paper explores the use of the technique of back-propagation of error (Rumelhart, Hinton, & Williams, 1986) to invert the mapping performed by a connectionist network. While the technique was introduced in that paper as a means of finding a set of weights which would achieve a certain mapping in a network of given topology, it is equally applicable to the problem of finding what input pattern would give rise to a specified output pattern in a network with given weights. Table 1 summarizes how it is possible to solve for any one of the three items *input pattern*, *weight matrix*, and *output pattern* given the other two, such that a network with that weight matrix maps that input pattern to that output pattern. The back-propagation input adjustment algorithm will be described further in

Table 1

Given	Solve For	Using
input pattern, weights	output pattern	forward propagation
input pattern, output pattern	weights	back-propagation learning
output pattern, weights	input pattern	back-propagation input adjustment

the next section, following which its use in inverting a particular network mapping will be explored.

Although no specific network implementation of the back-propagation mechanism is proposed here, these ideas hint at the intriguing possibility that there might exist a design for a network which contains within it the means for production, for comprehension, and for learning, all integrated together.

THE BACK-PROPAGATION INPUT ADJUSTMENT ALGORITHM

Just as the back-propagation learning algorithm (Rumelhart, Hinton, & Williams, 1986) is an incremental procedure for adjusting the weights in a network, the back-propagation input adjustment algorithm to be explored here is an incremental procedure. This means that it requires an initial "guess" at the input vector, which it successively modifies until the resulting input vector gives the desired output in the given network. The mathematical details of this algorithm, although straightforward, will not be given here, in the interest of brevity; suffice it to say that the algorithm moves down the negative of the gradient of the same squared-error performance measure used for the back-propagation learning algorithm. The difference is that this gradient is computed in the space of input vectors rather than in weight space. The derivation of this algorithm proceeds almost identically to the derivation of the learning algorithm, with the chain rule for partial derivatives giving rise to its back-propagation flavor, in which error-correction information is required to flow backwards along the connections in the network.

Just as in the learning case, back-propagation of error-correction information must be interspersed with forward propagation in the net to determine what additional adjustments are necessary. Thus the entire algorithm for inverting the network mapping for a particular specified output pattern consists of starting with an initial input pattern and modifying this pattern by repeated application of what will be called a *basic adjustment cycle*. Such a basic adjustment cycle consists of propagating activity forward in the network, back-propagating the error-correction information, and incrementing the input vector accordingly.

SOME SIMULATION RESULTS

The NETtalk text-to-speech network of Sejnowski and Rosenburg (1986) is a network whose input represents a seven-character window on a potentially much longer string of text and whose output represents the single phoneme which is appropriate for the character at the center of the window in the context of the remaining six characters. The network operates on an arbitrary-length text string by successively sliding its seven-character window along this string by one character at a time. Here we consider a drastically simplified version of such a net and study its ability to run "backwards" — i.e., its ability to spell a word given its phonetic representation. Furthermore, since the network represents only a single phoneme at a time, this problem will take on a sequential interpretation flavor: as successive phonemes are presented, the system will be forced to update its "preferred spelling" as these phonemes come along, rather than in parallel. It thus becomes interesting to examine the sequence of results obtained by the system.

The network used in these experiments is depicted in Figure 1. The input units encoded eight characters in each of three positions, with unused character/position combinations eliminated. The output units encoded nine phonemes. Both the input and the output representations were local rather than distributed. This was done strictly as a matter of convenience in setting up the network and also to make it easier to interpret arbitrary pattern vectors in a reasonable way, as will be discussed below.

The network was first trained (using back-propagation learning) to produce the appropriate phonemes for seven words. In the manner of NETtalk, each word was trained in each relevant position in the 3-character window. The training data is listed in Table 2. These words were chosen

Table 2

Input Word (3 positions)	Output Phonemes
can	/k/, /@/, /n/
car	/k/, /a/, /r/
con	/k/, /a/, /n/
wan	/w/, /a/, /n/
war	/w/, /c/, /r/
was	/w/, /^/, /z/
won	/w/, /^/, /n/

because of the interesting problems they present for a system trying to determine the spelling as the phonemes appear sequentially. Note that determination of the correct vowel cannot be made for some of these before the final consonant phoneme has been presented. Thus particular interest in these experiments was centered on the ability of the network to infer the vowel as each phoneme was presented.

Once the network had been trained to achieve essentially perfect performance, the weights were fixed; this network then formed the basis of the system on which all the experiments reported here were performed. In all these experiments, character positions to be determined had all their character units' outputs initialized to the same nominal value (typically .1). Also, for all the experiments the "solution" obtained by the system for any particular character position was interpreted to be that character having the largest output. This is consistent with the Sejnowski & Rosenburg interpretation of the "best guess" output of their system as the vector making the smallest angle with the output vector.

The results of the experiments are summarized in Table 3, where an asterisk is used to denote a character position which is to be determined. Experiments 4-7 involved presenting sequences of phonemes to the system, and these experiments were run as follows:

(1) The letter units were initialized to nominal values.
(2) The given phoneme was selected as the target output pattern to be achieved.
(3) The character units had their values adjusted via several iterations (typically 50, using a rate parameter of .1) of the basic adjustment cycle. At this point the target output was well matched by the output actually achieved using the adjusted input pattern.

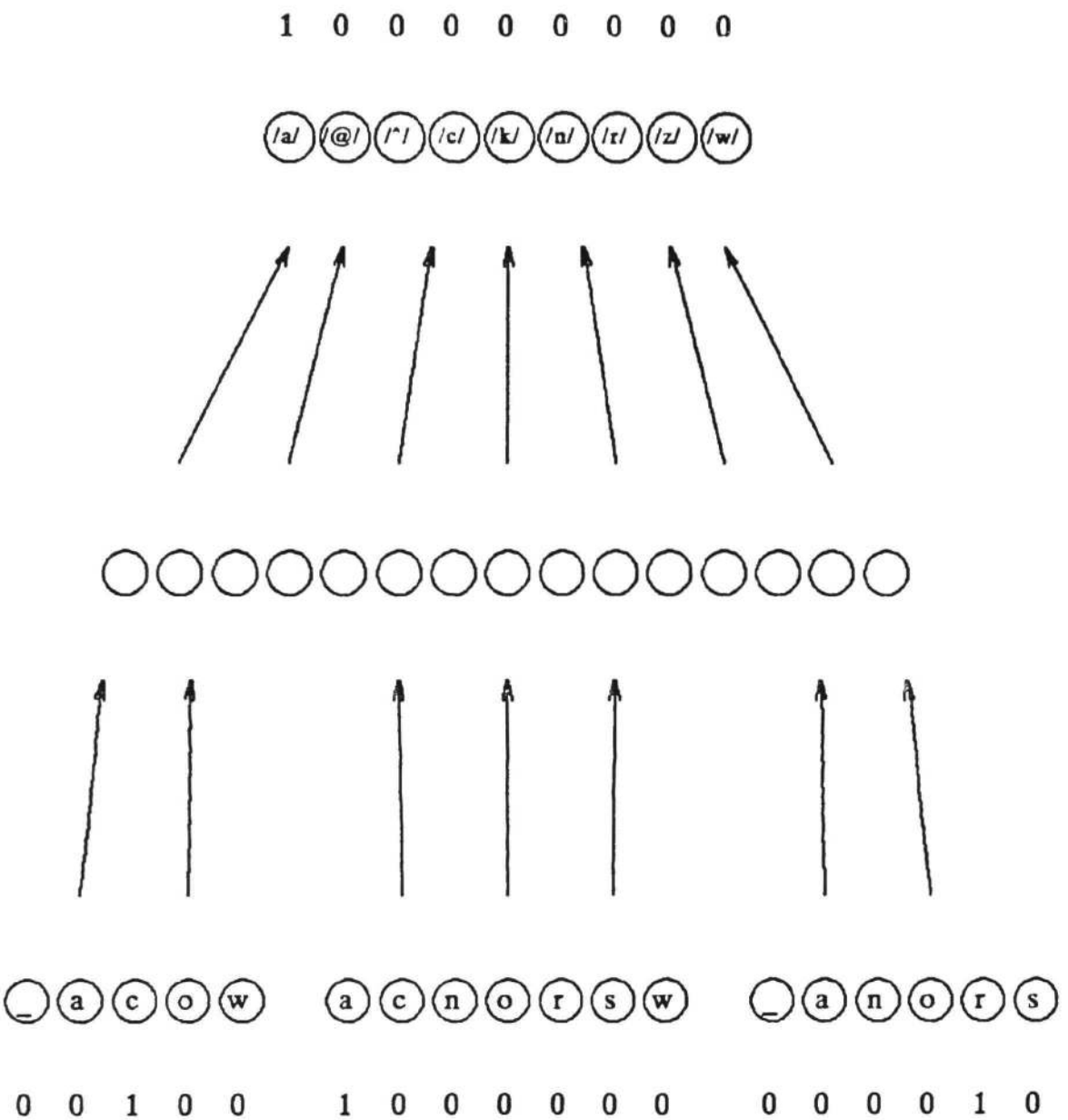

Figure 1. The network used in the experiments. The input layer is divided into three parts, one for each of three character positions. There is complete connectivity between layers. The input vector shown represents the word *car*. The corresponding output vector shown represents the appropriate phoneme /a/. The printed representation used here for these phonemes is taken directly from Sejnowski and Rosenburg (1986). The sounds they denote can be inferred from the description of the training data for the network given in Table 2. The underscore represents the <*space*> character.

(4) The output values of the character units were shifted one slot to the left in preparation for receipt of the next phoneme. This set of output values represented the starting point for the attempt to match the next phoneme. (The rightmost character slot had all its units' outputs set to the default nominal value.)

Steps (2)-(4) were repeated for each phoneme in the string of phonemes which the network was to spell. At each iteration of this cycle of steps the pattern of activity in the character units was examined at the end of step (3).

Table 3

Experiment	Target Phoneme	Character Units	
		Initial State	Winners After Convergence
1	/@/	•••	can
2	/a/	c•r	car
3	/a/	c•n	con
4	/w/ /^/ /n/	•••	_wa wos on_
5	/w/ /^/ /s/	•••	_wa wos as_
6	/k/ /a/ /r/	•••	_cr car ar_
7	/k/ /a/ /n/	•••	_cr cor on_

In every one of these sequential experiments there was ambiguity concerning the correct choice of vowel at the time that phoneme was presented; subsequent presentation of the next phoneme provided disambiguating information which enabled the system to choose the correct vowel even after the corresponding phoneme was no longer available to the system. In every case the system made the correct vowel its clear favorite once this disambiguating information was made available.

DISCUSSION

There are several remarks to be made here. First, the particular network used for these experiments was quite small and the inversion problems posed were quite simple. It will be interesting to see whether similar results are obtained if corresponding experiments are performed in a much larger network having many more input/output pairs "stored" in its weights.

Second, the problem of inverting a NETtalk-like grapheme-string-to-single-phoneme mapping was chosen because it illustrated not only the notion of inverting a mapping but also some other issues which are based on the observation that the network inversion problem and the network learning

problem are dual instances of the same general mathematical problem. As such, they are both underdetermined, in general, so that what must be sought are simultaneous solutions to multiple instances of such problems, or else solutions which are nearest, in some sense, to a given starting point.

Furthermore, the sequential nature of learning problems — in which the items to be learned are assumed to be experienced sequentially — is generally taken for granted, while the connectionist approach often suggests solutions to comprehension problems in which a great deal more parallelism is assumed (often by simply buffering a temporally extended input stream in order to make all components of it available for processing simultaneously). In the example considered here, a strictly sequential process was invoked in which the system was only allowed to examine one phoneme at a time. An interesting question is what to do at each step in order to get optimal convergence to the "correct" answer. The engineering topic of *recursive identification* (Ljung & Söderström, 1982) addresses such questions, although most such algorithms may be of limited applicability to these network problems since they are based on linear approximations. The importance of these sequential issues can be seen by noting that such algorithms applied to the learning problem would provide one-trial learning. While a parallel approach is possible even in the learning case, it is clearly inappropriate.[1]

Another remark concerns the implication of the duality between input vector and weight matrix for the connectionist approach. It is intriguing to speculate on how it might be possible to create models in which activations and weights play a more symmetric role. There are certainly precedents for such an enterprise. For example, models have been proposed which essentially replace weights by activation of units via *gating* connections on second-order *sigma-pi* units (Hinton, 1981; Rumelhart, Hinton, & McClelland, 1986; Williams, 1986). One such model is that of McClelland (1986). Also, some models have been studied which call for fast short-term changes in weights, which might be considered a means by which network weights are made to take on a role more like that of activation of units.

Finally, note that unlike most reports on connectionist-style research, this paper does not actually propose a network implementation of the computational technique suggested here. Rather, it suggests the utility of the back-propagation formalism in another setting besides that for which it was originally proposed. Results such as these suggest that treating back-propagation as a functional primitive in networks may lead to a number of elegant solutions to connectionist-style problems. It remains an open question just how such functionality may be implemented in more conventional forward-propagating network fashion. It is clear that the computation explored here has the flavor of a sequence of settlings, suggesting that a possible implementation might consist of a network designed to carry out this settling behavior.

---

1. In fact, it is not hard to devise a general procedure for constructing a larger network, with certain weights constrained to be equal, such that the sequential problem of determining the weights for a given collection of input/output pairs in a given network amounts to a single training instance in this new (possibly gigantic) network. Furthermore, the usual technique of interspersing the pattern presentations throughout training can be viewed as a time-shared, serial implementation of this parallel process.

REFERENCES

Hinton, G. E. (1981). A parallel computation that assigns canonical object-based frames of reference. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, 683-685.

Ljung, L., & Söderström, T. (1982). *Theory and Practice of Recursive Identification*. Cambridge: MIT Press.

McClelland, J. L. (1986). The programmable blackboard model of reading. In: Rumelhart, D. E. & McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 2: Psychological and Biological Models*. Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986). A general framework for parallel distributed processing. In: Rumelhart, D. E. & McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In: Rumelhart, D. E. & McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. Cambridge: MIT Press.

Sejnowski, T. J. & Rosenburg, C. R. (1986). *NETtalk: a parallel network that learns to read aloud*. Technical Report 86/01, Department of Electrical Engineering and Computer Science, Johns Hopkins University.

Williams, R. J. (1986). The logic of activation rules. In: Rumelhart, D. E. & McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. Cambridge: MIT Press.